TTDS Group Project – MIO – Movie-in-One Search Engine

S2166777, S2149550, S2154448, S2025818, S2159302, S2021327

1 Introduction

Nowadays, film has become the main entertainment and artistic expression of people. In the meantime, movies have become a common social topic in people's daily lives. Under social circumstances, people are always talking about new movies or classic movies. At this point, a lightweight and convenient movie search tool, providing fast and convenient movie search and display services, is urgently needed by users.

Besides, in leisure time, people tend to find a suitable movie to watch according to some existing information. The information could be the name of the movie of snipped information about actors, while it can also be the type of the film or the year it was released. Sometimes, they have a specific title or name of the file to search. Often, they just have a vague genre or release year.

MIO, the search engine, is targeted at users who have a specific movie search purpose or just want to find a proper movie with some filtering information. For users who could remember the title of the film, MIO could return the proper film quickly. On the other hand, if users just have an actor's name or a movie category, MIO could return satisfactory searching results with the ranking of film scores, with which users could have some ideas and advice for movie choice. At the same time, MIO provides well-designed user interfaces as well as fluent interaction experience, sparing no efforts to make users comfortable.

To be able to retrieve results with high accuracy, a large database of movies and relevant information is necessary. A public movie database, IMDB, was employed by us to retrieve all relevant data we needed. We grabbed all the data from the IMDB database for around 100,000 movies, which covers almost all movies from January 1, 2010, to December 31, 2021. For each movie, we collected all relevant information about it, containing 25 parameters. In total, there are around 2,500,000 data slots in our database.

To arrange this data better and make it quicker for calling with the front end, we decided to use MongoDB (MongoDB, 2022). Since MIO provides various ways for users to conduct searches, we created an inverted index aiming at different searching ways, including title, year of release, celebs, and genre. In each index file, we extracted necessary data with the corresponding 'oid' to call the movie in MongoDB. Besides, we choose the 'pkl' format to store all inverted indexes, thus speeding up the searching and matching.

For various searching ways, we divided them into keywords search and category search. We employed the BM25 ranking algorithm for dealing with the keywords search. At the same time, we defined a ranking algorithm with scoring and calculating weights. Besides, we used some weighting algorithms to create a 'list', making the search results more consistent with user expectations.

What's more, we also implemented some advanced features. We developed a global search function, which would shorten users' operation paths significantly. Based on the keywords entered by the user, MIO would conduct a global search and present the results to the user in groups. Meanwhile, MIO has also been equipped with a spell-check function to help users avoid errors caused by wrong spelling.

The report is organized as follows. In Section 2 we describe the system architecture and the technologies used in the project. In Section 3, we explain the methodology used for collecting and storing the dataset of movies. The pre-processing methods and indexing algorithm of the raw movie data are introduced in detail in Sections 4 and 5. In Section 6, we illustrated BM25 and weighting algorithms for keywords search and category search respectively. And some advanced features we designed for MIO are shown in Section 7. The design and implementation process for Graphic User Interfaces and interactions are well introduced in Section 8. Section 9 tells the design of the API. The evaluation of the project and future work is discussed in Section 10 and Section 11 contains individual contributions of each team member.

2 System Design

Designing the whole system architecture, functions and interactions is the first step of our work. MIO system can be split into two parts, the server-side and the client-side. Server-side modules process the search requests that can be received from multiple clients and return the results while the client-side consists of the GUI and its parts for connecting with the server-side. All parts related to the server-side are introduced in Sections 3, 4, 5, 6, and 7. And the GUI and interaction parts are discussed in Section 8 while all APIs for connection is written in Section 9.

3 Data Collection and Storage

To implement MIO, the search engine, one of the significant steps is collecting a large dataset of movie information, including titles, year of release, actors, and genres. All movie information was taken from the IMDB database. IMDB provides public datasets containing various information associated with every unique 'oid'. Moreover, we retrieved more detailed information about each movie that is only available on the HTML page of the IMDB website (IMDB, 2022a).

3.1 IMDB Information

From IMDB public dataset, we choose movies with the year it was released. We decided to use popular movies released in the last 11 years. We ranked all the movies from those years in ascending order of popularity and selected and downloaded the top 100,000 movies out of a total of 130,000.

Based on the requirements and data collecting regulations, we built a "spider" with regular expressions in Python to extract all information for each movie from the IMDB website. Finally, a txt file was generated, in which all movie information was stored with specific

format and parameters. During the extraction process, we visited the page of IMDB websites:

- imdb.com/search/title/[filtering condition] (IMDB, 2022a) - in which movies are filtered with a time of release. We selected films released between 2010-01-01 and 2021-12-31 (Sorted by Popularity Ascending). This web page contains all the information we need for each movie.

All features and parameters that we get from IMDB for each film are Title, Year, Rated, Released, Runtime, Genre, Director, Writer, Actors, Plot, Language, Country, Awards, Poster, Ratings, Metascore, IMDb rating, IMDB votes, imdbID and so on (shown in Figure 1).

3.2 Database

After extracting all the data, we needed from IMDB public database, we decided to use MongoDB for data storage and building the database we would employ in further implementation due to its natural compatibility with JSON format since our API is RESTful and the JSON data can be easily transferred to the client to construct JavaScript object and be used to render a webpage.

Since each MongoDB document has an oid field that uniquely identifies itself, we plan to construct the index into a python dictionary file that is small enough to fit into the memory and retrieve the oid of relevant movies based on the given query using that index. Then use pymongo to construct a mongo client to get all movie data contained in the retrieved oid list from the MongoDB and serve that as JSON response to the client.

4 Pre-processing

To pre-process the data, we followed the four steps: 1.Tokenization, 2.Case-folding, 3.Stop-word checking, 4.Stemming

For Tokenization, we simply used the '*findall*' function in package '*re*' to extract all the sequences of words and numbers as tokens. With such a function, all letters and numbers were preserved but punctuations were discarded. Brackets will be treated as operators and separated from the operand if it is used for specifying the precedence. In other words, the brackets used for indicating precedence are differentiated from ones in a proximity query. With such an assumption, splitting with the space character is the first step for tokenization. At the same time, the proximity and the phrasal operands have been broken because they contain a space character. With tracking the start and the end symbol of each operand, fractions could be combined as a whole, thus finishing the tokenization.

During the process of case-folding, all letters in the query were directly transferred into the lower case with the function '*lower()*'.

Similarly, for Stemming (Barla Cambazoglu, 2010), the '*Porter Stemmer*', from the '*nltk*' package, was employed to stem each token.

5 Indexing

Having finished pre-processing raw data of films, we put attention on creating indexes for the data. MIO provides users with various ways to search movies, including by the title, actors, year of release and genres. For different search methods, we need to build different indexes.

For searching in the 'Genre' category, we extracted all genres, which stored as 'Genre' parameter, in data, and stored them one by one with 'dictionary' data structure accompanied with the 'oid' of the specific film. When traversing all movies, if the movie genre is the same, the movie will be stored in the same dictionary. In this way, while building the index, we also achieved a simple classification of movies.

Similarly, for searching in the 'Year' category, we extracted all years, which stored as 'Year' parameter, in data and stored them one by one with 'dictionary' data structure accompanied with the 'oid' of the specific film. When traversing all movies, oids with the same year will be stored in the same key of the dictionary.

With the same method and process, we also built indexes for 'people' and 'title'. Finally, we wrote indexes into 'pkl' files, to improve the search speed.

6 Ranked Retrieval

According to W. Bruce (Barla Cambazoglu, 2010), the output produced by a good model should correlate well with human decisions about relevance. This is not only dependent on a good performance indexing method, but also on a set of ranked retrieval algorithms that can return results with high relevance to the user's search. In the MIO system, the ranked retrieval can be divided into two main functions: keywords search and category search. Keywords search means that the system returns a list of movies that are most relevant to the search term or key phrase. Category search means that after users select a year or a movie genre, the system returns a list with the most authoritative movies in this specific year or genre.

6.1 Additional Features

We use authoritative of each movie as additional features. Our definition of authoritative is based on the ranking scores and the number of votes of each movie. Specifically, we get the movie’s rating scores and the number of votes from the movie information database of iMDB. By exploring the data of the movie, we find that the scale of these two indicators is very different, which the number of votes ranges from tens to millions, while the rating scores range from 0 to 10. Therefore, we normalize these two kinds of features to the range from 0 to 1. We also find that the distribution of these two features is very different, which the rating scores is more like a normal distribution, while the number of votes is close to long tailed or tilted distribution, so we use Log transformations to let the number of votes to be more like a normal distribution before normalizing.

Therefore, the procedures of building additional features are:

(1) We first use Log transformations to the number of votes to make it more like a normal distribution.

(2) Then, we normalize both rating score and the number of votes of each movie and add them together using below formula:

$$Authoritative\ score = \frac{rating - \min(rating)}{\max(rating) - \min(rating)} + \frac{log(votes) - \min(log(votes))}{\max(log(votes)) - \min(log(votes))}$$

6.2 Category Search

The category search includes Genre search and Year search. When users search a particular year or genre, the system will firstly use inverted index to find all the related indexes of movies for this particular year or genre. Then, the system will rank all related movies by the authoritative score of each movie we create in 6.1. Therefore, the system can get a list of indexes ("oid" in this case) which is ranked by authoritative score, then related pages can be find based on these indexes. Finally, the result is displayed on the search result page.

6.3 Keywords Search

For the Keywords search part, we finally decided to use the BM25 algorithm (Barla Cambazoglu, 2010). At the beginning of the project, we conducted effect testing experiments on both TF-IDF (Barla Cambazoglu, 2010) and BM25 algorithms. The experimental results show that the TF-IDF algorithm increases the score as the value of word frequency (TF) increases. The BM25 algorithm, on the other hand, tends to a constant value of the score as the word frequency (TF) is gradually increased. BM25 improves the stability of the TF-IDF algorithm to some extent. Moreover, the results of TREC retrieval experiments show that the BM25 algorithm performs well in commercial search engines, especially web search engines (Barla Cambazoglu, 2010).

The keywords search method is used for both title search and celebrity search. The model is shown as below：

$$BM25(t,d) = \frac{tf_{t,d}}{k * \frac{L_d}{\overline{L}} + tf_{t,d} + 0.5} \times log_{10}(\frac{N - df_t + 0.5}{df_t + 0.5})$$

Assume k = 1.5.

We first preprocess the query and transform them into a list of tokens, then use the inverted index created in section 5 to retrieve all related indexes. In this section, a dictionary that contains indexes and related titles or celebrities' tokens is also created and named as the token dictionary. We use the token dictionary to output the related tokens corresponding to related indexes we get from inverted indexes. The BM25 algorithm is used to obtain the correlation scores of query tokens and related tokens in the token dictionary. The scores are normalized using the same formula in 6.2, then the **correlation score** and the **authoritative score** described in 6.1 will be added together as a **general score**. The general score can represent the popularity, movie quality, and relevance to query at the same time. Then the general scores are sorted from the highest to the lowest to form a

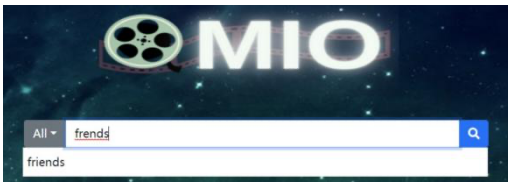
dictionary set under this search, with indexes of movies and scores corresponding to each other. Finally, the information related to movies with high BM25 scores is displayed on the search result page from highest to lowest.

7 Advanced Search

After confirming that all features in our search engine worked well, we investigated some advanced features including spell-checking, global search, and search prompt. After studying several approaches on search prompts, we decided not to implement this feature. Because it would reduce the system performance and cause some errors.

7.1 Spell-check

The spell-check function speculates on returning movie results that are more in line with the user's expectations. For example, when a user enters the word 'frends', the system will suggest a correct spelling 'friends'. The user can be prompted to click on the prompted word to search again.



After testing the effectiveness and performances of various spelling algorithms, we chose a statistical method based on Bayesian inference, which can process a large amount of text in a short time and return a high accuracy (90%). This method can select the most probable correct word among many alternative words in case the user enters the wrong word. The formula is as follows:

$$P(c|w) = \frac{p(w|c) * p(c)}{p(w)}$$

Firstly, a text library (text) is created and each word in the text library is taken out. The dictionary structure is built by the ‘Train’ function and the number of appearances is calculated. Combine with the ‘words’ function to generate a dictionary of word frequencies and calculate the distance between the words entered by the user and the words in the lexicon. Finally, the most likely word that the user wants will be displayed on the home page.

7.2 Global Search

In the first version, users need to select the search category first then search movies under that category. We add the global search function to improve the user's search experience. After users enter the content, the page displays information about the three most relevant movies under each category, which is more convenient.

8 GUI and Interactions

For the Graphical User Interface (GUI), after collecting users' needs with our team members, we designed the low-fidelity prototype in Figma, in which all interfaces and interaction ways are well designed.

8.1 The Interaction Process

When users use MIO, the whole operating process is shown in Figure 2.

Once they open the MIO website, the initial page, the large search box, will be presented to them. After users choose the searching category, including title, year, genre and celebs, the searching content can be entered. Based on the search results, different pages will be displayed to users.

If there is no matched result got from the database, MIO will show the 'No Result' page to users.

Otherwise, search results will be displayed with some regulations and restrictions. Once users press one of the searched movies, the website will jump to the detail page of the movie, in which users can check detailed information, including actors, directors, year of release, etc.

8.2 GUIs in detail

With the whole interaction process loop, we designed specific GUIs.

As shown in the following figures, interactions for the initial page are illustrated. Users choose the search category with the scrolling choice box, which contains All, Title, Year, Genre and Celebs. After the search content is entered, the search action will be conducted with the press of 'GO!'. (Figure 3 and 4)

If there is no result from searching, an apology will be presented to users with some prompt. (Figure 7)

The search results will be displayed with some specific regulations. All search results are sorted in order of relevance. (Figure 5)

- For the same relevance scores, results are sorted by movie Ratings.
- If users choose the filter as 'Year', the search results are movies released in the specific year, sorted by movie Ratings.
- If users choose the filter as 'Genre', the search results are movies tagged as the specific genre, sorted by movie Ratings.
- If users choose the filter as 'Celebs', the search results are movies acted by the specific person, sorted by movie Ratings.

At the same time, if users choose the filter as 'ALL' at the beginning of the search box, results will be grouped with 'Title', 'Year', 'Genre' and 'Celebs' automatically. In each group, the relevant results will be presented. (Figure 6)

On this page, each result of movie will be illustrated on a specific card, which shows some brief information about the movie to help users judge and identify. The left side of the card

is the main poster of the movie while the right side shows the title, runtime, genre, year of release, rating, plot outlines, directors, writers, and actors. When users press at the card, the detail page of the movie will be navigated to. (Figure 8)

The detail page shows all information we have in the database about the movie, providing users with comprehensive and reliable information.

8.3 Implementation of GUI

For implementing GUIs we have designed, we employ the Vue framework, which is a JavaScript framework for building user interfaces (VueSchool, 2022). It builds on top of standard HTML, CSS and JavaScript, and provides a declarative and component-based programming model that helps us efficiently develop user interfaces, be it simple or complex.

We mainly employed two core features of Vue:

- **Declarative Rendering**: Vue extends standard HTML with a template syntax that allows us to declaratively describe HTML output based on JavaScript state.
- **Reactivity**: Vue automatically tracks JavaScript state changes and efficiently updates the DOM when changes happen.

With the help of Vue framework, we finalised all web pages, and they are shown from Figure 9 to Figure 14.

9 API

We use Flask to build backend RESTful API, an architectural style for an application program interface (API) that uses HTTP requests to connect client and server and use pymongo to connect to MongoDB for retrieving movie data based on result return by retrieval model.

10 Evaluation and Future Work

All the basic functions of the search engine worked well, and the search was fast without noticeable delay. After comparing several movie search websites, we found that the search results from the MIO search engine are familiar with these websites' results. In other words, our algorithm is very similar to the results achieved by those movie search websites.

We also realize that our system has limitations. The system currently uses static data, creating the entire project based on one download, and then updating it periodically to add data. In terms of algorithms, spell check relies on the found text library(text) and needs further optimization. In addition, we need to optimize the interaction design to create a better user experience. For example, we studied plot search, which can help users to find movies when they cannot remember the movie title. Users can enter the description of the movie and our system would return a movie list as they wanted.

11 Individual Contributions

- **s2149550-Dai Xu**

Configure MongoDB on remote cloud for storing movie data; Construct serve side of the application using Flask to build backend RESTful API for serving ajax request from client and use pymongo to connect to MongoDB for retrieving movie data based on result return by retrieval model; Construct client side of the application using Vue.js to build a single page application for frontend; Final deploy on google cloud VM.

- **s2154448-Peng pai**

I worked on data collection, GUI and architecture of the website. To be more specific, I implemented many data crawler functions to crawl various movie information on IMDB web pages, including movie title, actor, brief introduction and so on. All kinds of movie information are extracted by crawler using regular expression, and the JSON file of movie information is returned, so that MongoDB database can better process the data. Besides, I also used Vue framework to write a lot of website interfaces, such as movie details interface. Many of the interfaces on the site are co-written by Dai Xu and me, each responsible for different components. In the search result display interface, Dai Xu was responsible for container and Pagination of movie list, while I was responsible for the background of the website and various components styles. In addition, I set up Vue environment, which is responsible for the interface, and connected it to the Flask environment, which is responsible for transferring data forward. This architecture can greatly reduce the development effort.

Finally, DAI Xu and I participated in the deployment of the website in the Google Could Platform. We configured Nginx to connect http://104.155.34.47 to www.newbee123.xyz.

- **S2166777-Rui Wang**

I mainly worked for designing functions, User Interfaces and interactions and wrote the project report with Di Wu. At the beginning of this project, I made some competitive research on several major movie search websites, including IMDB and Douban. Based on functions the websites provide and our user requirements, I designed main functions of MIO and the user interaction logic process. The whole MIO system contains 4 main pages and provide movie search service from title, year of release, celebs and genres aspects. With the help of the logic process, I designed the low-fidelity prototype, user interfaces, with Figma from 0 to 1. Based on the UI design, the front end of our team implemented MIO interfaces. Finally, I was responsible for writing the report with Di Wu, and I designed the whole structure of the report and finished the Introduction, System Design, Data Collection, Pre-process and GUI parts.

- **S2025818-Di Wu**

I worked on product design, GUI design, interaction design, and completed this report with Rui Wang. For the product design part of the system, I completed the product analysis of

mainstream movie search websites and finally form the product analysis report. Based on the final team discussion results, we determined the product features and main algorithms for both versions (V1 & V2), including search sorting, category search, global search, spell check, etc. In the GUI part, I finished the UI design of the second version and optimized system interaction. Basic system testing was also completed. In the report writing part, I finished the part of Ranked Retrieval, Advanced Search, API, Evaluation, and Future Work, and proposed the system optimization suggestions.

- **S2159302-Hanxu Hu**

I worked on data preprocessing, which includes tokenization and stemming, the specific stemming algorithm is porter stemming. And in order to prevent the search error, I replace punctuations with spaces, and make all tokens as lower case.

I also create inverted indexes of all four kinds of search mode, which is Genre, Celebrity, Title, and Year, in order to speed up searching, I save them as pickle files at local.

And I design the logic of search in Genre and Year. The search logic is based on the Authority of a movie. Specifically, the measurement of a movie's Authority is based on two aspect in this case: the number of votes and the rating score. I design the method of computing Authority score, which includes normalize both of these two metrics, and changing the distribution to them, and adding them up. My computing method of Authority score is also used in Yunqing Liu's Celebrity and Title search.

Finally, I implement the spell checker based on edit-distance and bayes method as an auxiliary functions for search, which could return candidate words when searching.

I also implement plot search functions which could search related plot of the movies using BM25. And this function will be updated on the web as future work.

- **s2021327-Yunqing Liu**

My main work is the creation of search algorithms. I implemented both tf-idf and BM25 algorithms and, depending on the practicalities, I ended up choosing to use the BM25 algorithm to score the results. On top of this, I normalized the scores obtained by BM25 and weighted the scores together with the number of votes and the rating scores to obtain the final scores and give a ranking of the search results. This applies to searches based on people's names, searches based on movie titles and global searches. In addition, I have optimized the BM25 algorithm. I take the relevant documents containing the query as the overall number of documents recalled, which greatly improves the search efficiency of the model.

I also assisted Hanxu Hu in building the inverted index by adding some needed data processing.

Appendix


```
{
  "Title": "The King's Man",
  "Year": "2021",
  "Rated": "N/A",
  "Released": "N/A",
  "Runtime": "131 min",
  "Genre": "Action, Adventure, Thriller",
  "Director": "Matthew Vaughn",
  "Writer": "N/A",
  "Actors": "Ralph Fiennes, Gemma Arterton, Rhys Ifans,
    Harris Dickinson",
  "Plot": "In the early years of the 20th century, the
    Kingsman agency is formed to stand against a cabal
    plotting a war to wipe out millions.",
  "Language": "N/A",
  "Country": "N/A",
  "Awards": "N/A",
  "Poster": "https://m.media-amazon.com/images/M
    /MV5BMDEzZDY2ZDktNTlmOS00NThjLThkNTEtMjE5MzI5NWEwZmRjXkEy
    XkFqcGdeQXVyMDA4NzMyOA@@._V1_UX67_CR0,0,67,98_AL_.jpg",
  "Ratings": "N/A",
  "Metascore": "N/A",
  "imdbRating": "6.4",
  "imdbVotes": "76,740",
  "imdbID": "tt6856242",
  "Type": "movie",
  "DVD": "N/A",
  "BoxOffice": "N/A",
  "Production": "N/A",
  "Website": "N/A",
  "Response": "True"
}
```


Figure 1 All parameters and data for a movie

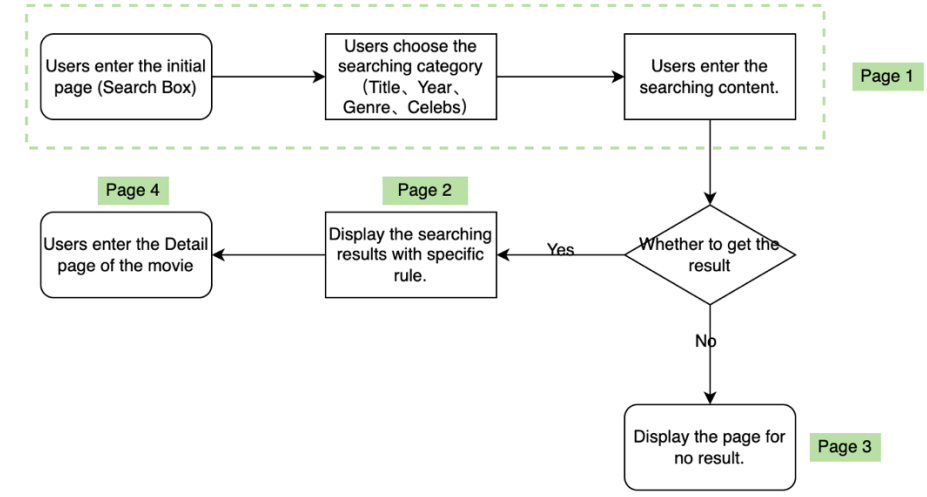


Figure 2 The process of users' operation with MIO

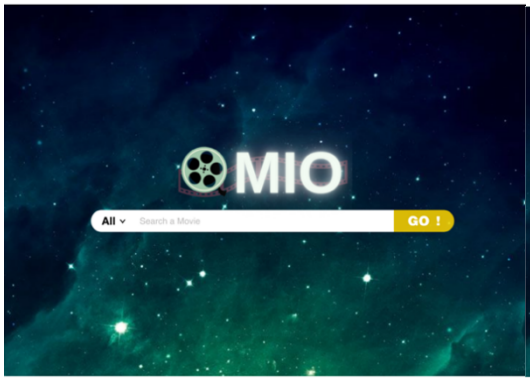


Figure 3 The Page-1 of MIO

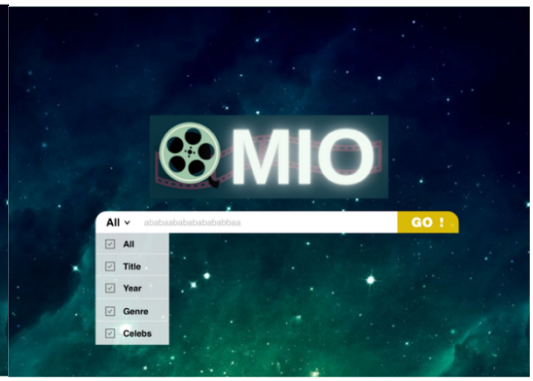


Figure 4 All filters in searching box of MIO

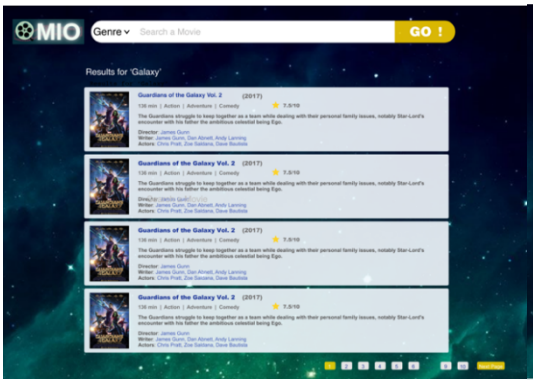


Figure 5 Page-2: Searching Results

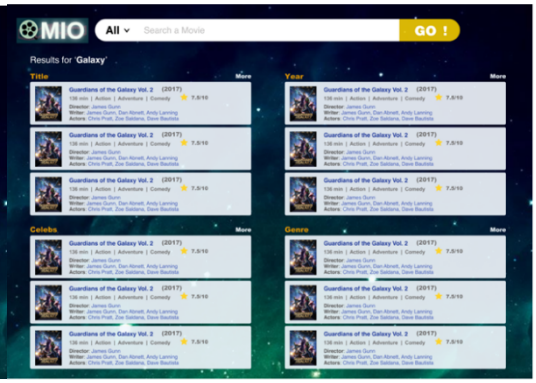


Figure 6 Page-2: results for global search

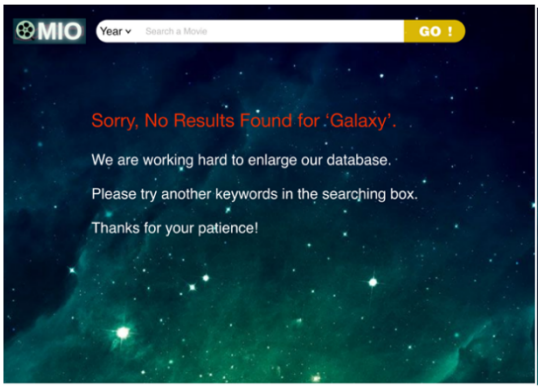


Figure 7 Page-3: No results

Figure 8 Page-4: Movie detail

Figure 9 Page-1: the search box

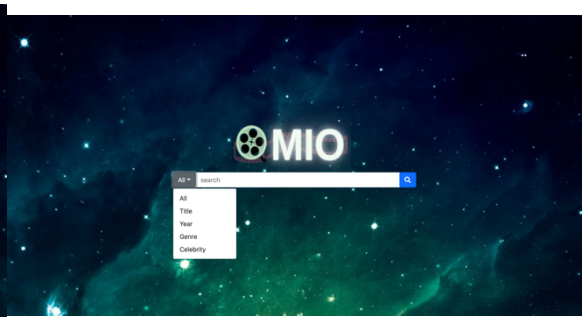


Figure 10 All filters in searching box of MIO

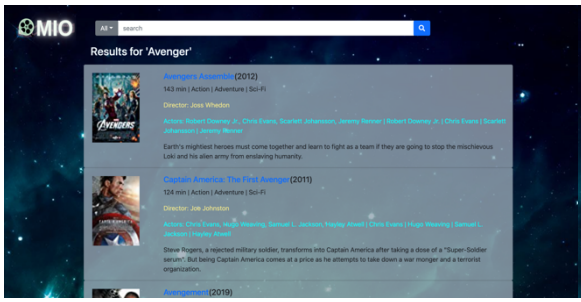


Figure 11 Page-2: Searching Results

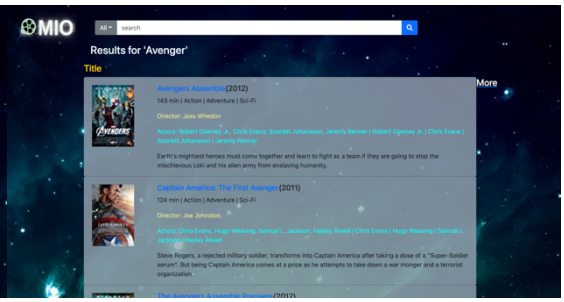


Figure 12 Page-2: results for global search

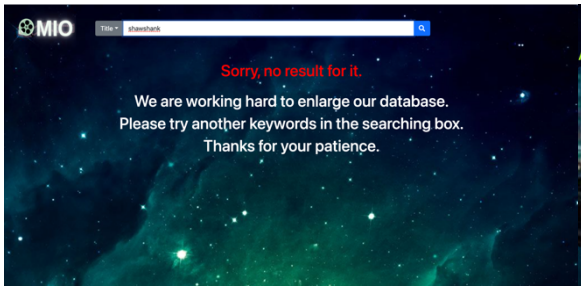


Figure 13 Page-3: No results

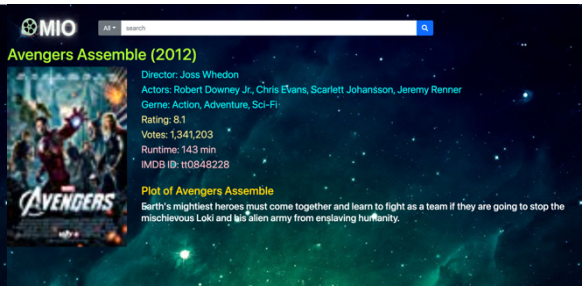


Figure 14 Page-4: Movie detail

Reference

Barla Cambazoglu, B. (2010) 'Review of "Search Engines: Information Retrieval in Practice" by Croft, Metzler and Strohman', *Information Processing and Management*, 46(3). doi: 10.1016/j.ipm.2009.12.009.

IMDB (2022a) *IMDB*. Available at: https://www.imdb.com/search/title/?title_type=feature&year=2010-01-01,2021-12-31&ref_=adv_prv.

IMDB (2022b) *IMDB - Top250*. Available at: https://www.imdb.com/chart/top/?ref_=nv_mv_250.

VueSchool (2022) *Vue.js*. Available at: https://vuejs.org/guide/introduction.html#what-is-vue (Accessed: 3 March 2022).

MongoDB (2022) *MongoDB: The Application Data Platform.* Available at: <https://www.mongodb.com/>.