Project Report Part 4

Project GitHub URL: https://github.com/yuyanwang03/IRWA

Project GitHub Tag: IRWA-2024-part-4

The steps required to execute the Search Engine Application are detailed in the README.md file in the Part4 directory. It is important to ensure that the static folder contains the `farmers-protest-tweets.json` file in order to make sure that the program works.

User Interface

In the previous part of the project, we have implemented the search and ranking algorithms that form the core functionality of the search engine. Building on this foundation, the final part focuses on developing a user-friendly interface for the search engine. To simplify the process, we have used the "Toy Search Engine" example provided during class as a template, modifying and customizing it to align with our project's needs. We have also used Flask, a simple python's web framework to implement the user interface together with the web analytics.

The main file of the search engine interface is 'web_app.py' where we set up the Flask application and it handles the interaction between the user and the search engine. We first load the corpus from the CSV file "processed_data" into memory, and then we create an instance of the class SearchEngine, which handles the core functionality of doing search queries on the corpus. For web analysis purposes, we tract the session data, such as the last query search or the result count, and we parse the user-agent to gather information about the user's browser and device. Finally, we render index.html and results.html to display the interface of the home page and the search results.

Additionally, we load the JSON file 'farmers-protests-tweets.json' to make sure that we only load valid entries into memory. In other words, by also loading this JSON file, we are also integrating both, static (from processed_data.csv) and dynamic (from farmers-protest-tweets.json) data to better manage search inputs and results.

Then, we have a folder, 'search', where the files that build up the search engine are located. 'load_corpus.py' loads the tweets into a structured format, 'objects.py' gives the data structures to represent the tweets and the search results and 'search_engine.py' implements the search and ranking of the results and returns then in a structured format to display them to the user, using several algorithms, such as TF-IDF, BM25 and the custom scoring algorithm we previously developed.

In the file where we load the corpus, we convert each row of the CSV file into a Document object, representing a tweet and each row is validated so we make sure that the required columns are present. Then, the set of Document objects are stored in a dictionary, an in-memory representation of the corpus. Inside the objects file is where we define the two core classes that will be used to structure the corpus and the results. The Document class represents the attributes of a tweet from the corpus and the ResultItem class represents individual search results.

Finally, the search_engine file defines the SearchEngine class that implements several search and raking algorithms by taking a set of documents (tweets) and it gives methods to search for relevant documents (tweets) based on a query. The search and ranking algorithms we have used are TF-IDF, BM25 and a custom scoring algorithm that is combining scores from the text score given by TF-IDF, the recency scores based on the tweet's dates and the social scores that combines likes and retweets to assess the popularity of the tweet. At the end, search results are represented as ResultItem objects with detailed metadata for a user-friendly display, which makes it easier to interact with the results and to have a better analysis.

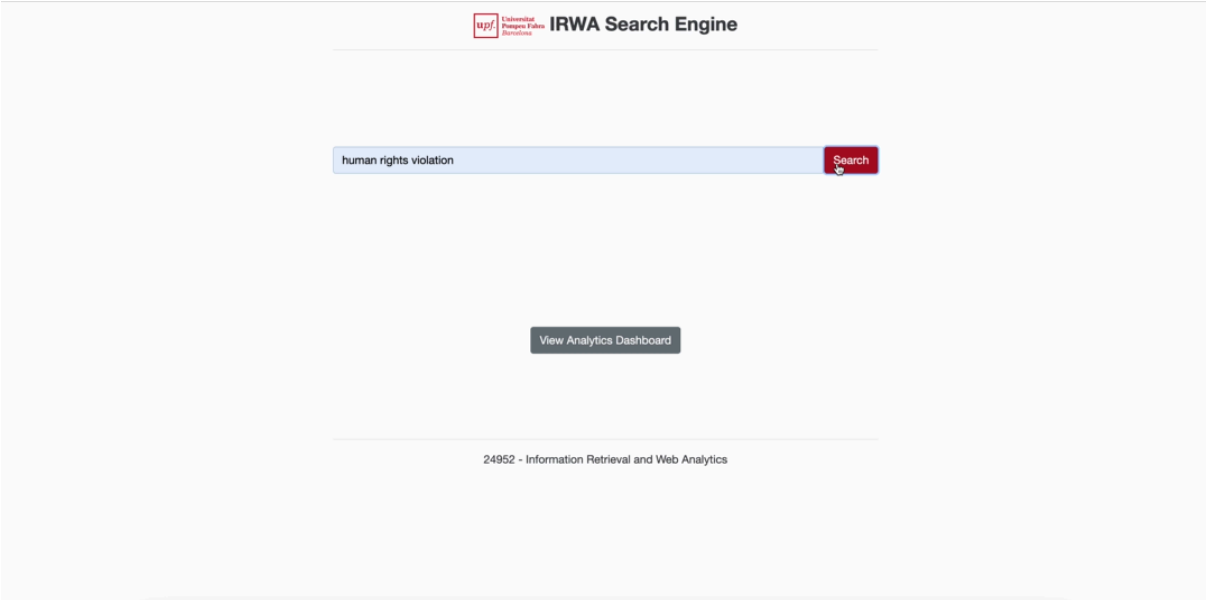
To define the structure and layout design of the web, we used the HTML files to develop the

navigation menus and the search engine interface and to render the dynamic content. The file that defines the main structure of the interface is ‘base.html’, which provides a design that is going to be consistent across all pages, meaning that other html templates inherit from base.html. In this way, we don’t need to repeat headers or footers across all html files, we reduce redundancy and we make updates easier.

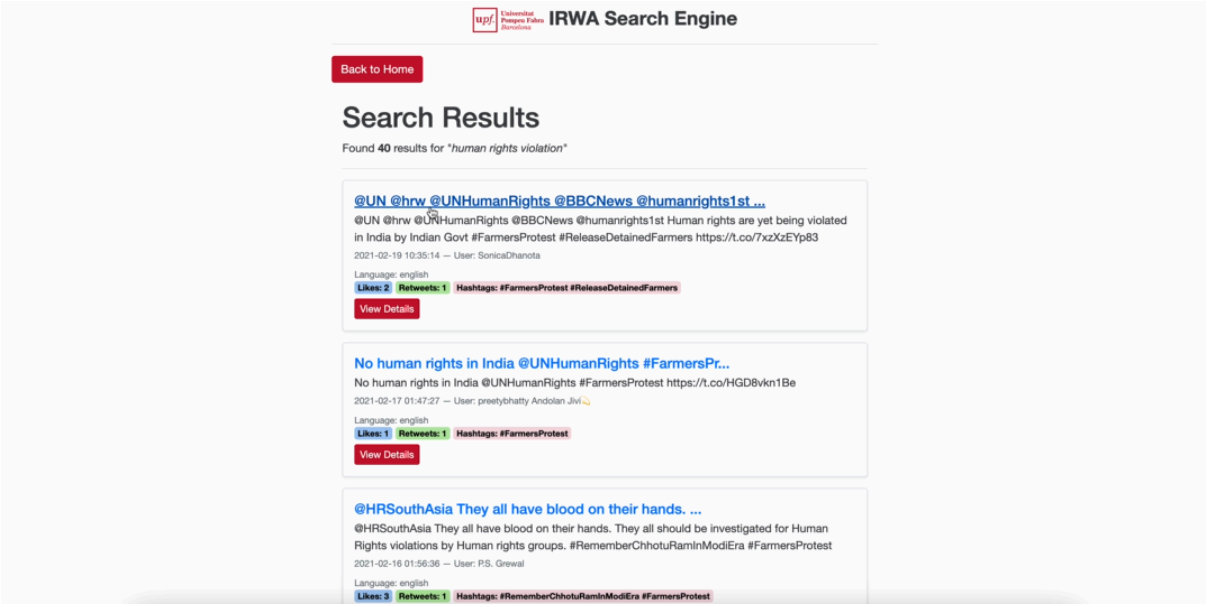
As base.html is the parent for all html templates, we developed ‘doc_details.html’ used to implement the interfacte that will show the detailed information of a specific document. It displays the key elements of a tweet such as the content, the date it was published, the number of likes and retweets, the hashtags, the language it was written in and the URL. If it is available, it shows the user associated to each tweet and, also, its personal information. We also provide a button to return back to the page where there are all the results, so the user can contine with its search.

‘index.html’ is the homepage of the search engine, working as the main point where users can input the search query. It also acts as the gateway to the application’s search functionality. Once, the user has initiated a search query and clicked on the ‘search’ button, it redirects the user to the page where all the results are displayed, managed by ‘results.html’. This file is in charge of providing a detailed view of the results obtained for the query, and allowing the user to 2click on a specific tweet to see additional information related to it. It includes metadata for each tweet, such as the number of likes and retweets, the date, the user information (if available) or the language the tweet was written in. From this page, we can also return to the homepage (index.html).

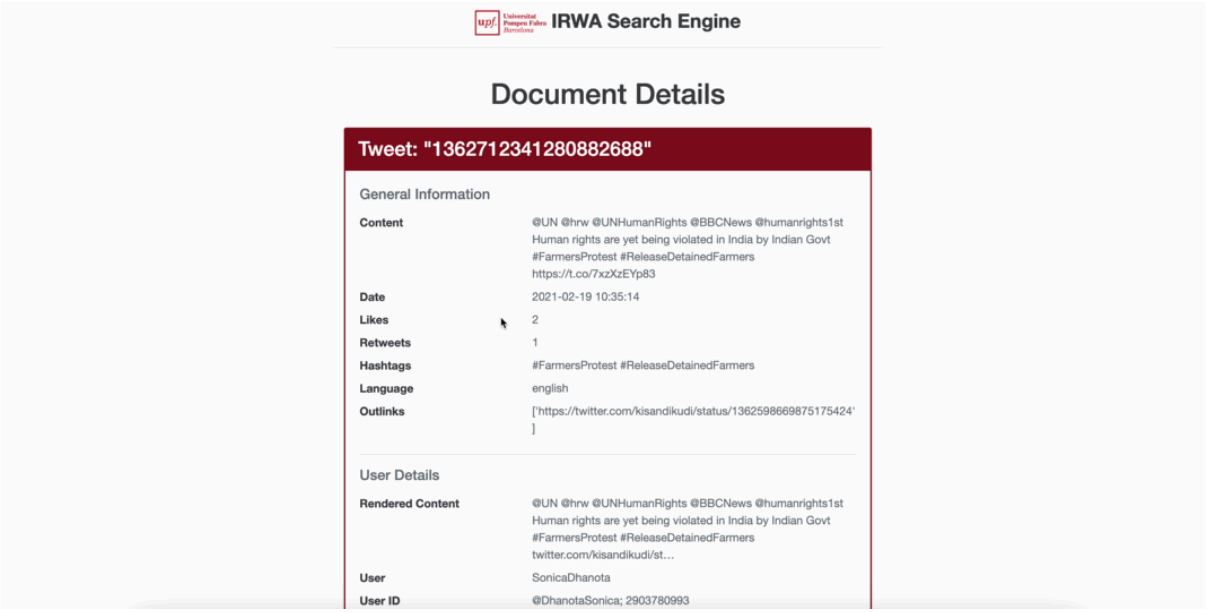
The screenshots below show the different interface designs for each of the pages of the main functionality of the Search Engine Application.



Home page of the Search Engine Web Application



Results page of the example typed search query



Document Page to show all the details related to a tweet.

Web Analytics

As required we have implemented web analytic metrics so we can observe user behaviors and keep track of the search engine activity. The analysis gives important information about how users interact with the application and the search engine. We will mainly focus on understanding user interactions, such as clicks and session data, while also analyzing search query trends, and visualizing statistical data, which can offer us a clear way of evaluating the search engine performance.

For this Web Analytics part of the project, we decided to make it simple and, instead of using the cache of the web to store the information related with each session, we hardcoded a variable when the application initializes, called 'session', where we keep the different information that we want to gather from the user and the search engine. Additionally, at the end of the session, that is when the user exits the application, the session's information is also stored in a JSON file, so it does not delete.

The metrics we are using to evaluate the performance of the search engine, as well as the behavior of the user, can fall between different categories of key indicators. Request metrics track the accessed URL endpoints, HTTP methods, each access timestamps, and the IP addresses, which are logged after every user interaction. Query metrics help to assess the search engine effectiveness, and it includes the full text of the query, the query ID and the number of terms, as well as the timestamp of the query and the number of results for each of them. To evaluate the results relevance, we have used click metrics that capture the clicked document ID and its associated query ID, the document rank in the results and the timestamp of the click. User context metrics gather information on the user's environment, such as the browser, IP address and session details, also we gather optional data like the user's city and country, using their IP address. Since the application is running locally, the city and the country variable's value is 'Unknown', but they will show the actual location once it is deployed.

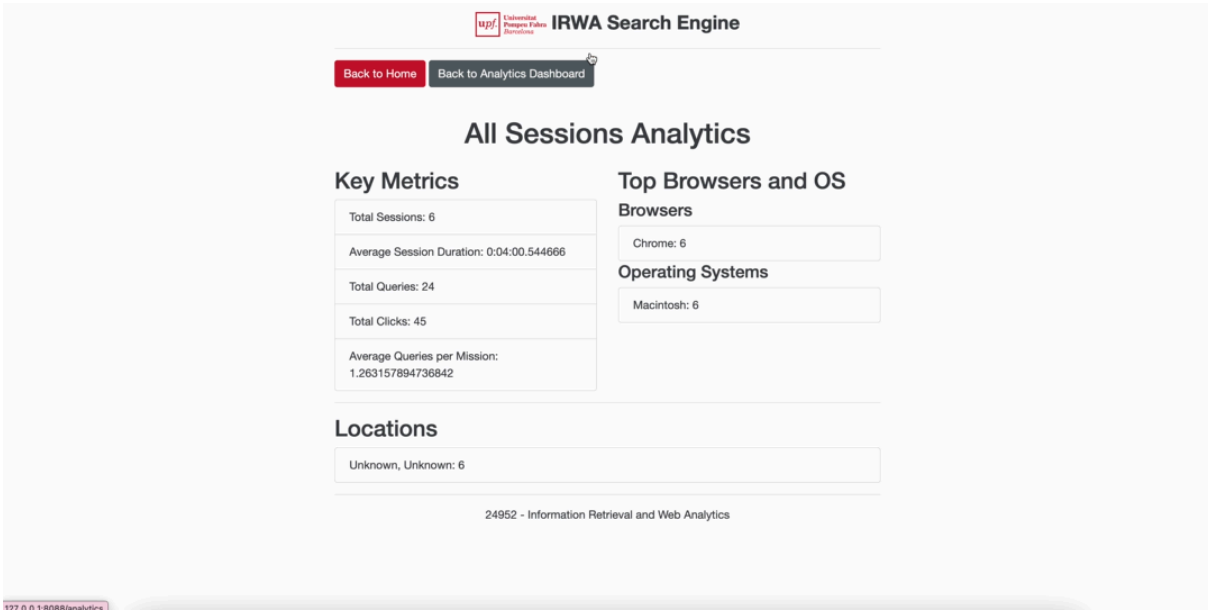
Lastly, session analytics help track navigation paths and document metadata store details about the retrieved documents for each query. For this metric, we use a variable called 'last_activity' where we store the timestamp, and it allows us to calculate the duration of the user's session with a variable called 'duration' and it shows the total duration of the session. Additionally, we group queries into missions that are sequences of queries with a common goal, so we can track logical sessions across a single or multiple physical sessions and these queries are stored under the same mission ID. The collected data during a session is stored in a session dictionary during runtime and saved in a JSON file at the end of each session, as we have mentioned before.

We have developed two HTML templates files to display all the analytics in our web application. The 'all_sessions.html' file presents all the aggregated data across all user sessions, providing a full view of long-term trends and overall patterns among all sessions. It displays an analytics dashboard, where the key metrics shown are total number of sessions, the average session duration, the total number of queries, the number of total clicks and the average number of queries per mission. Moreover, it displays extra information across all sessions, such as the most popular browsers and OS among all users, a distribution of users by location, and the total number of users from each area.

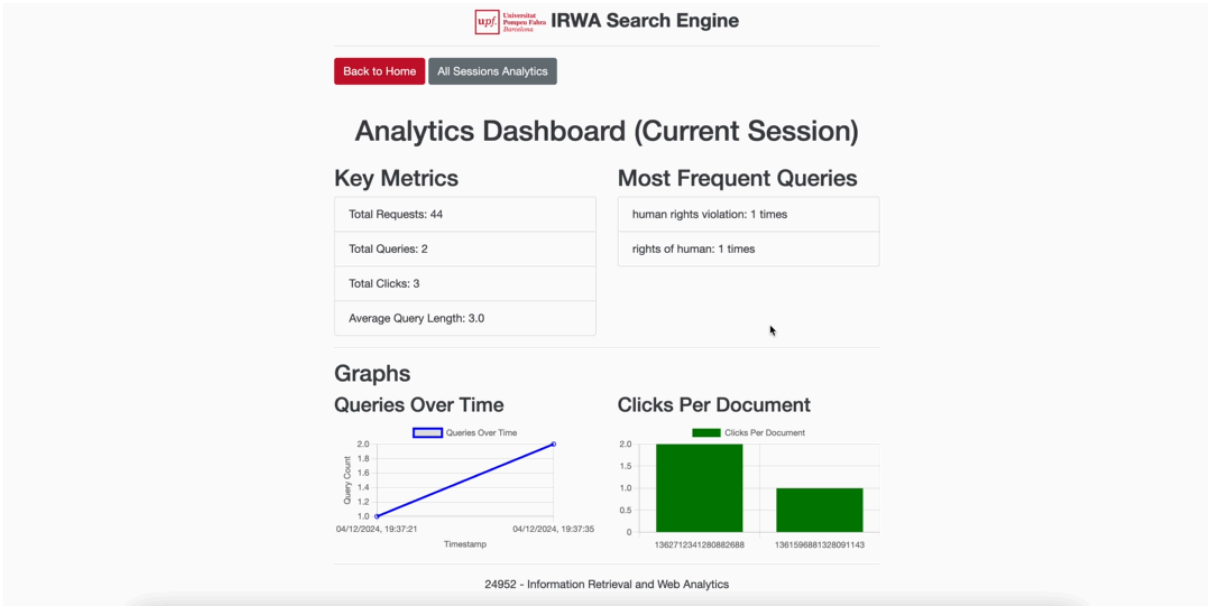
In contrast, the 'analytics.html' file mainly focuses on real-time information of the current session, providing a real-time understanding of the ongoing session with some interactive visualizations to help direct analysis. The page displays the key metrics we have mentioned above that are specific to the current session, such as the total number of requests, queries, clicks, and the average query length, as well as the most frequently executed queries in the session, which is done using a dynamic loop. Until this point, both files are really similar, but what sets them apart is that 'analytics.html' includes a line chart to show the number of queries over time and a bar chart to show the number of click per document. For the 'Queries Over Time' chart, the used timestamps are turned to a human readable format, so the user can understand it easily, and for the 'Clicks Per Document' chart, we dynamically map document IDs to their respective click counts.

The search engine interface has been designed to be intuitive and easy to use and find the information that the user needs. The homepage of the engine is where the user can type the query search into the search bar and click the 'search' button to begin the search. Then, the user will be redirected to the results page, which displays the total number of results that are found. For each result, the user can click its title or the 'view details' button to open a detailed view of the document. Once, the user is in the document details page, it will find the metadata related to the document, such as the language, the user details and link to the original source, if available. The user can also go back to the results page or even the home page to start another search.

The screenshots below show the interface design of the Analytics Dashboard of the Search Engine Application, and, as we have explained, we have two pages which correspond to the data summary of all sessions and the data of the current session.



All Sessions page with the data across all sessions



Current Session page with the data from the current session