****

**Capstone Project Phase B**

**News Website Content Filtering**

**To browse the news websites in a personalized manner**

**23-2-D-15**

****

**Supervisor: Dr. Natali Levi** [**NatalyL@braude.ac.il**](mailto:NatalyL@braude.ac.il)

**Students: Ronen Zeyan** [**Ronen.zeyan@e.braude.ac.il**](mailto:Ronen.zeyan@e.braude.ac.il)

**Adham Asaad** [**Adham.asaad@e.braude.ac.il**](mailto:Adham.asaad@e.braude.ac.il)

**Table of contents**

[1. Introduction 4](#_Toc165142589)

[2. Project Review and Process Description 5](#_Toc165142590)

[**2.1 Package diagram** 5](#_Toc165142591)

[**2.2 Activity diagram** 6](#_Toc165142592)

[**2.3 Database description** 7](#_Toc165142593)

[**2.4 Solution** 8](#_Toc165142594)

[**2.5 Engineering Process** 8](#_Toc165142595)

[**2.6 Implementation** 9](#_Toc165142596)

[**2.6.1 Base Definition** 10](#_Toc165142597)

[**2.6.2 Filter Implementation – Detailed Breakdown of the Filtering Process** 10](#_Toc165142598)

[**2.6.3 Website Implementation – Detailed Breakdown of the Website Development Process** 15](#_Toc165142599)

[**2.7 Decisions** 16](#_Toc165142600)

[**2.8 Frameworks** 17](#_Toc165142601)

[3. Challenges 19](#_Toc165142602)

[4. Testing 23](#_Toc165142607)

[5. User Documentation 27](#_Toc165142608)

[**5.1 User's guide – Operating instruction** 27](#_Toc165142609)

[**5.1.1 General Description** 27](#_Toc165142610)

[**5.1.2 Operation instructions** 28](#_Toc165142611)

[**5.2 maintenance guide** 37](#_Toc165142612)

[6. Results and Conclusion 41](#_Toc165142613)

[7. REFERNCES 43](#_Toc165142614)

**Abstract**

In our digital age, where the information technology flourishes incessantly, we grapple with an endless stream of data and news flowing at all times through the internet, which serves as our primary source of information. This access to information, unprecedented in human history, provides us with opportunities and access to knowledge that did not exist before. However, it also submerges us in new challenges, particularly the difficulty of filtering content that is either unwanted or irrelevant to us. To tackle this issue, this book introduces a solution, which it's a filter integrated within a website, specifically designed for English news websites. This system enables users to efficiently delete unwanted content by entering the URL of a news website and choosing from two filtering options—category-based or word-based. The category-based method eliminates articles from specified categories, while the word-based method removes articles containing specified keywords or their conjunctions. The filtering process is executed in two stages: Stage A focuses on the extraction, and processing then identification of the unwanted content, utilizing natural language processing (NLP) techniques such as stemming. Stage B involves the precise deletion of the unwanted content through hierarchical traversal. This dual-component system includes a user interface for input management and a filter that processes and filtering data based on user settings. The goal of this system is to ensure that users are not exposed to unwanted content and can navigate news websites safely.

**Keywords**

Stemming, news website, WebCrawler, Flask Framework, Filtering, customized website, user oriented news.

# **Introduction**

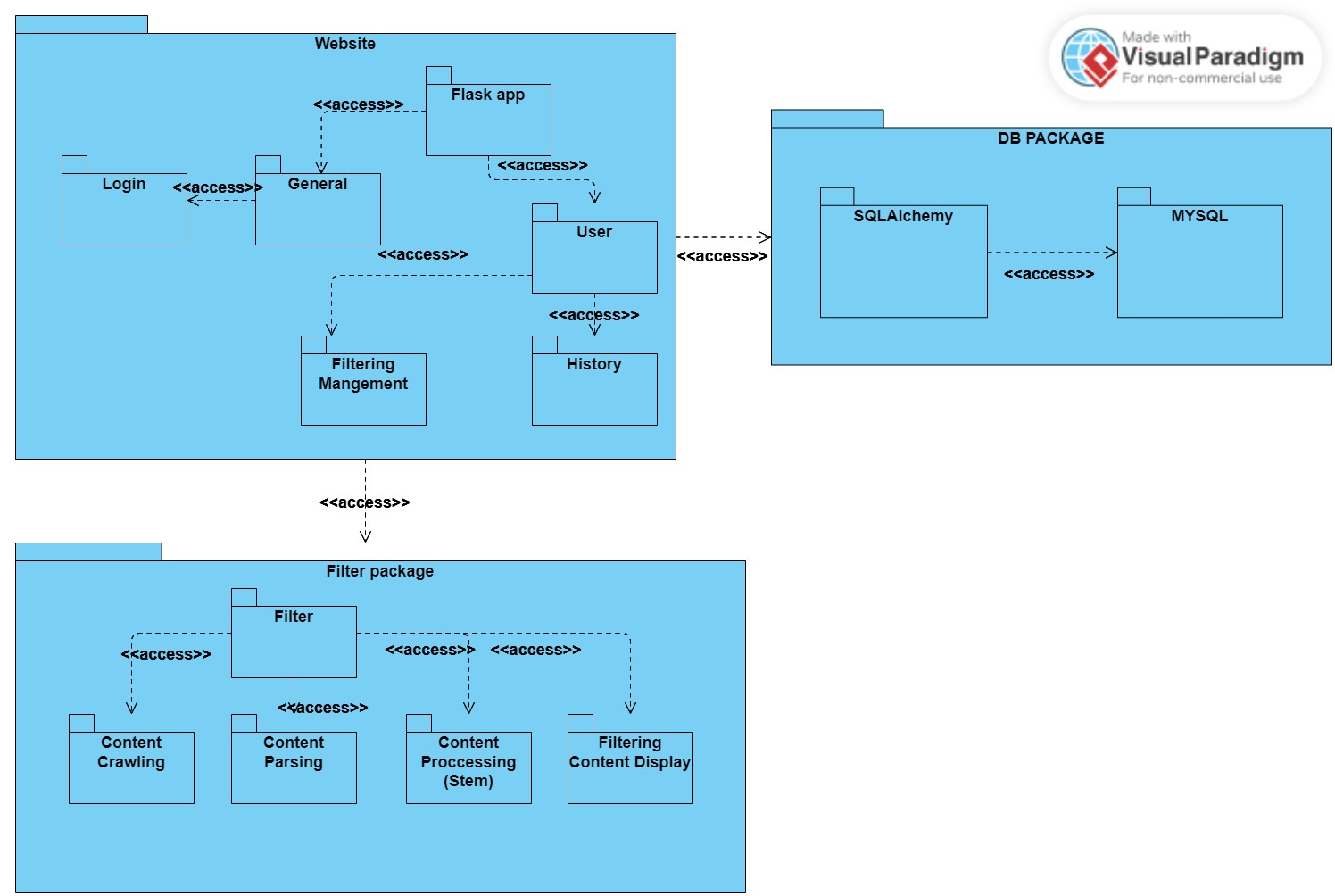
Nowadays, we find ourselves struggling with content that is not only uninteresting to us but can also be mentally exhausting and sometimes even harmful for us or for our children. Part of this challenge arises from the fact that the information streaming to us does not always consider the needs or personal preferences of the specific reader, but is presented universally, without any attempt to tailor itself to a specific reader.

Numerous studies have clarified the high cost we pay as a digital society for unregulated news consumption, especially when the content we encounter is considered negative. Prolonged exposure to such information not only burdens us mentally but also can adversely affect our physical health. Phenomena such as anxiety, depression, and feelings of loneliness are just some of the psychological repercussions of exposure to detrimental information for us. [1]

The necessity to address the information overload and to provide users with the capability to control the content displayed to them has led us to develop our project. The aim is to create a system that enables smart and targeted content filtering, considering each user's personal preferences and filtering the information based on the user's request. Our solution is a website, developed using the Flask web framework [2] , utilizing advanced content extraction (web scraping) techniques [3] using requests [4] and BeautifulSoup [5], and employing Stemming algorithms to filter out unwanted content, before presenting the relevant and quality content back to the user in our website. Thus, our project offers a solution that gives users control and autonomy over the information delivered to them, significantly improving their digital experience.

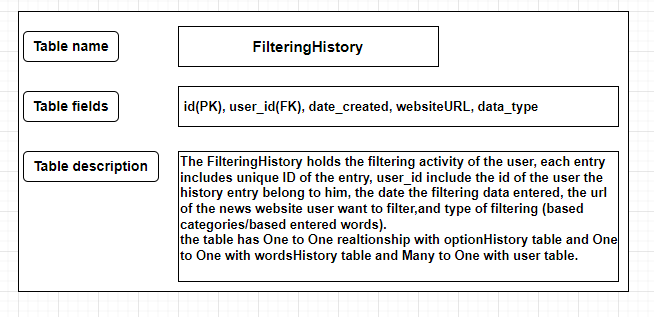
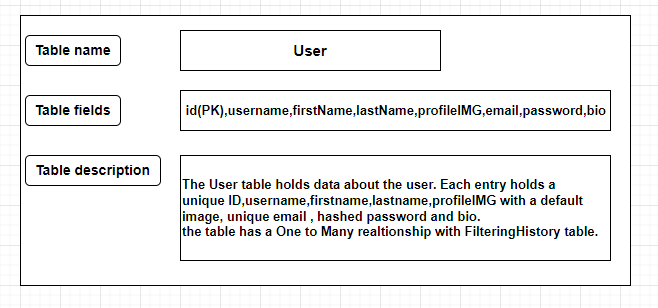
# **Project Review and Process Description**

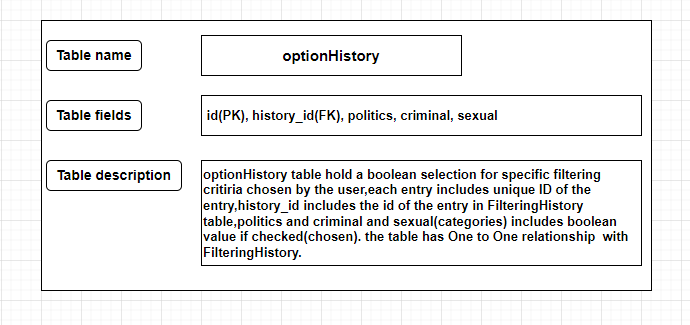
## **2.1 Package diagram**

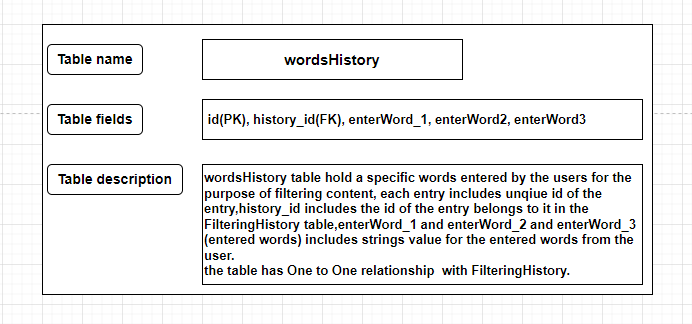


## **C:\Users\אאא\AppData\Local\Packages\Microsoft.Windows.Photos_8wekyb3d8bbwe\TempState\ShareServiceTempFolder\Untitled.jpeg2.2 Activity diagram**

## **2.3 Database description**







## **2.4 Solution**

Our solution is a filter implemented within a website we built using the Flask framework, It is designed to allows the user to filter news websites by inserting the site link and choosing the type of filtering. There are two types of filtering that the user can choose: category-based filtering and words-based filtering. In category-based filtering, the user is offered three categories to select from, indicating the category of articles they are not interested in viewing. In word-based filtering, the user has the option to input specific words, and articles containing any of those words or combinations thereof are filtered out and not displayed to the user. The filter utilizes Python libraries such as Requests for fetching/extracting the textual content from the pages and BeautifulSoup for parsing this content. Additionally, it used NLTK library which used for natural language processing (NLP), particularly stemming, which simplifies the words in the textual content to their root forms. This process increasing the precision of article classification and identify the unwanted articles. The final outcome is a personalized display of the news website according to the user's request within our website.

In our solution, filtering is performed on a per-page basis. Meaning, as long as the user has not requested a specific page, the filtering stage will not be executed on it. Every time a user wants to filter a page, their link is sent along with the filtering data (words/categories), and then the filter performs the filtration and returns the personalized page to the user.

.

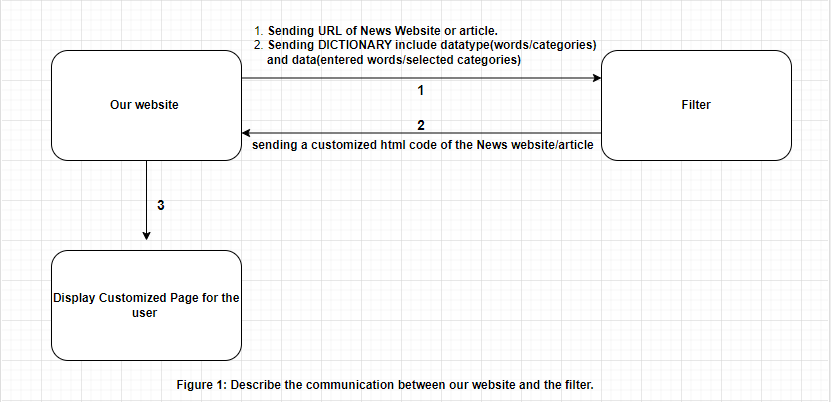
## **2.5 Engineering Process**

Our engineering process began with an in-depth investigation and understanding of the REQUESTS and BeautifulSoup libraries, which we chose to use in Phase A. In addition, we conducted extensive research on other recommended Python libraries and methods for fetching and analyzing web content, comparing them to BeautifulSoup and REQUESTS. At the end of this process, we decided to continue using these two libraries because they offer ease of use and efficiency in data processing, making them the preferred choice for our project. As part of this process, we performed a comprehensive review of the structure of various news websites. Our goal was to identify common patterns in their structure and code, enabling the development of a generic crawler suitable for a variety of these sites. Concurrently, we began learning the principles of website development using the Flask framework, aiming to understand its unique features and find an efficient way to integrate the site with our filtering system.

## **2.6 Implementation**

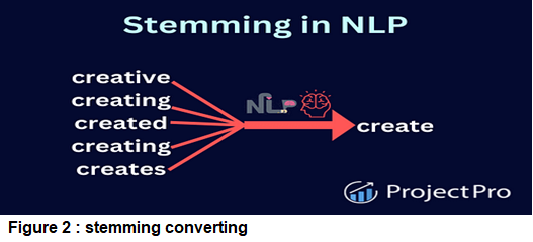
Our implementation is divided into **two** separate parts: the implementation of the filter and the implementation of the website itself. The website communicates with the filter as a service, sending it the necessary data, which includes the address of the news website or article the user wants to filter and the type of filtering (category-based/words-based), along with the specific filtering data (entered words/marked categories).

The filter receives the data and parses it to determine the type of filtering, then performs the required filtration on the website entered by the user. After the filtration process is completed by the filter, the sanitized code of the link that was sent is returned to the website. This ensures that the requested page in the news Website is personalized for the user and doesn't contain unwanted articles. (See Figure 1)



### **2.6.1 Base Definition**

* **NLTK.STEMMING**

Stemming is a natural language processing (NLP) technique used to simplify words by reducing them to their root or base form, known as the stem. This process aids in text normalization by removing affixes such as prefixes and suffixes, thus reducing multiple inflected forms of words to a common stem (see figure 2). For example, consider the words "connections," "connected," and "connecting." Applying stemming to these words would yield the common stem "connect." While the resulting stem may not always be a valid word in the language, it preserves the core meaning of the original word. Stemming is widely used in various NLP [6] tasks, such as information retrieval and sentiment analysis, to improve efficiency and accuracy in processing textual data. [7]

### **2.6.2 Filter Implementation – Detailed Breakdown of the Filtering Process**

Our filter implementation is divided into **Two Main Stages**:

**Stage A.** Identifying/Determining undesirable articles on the page.

**Stage B.** Manipulating the page and assessing it to personalize it for the user.

**Stage A: Determining undesirable articles**

This stage consists of **three** parts:

**Part 1.** Extracting the content and links from the requested page.

**Part 2.** Extracting content from each link.

**Part 3.** Checking the content and determining which links/articles to filter.

**In the first part**, we extract the HTML content of the main page of the news website using the `requests` library. Then, we extract the links from the content using `BeautifulSoup` by searching for all `<a>` tags in the extracted page and saving the `HREF` attribute of each tag. These links represent articles on the website as well as other links.

**In the second part**, combined with the **third part**, we take all the links on the page and start the process of extracting the content of each link. Then, we extract the textual content, such as the content of a specific article. After extracting the textual content, we perform stemming on each word in the textual content. **Finally**, we check if the converted content contains any undesirable content, i.e., one of the words entered by the user. Figure 3 illustrates this entire process.

**Filtering Techniques:**

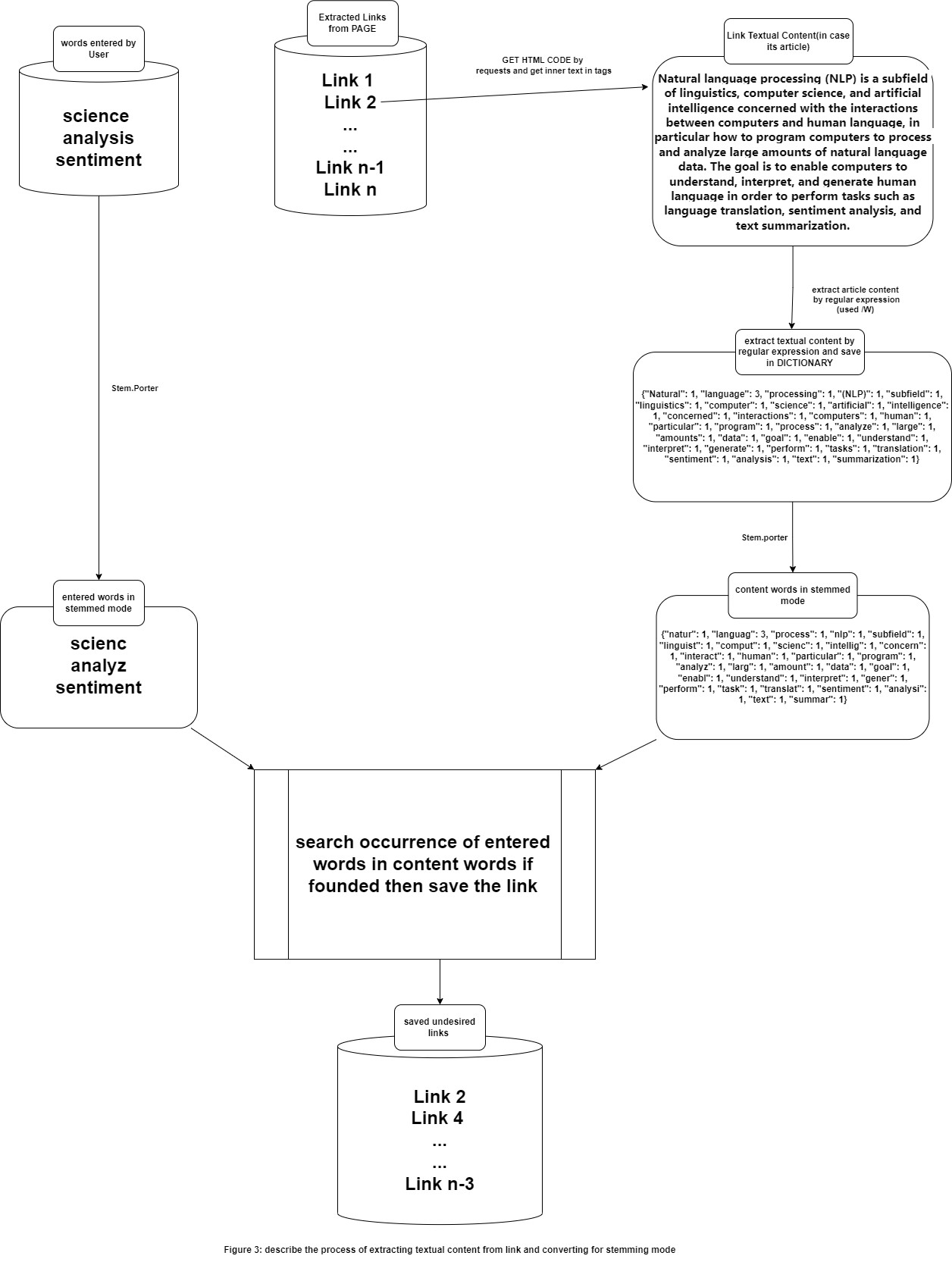
In our project, we have implemented two distinct filtering techniques, each one utilizing a different approach to address specific types of unwanted content but both techniques employing stemming process. These techniques are designed to enhance the user experience by allowing for a more tailored browsing environment, effectively minimizing exposure to undesirable information. Here, we detail the processes involved in each technique, explaining how they function and are applied within our system.

**1. Word based filtering:**

In word-based filtering, the user selects the option "ENTERED WORDS" and enters up to three words that he do not wish to see in articles. For example, if the user enters the word "murder", any article on the website containing the word "murder" or its variations is hidden and not displayed to the user. The way we implemented this is by directly searching for occurrences of the entered words within the article content. If any occurrences are found, we save the link to the article for removal later. To elaborate further, as described above, we extract the textual content of the article using BeautifulSoup and regular expressions, extracting word by word from the page and storing them in a dictionary. Then, we perform stemming on each word in the dictionary to convert them to their root form using nltk.stemming in Python. Additionally, we convert the entered words to their root form. Finally, we check for the occurrences of the entered words by the user with the words in the dictionary.

**2. Category based filtering:**

In category-based filtering, the user is presented with three main categories and can select the desired categories they want to filter articles associated with. Our solution here is to prepare an array in advance for each category, in which we store the most common and relevant words related to the category, preserving the words themselves in their root form. Then, in the final stage, we search for occurrences of the words in these arrays within the words in the dictionary, which essentially represent the content of the article. If any of the words appear, the article is marked as undesirable, and its link is saved to hide it during the page editing and display stage.



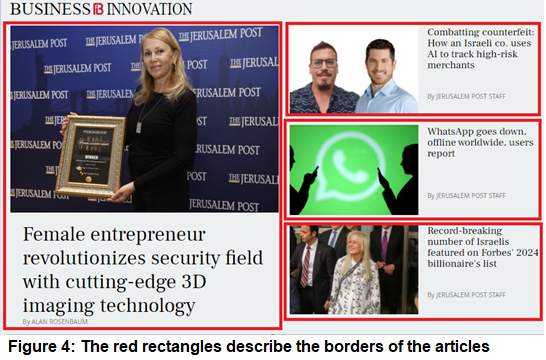
**Stage B: Page editing**

This stage is divided into **two parts**:

**Part 1.** Removing unwanted articles from the page and converting relative links to absolute links.

**Part 2.** Editing the HREF attribute of articles/pages on the news website.

After completing Stage A, where we identified all the unwanted links, we move to Stage B to edit the HTML page and remove these unwanted articles. In the **first part**, we begin by using BeautifulSoup to search for the **<a>** tags associated to unwanted links. We then delete all tags related to these unwanted articles, which includes any tags within the boundaries/borders of the unwanted articles (see Figure 4 for article borders). After deleted these tags, we proceed to the **second part**, where we convert the HREF attributes of all remaining articles/pages. This conversion ensures that when users click on links, they make GET requests to our website instead of to the original news website. This allows us to filter the page content before displaying it to the user (for a full explanation, refer to section **2.7** ). Finally, we return the modified HTML code from the filter back to our website, which then presents the cleaned and personalized page to the user.

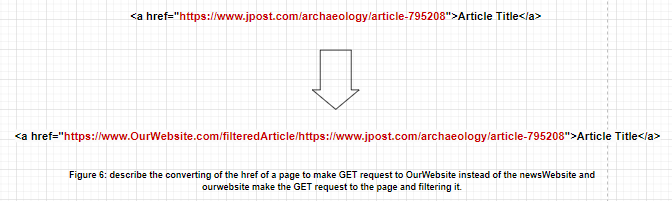


### **2.6.3 Website Implementation – Detailed Breakdown of the Website Development Process**

Our website development, which incorporates a filter, was carried out using the Flask framework. During the construction of our website, we began by creating the web pages (HTML), and designing them which achieved through Tailwind CSS and standard CSS. Alongside this, we added dynamics to the pages using JavaScript and Jinja2 [8], a built-in templating engine in Flask. For example, in using JavaScript, we stored data in Local Storage on the client side, such as preferences for dark or light mode, and performed various interactions such as enabling and disabling form/inputs fields, hiding and showing fields, changing field colors, and setting listeners for specific actions. All this was done to provide the user with an interface that is easy, fast, understandable, and efficient. With Jinja2, we displayed dynamic content that was passed as parameters with the page, such as historical table data and various forms pulled from the database. The HTML pages of the site were stored in a folder called 'templates', while CSS files and images were kept in a folder named 'static'. We used SQLite as our database along with SQLAlchemy, which helped us manage the database and build queries efficiently and easily. To secure the site, we performed actions such as hashing passwords and setting access requirements (annotations of required) for users who are not logged in or are unauthorized. Additionally, we created customized Error pages like 404 and 401 etc., to present clear messages to users about errors and help them understand what went wrong. The site was divided into separate scripts, each dedicated to a specific area: the 'routes' script included all the paths to pages and the presentation of each page upon reaching a certain path, another script dealt with 'models', i.e., database queries. Another script handled 'forms', which included all the different forms we used on the pages. This division facilitates the management and maintenance of our website and allows updates and changes to parts without affecting the entire system, which enhances the efficiency of development and maintenance.

## **2.7 Decisions**

* During the development process, we made several critical decisions regarding our approach to filtering and displaying pages. For example, initially our approach to filtering and displaying pages was to crawl all the website's pages, filter out all unwanted links, perform preliminary editing on all pages, and customize them before displaying them. However, this approach was not very efficient because it took a long time for the filter to complete checking and filtering all the website's pages. Additionally, it resulted in a waste of resources and editing many pages that the user might not request while browsing the site, causing a heavy load on the news website's server due to the performance of many GET requests in a short time.

Therefore, we **decided** to change direction and perform real-time filtering. As long as the user does not request a page, there is no need to filter it. When a user requests a page, the filter performs filtering, and the filtered page is displayed to the user. So, we proceeded with stage two, editing the links of `<a>` tags, i.e., editing their HREF attribute (see figure 6). Instead of containing a direct link to the page, they contain a link to our website. Then, our website fetches the content of the desired page using `requests` and displays it within our site customized.

* **Another decision** we made during our project development was choosing between using the Requests library, which pulls page content directly from the server, or replace it with an automation tool like Selenium, Pyppeteer, which simulates a full user interaction. This dilemma arose because Requests sometimes does not extract all the functioning page functionality, such as non-working buttons in the Navbar, search button not working, content that does not load during scrolling, and advertisements that do not appear. Initially, we think to switch to a tool that could extract all the dynamic content of the page, including JavaScript, which enables full functionality. We found that automation tools like Selenium are capable of extracting all the content and functionality required, but they suffer from slowness [9] and responsiveness issues compared to requests because they simulate real interactions like real user with the page, load it in a headless browser, and perform full interactions before extracting the content. Due to its slowness, we ultimately decided to stick with Requests. Its speed and efficiency in fetching static content were preferable for our project needs, especially since the news sites are primarily intended for reading and do not require complex dynamic loading or complex user interactions with the page, and many times requests fetch the page completely and functional including all its functionalities.

## **2.8 Frameworks**

Throughout the development of our website, we harnessed a suite of essential frameworks. These frameworks were pivotal in shaping our website and the filter, equipping us with the tools needed to craft a robust and user-friendly solution. In the upcoming section, we'll introduce these key frameworks, providing insights into their functionalities and underscoring their profound impact on our website's development trajectory and the filter.

* **Flask**  
  Flask is a Python micro web framework known for its simplicity and adaptability. It has been widely adopted by major platforms like Pinterest and LinkedIn, showcasing its scalability and reliability. What sets Flask apart is its intuitive syntax, extensive documentation, and strong community support, making it accessible to developers of all skill levels. With its broad range of extensions and seamless integration with other tools, Flask is favored for projects of varying sizes, from small applications to large enterprise solutions. Its minimalist design and proven performance in real-world scenarios make it a top choice for modern web development, solidifying its position as a leading framework. [2]
* **Tailwind**

Tailwind CSS is a unique open-source CSS framework known for its unconventional approach. Unlike traditional frameworks such as Bootstrap, Tailwind does not provide pre-defined classes for elements like buttons or tables. Instead, it offers a wide range of "utility" CSS classes, enabling developers to style elements by combining them. This utility-first concept distinguishes Tailwind and empowers developers with granular control over styling elements. Tailwind requires applying a combination of classes like bg-yellow-300 and font-bold to achieve the same effect. [10]

* **SQLAlchemy**

SQLAlchemy is a Python toolkit and Object Relational Mapper (ORM) that simplifies the use of DB like SQLite, MySQL, and PostgreSQL etc… by harnessing the full power of SQL. It offers a comprehensive range of enterprise-level persistence patterns, optimized for high-performance and efficient database interaction, all integrated into an easy Python language. [11]

* **SQLite**  
  SQLite is a lightweight relational database management system (RDBMS) known for its simplicity. SQLite can operate without a separate server, in addition, SQLite stores all the data in a single file, which makes it highly portable and widely used across various web apps and applications. [12]
* **BeautifulSoup**

Beautiful Soup is a Python package used for parsing HTML and XML documents, even those with malformed markup. It constructs a parse tree for documents, enabling extraction of data from HTML, which is particularly valuable for web scraping tasks. In terms of usage, Beautiful Soup represents parsed data as a tree structure, allowing it to be searched and iterated over using standard Python loops. For example, it can be used to load a webpage (e.g., News Website main page) using the urllib library [13], then parse the document and search for specific elements, such as links, within it. [5]

* **Requests**

Requests is an HTTP client library for the Python programming language. It simplifies the process of making HTTP requests in Python programs, providing an easy-to-use interface for sending HTTP requests and handling responses. With Requests, developers can interact with web servers, fetch data from APIs, and perform various HTTP operations with minimal boilerplate code. It is widely used for web scraping, API integration, and building web applications in Python. [4]

# **Challenges**

Throughout the semester, we faced a variety of challenges some of which were related to the implementation of phases A and B, and some that were general for the entire development process. In the subsequent discussion, we will outline these issues and describe the approaches we used to overcome them.

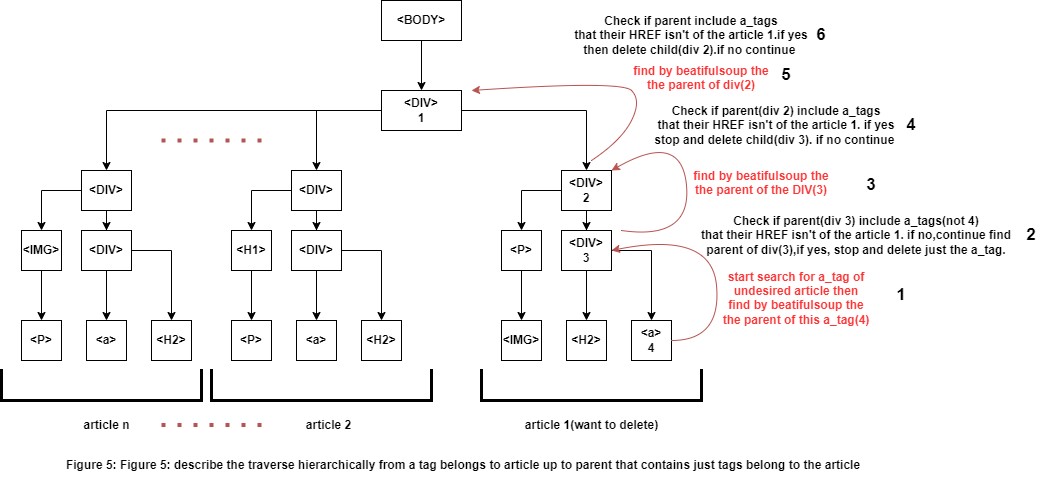
**Challenges in Stage A**

The **challenges** we faced in this stage and our approach to handling them are as follows:

One of the challenges was how to extract the textual content of the page since there is no uniform structure for HTML pages, and it's not known which tags contain the texts. We addressed this challenge by using beautifulsoup and regular expressions [14], particularly soup.get\_text() and `\W`, to extract the textual content and store it in a dictionary. Essentially, we split the entire content of the page into words stored in a dictionary. Using a dictionary, especially for storing the words of the article, was crucial to expedite the search for words because Python dictionaries are implemented using hash tables, enabling word search in constant time complexity O(1). [15]

**Challenges in Stage B**

At this stage, we faced a **challenge** of determining the borders of the articles or explicitly identifying which tags on the page belong to the article we want to delete and which do not. To overcome this challenge, we examined how article-related tags are structured within the HTML code on many news websites and found that in most cases, article-related tags are nested within a parent tag that contains them all without tags that not belong to the article. Therefore, our task became to find and delete this parent tag.

Our approach to tackle this challenge and find the parent tag (article borders) was to **hierarchically traverse** from one of the HTML tags belonging to the article up to its parent tag, then check if it contains other tags related to another article. If so, we delete only the child tag. If not, we continue traversing up to the parent of the parent tag (grandparent) and repeat the process until we reach a parent tag that contains tags belonging to another article. Then, we delete the child tag. Practically, using BeautifulSoup, we found all `<a>` tags on the page and checked those whose HREF contains links to unwanted articles (links received from the first stage). Starting from the `<a>` tag of the article, we found its parent tag using BeautifulSoup (`a\_tag.parent`) and checked if it contains other `<a>` tags. If so, we examined their HREF attributes. If they were different from the link to the article or to another unwanted article, we deleted the child tag. If not, we continued searching for the grandparent tag (`a\_tag.parent.parent`) and repeated the process. (See Figure 5)

**Another challenge** we encountered was that some websites use relative paths for articles, images, or CSS and JS files, rather than explicitly specifying the path. To address this, we searched for tags that could contain path fields like link, script and img tags using BeautifulSoup and checked if they were relative or absolute. If they were relative, we converted them to absolute paths by adding the website's domain name. This check and conversion were performed using the `urllib` library in Python.

**General Challenges**

During the development phase, we faced a general challenges including:

## **Uniform structure for HTML pages:**

One of the most significant challenges we face is the lack of consistent structure in HTML pages. The structure of web pages varies from site to site and even from page to page within the same site, depending on how the developers built them. For example, when extracting textual content from articles, we may not know which tags the content resides in; it could be within `<div>`, `<p>`, or other tags. Similarly, when removing an article from a page, we may not know which tags belong to the article. Therefore, we had to find a consistent approach that could accommodate different page structures. In the solution section, we explain how we addressed these issues.

## **Filtering Time (Flask Session & Requests Session):**

The process of crawling a page and extracting all the links, then going through each link and extracting its content to determine which pages or articles to filter within the news website, is a time-consuming task that varies depending on different factors. These factors include the number of links present in the page, the response speed of the requested server, the extraction of textual content using BeautifulSoup and regular expressions, and the removal of unwanted parts from the page that will be displayed to the user. In our project, this was one of the most critical issues we tried to address, as the filtering process on a single page of the website took about two minutes to complete and present to the user. To tackle this challenge, we checking many pages on news websites and noticed that each page contains many repeating links, such as navigation and footer links, as well as some articles. Therefore, we looked for a way to avoid re-extracting the content of these links and rechecking them on every page that user request to filter. We saved the links, including their classification (whether they are desired or not desired), and thus, for each new filtering request, we first check if the links exist in our saved repository. If they do, we do not check them again, if not, we extract the link content and save it for filtering future pages. To implement this action, we found that Flask offers a session (Flask session) [16] for each user, similar to cookies in the browser, which allows us to save information about the specific user and useful data during browsing. Within this session, we stored a dictionary containing a key (link) and a value (yes/no, for filtering/not for filtering), the result is that only filtering the main page takes a bit more time than usual, while other pages do not, because many of the links' classifications are already known, Additionally, we found that the Requests library takes a few seconds per content request for each link because it sends a new request to the server each time, including all settings and permissions, which slows down the process. Therefore, we explored how we can improve this by using what's called a Requests session, which maintains an open connection between the session and the website from which we request content, known as an HTTP persistent connection. This technique eliminates the need to resend all settings and permissions with each request, significantly reducing the time required for filtering. This is an efficient way when we make multiple requests to the same server. Ultimately, this approach worked, and we managed to reduce the filtering time to less than a minute for some sites. As previously mentioned, this also depends on the physical location of the server you are querying and how quickly it responds—for example, filtering times for Israeli news sites were much faster compared to a news site located in the United States, likely because the server domain is physically closer to us than one located far away.

**Blocking by news Websites servers:**

One of the significant challenges we encountered was having our requests blocked by news websites. Our system, which is designed to extract all links from a page, then extract and process the textual content for each relevant link was occasionally suspected of a DoS attack [17] due to the high frequency of requests it generated. Therefore, we found a solution by changing the order of operations: rather than extracting all the links and then their textual content, we began by directly extracting the textual content from each relevant link we extract. We then immediately converted this content to its root form (stemming), checked the occurrence of words, and classified it. This method created a sufficient time gap between each request, allowing us to continue with our crawling and extraction operations without being blocked by the news servers.

## **Scraping:**

Scraping (crawling) is an operation that requires a thorough understanding of the specific structure of the website from which data is to be extracted. [18] This requirement posed a significant challenge for us when we aimed to scrape multiple news websites, each with its own unique layout and data format. To address this issue, we tried to create a generic codes that could adapt flexibly to various website architectures. This solution enabled our scraper to effectively handle different structures, ensuring efficient data extraction across a diverse range of news platforms. By creating this versatile scraper, we successfully navigated the complexities of working with multiple web environments, streamlining our scraping processes.

# **4. Testing**

To ensure proper functionality and reliable performance of the system we build, we adopted a comprehensive approach to system testing. The testing process was divided into three main parts: unit testing, integration testing, and end-to-end (E2E) testing. Additionally, the tests were further divided into three separate sections: **tests for the filter**, **tests for the server (Back End)** side of the website in conjunction with the database, and **tests for the client side (Front End)**. Each section of the tests included a combination of different scenarios to cover all the required functionality and potential failure modes.

**1. Unit Tests for the Filter:** Initially, we conducted unit tests on the filter itself. The purpose of these tests was to ensure that the filter accurately performs the filtering and editing of the website pages collected through data analysis techniques. Every functionality of the filter was thoroughly tested to ensure it correctly identified and removed undesirable content.

**2. Unit Tests for the Server:** Concurrently, we conducted unit tests for most of the functions activated on the server side of the website. This included integration with the database and verification that information was correctly transferred between the database and various interfaces. It was crucial for us to ensure that the server could handle a high load and properly manage all user requests.

**3. E2E Tests for the Client Side:** Finally, we conducted end-to-end tests using Selenium, which is a software testing automation tool that simulates user actions on web interfaces. Selenium allows for the execution of scripts that emulate real interactions with the browser, including page navigation, clicking on elements, and entering text. The aim of these tests was to ensure that users could smoothly and intuitively navigate through the various pages of our website and achieve the desired results without errors or issues or bugs. [19]

Overall, the combination of these three types of tests allowed us to build a complete picture of the system's stability, reliability, and efficiency. Each scenario we tested was a vital testing tool that contributed to the project's success and the final quality of our product.

We **summarized** these tests into the following cases, which covered various scenarios:

|  |  |  |  |
| --- | --- | --- | --- |
| **Case** | **Purpose** | **Headline** | **Results** |
| **Filter** | | | |
| Crawling process | Ensure that the filter extract all the relevant links in the html page | Extract all links | Extract all relevant links from the html code and save in dictionary and set |
| Empty html code links extract | Return empty set and dictionary (without links) |
| Textual Content Extractor process | Ensure that the filter extract the textual content from a html code | Extract textual content | Extract the textual content from the html code by BeautifulSoup and regular expression and split for words and save them in dictionary. |
| exception handling in Extract textual content | The exception handling and returned an empty dictionary |
| Stemming words and searching words occurrences process | Ensure that the filter stemming the words correctly | Stemming words of the textual content | The words in the dictionary converted to the stemming mode |
| Stemming the entered words | The entered words converted to the stemming mode |
| Ensure that the filter check occurrence of stemming entered words in stemming textual content of page | Check stemming exist | Return True (meaning the stemmed entered words occurrences in the content) |
| Check stemming not exist | Return False (meaning the stemmed entered words not occurrences in the content) |
| Remove stop words | Stop words deleted from the dictionary of the textual content |
| Editing Page process | Ensure that the filter delete the undesired article from page in different situation | Delete html tags belong to undesired article with another articles in same parent tag | Parent tag that contains All tags belongs to the undesired article removed and not exist in the html code |
| Delete html tags of undesired article in a-tag | Just the a-tag of the undesired article removed and other articles not deleted |
| Ensure that the convert of relative paths to absolute paths done correctly | Convert relative path of links to absolute | The relative path changed to absolute path |
| Convert Relative path in SRC of img tag to absolute | The relative path of img path converted to absolute path |
| Change Attributes of images to SRC | All the IMG tags that not contain SRC(data-SRC or others) converted successfully to SRC |
| Ensure that the filter Convert the HREF in a-tags in pages to Our Website instead of the news website | Convert a\_tags HREF | All the relevant a\_tags HREF converted successfully in the html page |
|  |  |  |  |
|  |  |  |  |
| **Back End – Flask & DB** | | | |
| Registration and Login and update user data | Ensure that the inputted information is accurately processed and securely stored. | Registration | The new account was added to the database successfully, display success message |
| Failed Registration | The account with the same email or username exist in the database, returning failed message |
| Log in | The account Email and password was retrieved from the database, display success message |
| Failed Log in | The account with the same email and password not found or wrong in the database, returning failed message |
| Update profile | the account updated in the database, display success message |
| Failed update profile | Exist username or email in database, display failed message |
| FilteringHistory and optionData and enteredWords data | Ensure that the filtering data entered by the user stored correctly and accurately in the correct table | Filtering History table updated | The data added to the database successfully in the history table |
| OptionHistory table updated | The data added to the database successfully in the OptionHistory table |
| enteredWordsHistory table updated | The data added to the database successfully in the enteredWordsHistory table |
| User Access pages | Ensure that the user can access some pages just in case he is logged in and redirect to the correct pages | Login page in case user already logged in | The user redirect to the home page instead of login page(because he already logged in) |
| User home page instead of global home page in case user logged in | The user redirect to user home page instead of global home page (because he already logged in) |
| Access denied : Entered Filtering data page in case user not logged in | The user will get a page with error (status 401 (mean that he cant access this page) |
| Access success : Entered Filtering data page in case user logged in | The user will get the desired page (status 200) |
| Access denied : Entered update profile page in case user not logged in | The user will get a page with error (status 401 (mean that he cant access this page) |
| Access success : Entered update profile page in case user logged in | The user will get the desired page (status 200) |
| Access denied : Entered history page in case user not logged in | The user will get a page with error (status 401 (mean that he cant access this page) |
| Access success : Entered history page in case user logged in | The user will get the desired page (status 200) |
| Error pages (401,403,..,500) | Ensure that the user get the correct error page | Not exist Route (entered wrong link) | The user will get a page with error (status 404) |
|  |  |  |  |
| **Front End - SELENIUM** | | | |
| Registration and Login and update user data | Ensure that the website display correct message for the process done by the user and redirect to correct pages | Registration | Redirect to login page and Display success message "Account Created successfully for 'username' " |
| Failed Registration | Display failed message "Username already exist, Please choose another one!!!" |
| Log in | Redirect to user\_home page and Display success message "you have been logged in successfully" |
| Failed Log in | Display failed message "your email or password is wrong,…" |
| Update profile | Display success message "your profile has been updated successfully" |
| Failed update profile | Display failed message "the username already exist…" |
| history | Ensure that the filtering data that user enter, displayed in the history table | History table updated | History table updated with add new row to the table included the filtering details entered by the user (URL,filteringOption,filtering\_data) |
| Filtering process | Ensure that the filtering process done and display the appropriate message/data | Success filtering display(exist website) | The desired news Website entered by the user displayed customized. |
| Failed filtering display (not exist website) | Redirect for page with message "Filtering not success" |
|  |  |  |  |

# **User Documentation**

## **5.1 User's guide – Operating instruction**

### **5.1.1 General Description**

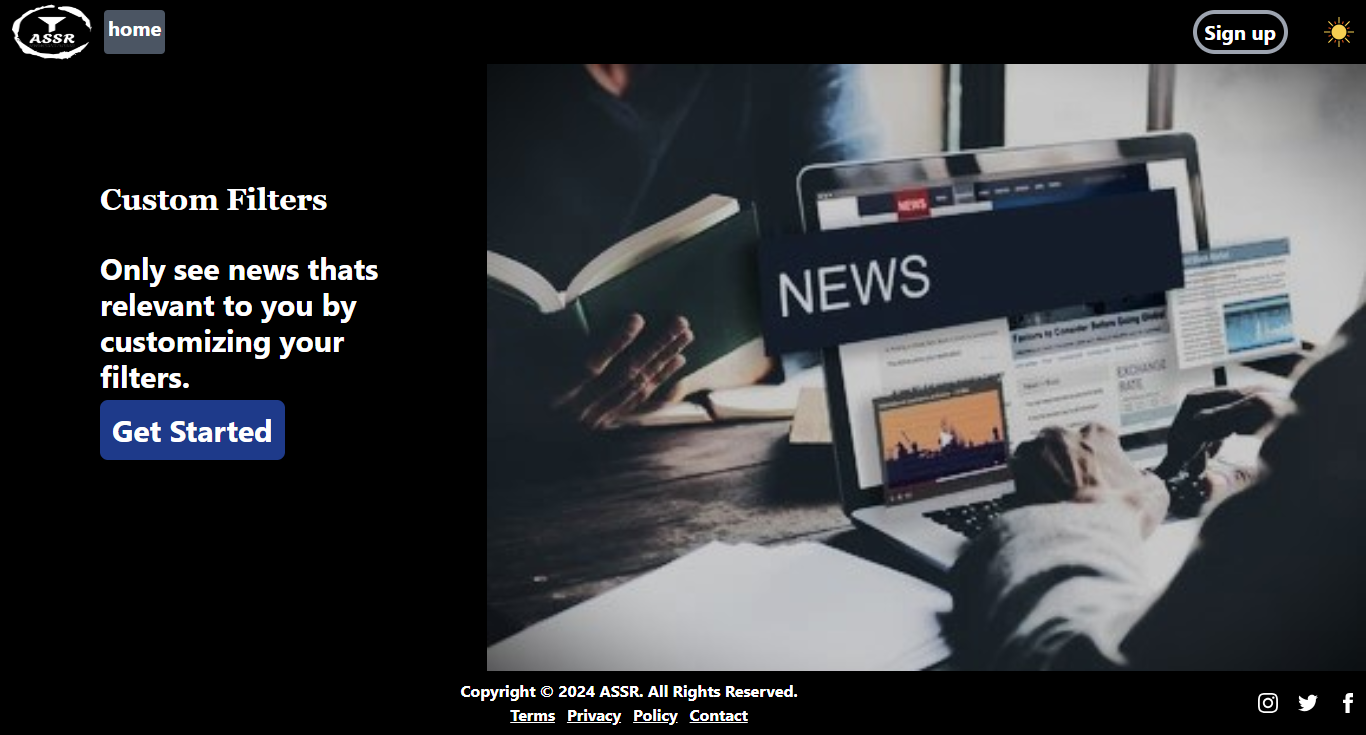
The website offers a unique filter designed for news websites, allowing users to customize their reading experience according to their personal preferences. Initially, users are required to enter the main link of the news website they wish to filter. Subsequently, they can choose from two filtering options: word-based filtering or category-based filtering. In word-based filtering, the user is asked to enter up to three words they want to filter articles that contain any of those words or their combinations. Alternatively, in category-based filtering, the user is offered three categories to choose from: politics, crime, and sexual content. Any article included in these categories will be filtered out and not displayed to the user.

After choosing the type of filter and entering all the required details, our system processes the request and displays the news website in a personalized version, with all unwanted content removed. Additionally, the system provides users with the option to view their filtering history, including a list of the URLs of the news websites that were filtered, the type of filter chosen (words or categories), which words or categories were selected, and the exact time and date of each filter.

The ability to monitor their filtering history enables users to verify and modify their filter settings as needed, allowing them to adapt to changes in their preferences or to improve their browsing experience at any time.

### **5.1.2 Operation instructions**

**General Pages for all users:**

**Home Page (Before logged in)**

this is the home page of the website its include an explanation and details about the website and what it offer for users and what types of things it can do.

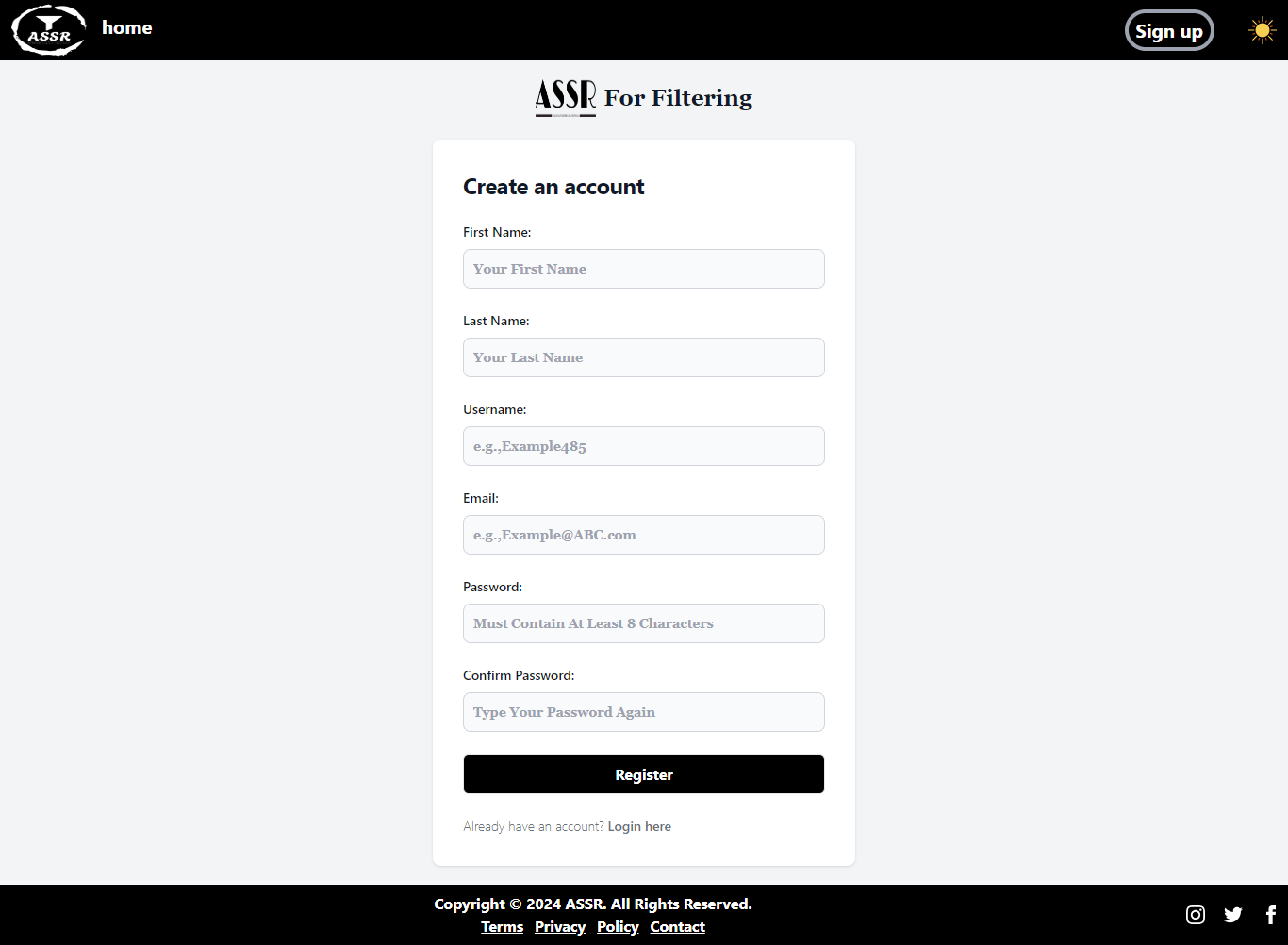


* Click on 'Get Started' to be directed to the login page. Here, you can enter your login details or proceed with logging in if you are did not registered yet.



* If you are not yet a registered user, click on 'Sign Up' to navigate to the registration page. Fill in the required details to create a new account and start enjoying the services we offer.

**Registration Page**

****

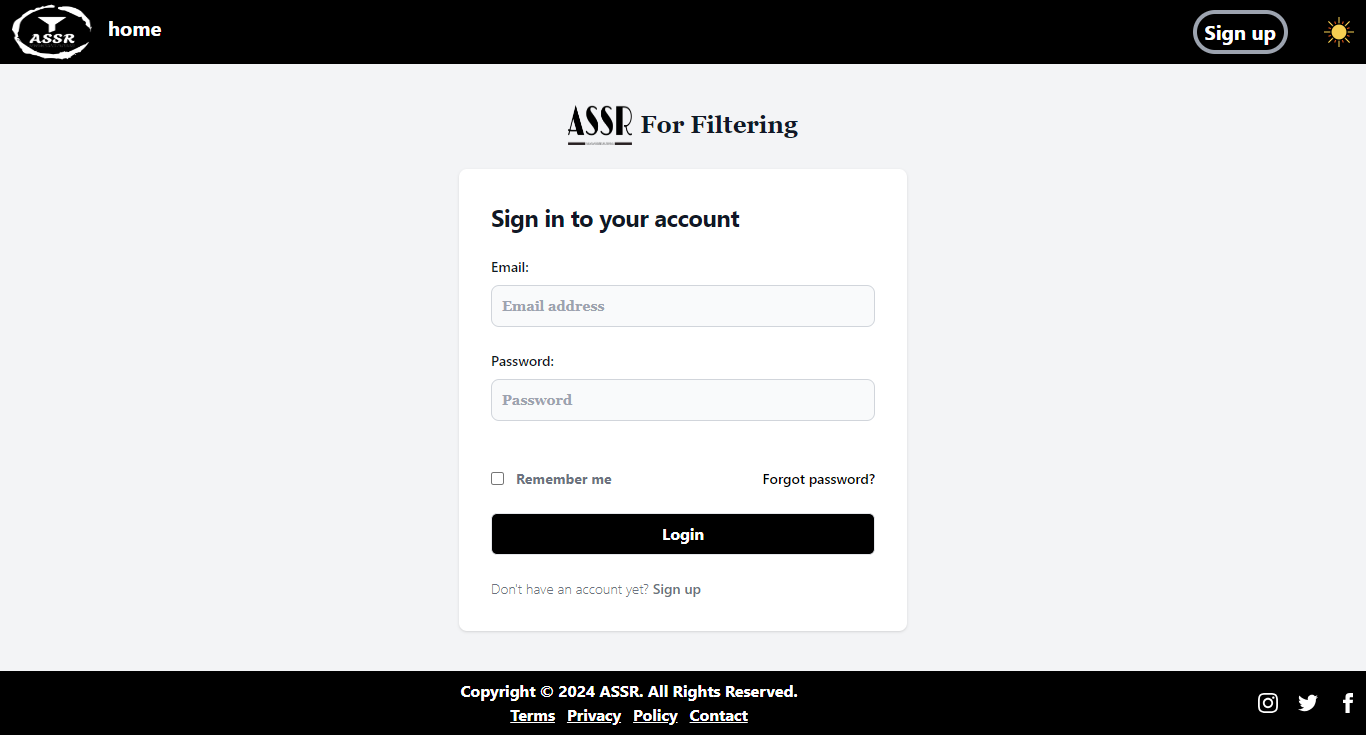
**Creating An Account :**

* Fill in all the fields of the form, then click the  button to create your new account. After registration completed successfully, you will be redirect to login Page.



* If you already have an account, click on 'Login here' to be directed to the login page**.**

**Sign in Page**

****

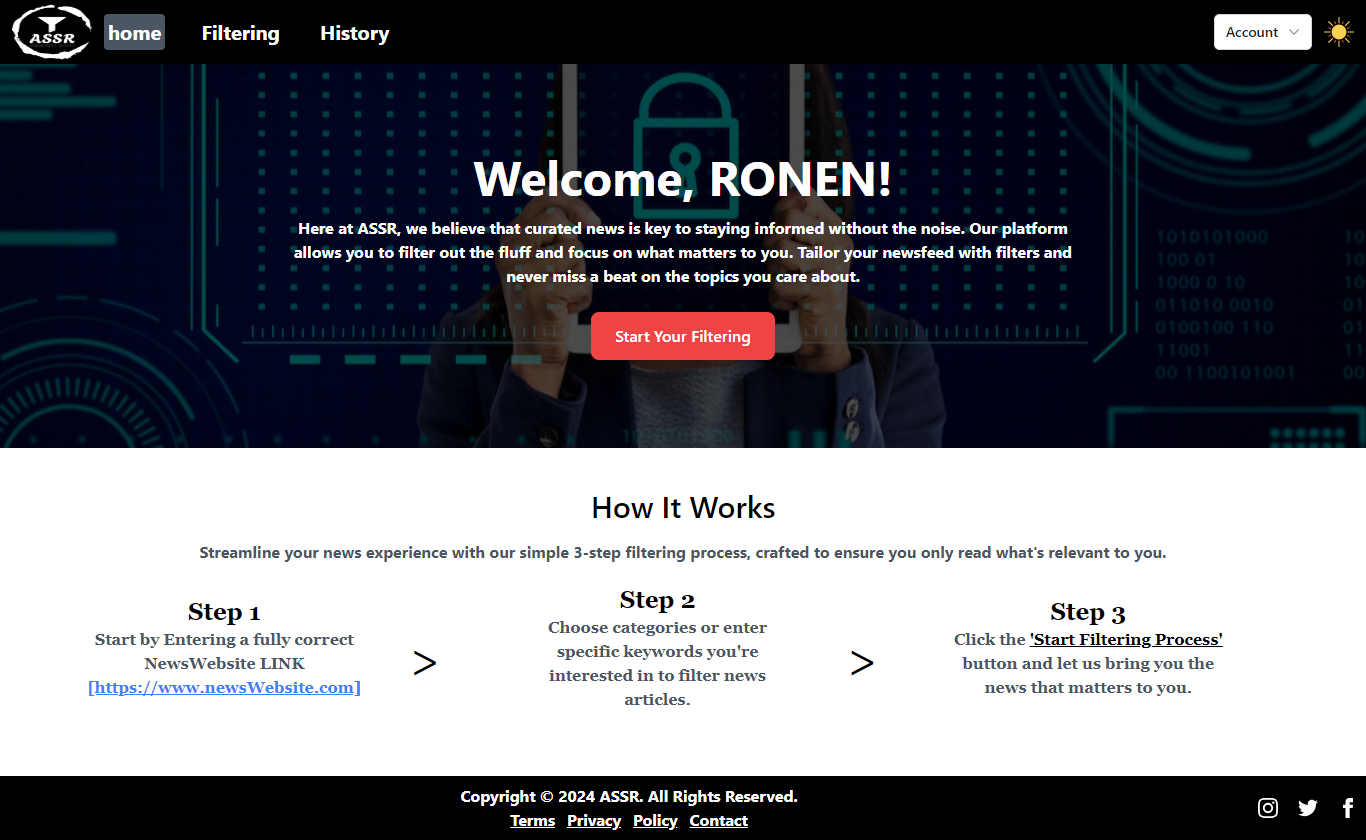
**Accessing your account:**

* Enter the email and password you registered with them

****

* If you did not have an account with us, click on 'Sign up' to be directed to the registration page.

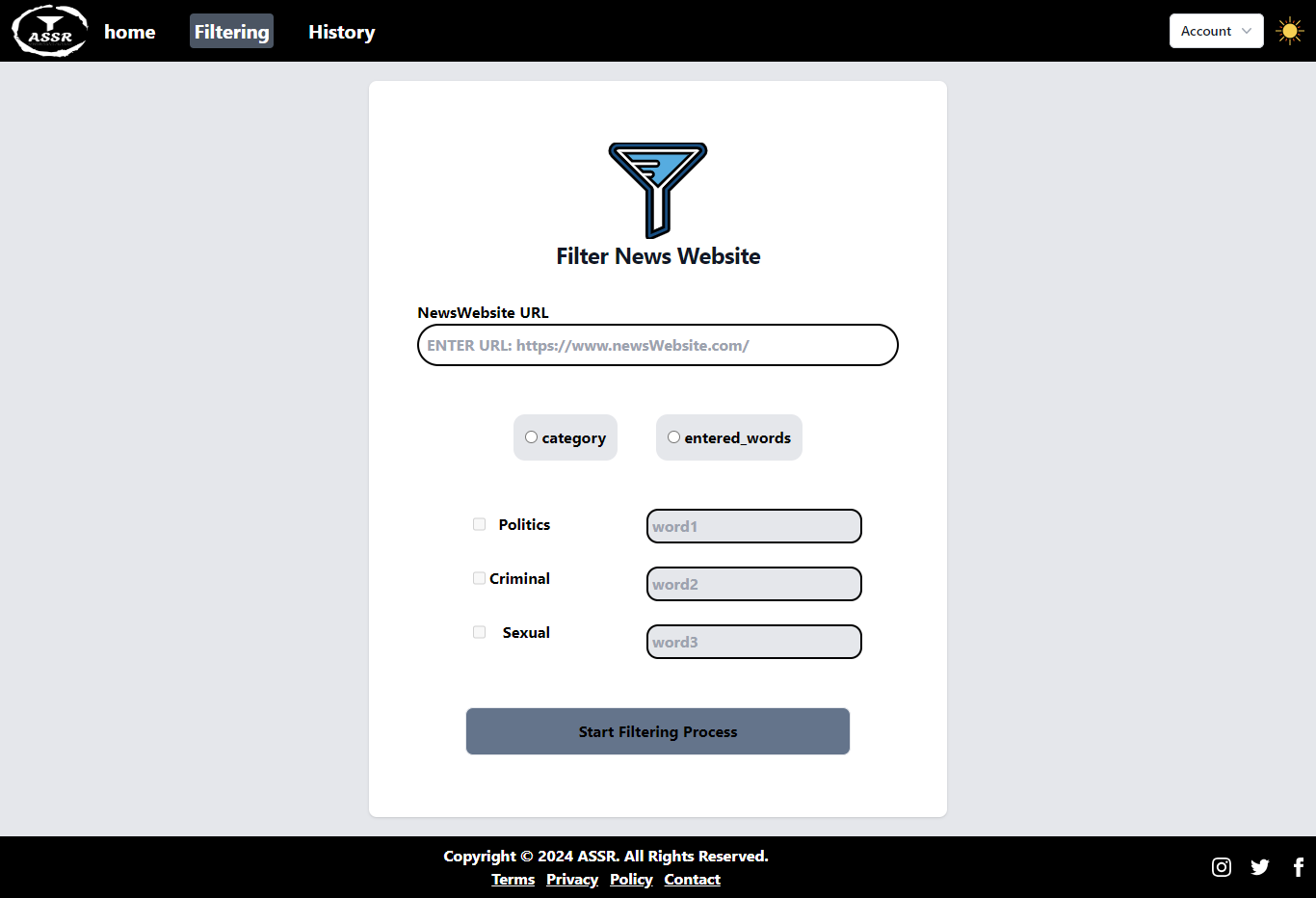
**User Home Page (After logged in)**

****

This is the home page for the user of the website. It includes an explanation and details about the website and how to start the filtering Process.



* Click on 'Start Your Filtering" to be directed to the filtering details page. Here, you can enter the news website you want to filter and details about the filtering

**Filtering Details Page**

The Filtering Details page allows you to tailor the news content according to your preferences.



* Enter the URL of the news website you want to filter content from.



* Choose the type of filtering you want.

**In case you choose filtering based categories :**

****

* Choose the categories you want to filter articles classified/belong for them.

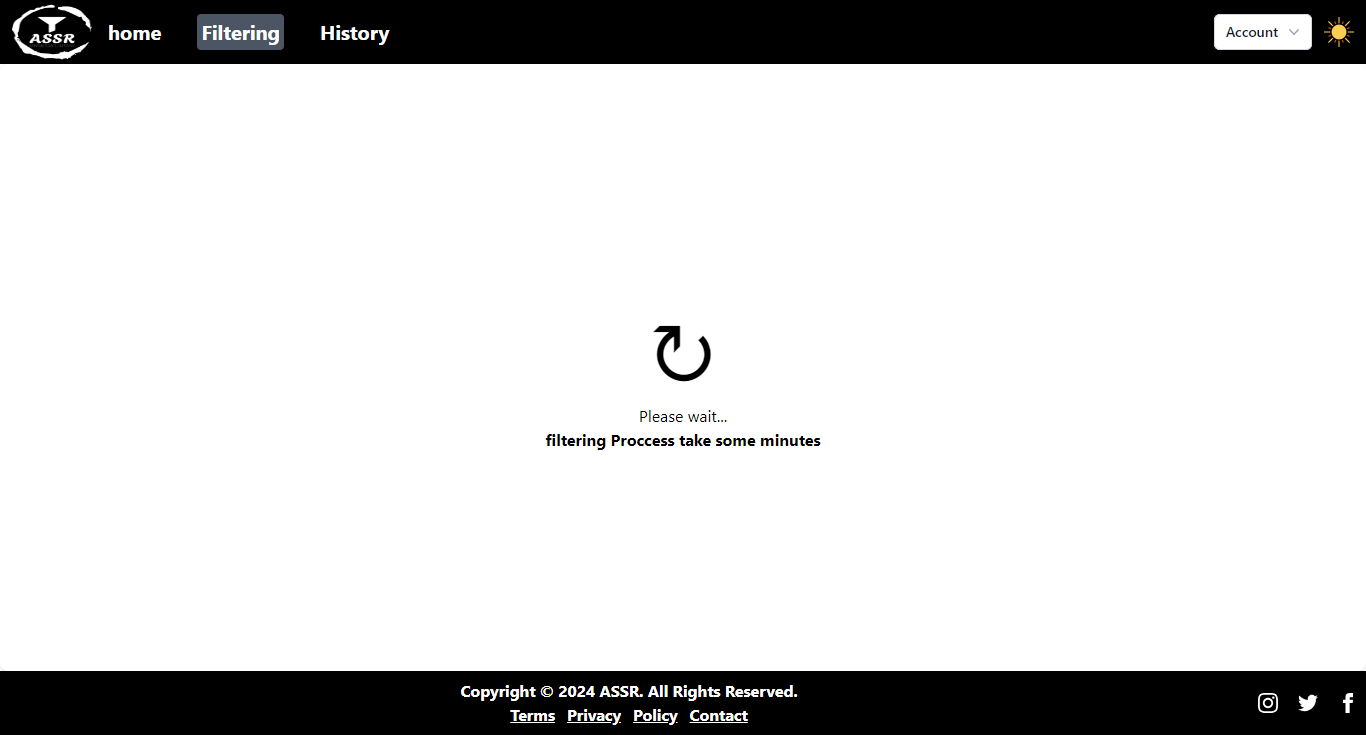
**In case you choose filtering based entered words:**

****

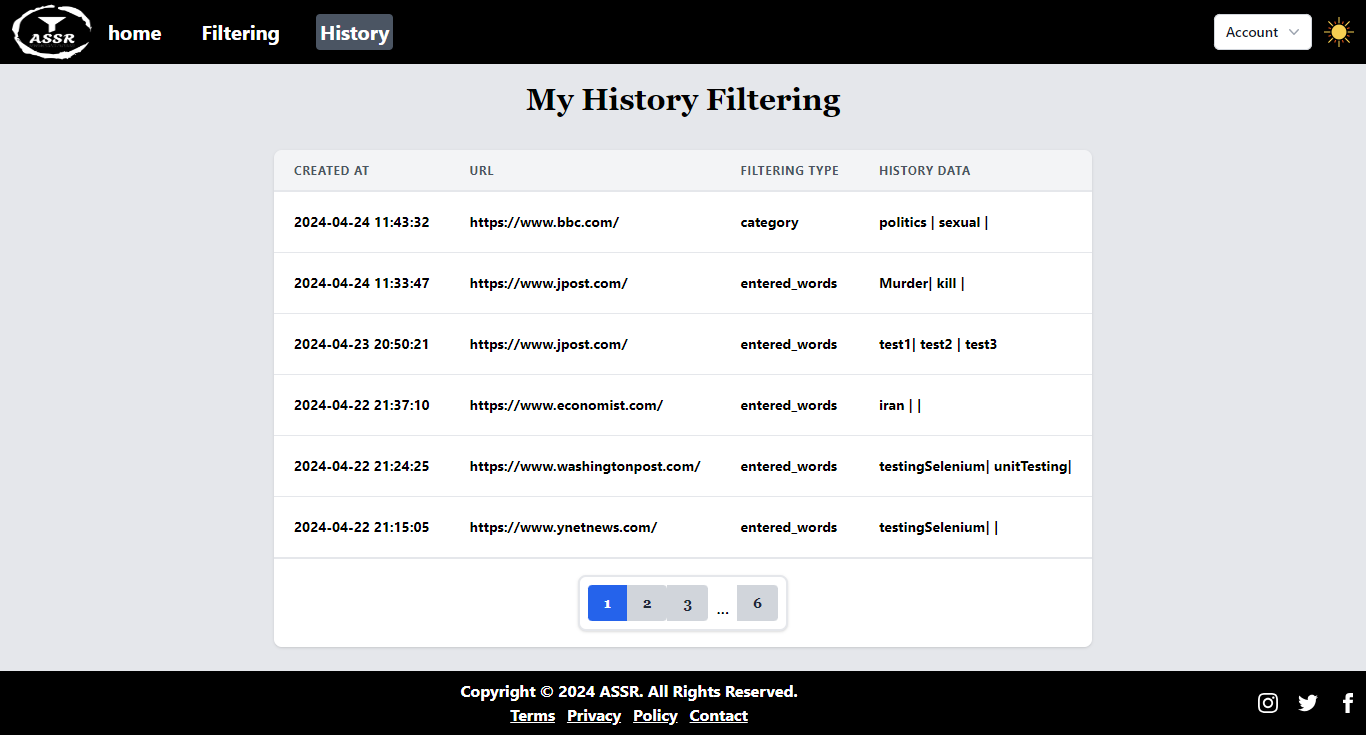
* Enter until 3 words you want to filter article included them or their conjunction

****

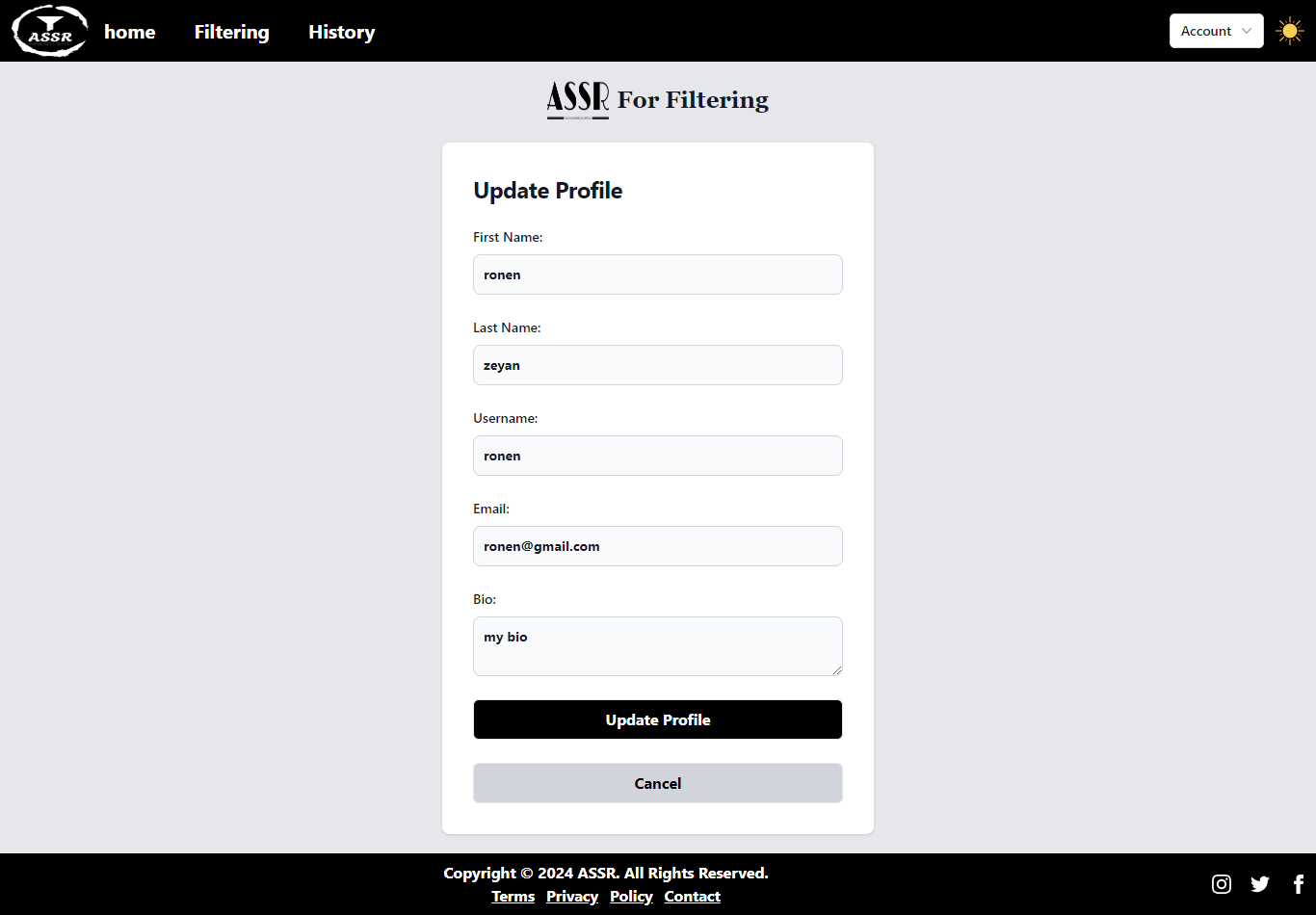
* Click on 'Start Filtering Process" to be start the filtering process.

**Waiting Filtering Process Page**

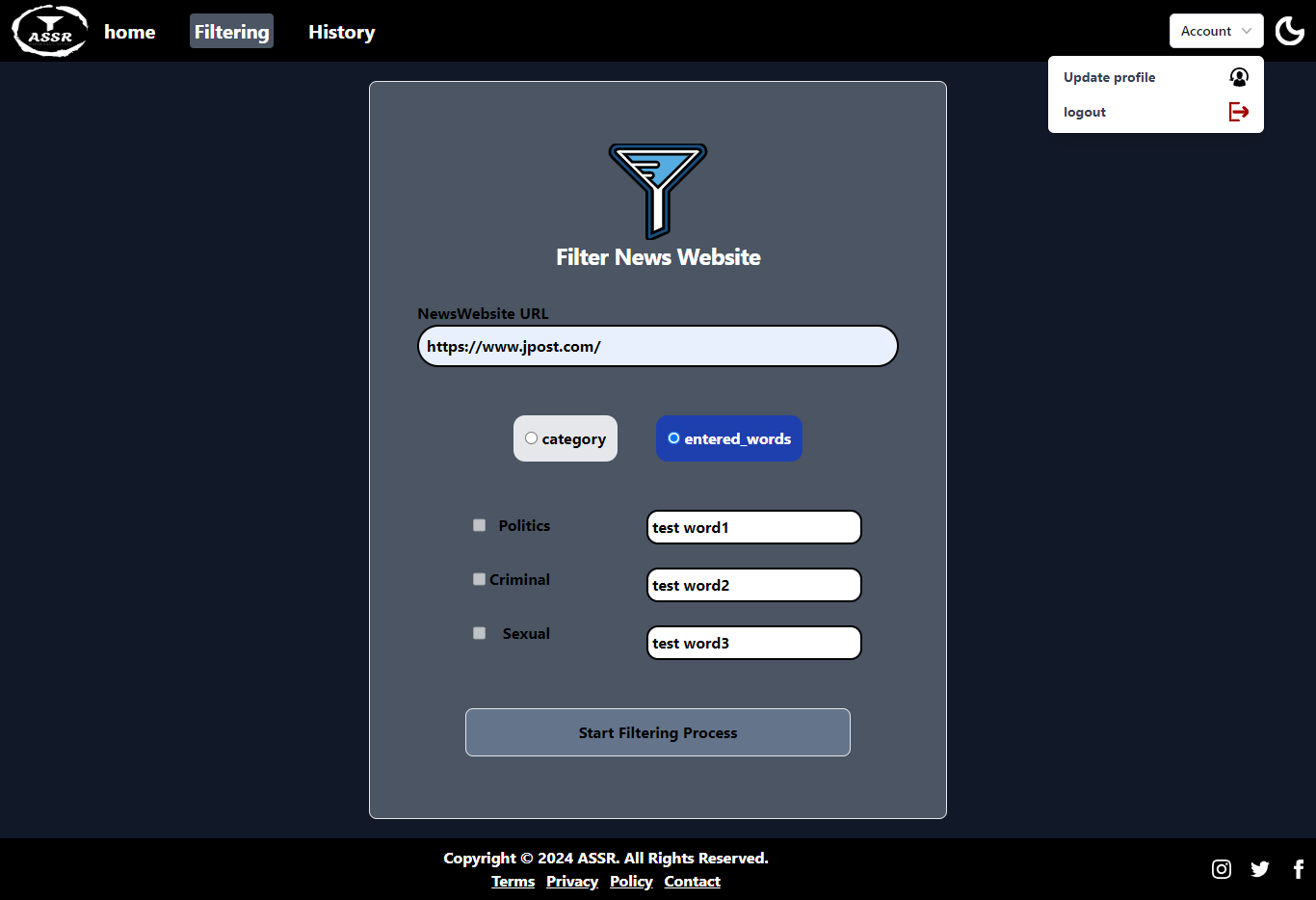
After entering your filtering criteria, a processing screen will appear while the system filters your content. The duration of this process can vary, **typically taking 1-3 minutes**, and may depend on your computer's performance, internet speed, and the response time from the news website's server. Please there is **no need to refresh** the page, your tailored news content will be displayed automatically once the filtering is complete.

**History Page**

On the History Page, you can review your past filtering activities. This page displays a comprehensive log of the news websites you've filtered, the specific filtering criteria you've used, and the keywords or categories you entered. The history presented in chronological order, with the most recent activity at the top, allowing you to easily track and revisit your filtering preferences and results. This table starts from your very first filtering action upon using the website**.**

**Update Profile Page**

On the Update Profile Page, you can easily modify your personal details such as your name, username, email, and bio.

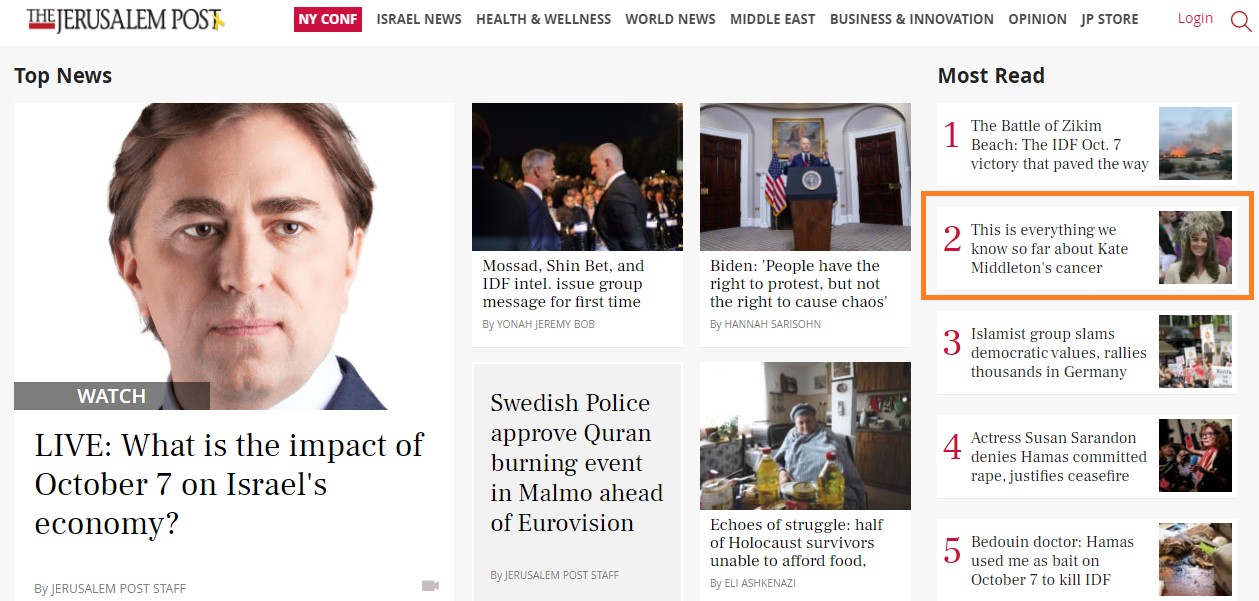
**Settings in all pages**

**Switch between Light Mode and Dark Mode by press  or for dark mode  .**

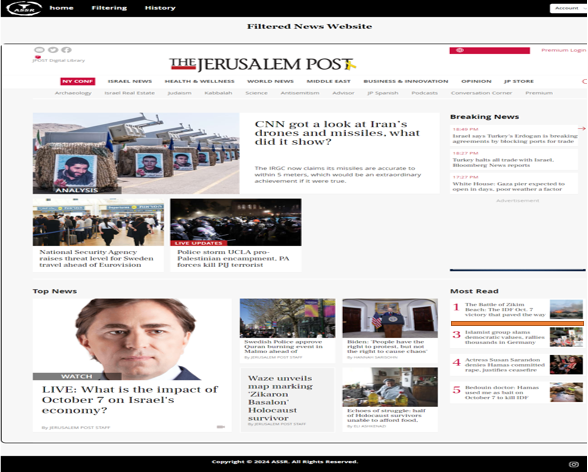
**Website Filtering Page (including an example of filtered news website)**

This is the page display how a news website appears after undergoing our filtering process. In this example we choose to filter out articles and pages discussing cancer.

**News website Before Filtering:**

Here is the news website before filtering. In the orange rectangle, you can see an article about cancer.

**news website after Filtering:**

and here is the news website after filtering, displayed within our website. The article that discussing cancer, previously marked in orange, has now been removed.

## **5.2 Maintenance guide**

Our project components have been organized in a way that ensures clarity and ease of maintenance. Below is the structure of the project files and description and role about the main folders and files.

## **Project Files Structure**

├─ 📁ASSR

│ ├─ 📁Templates

│ │ ├─ 📄Home.html

│ │ ├─ 📄displayHistory.html

│ │ ├─ 📄Base.html

│ │ ├─ 📄Header.html

│ │ ├─ 📄filterPage.html

│ │ …

│ │ └─ 📄newsWebsiteIframe.html

│ ├─ 📁static

│ │ ├─ 📁style

│ │ │ ├─ 📄style.css

│ │ ├─ 📁images

│ │ │ ├─ 📄logo.png

│ │ │ ├─ 📄…

│ ├─ 📄\_\_init\_\_.py

│ ├─ 📄filter.py

│ ├─ 📄models.py

│ ├─ 📄forms.py

│ ├─ 📄routes.py

├─ …

├─ 📄FilterTests.py

├─ 📄backEndTests.py

├─ 📄frontEndTests.py

├─ 📄run.py

## **Main Files of the project**

1. **ASSR Folder**

**Description:** Contains all the folders and files of the website and the filter files.

1. **Templates Folder**

**Description:** Contains all the HTML pages of the website.

1. **Static Folder**

**Description:** Contains all the static files in the website,divided into 2 files,CSS,IMAGES

* CSS Folder: Contains the website CSS files.
* IMAGES Folder: Contains all the images used in the design of the website

1. **Routes.py**

**Description:** Responsible for managing and displaying the appropriate HTML page.

**Role:** each method in this file responsible about display a html page and send & receive data from html pages and save in DB.

1. **Forms.py**

**Description:** Responsible for defining the forms on HTML pages, such as the registration and login pages, and for entering filter data, etc.

**Role:** Provides a convenient and secure interface for the user to collect data and perform form validations like maxLength,RequiredField etc…

1. **Models.py**

**Description:** Responsible for the tables that will appear in the database and includes the definitions of tables and fields.

**Role:** SqlAlchemy used the tables in the file to make CRUD operations and also it includes general functions about saving logged-in users etc...

1. **Filter.py**

**Description:** Responsible for the filtering process of the news website.

**Role:** Acts as the project web crawler/scraper, performing filtering based on predefined criteria.

This division into separate files for each component allows for better and more flexible management and maintenance of the website.

**Main functions in filter.py file :**

**Function Name:** 

**Purpose:** fetch the data from a given link(news website link or any article link).

**Inputs:**

* url(string): the url from which the data will be fetched.
* session(object): a object of requests to fetch the data. (faster than requests.get)

**Outputs:**

* soup or none: the parsed html code of the page,none in case of exception.

**Function Name:** 

**Purpose:** extract all the relevant links in the parsed soup.

**Inputs:**

* Soup(object) : a object of beatifulSoup including the parsing of the html page
* url : url of newsWebsite(used to identify the relevant links and convert relative links to absoute)
* set\_checkLinks: dict including the relevant links we want to check and classify.

**Outputs:**

* setLinks : set including the relevant links we extracting from the html code.
* links\_absoluteLinks : dict including the absolute link as key and link(link how they looked in the HREF of a tag)
* absolute\_Links : dict including the HREF of link and his absolute link (used in manipulate\_toDisplay\_htmlPage function)

**Function Name:** 

**Purpose:** edit the page. delete unwanted articles,change HREF’s of articles.

**Inputs:**

* Soup : a object of beatifulSoup including the parsing of the html page
* Links\_to\_delete : set included the links of unwanted pages.
* Absolute\_links: dict including the absolute links (helping for convert the HREF of survival pages/articles)
* url : url of the news website.

**Outputs:**

* Edited soup

**Function Name:** 

**Purpose:** classify the relevant links we extract and identify the unwated pages.

**Inputs:** including 5 parameters. (used by functions(stemming,extracting,classifing…)

**Outputs:**

* deleted\_links:links should be deleted.
* Classify\_links:classify of links (yes to delete/no).

**How to start our website in localhost:**

Currently, the website has not yet been deployed to a public domain, so it is not accessible through internet browsers. Until the deployment process is complete, the website can only be accessed through the local server. We are working diligently to finalize the deployment, and soon it will be available for access via the cloud. To start the application on your computer you have to follow the instructions below, in the future a public domain will be available.

**Website Maintenance guide:**

In order to maintain the code, you first need to install a few tools and setup your environment on your computer.

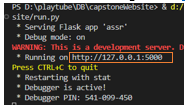
1. Python – [download python](https://www.python.org/)
2. VS Code – [download VS code](https://code.visualstudio.com/)
3. Clone the project from the following GitHub repository:

[[https://github.com/RonenZeyan/Capstone-Project-D-15-23-2-Phase-2].](https://github.com/RonenZeyan/Capstone-Project-D-15-23-2-Phase-2)

1. Start the VS Code and open the project folder within VS Code.
2. Open the terminal within VS Code and execute the batch file to install the necessary libraries and dependences by typing the following sentence **“.\install\_requirments\_for\_project.bat” (without “ “)** and wait for the installation to complete.

**\*\*(Note: Pay attention to which DIRECTORY you are in(**

1. Run the project by running the file **“run.py”** or write in the terminal **“python.exe run.py”.**
2. Open your web browser and navigate to <http://127.0.0.1:5000/> .



# **6. Results and Conclusion**

The main goal of the project was to create a website that allows users to filter news content based on personal preferences. Our site is based on advanced filtering methods, including filtering by categories and keywords, which help users tailor the news they see to their needs and interests and filter out unwanted news. The site operates by receiving a news website URL and performing content filtering based on the user-selected settings. The development included the use of libraries such as Requests and BeautifulSoup for extracting and analyzing website content, as well as the use of NLP techniques like Stemming to improve the efficiency of filtering and achieve high accuracy in the results.

During the development, we encountered several challenges, including optimizing the filtering process and ensuring the website's responsiveness and the filtering process time. For each challenge, we held creative thinking meetings where we analyzed the problem, explored possible solutions, and chose the most suitable solution. The choice of solutions was made with consideration for implementation simplicity and alignment with best engineering practices.

In hindsight, we would have opted for a different tool instead of REQUESTS. As previously mentioned, REQUESTS can fetch content, but occasionally not in its entirety, leading to inaccuracies or incomplete filtering across the entire page. Moreover, REQUESTS lacks the capability to extract the page's content, including its JavaScript, resulting in the loss of dynamics and certain functionalities of the site. Sometimes times, it also causes failure to display images especially those rendered using JavaScript within the page we edited and displayed for the user. Consequently, we transitioned to automation tools, despite their slower performance, particularly Selenium, despite our prior decision not to use it in decisions section above. We will choose it and try to optimize more our filtering process to compensate for the slowness of it.

The special with Selenium is that it retrieves content after the full page loads, not just the initial page from the server. This ensures comprehensive content extraction, facilitating complete filtering and ensuring the page's functionality by extracting JS content.

In addition, in our project, the Flask server manage the website and perform the filtering, in hindsight, We would have preferred to separate the website management and the filtering process into two distinct servers and establish an API for communication between them. This step would have allowed us to reduce the load on Flask server, which could then focus solely on running the website, while the filtering would be handled by a separate server. This will improve the system performance and responsive.

In the future, we intend to expand the repository of categories and words for filtering. In addition, we aim to extend our filtration capabilities beyond English-language news websites to also include Hebrew and Arabic. We will also consider developing a browser extension that would be more practical and allow quick and flexible access to filtering services.

Despite the challenges that we faced, we believe that the project has proven its effectiveness and offers a personalized browsing experience, providing users with maximum control over the content they consume. We are committed to continuing to improve the system to ensure user satisfaction and expand the impact of the project.

# **7. REFERNCES**

|  |  |
| --- | --- |
| [1] | “https://www.goodrx.com/health-topic/mental-health/is-news-bad-for-your-mental-health,” *Eric Patterson, LPC,* 2022. |
| [2] | “https://en.wikipedia.org/wiki/Flask\_(web\_framework)”. |
| [3] | “https://en.wikipedia.org/wiki/Web\_scraping#:~:text=Scraping%20a%20web%20page%20involves,fetch%20pages%20for%20later%20processing.”. |
| [4] | “https://realpython.com/python-requests/”. |
| [5] | “https://www.crummy.com/software/BeautifulSoup/bs4/doc/”. |
| [6] | “https://en.wikipedia.org/wiki/Natural\_language\_processing”. |
| [7] | “https://www.analyticsvidhya.com/blog/2021/11/an-introduction-to-stemming-in-natural-language-processing/”. |
| [8] | “https://www.geeksforgeeks.org/templating-with-jinja2-in-flask/”. |
| [9] | https://www.zenrows.com/blog/selenium-vs-requests#which-is-best. |
| [10] | “https://en.wikipedia.org/wiki/Tailwind\_CSS”. |
| [11] | “https://en.wikipedia.org/wiki/SQLAlchemy”. |
| [12] | “https://en.wikipedia.org/wiki/SQLite”. |
| [13] | “https://medium.com/nishkoder/working-with-pythons-urllib-library-a-beginner-s-guide-79bcad776a8e”. |
| [14] | “https://en.wikipedia.org/wiki/Regular\_expression”. |
| [15] | “https://code.likeagirl.io/time-complexities-of-python-dictionary-and-set-operations-ee13511a2881”. |
| [16] | “https://testdriven.io/blog/flask-sessions/”. |
| [17] | “https://www.cloudflare.com/learning/ddos/glossary/denial-of-service/”. |
| [18] | “https://medium.com/geekculture/web-scraping-with-python-a-complete-step-by-step-guide-code-5174e52340ea”. |
| [19] | “https://www.selenium.dev/documentation/”. |
| [20] | https://en.wikipedia.org/wiki/HTTP\_persistent\_connection. |