Search Engine for Song

Submission For
ITCS414_Information Storage and Retrieval
The Project Phase 2

MAHIDOL UNIVERSITY
FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

PREPARE BY

6488129 Karuetat Panaspraipong

6488196 Waritthorn Mahasiripanya

Introduction

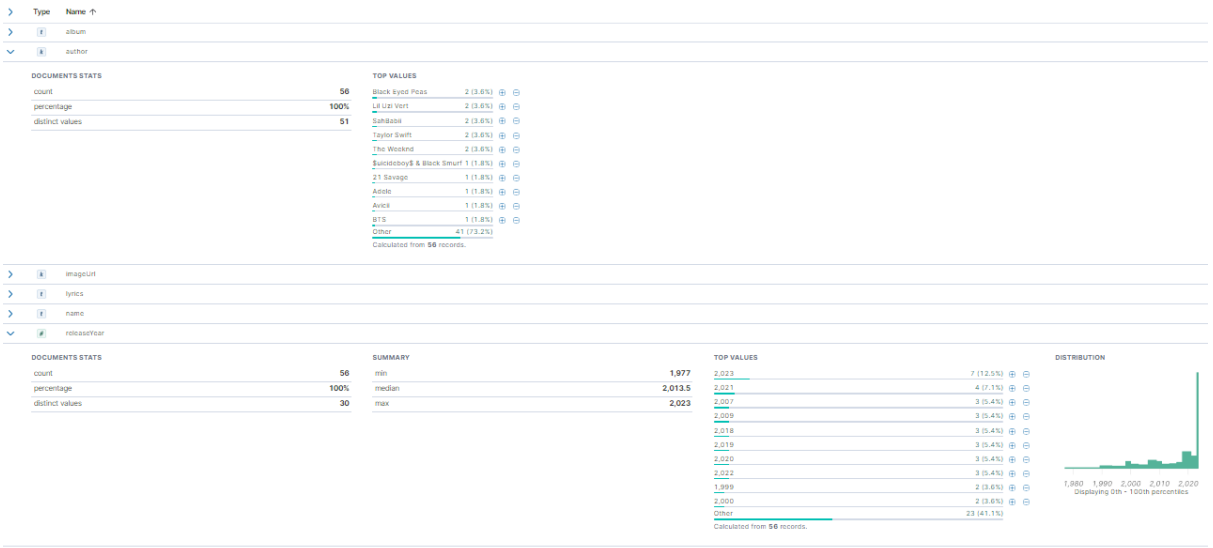
The implemented system aims to provide a search functionality for a music database stored in Elasticsearch. The dataset is in JSON format and contains information such as song name, author, release year, album, lyrics, and image URL. The primary goal is to allow users to search for songs based on various criteria and retrieve relevant results. We try to solve these problem: Efficient Music Search, Partial Matching, Fuzzy Matching, and Scalability.While there may be existing search engines for music databases, the specific implementation using Elasticsearch, Python, and associated technologies provides a customizable and potentially more efficient solution. The choice of tools like Visual Studio Code, HTML, CSS, Python, Elasticsearch, and Kibana allows for a flexible and powerful system.

Implementation

i. Data collection, example documents, and data statistics

File type: JSON

Data Collection: name, author, relaseyear, album, lyric, imageUrl.



ii. Tools and software

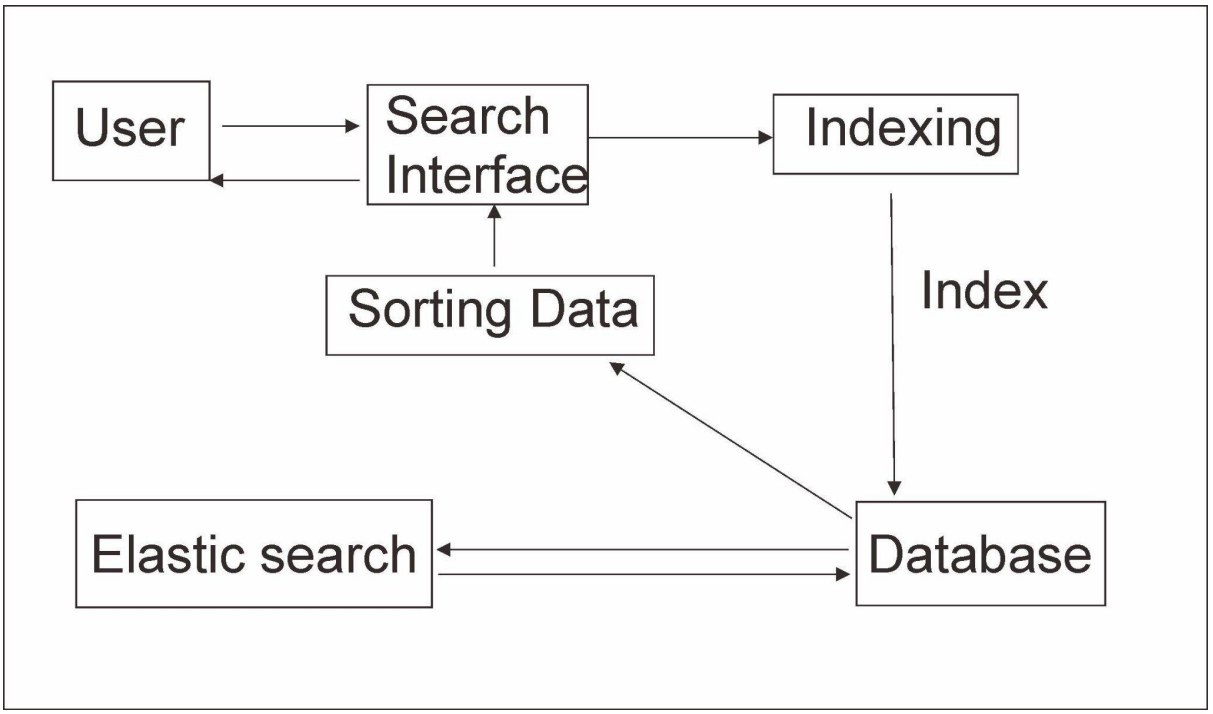
Visual Studio Code / HTML, CSS

Elastic Search / Search

Kibana / Search

Python / Python Running code

iii. System diagram



iv. Snapshots of the system

search_app.py

```
es = Elasticsearch("https://localhost:9200", http_auth=("elastic", ELASTIC_PASSWORD), verify_certs=False)
app = Flask(__name__)

@app.route('/')
def index():
    return render_template('search-song.html')

@app.route('/search')
def search():
    page_size = 10
    keyword = request.args.get('keyword')

    if request.args.get('page'):
        page_no = int(request.args.get('page'))
    else:
        page_no = 1

    body = {
        'size': page_size,
        'from': page_size * (page_no-1),
        'query': {
            'multi_match': {
                'query': keyword,
                'fields': ['name^5', 'author^4','releaseYear^4','album^4','lyrics^3'],
                'fuzziness': 'auto',
            }
        }
    }

    res = es.search(index='song-bulk', body=body)

    hits = [{'name': doc['_source']['name'], 'author': doc['_source']['author'], 'releaseYear': doc['_source']['releaseYear'],
            for doc in res['hits']['hits']]
    page_total = math.ceil(res['hits']['total']['value']/page_size)
    return render_template('search.html',keyword=keyword, hits=hits, page_no=page_no, page_total=page_total)
```

search.html

```
    <h1>Search Song</h1>
</header>

<div class="search">
    <div class="text">
        <p>Your search results:</p>
    </div>

    {% for doc in hits %}
        <div class="result-item">
            <h2>{{ doc['name'] }}</h2>
            <p>{{ doc['author'] }}</p>
            <p>{{ doc['releaseYear'] }}</p>
            <p>{{ doc['album'] }}</p>
            <p>{{ doc['lyrics'] }}</p>
            <img src="{{ doc['imageUrl'] }}" alt="Album Cover">
            <hr>
        </div>
    {% endfor %}
</div>

<nav aria-label="Page navigation">
    <ul class="pagination justify-content-center">
        {% if page_no != 1 %}
        <li class="page-item"><a class="page-link" href="?keyword={{ keyword }}&page=1">First</a></li>
        <li class="page-item">
            <a class="page-link" href="?keyword={{ keyword }}&page={{ page_no-1 }}" aria-label="Previous">
                <span aria-hidden="true">&laquo;</span>
            </a>
        </li>
        <li class="page-item"><a class="page-link" href="?keyword={{ keyword }}&page={{ page_no-1 }}">{{ page_no-1 }}</a></li>
        {% endif %}
        {% if page_total != 1 %}
        <li class="page-item active"><a class="page-link " href="?keyword={{ keyword }}&page={{ page_no }}">{{ page_no }}</a></li>
        {% endif %}
        {% if page_no < page_total %}
```

search-song.html

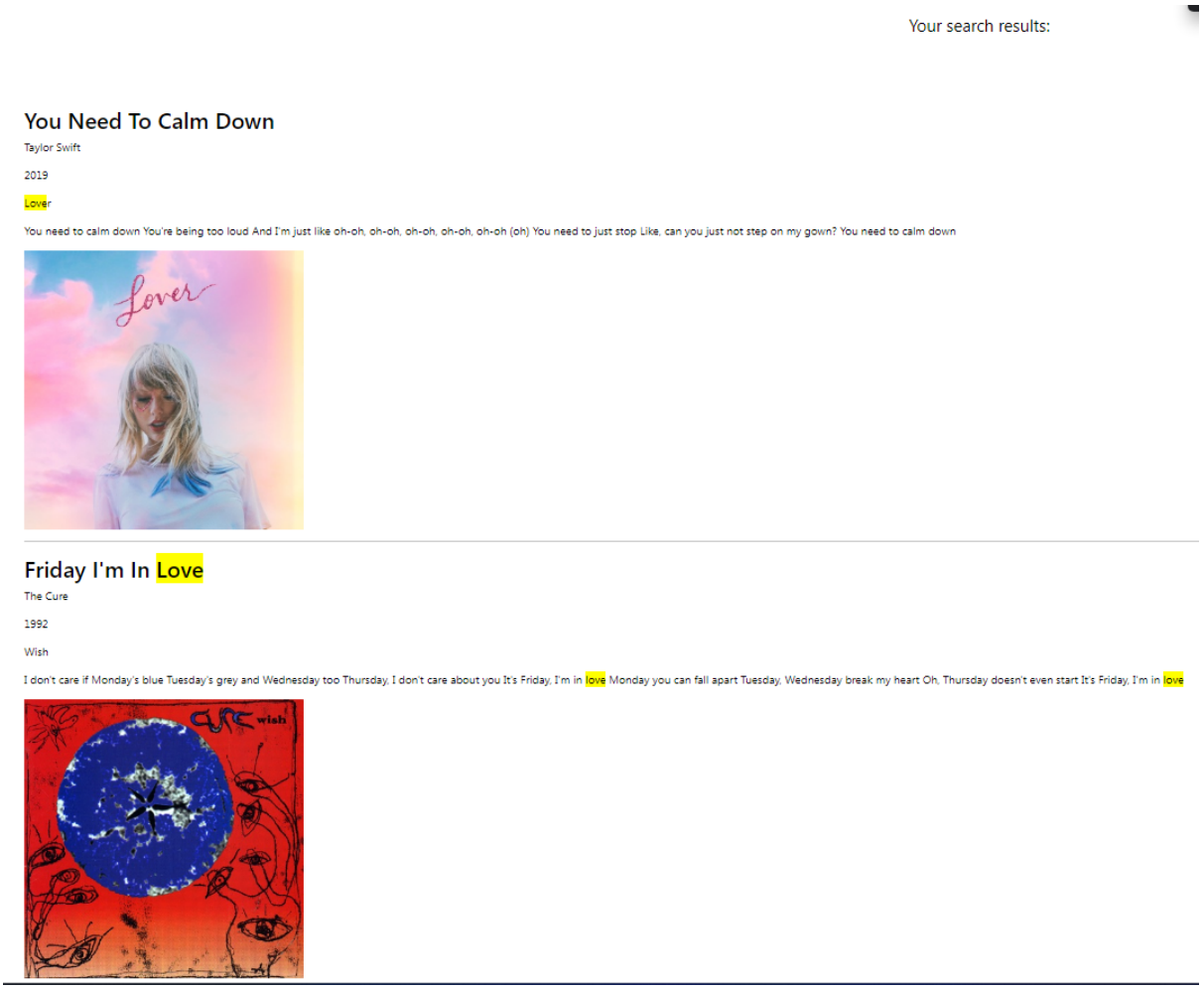
```
<nav class="user-navbar">
    <ul>
        <li><a href="/">Search Song</a></li>
    </ul>
</nav>

<header>
    <h1>Search Song</h1>
</header>

<section class="search">
    <div class="text">
        <h2>Search For Song</h2>
        <h2>Search here!</h2>
    </div>

    <!-- please make all buttons orange, including this one! -->

    <div class="search-container" >
        <form class="search-bar" method="GET" action="{{ url_for('search') }}">
            <br>
            <input class="keyword" type="text" name="keyword" placeholder="Insert
            <br>
            <button type="submit" class="button-search">Search</button>
            <br>
        </form>
    </div>

</section>

<section>
    <div id="results" style="background-color: black;"></div>
</section>

<footer>
    <p>SEARCH ENGINE FOR SONG</p>
</footer>
```

v. Example step-by-step search sessions that highlight the following functionality:

1. One-word query

Search Input: love

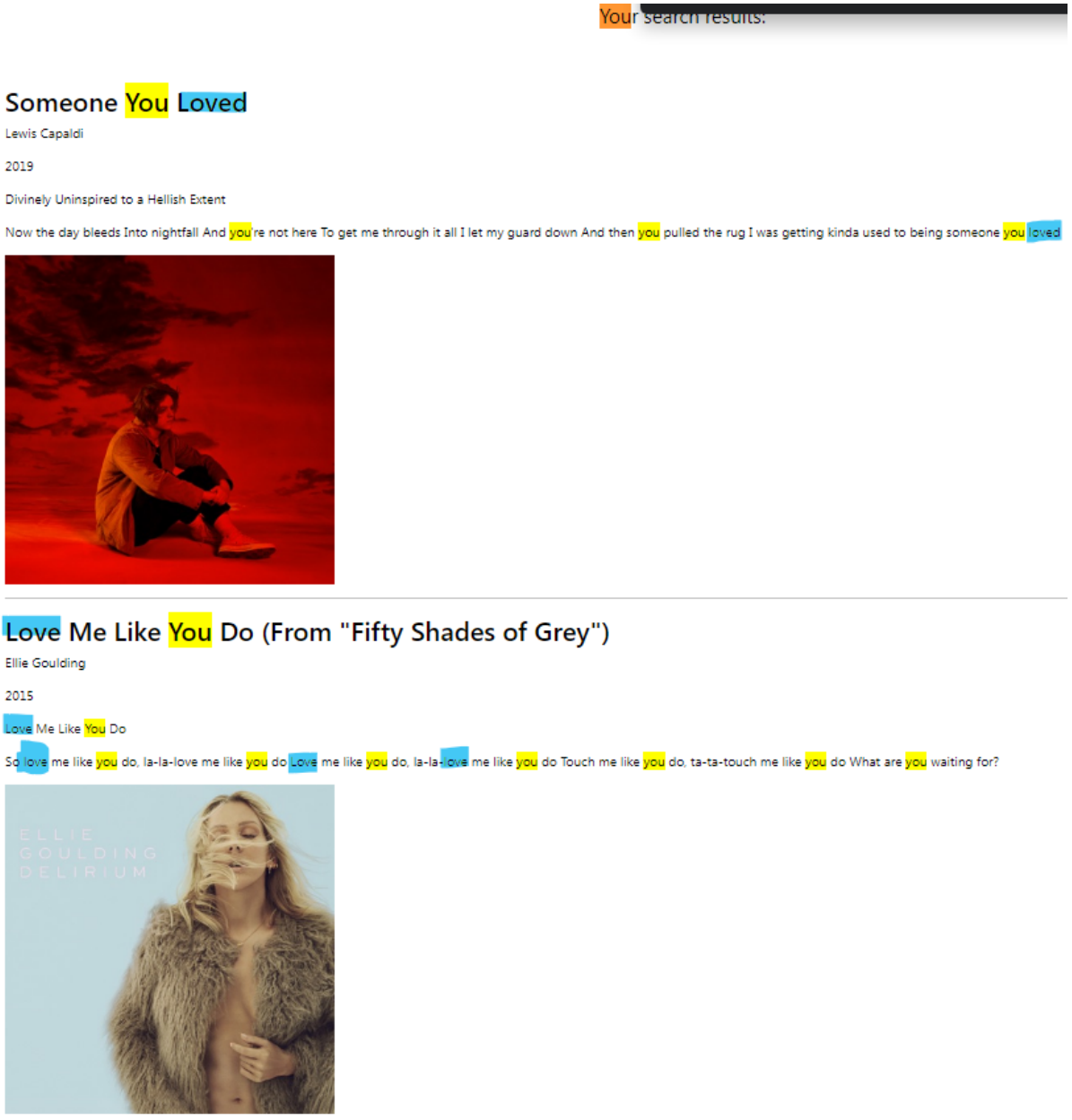
Album Lover got a higher score since we employed the fuzziness tool. The elastic search then transforms lover to love, giving in a higher score than Friday I'm in Love, which includes more "love" phrases.



2. Multiple word query

Search Input: love you

The first song, "someone you loved," receives a better score since it contains fewer words than the second, "love me like you do," which contains several queries of the words "love" and "you," but receives a lower score.



3. Partial match

Search Input:hat

"Thong Song" is first because the query "hat" is close to each other, despite the fact that all about that have "hat" in the song title.

Your search results:

Thong Song

Sisqó

1999

Unleash the Dragon

She had dumps like a truck, truck, truck Thighs like what, what, what All night long (c'mon) Let me see that thong

All About That Bass

Meghan Trainor

2014

Title (EP)

Because you know I'm all about that bass 'Bout that bass, no treble I'm all about that bass, 'bout that bass, no treble

4. Ranking

To match the question, we employ multi-match. The elastic sort high to low search score is as follows: name is highest, author, release year, and album have the same score is second, and the final lyric has the lowest score. To rectify the term, we employ auto fuzziness.

```
body = {
    'size': page_size,
    'from': page_size * (page_no-1),
    'query': {
        'multi_match': {
            'query': keyword,
            'fields': ['name^5', 'author^4','releaseYear^4','album^4','lyrics^3'],
            'fuzziness': 'auto',

        }
    }
}
```

Discussion

i. Limitations of the ElasticSearch code snippet:

The code snippet does not handle potential exceptions or errors that may occur during the Elasticsearch search operation. It lacks input validation for the page_size and page_no parameters, making it susceptible to unexpected values. The 'fuzziness' parameter in the multi_match query might lead to broader search results, impacting precision.

ii. Technical difficulties, challenges, and lessons:

Dealing with large datasets could lead to performance issues, and optimizing the search operation may be challenging. Managing and handling Elasticsearch index mappings, especially when dealing with dynamic data, can be a complex task. Ensuring proper security measures, such as authentication and authorization, for Elasticsearch access can be challenging.

iii. Opportunities for future improvements:

Implementing result pagination more efficiently, considering potential performance implications with large datasets. Enhancing error handling to provide meaningful feedback in case of issues during Elasticsearch queries. Exploring and adopting the latest features and improvements in Elasticsearch to optimize search performance. Considering additional search functionalities or filters to improve the user experience, depending on specific use cases.

Conclusion

In conclusion, the implemented music search system demonstrates a robust approach to retrieving relevant information from a music database using Elasticsearch. The system accommodates various search scenarios, including one-word queries, multiple word queries, and partial matches, providing users with a versatile and user-friendly experience.

The choice of tools, including Visual Studio Code for development, HTML and CSS for the user interface, Python for handling backend logic, and Elasticsearch for efficient search operations, contributes to the system's flexibility and scalability. The integration with Kibana enhances the visualization and monitoring capabilities, allowing for a more comprehensive understanding of the data and search performance.

Despite the success of the implementation, it is essential to acknowledge certain limitations. The code snippet provided lacks comprehensive error handling, and potential issues with input validation could be addressed for a more robust system. Additionally, while the system is designed to handle large datasets, ongoing optimization may be necessary to maintain optimal performance as the database grows.

Looking ahead, there are opportunities for future improvements. Enhancing result pagination, exploring additional Elasticsearch features, and implementing more sophisticated error handling are areas where the system can evolve. Moreover, considering security measures, such as authentication and authorization for Elasticsearch access, should be a priority for a production-ready system.

The implemented music search system addresses specific challenges related to efficient and flexible search functionality within a music database. As technologies evolve, continuous refinement and adaptation will be crucial to ensuring the system remains effective and meets the needs of users in an ever-changing landscape.