# *Smart Search Engine*

***Interim Project-2 Plan***


**PROJECT MEMBERS:**

YELISETTI KRISHNA TEJA (NET-ID: KXY200016)

SATYA SOMEPALLI (NET-ID: SXS190436)

SAKETH DASAVATHINI (NET-ID: SXD190016)

HARSHITH RAVIPROLU (NET-ID: HXR180005)

SAI PRANAV REDDY DONTHIDI (NET-ID: SXD200125)

APUROOP PARAVADA (NET-ID: AXP210033)

SANJANA PENMETSA (NET-ID: SXP190149)

THOTA JAYASHREE SANTHOSHI (NET-ID: JXT210011)

B MOUNIKA (NET-ID: MXB210007)

PREETHAM RAO GOTTUMUKULA (NET-ID: PXG210001)

YOGESH BALA (NET-ID: YXB200007)

SRINATH REDDY MAVILLAPALLY (NET-ID: SXM210047)

KIRAN DEVARAJ RAJ (NET-ID: KXR190038)


https://github.com/krishnaty3/SmartSearchEngineDocs.git

**Submitted for: CS 6359.001**

**Project-II**

*The following document is evolving in nature and covers what was implemented in phase-1of the project and what changes/updates/additions have been made to phase-2 of the project.*

## I. Summary:

Smart Search Engine is a movie-based search engine where a user can search for movies using genre(s) or movie name or actor name(s) or any word related to movies. The most relevant results (in this case: most relevant movie related Wikipedia-URLs based on the word(s) searched) will be displayed at the top. This search engine also supports multi-phrase queries/words (example: multiple-genres based search). User can also click on a specific link displayed in the results and navigate to that link for more details on that result. User will be displayed with a two-line description below every Wikipedia-URL in the results. User will have an option of navigating through multiple results using the pagination functionality implemented in the phase-two.

*Note: All the functionalities implemented/updated/added have been explained in-detail below.*

## II. Functional Requirements:

Following are the functional requirements which are covered in the Smart Search Engine project:

- **Domain specific search Engine:** One word/multiple words search which will fetch results from a database which has movie-related information. User has to search something related to movies, for example, genre(s), movie name, director name, actor names(s) or actress name(s), etc. For example, user can search for ‘drama’ or ‘horror’ or even ‘drama horror’ together.
- **Logical AND/OR Implementation:** Logical OR based search was implemented in the first phase of the project. In phase-2 of the project, Logical AND based search was added along with OR based search. User will have a drop-down list present in the User Interface where user can select either ‘AND’ or ‘OR’ and then search. If user doesn’t select any clause in specific, then the default OR-clause will be taken into consideration while displaying the results. For example, if we search for ‘drama horror’ and choose the ‘AND’ option, then the links which mandatorily matches both the words will get displayed in the results, that is a Wikipedia link containing both the words in it. Another example would be ‘drama horror’ with ‘OR’ clause which will display all the results with either of the words present including ‘AND’ case results too.
- **Ranking/Ordering of URLs:** In this project, we are displaying the most relevant URLs first or at the top. Here, the relevancy is being calculated based on TF-IDF scores of the words searched in the displayed URLs. For example, if user searches for ‘drama’, then the most relevant URL will be a link with the highest TF-IDF score of the word ‘drama’ in the corpus of URLs. Here, results will be displayed in the decreasing order of TF-IDF scores.

- **Filter Stop-words:** In this project, stop words will be omitted from the query which user searches for. For example, if user searches for 'Drama Horror is' then 'is' here is a stop-word which will be filtered. Examples of some stop-words are: 'the', 'and', 'or', 'do', 'for', etc.
- **Stemming:** Stemming of the word means bringing the word to it's root form. Here, the word(s) being searched will be stemmed to their root forms and based on the root forms of the words, it will be compared with the database and relevant results (URLs) will be outputted. For example, if the user searches for 'Flying' then the suffix 'ing' will be removed and the input will be brought to the root/stemmed word which is 'Fly'.
- **Auto-suggestion:** In phase-2 of the project, autosuggestion of the input has been implemented. When user searches for something, he/she will be suggested with a query of word(s) aligning with the words he/she is typing. For example, if user starts the search by typing 'dr', then the user will be suggested with a query of word(s) such as drama, drama movies, drama film, drama horror, etc.
- **Pagination:** In the phase-2 of the project, pagination has been implemented for better UI support. A maximum of 10 results will be displayed per page and all other results will be distributed accordingly (to next pages). For example, if there are a total of 32 results, then there will be 4 pages to navigate where the first three pages will have 10 results per page and the last page will have the last 2 results.
- **404 Error page:** In the phase-2 of the project, Page-404 has been implemented where if there are no results based on the user search, then user will be navigated to the Results-not-found page. This functionality has been added for better UI implementation compared to Phase-1 of the project. For example, if user searches for an irrelevant word 'aaabbbcdddeeeg' and since there's no match in the movie database with the mentioned word, the browser will display 'No Results found' Page.

## III. Non-Functional Requirements:

Following are the non-functional requirements which are covered in the Smart Search Engine project:

- **URLs-addition:** To expand the movie database, 1000 URLs have been scrapped and added to the movie database in the phase-2 of the project to fetch multiple and more relevant results based on user-search.
- **Localization:** The project has been locally hosted and can be hosted on any local system or machine. A readme file along with a minimum package requirements file will guide on how to run this project or how to deploy this project locally.
- **Maintenance:** In the phase-2 of the project, logging has been implemented which will help resolve the issues quickly as each and every module implementation will be logged into a

file specifically (this file will log the flow of events or function calls happening in the back-end) through which problems can be understood or analyzed.

- **Removal of outdated URLs & Addition of new URLs (Consistency of the Database):** To remove the expired URLs and add new URLS in the movie database, admin can run the pre-processor frequently (for example, weekly once) which will scrape the newly added links to expand the database and the expired links in the existing database will be removed. In this way, consistency of the database can be retained. Note: This step also comes under maintenance of the project.
- **Friendly User-Interface:** In the phase-2 of the project, the user interface of the Smart Search Engine has been changed compared to phase-1 where user will have an additional option to use the and/or clause while searching for multiple words. Pagination has also been implemented in phase two for better navigation among the links. To show the throughput, number of results along with time taken to display the results are also being displayed in the UI in phase two.
- **Portability & Usability:** As previously mentioned in the Localization non-functional requirement, Smart Search Engine will have an option of portability where the project can be easily deployed on any local machine following a guide on how to setup the project using the read-me file and the requirements file. Smart Search Engine is very easy to use in nature taking both admin and user into consideration.
- **Low-Latency and High-Throughput:** In phase-2 of the project, Low latency and High Throughput is maintained while displaying the results. Although the movie database has been expanded and multiple URLs have been scrapped, the low latency is still maintained and as mentioned previously the user interface will show the time taken to display the number of results which supports the high-throughput and low-latency functionality.

## IV. UML-Diagrams:

In phase-1 of the project, following UML diagrams were drawn:

- **Class Diagram:** A basic class diagram to understand the main and primary external or system actors was drawn.
- **Usecase Diagram:** A usecase diagram which covered only the main usecases in the proposed project was drawn.
- **Sequence diagram:** Two sequence diagrams were drawn where the basic flow of events between admin and pre-processor were shown in one of the sequence diagrams and in the second sequence diagram (main sequence diagram), the basic flow of events between user and the SearchEngineModule (system actor) were shown.

Note: In addition to all the above UML diagrams, a basic flow of the project was drawn and presented in the phase-1 of the project.

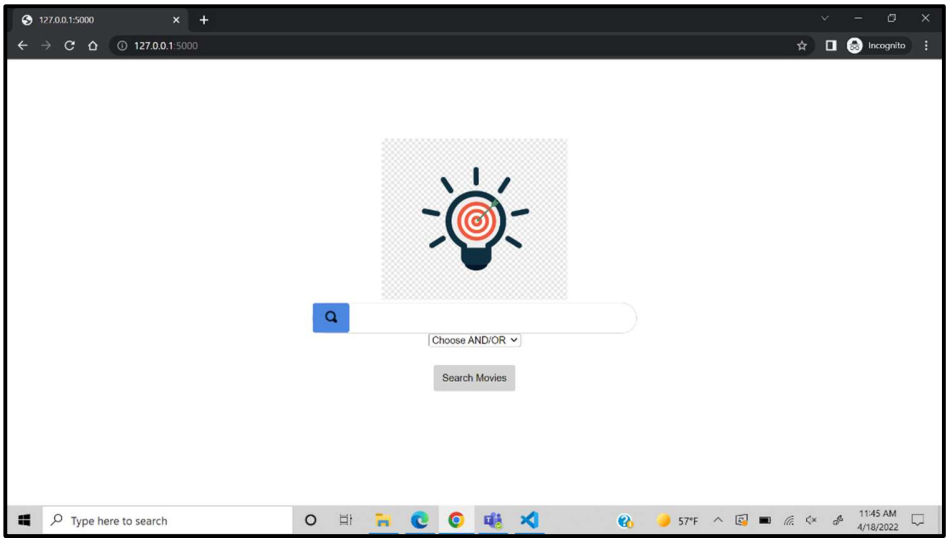
In phase-2 of the project, following UML diagrams will be drawn or updated:

- **Class Diagram:** An updated class diagram which explains the relationships between different classes more clearly will be drawn/presented in the phase-two of the project.
- **Usecase Diagram:** A detailed usecase diagram which covers all the actors and their specific usecases will be drawn/presented in the phase-two of the project.
- **Sequence diagram:** Sequence diagrams will be updated accordingly and additional sequence diagrams will be drawn/presented in the phase-2 of the project which will help understand the flow more clearly.
- **State-transition diagrams:** State-transition diagrams describe all of the states that an object can have, the events under which an object changes state (transitions), the conditions that must be fulfilled before the transition will occur (guards), and the activities undertaken during the life of an object (actions). State-transition diagrams are very useful for describing the behavior of individual objects over the full set of use cases that affect those objects. In the phase-2 of the project, a state-transition diagram will be drawn/presented which explains the dynamic understanding of the flow of the Smart Search Engine.
- **Activity diagrams:** Activity diagram is another important diagram in UML to describe the dynamic aspects of the system. Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc. Activity is a particular operation of the system. Activity diagrams are not only used for visualizing the dynamic nature of a system, but they are also used to construct the executable system by using forward and reverse engineering techniques. In the phase-2 of the project, an activity diagram will be drawn/presented.

Note: All the UML diagrams drawn/shown/presented in phase-1 of the project will/might be corrected/updated/edited in phase-2 of the project.

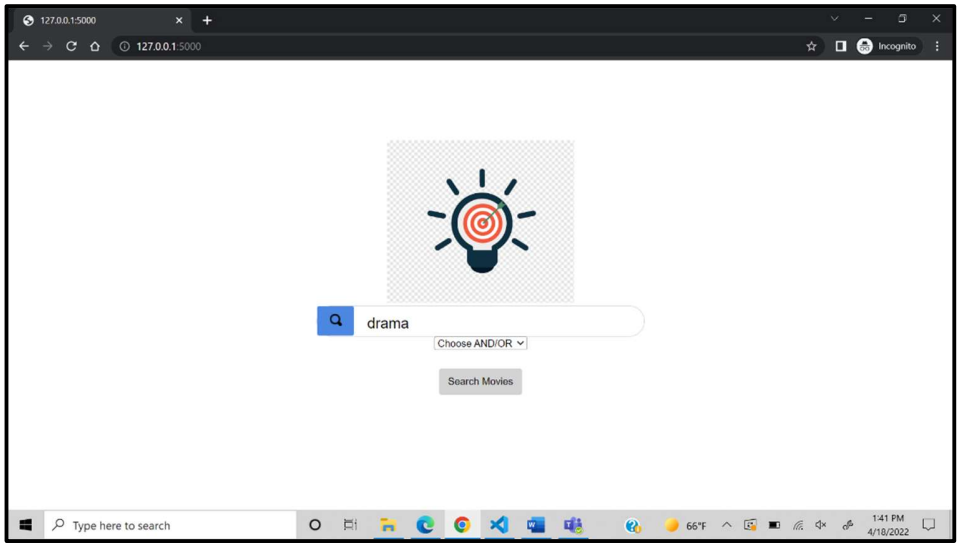
## V. Demo-Screenshots of the Smart-Search Engine:

Following is the screenshot of the main(index) page of the Smart Search Engine which is locally hosted.

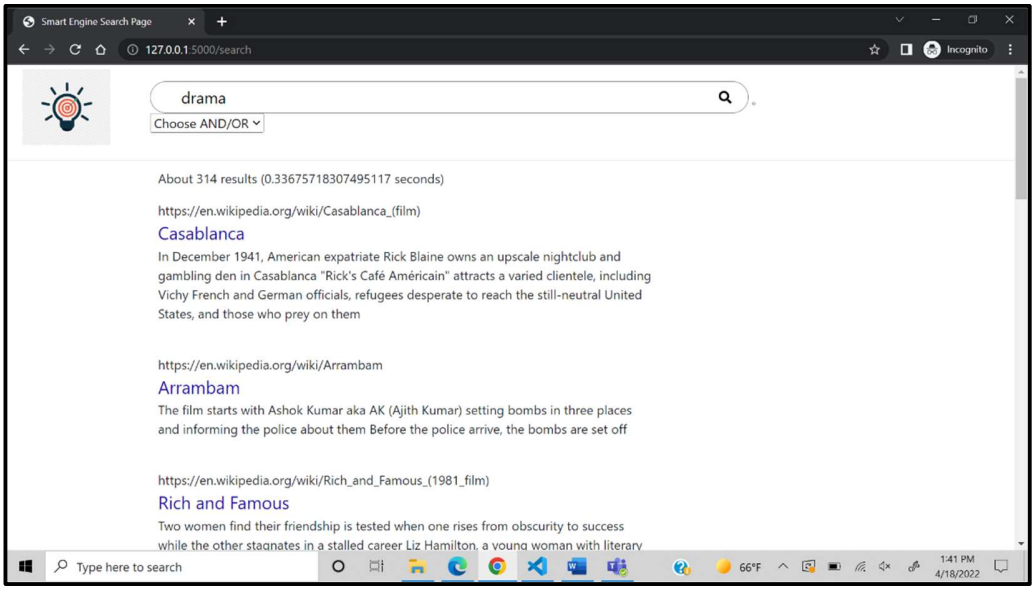



In the above screenshot, user can select AND/OR clause while searching for multiple words.

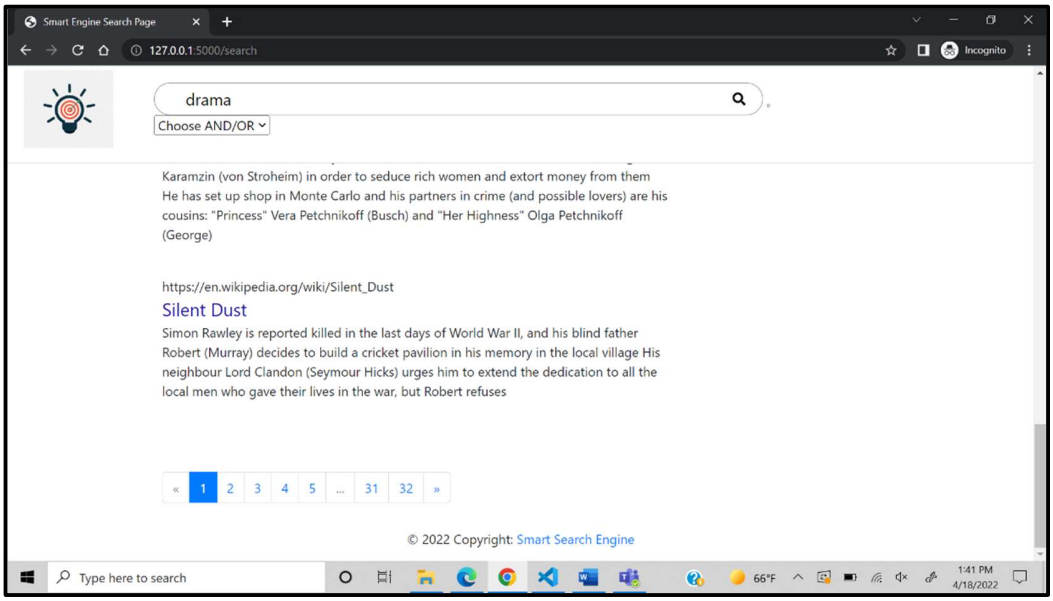
Following is the screenshot of one-word search ‘drama’ being searched by the user.



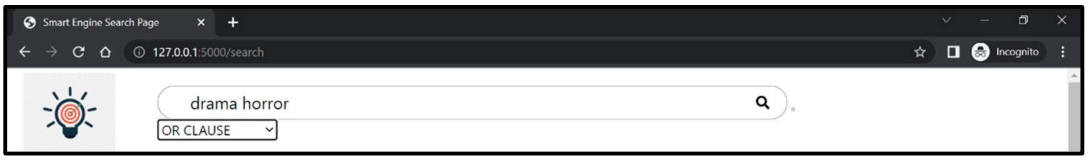
We can see that 314 results have been displayed in a span of 0.336757 seconds for one-word ‘drama’.



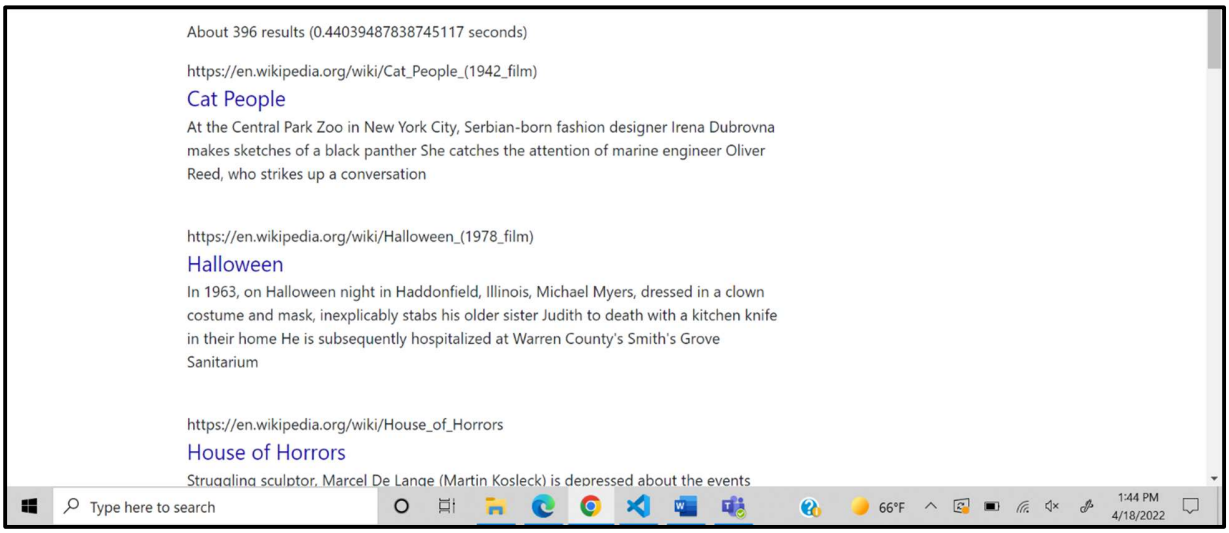

We can see below those 314 results have been divided into 32 pages (pagination functionality).



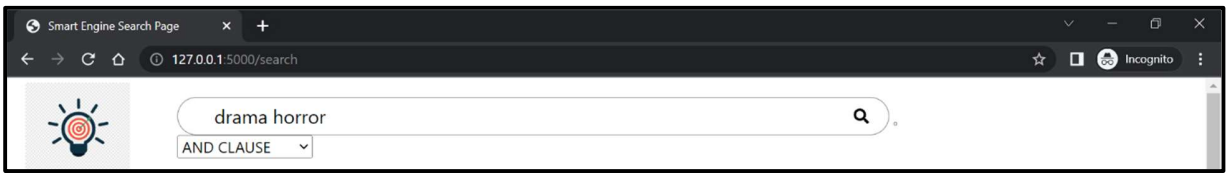
Following is the screenshot of multiple word query search with ‘OR’ clause:



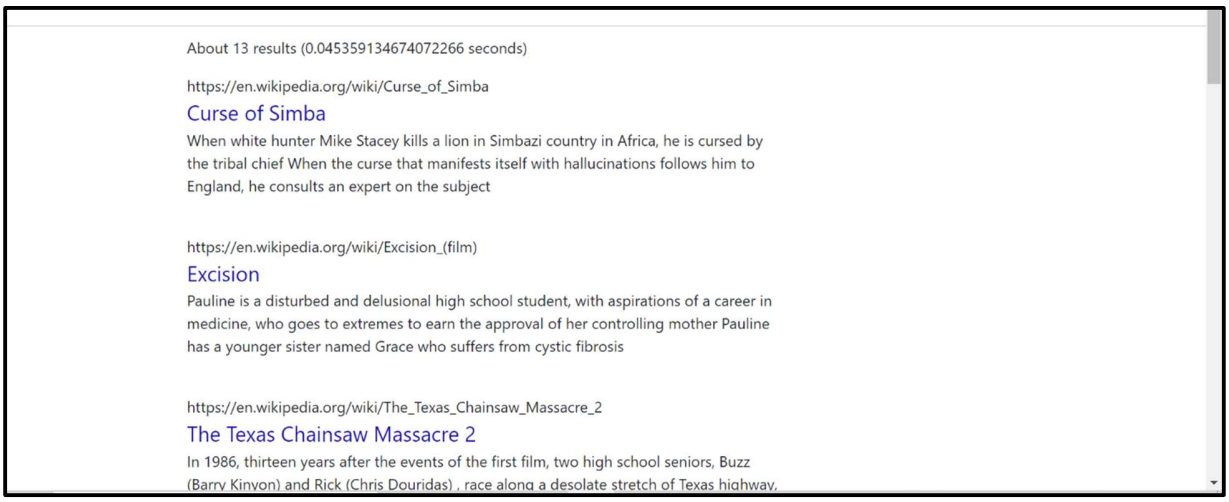

Following is the screenshot of multiple word query (‘drama horror’) search result using ‘OR’ clause:




Following is the screenshot of multiple word query search with ‘AND’ clause:



Following is the screenshot of multiple word query (‘drama horror’) search result using ‘AND’ clause:




We can see that the number of results is comparatively low in case of ‘AND’ clause when compared to ‘OR’ clause (13 < 396).