University of Regina

Improved Search Engine Based on Sentimental Analysis

Final Report

Submitted to Dr. Samira Sadaoui

Department of Computer Science

CS 872-Software Engineering

Winter 2020

By

Adarsh Koppa Manjunath - 200397257 - akb049@uregina.ca

Tanu Nanda Prabhu - 200409072 - tnb735@uregina.ca

Regina, Saskatchewan

March, 2020

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Problem Definition

Search engines are becoming an essential day-to-day tool for finding required information located somewhere on the web. There are different search engines such as Google, Firefox, Yahoo and many more available on the market today. But all these search engines **display only 2-3 lines of required information** along with the URL of the website for any search. Here, the newsreader has to open every link sequentially which is very time consuming. If the website is spam, the newsreader will never know unless they open the website which would expose the newsreader's system to any security threats. For news, the newsreaders are very specific about the summary. Some newsreaders wish to see only positive content, some want only negative content, and others don't mind reading the overview. The existing search engines have the above-stated problems. The application that we are proposing will display the content up to 30 lines of related information about the searched article, the source website address would also be displayed. With the help of this application, we can avoid opening multiple links sequentially for reading the summary. A unique feature called sentimental analysis is being added to the application. With the help of sentimental analysis feature, we will display results in positive or negative sentiment order depending on the input given by the newsreader. If the newsreader

inputs sentiment as neutral, the order of the result is the same as in google.

1.1 Application Domain

Application domain for this project is **information retrieval** as we will remove the noise and extract the text blocks from the web source. Noises here include the HTML tags, CSS (Cascading Style Sheets), images, js (JavaScript), digits, symbols etc. and useful information will be the text contents of the website which tells overview and not necessarily meaningful sentences.

1.2 Motivation

As we stated in the problem statement, current newsreaders find it tedious to open multiple links sequentially to know the web source content. Considering tedious issues aside, there are other issues like security, spamming and the sentiment. Current search engines display accurate search results within a second, but again opening these results sequentially would lead to the above-stated issues. Motivation behind this project is to use the existing search engine like Google and add extended features like sentiment and parallelly display the description of search results within 30 lines. We will avoid spamming as newsreaders don't have to open the real web source, as they will know whether the website is spam from the summary itself.

Chapter 2

Feasibility study

In this chapter, we will describe the benefits of the proposed application compared to the existing ones. Also, we will show whether we are developing a new system, extending or improving existing ones.

2.1 Proposed application

- The system we are developing is the **extended version** of current search engines. Since Google [1] is the most popular search engine these days, we are considering Google as a base and adding extended features like sentimental analysis, parallel displaying the descriptions of search results within 30 lines and an option to follow up with the search topic.

- The googlesearch [2] is a python library which would help us communicate to Google through python script and this library is the heart of our project. We are considering **googlesearch** library as base and also as current solution, on top of this we are adding enhanced features as mentioned above.

2.1.1 Benefits of the proposed application:

- Summary of the web source result will be displayed within 30 lines, which gives an enough idea about the actual web source. This enhancement will be helpful in two ways, one is by saving time and other is that the newsreader's system will not be directly exposed to the actual web source.
- Summary will be ordered according to sentiments. Newsreaders can give positive, negative or neutral sentiments as input.
- Newsreaders can also save their favourite topic along with the sentiments in their portal. List of URLs will be displayed in the newsreader's portal, corresponding to their given topic and sentiment. Whenever there is a change in the result, newsreaders will be notified and then the updated list of URLs would be displayed in their portal.

2.1.2 Proposed application Workflow

Below are the steps of the proposed application architecture:

1. **Newsreader Input**: Newsreader inputs the topic to be searched along with sentiment.
2. **URL Extraction**: Extracts all URLs related to the inputted topic using **googlesearch** [2] python library.
3. **Extract Content**: Extracts text content from all the URLs parallely using python multi-threading function. Beautifulsoup and re (Regular Expression) are the two python libraries used to extract web content by removing noise (HTML, CSS, JavaScripts tags, etc.).
4. **Clean Text**: Extracted text will have digits, symbols and redundant words which do not contribute to the sentimental analysis. This will be removed

using re and porter stemming libraries. In porter stemming every word will be reduced to its root word. Example **happiness** to **happy**

5. **Sentimental Analysis**: Cleaned text will be tokenized, and stop words will be removed using nltk (Natural Language Toolkit. Sentiment for each word will be calculated using the TextBlob python library. At the end, we will be having positive, negative and neutral sentiment counts for every URL.

6. **Display**: Results will be displayed to the newsreader in 30 lines for every URL and results will be ordered according to user inputted sentiment. Ordering is by considering the sentiment count. Every URL has all three types of sentiment count as mentioned above in step 5. If sentiment inputted by a user is negative, we will consider the URL with highest negative sentiment count at first and followed by lower sentiment count in descending order.

   Below shown is the proposed solution workflow

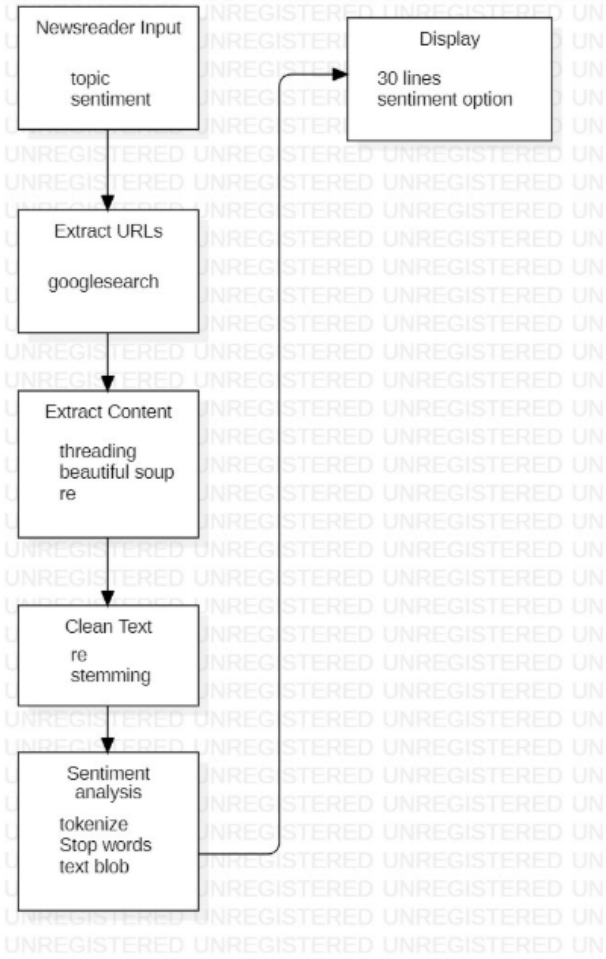

Figure 2.1: Proposed solution workflow

Example of proposed solution workflow

Newsreader Input

Topic	Canada Employment
Sentiment	Negative

Figure 2.2: Newsreader giving topic along with sentiment as input

ISE internal function

Steps				Library used
Extract URLs	[URL 1, URL 2, URL 3]			googlesearch
Extract content	**URL 1** People are getting laid off and everyone seems to be very sad and even employer is sad that he can't retain employees	**URL 2** People are worried that they will lose job and the worriness is not only about job, is also about finance	**URL 3** 83% of Canadians might get qualified for EI	threading, beautiful soup, re
Clean text	People are get laid off and everyone seem to be very sad and even employer is sad that he can't retain employees	People are worry that they will lose job and the worry is not only about job, is also about finance	of Canadian might get qualify for EI.	re and porter stemming
Sentiment analysis	People, laid, sad, employ, sad, retain, employee, {"**Positive**":1, "**Negative**": 3, "**Neutral**": 3}	People, worry, lose, finance {"**Positive**":1, "**Negative**": 2, "**Neutral**": 1}	Canadian, qualify, EI {"**Positive**":1, "**Negative**": 0, "**Neutral**": 2}	tokenize, stop words, text blob
Display	URL1, URL2, URL3			

Figure 2.3: ISE internal function

Search result – negative

URL1

"This is the URL 1 description"

URL2

"This is the URL 2 description"

URL3

"This is the URL 3 description"

Figure 2.4: Search Result for negative sentiment

Chapter 3

Software requirements elicitation and specification

In this chapter, we will describe the functional requirements for each user role, along with the use case diagrams. We will describe the most complex use cases with activity diagrams and also describe the software qualities.

3.1 Functional requirements

The application we are proposing will be used by newsreaders and admin. Newsreaders are mostly the people who are very interested in knowing the current trending topics, and they use the internet as a source to read. The admin will be us (project team) who developed this application. Below are the functional requirements for both newsreader and admin roles:

- **Newsreader:**
  - Searches for news by inputting the topic along with sentiment.
  - Newsreaders can add their favourite topic to follow up with in their portal.

  - Newsreaders can send feedback to the admin if they are not satisfied with the search results.

- **Admin:**

  - Add and delete newsreaders.
  - Check the newsreaders feedback and delete.
  - Admin can manually inspect newsreader stories. Here manual inspection is checking whether a newsreader's story is properly updated or not. And also, whether a newsreader violates any community guidelines or not.

3.2 Use case

As we know that there are two actors such as newsreader and admin who would use this application. We would give the detailed use case diagrams below. Also, we would describe in detail two use cases with activity diagrams.

3.2.1 Use case diagram for newsreader

Below shown is the use case diagram for newsreader

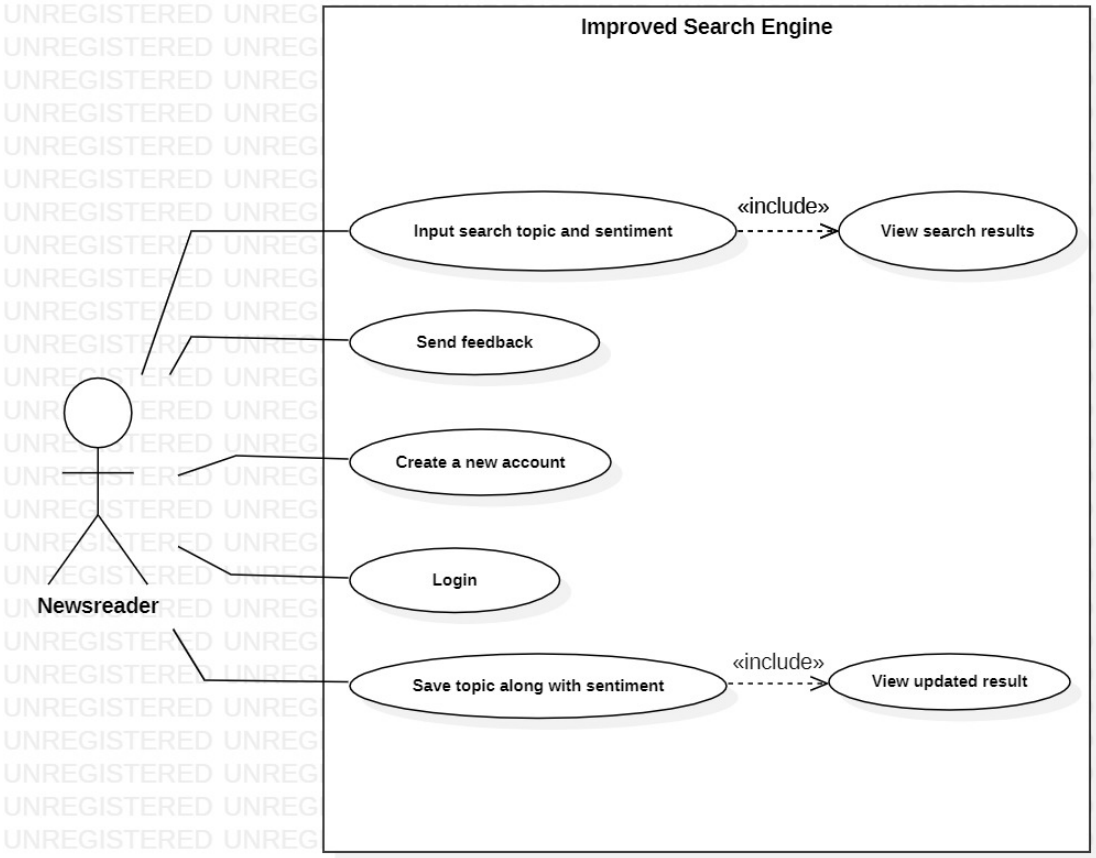

Figure 3.1: Use case diagram of newsreader

- **Newsreader:**
  - The newsreaders search for any news topics they are interested in. The topic that they need to search should be typed in the search bar, along with the sentiment they are looking for. Sentiment can be Positive, Negative or Neutral. For example: If a newsreader called John wants to search for the topic **Winter in Regina** and is interested only in the negative aspects of the topic, then he will search for the topic by typing **Winter in Regina** in the search bar, select the sentiment **Negative** and hit the enter button. After seeing the result, if the newsreader is not happy with the sentiment order or if he has anything to share with the admin, he can send feedback

to the admin. Before sending feedback, there will be an authentication check from the server side. If the newsreader is already logged in, he will be redirected to feedback forum directly, otherwise the newsreader will go through the login process. This is designed using a protected proxy pattern which is discussed in the next section below.

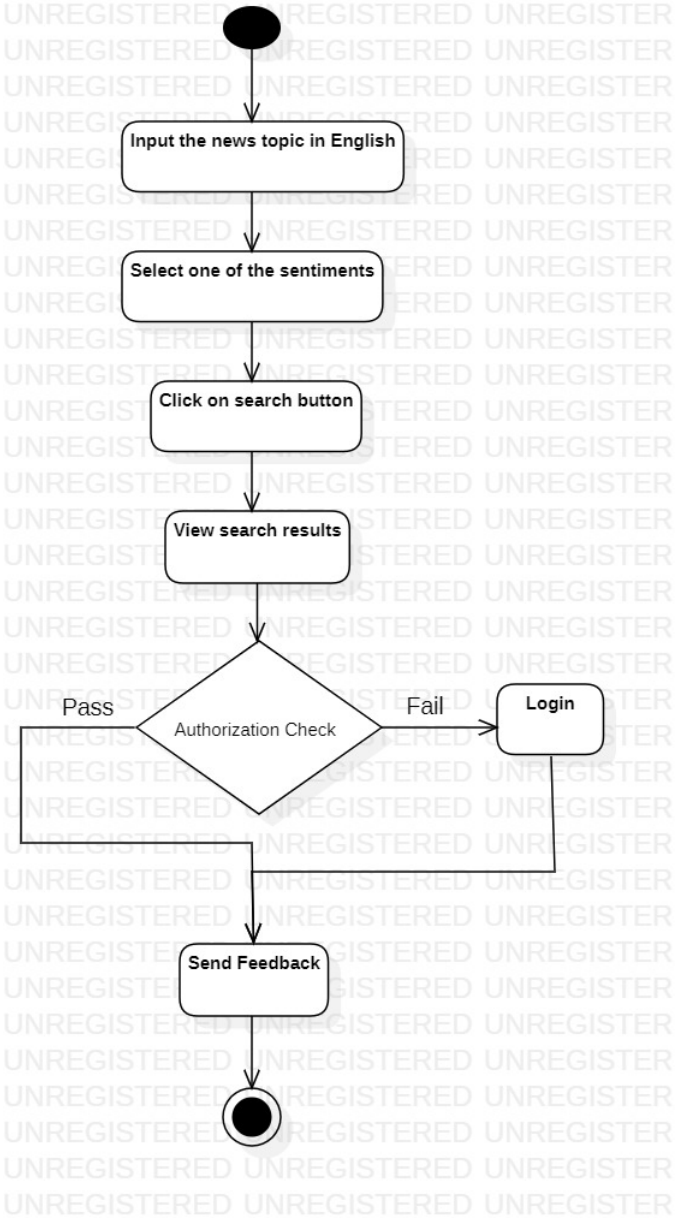

Figure 3.2: Activity diagram for a newsreader while sending a feedback

– Newsreaders can also follow up with the topic by inputting the topic along

with sentiment they are interested in. Newsreader has to register with the Improved Search Engine website first, after registering they can login and access their portal. The portal has an option of saving the topics to be followed up. The results (URLs) will be updated on the saved topic every day. Newsreaders can login and view the update on the topic they have been following up with.

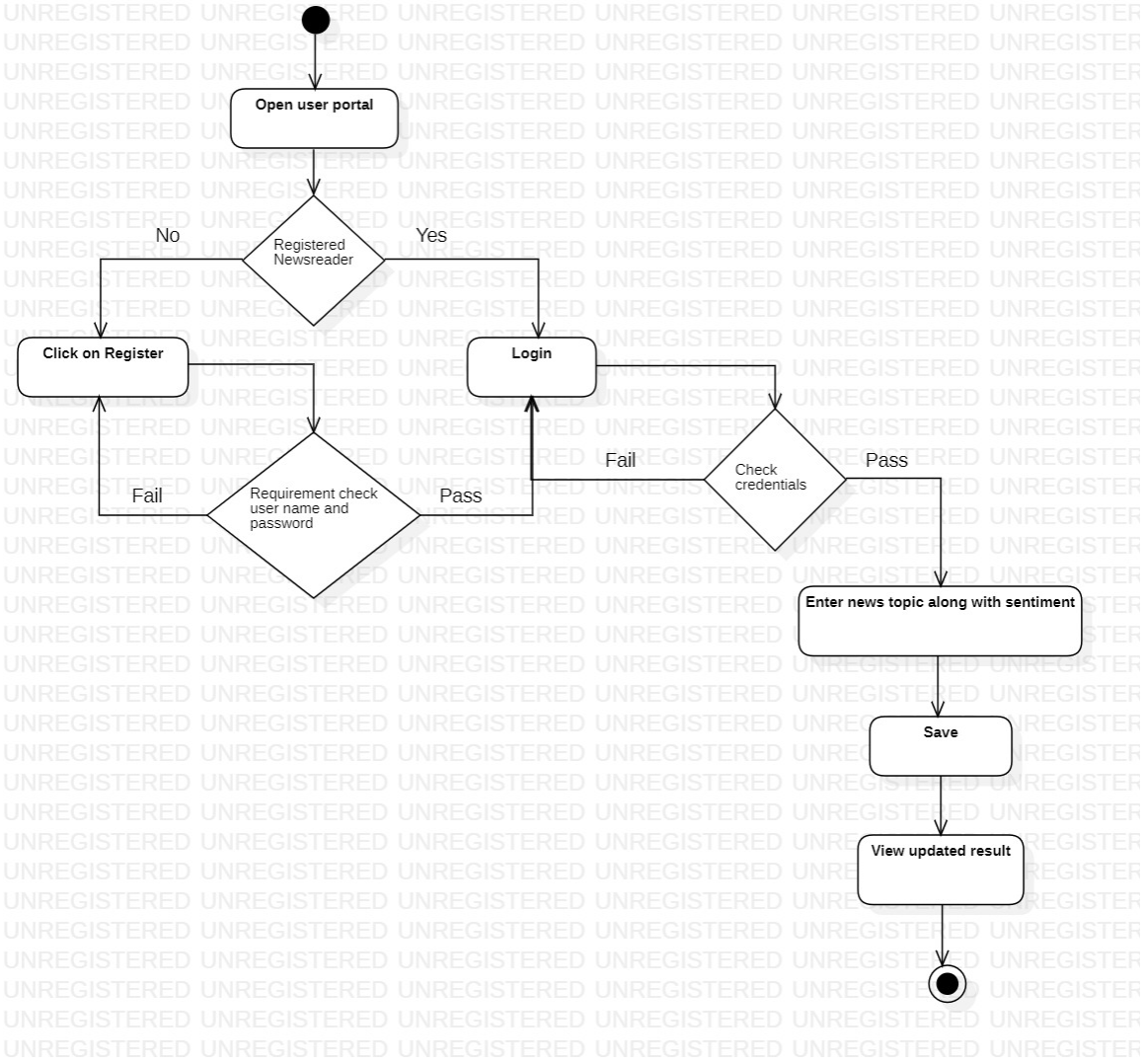

Figure 3.3: Activity diagram when the newsreader saving their topic to be followed up in their portal

3.2.2 Use case diagram for admin

Below shown is the use case diagram for admin

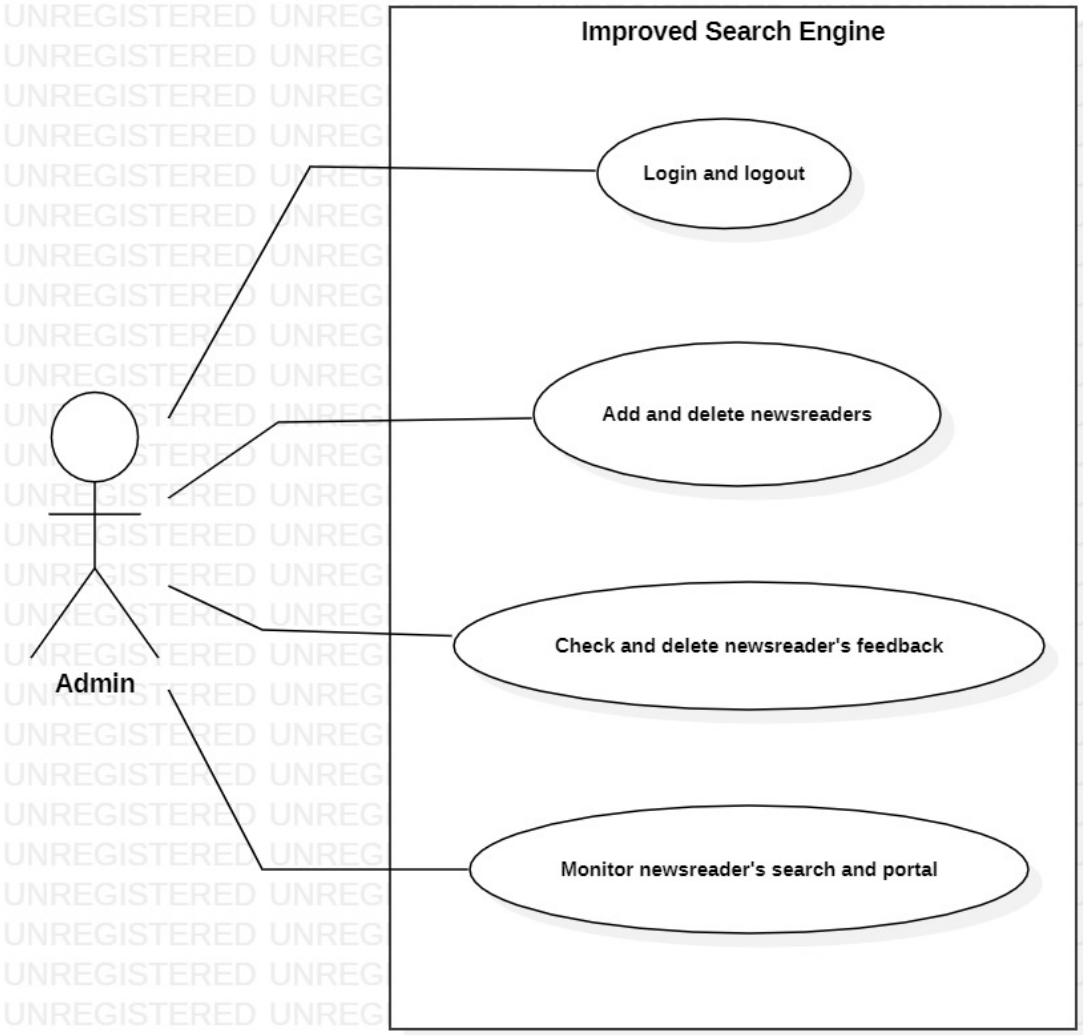

Figure 3.4: Use case diagram for admin

- **Admin:**
  - The admin controls the website. A key task for the admin is to maintain the website and handle all feedback and complaints using default Django admin UI. For example: If a newsreader called John searches the topic **Winter in Regina** and selects the sentiment **Negative** but all search

results show positive articles saying how beautiful it is in Regina during winter, then those search results will be incorrect. Now John can send his feedback saying, **improve the accuracy of search results** and the admin can handle this by logging into the admin portal, and decide whether to delete or consider the feedback.

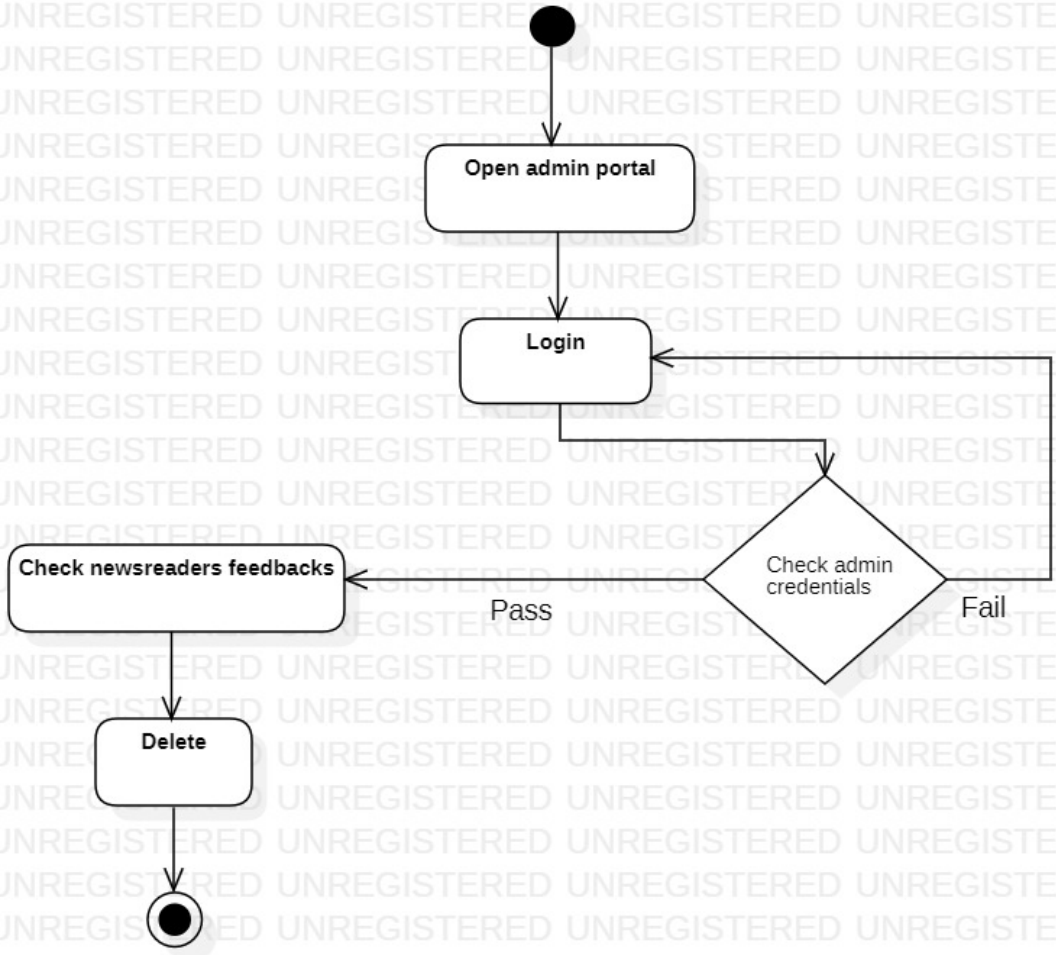

Figure 3.5: Activity diagram for admin checking newsreader's feedbacks

– Admin can monitor subscribed newsreader's accounts. If a subscribed newsreader saves or tries to follow up disturbing content or violates community guidelines, the admin can delete the newsreader account.

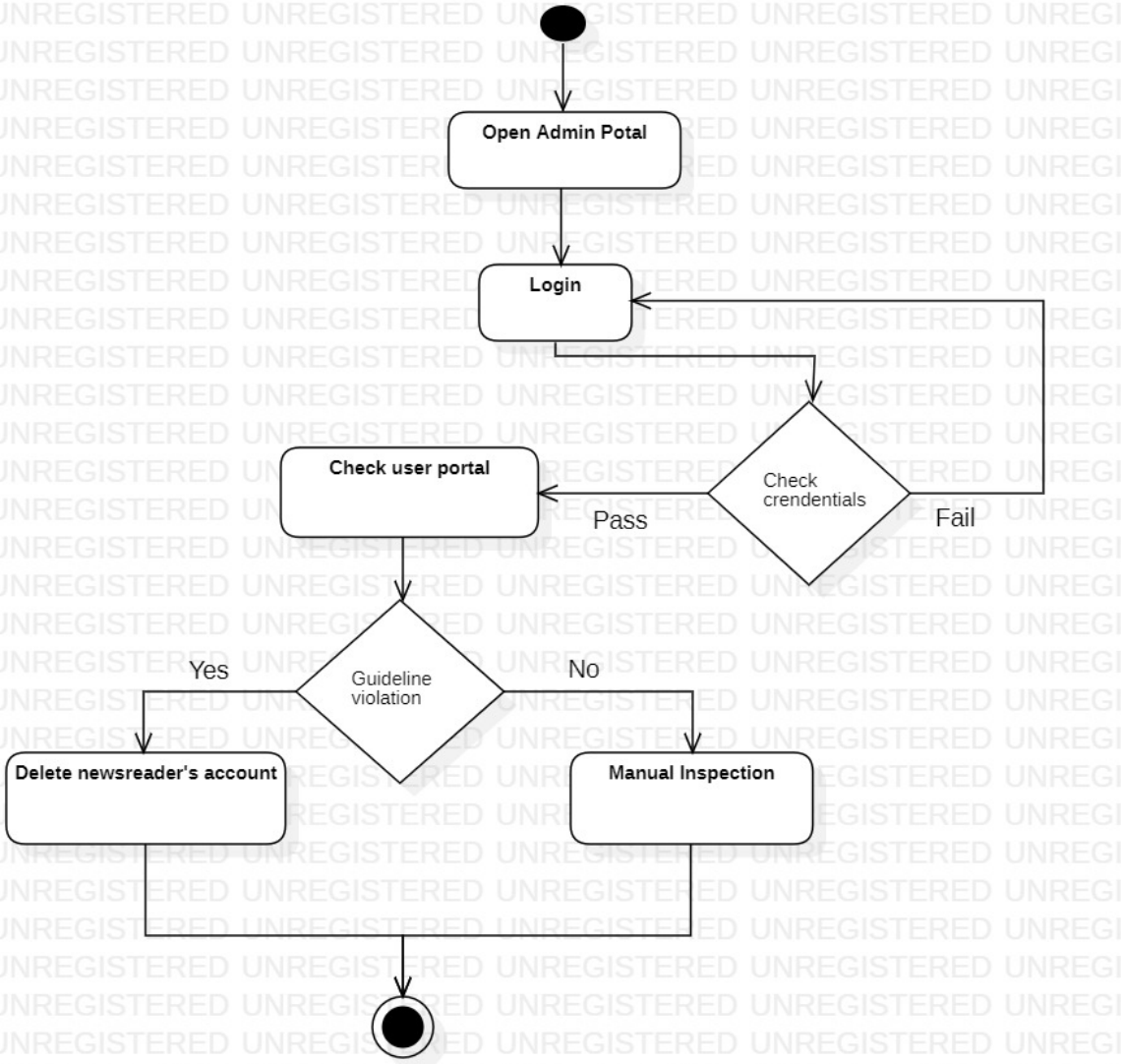

Figure 3.6: Activity diagram for admin monitoring newsreader's accounts and activity

3.3 Software qualities

In this chapter, we will discuss the software qualities with concrete examples. The software qualities of our application are as shown below:

- **Correctness**
  - Search results should match the newsreader's inputted topic.

  - If filter is **positive sentiment**, search result will display the URL with highest positive sentiment count at first and followed by URLs with low positive sentiments in descending order. The same would be followed for negative sentiments too.
  - Neutral will display the result in the same order as in Google.
  - Update newsreaders saved topic every day.

- **Time-efficiency**
  - All search results will be displayed within 30 lines and ordered according to the newsreader's inputted sentiment within 60 seconds.
  - All other web page functions will take 1-2 seconds to perform the desired operations.

- **Robustness**
  - Newsreaders will be prompted with the error messages if the inputted topic or sentiment is empty .
  - Error messages will be displayed if the newsreader's input topic is in other than English language.
  - Login error message will be displayed for invalid credentials.
  - Registration failed messages will be displayed if the user or email address is already taken.

- **Security**
  - Newsreaders accounts and feedback should be protected and kept confidential. For this, we have followed complicated password patterns. When the newsreader registers with the website, a password has to be set. This

password can not be commonly used or same as username. Length of the password should be 8 or more characters and should include both alphabets and numeric character. To implement this, we used the concept of Django's user authentication.

– We used Heroku Cloud application platform to deploy our application, Heroku by default providing SSL certificate, issued by DigiCert. Our application uses HTTPS instead of HTTP, so data sent through our application will be safe. Below figure indicates the secure connection of our application and the certificate information is also depicted.

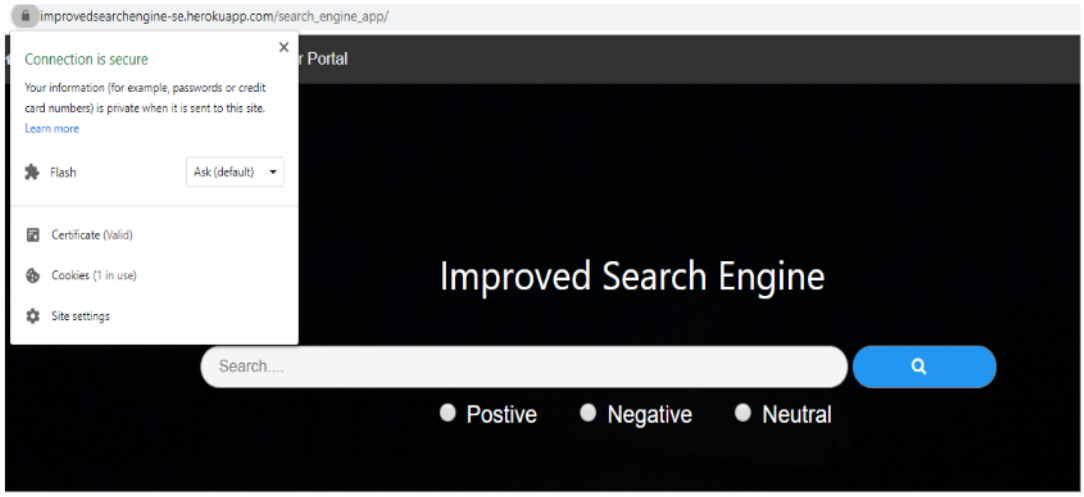

Figure 3.7: The application has a secure connection

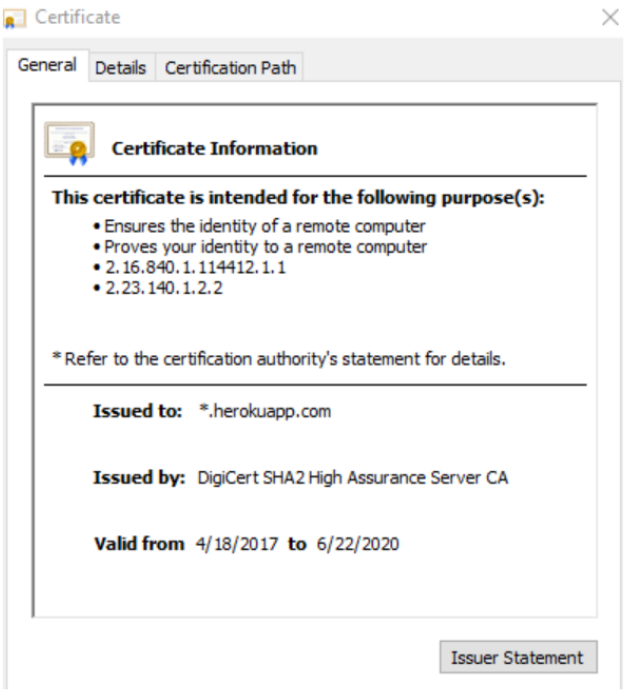

Figure 3.8: The certificate information of our website provided by DigiCert

– Protected proxy pattern is used for authentication [3]. To implement a protected proxy, a feedback model is considered. After the newsreader searches for a topic, if he or she is not happy with the results they can share feedback with the admin provided newsreader should be logged in. Feedback model is the real subject class and FeedbackView class acts as a proxy class. FeedbackView checks whether the newsreader is logged in or not. If yes, newsreaders will be prompted with a feedback forum or else newsreaders will be prompted with the login forum. Authentication is checked in the FeedbackView class without newsreader knowledge. If the newsreader is authorized and logged in, then he will be redirected to a feedback form. The class diagram of the protected proxy design pattern is as shown below.

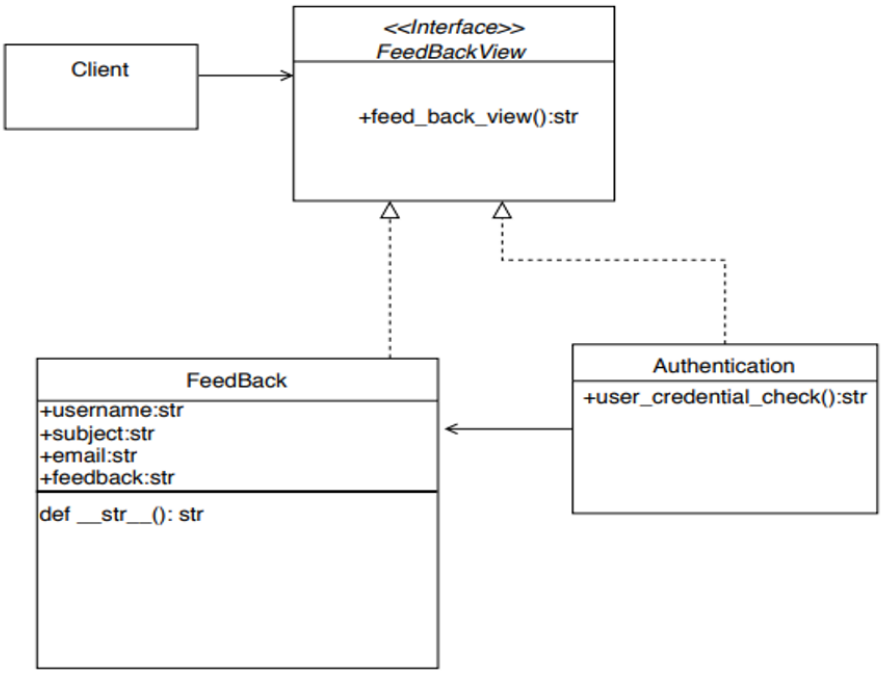

Figure 3.9: Class diagram of protected proxy

Chapter 4

Top-level and low-level design

In this chapter, we will discuss the MVT architecture and its benefits for our application. Also, we will describe the design patterns along with the algorithms we have incorporated in our applications through class diagrams.

4.1 MVT software architecture

We are using the Django web framework to build our web application. Django follows the model-template-view or model-view-template software architectural pattern. There is no difference between MVT or MTV, both are the same thing. The view with Django should not be confused with the controller in MVC, the view is still a view in Django. Let us understand Django's MVT software architecture. The Django's architecture comprises three main parts such as.

- **Part 1**: Set of tools that make working with databases much simpler and easier. Also known as "model" which is the data access layer responsible for handling the data.
- **Part 2**: The template system. Also known as the "template" which is the presentation layer responsible for handling the user interface.

- **Part 3**: The framework that handles the communication between the user and the database. Also known as the "view" which is the business logic layer responsible for interacting with the model and rendering the template.

  Now that we know about Django's MVT architecture, let us now see how we incorporated it into our application. The complete MVT architecture is depicted below in figure.

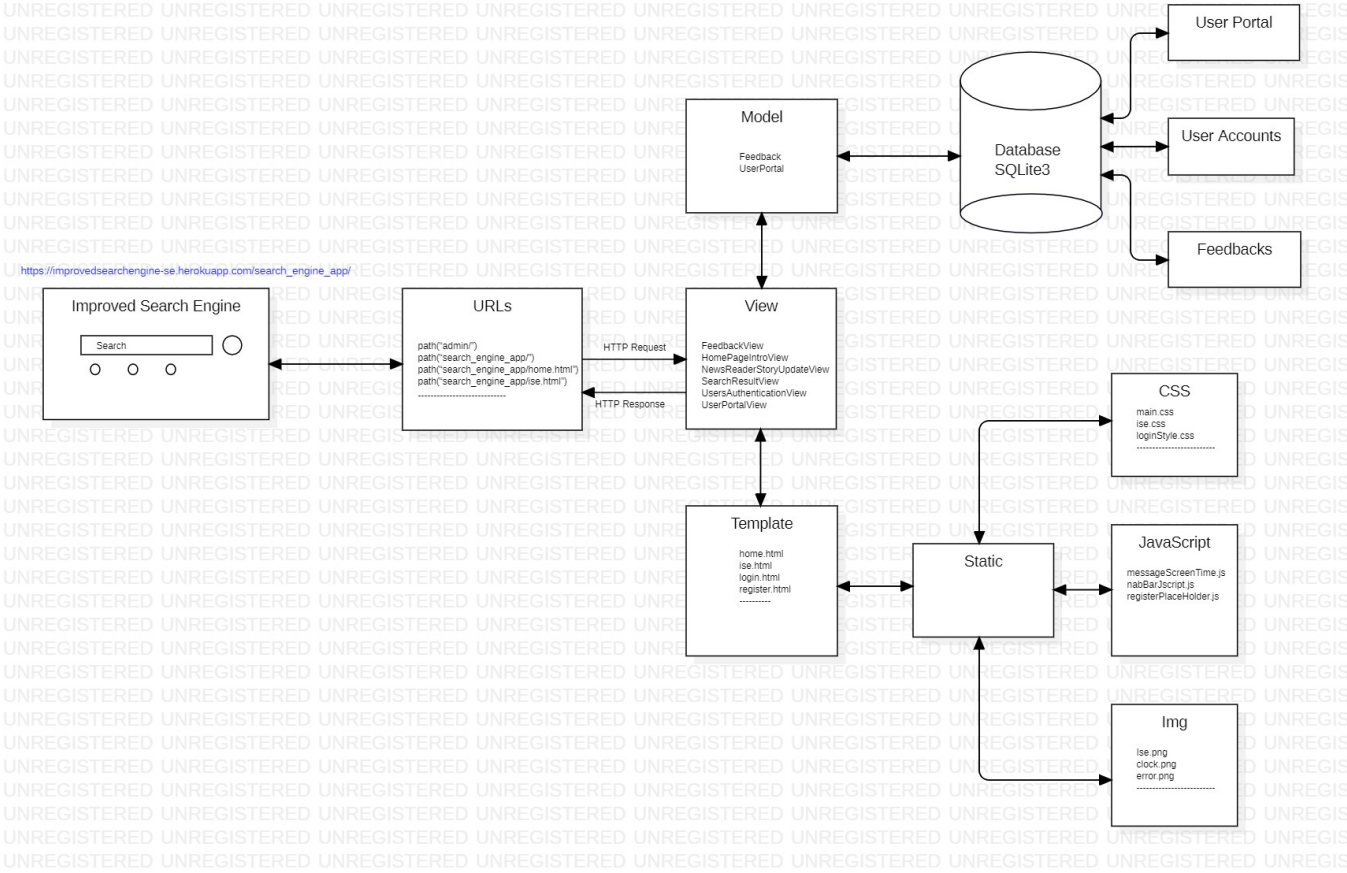

Figure 4.1: MVT software architecture

4.1.1 Working of MVT architecture

Django uses request and response to communicate between the client and the server.

- The newsreader navigates or requests an HTML page (Home Page) to be rendered.

- The HTML page (Home page) path would be stored in a list called "urlpatterns" which can be found in the urls.py file. The urlpatterns list routes URL to the respected views. Inside the urlpatterns, there is a function called path which accepts two parameters: the pathname and the view name (function or class). For example, the newsreader wants to go the home page, in this case, the path for the home page i.e. path ('search_engine_app/home.html', views.search) and the search class would be called which would be in the views. This is how Django's URL maps the request to the views.

- After the URL is mapped, and the associated class-based view method is called, the view would then interact with the model to store or retrieve the data and then also interact with the template by sending the response. This is how the view renders a template. For example, the newsreader gets an HTML page with a form field and they enter the data in those fields on clicking the submit button it would store the data from the forms in the database via the model. Later the view sends the template as a response; this is how the view responds to the client. Sometimes when the newsreader asks only for a simple HTML page, then the view does not interfere with the model and merely returns the template as the response. For example, when the newsreader searches for the topic along with the sentiment than on clicking the submit button, the view does not store the results to the database but displays the results in the form of an HTML page.

  To summarize, the urls.py file contains all the paths with the URLs, the view file contains all the class-based views which accept the request and the response via a template file (HTML). The views would also interact with the model, which is directly connected to the database, in this case, we are using the SQlite3 database. The template comprises all the front-end UI files, mainly HTML. This is what the newsreader sees as soon as they navigate to the website. Last,

there are static files which are mainly CSS files, Images and JavaScript files. We use these for the styling of the application.

4.1.2 Reasons for choosing MVT architecture

Model

We are using Django's model to store essential data. The model maps to a database table. Here we are using Django's SQLite default database. With the help of its automatically-generated database access API, we can directly store the values into the database writing no extra database query code. We are using two database tables named Feedback and User Portal for storing the newsreader's feedback and the topic details that they want to save. The feedback table comprises the details such as username, email, subject and the feedback message. And the table user portal comprises username, topic, URL, update date, and the sentiment of the article that the newsreader wants to save in their portal. With the help of Django's model field types such as Charfield and Textfield which provides us fields to store the data which would be automatically stored in the database. Suppose if we want to alter or edit the database attributes or tables then all we must do is go to the models.py file (which can be found in search_engine_app
models.py) and then make the modifications. There is no need to update any other files. This is one of the advantages of Django's MVT.

View

The view sources the data from the database (or external service such as text area or forms) and delivers it to a template. In our case, we represent the view as class-based views. The class-based views allow us to reuse the code and inherit them. The views in Django take a request and gives a response. For example, consider a view named user portal which contains the parameters topic and the sentiment of the article.

Here as soon as the newsreader enters the details via the template (HTML) page. Then the view requests and checks for the POST condition. Later, all the data such as the topic and the sentiment would be stored in the forms. And then the data from the forms would be stored in the database later the view returns a response saying the data was successfully saved or render the HTML page (userportal.html) via the template. The HTML page is what the newsreader sees when they render the application. Last, in the urls.py file, we need to give the path of the filename and make sure that we call the view class, by doing so the moment opens the browser and enters the path the HTML page would be rendered. Also, Django lets us design the URLs however we want to mean we can use some fancy name for the URLs, we don't have to follow any convention or rules for this. For example, we have used the name '/search_engine_app/' as the base URL. We can also add many views to our application without even touching the main code. In our project the views folder can be found in 'search_engine_app/views' inside this folder there are many class-based views such as feed_back_view.py, home_page_header_view.py, import_required_view_libraries.py, newsreader_story_update_view.py, search_result_view.py, user_authentication_view.py, and user_portal_view.py. With the help of Django's view, we can easily store the data and then render it on the web browser.

Template

In our project, the template folder comprises all the HTML files for the front-end. With the help of template inheritance, we can just write the HTML code once and then extend it to other pages. For example, we have written the base code for the navigation bar using HTML, CSS and Bootstrap rather than writing the redundant code in every single HTML page we can use Django's template inheritance and we can extend the base code to other pages by just saying **{{% include navBar.html %}}**. This was a handy feature of Django because as we all know we need the navigation bar

in most of the pages, in this case, Django's template inheritance comes in handy meaning we can use the same navigation bar in every single HTML page. This is like writing the code once and using it as much as you want. The template folder in our case comprises all the HTML files such as feedBack.html, feedBack_login.html, home.html, ise.html, login.html, mesages.html, navBar.html, projectTeam.html, signUp.html, results.html, userPortal.html, viewYourUpdatedResults.html. Also, these HTML files are associated with a CSS file for the styling which is in the Static/Css folder. Also, with the help of Django's template language, we could easily manage the tradeoff between python programming and HTML language. With this, we could easily use the python variables and then print them on the HTML pages.

One of the main advantages of Django MVT architecture that helped us is the loosely coupled feature. Every file or folders has its own value or style. If appropriate, the various layers of the system will not know each other. Meaning the template knows nothing about the database layer, the model knows nothing about the web requests. Suppose if we want to change the attributes of the database, then without further thinking, we need to make changes only in the model.py file, the same holds good with the UI changes, we need to only change the HTML file in the template. We can do all these changes without even touching the main code, we can make every minute change in their respected files. This is how Django's MVT was beneficial to our application.

4.2 Design Patterns

Observer Pattern and Factory method pattern are the design pattern we choose for this project. All these patterns were referred from **Gang of Four design patterns** textbook [3].

4.2.1 Observer Pattern

Reason for selecting observer pattern

One to many relationship nature. When one object changes dependent objects have to be notified, this property is essential and useful for our news publisher and subscribed newsreaders relationship. As we stated in the use case, newsreaders can save topic and sentiment, to follow up with updated URLs. In this case, we have a newsreader's database saved with the newsreader name, topic, sentiment and list of URLs for the saved topic. Whenever this URL list changes, we have to notify all subscribed newsreaders with updated URLs for their corresponding topic and sentiment.

To implement this use case, an observer pattern is used. We considered **NewsPublisher**, **NewsreaderStoryUpdateView**, and **CheckUpdate** classes for implementing the observer pattern. We have provided the algorithms and the programs for the corresponding to the pattern's important methods..

- **NewsPublisher is a subject class with three methods.**
  - **add_newsreader()** : Helps in adding newsreaders to the subscribed newsreaders list with topic and sentiment.
  - **remove_newsreader()**: Helps in removing newsreader from the subscribed newsreader list
  - **notify()**: notify any update in the newsreader list.

```
1 class NewsPublisher:
2     def __init__(self):
3         """
4         Declare private list of authenicated and subscribed users
5         """
6         self.notify_dict={}
7         self._db_authenticated_users=User.objects.all()
```

Algorithm 1 NewsPublisher class

1: **Step 1**: authenticated_users ← []
2: /* initialize the list of authenticated newsreader list */
3: **Step 2**: subscribed_users ← []
4: /* initialize the list of subscribed newsreader list */
5: **Step 3**: add_newsreader(newsreader_username, topic, sentiment)
6: /* save newsreader story in userportal model */
7: **Step 4**: remove_newsreader(newsreader_username)
8: /* remove newsreader from subscribed newsreader list */
9: **Step 5**: notify()
10: /* method to notify all changes */
11:

```
        self._db_subscribed_users=UserPortal.objects.all()
        self._authenticated_users=[index.username for index in self.
    _db_authenticated_users ]
        self._subscribed_users= [index.username for index in self.
    _db_subscribed_users]

    def add_newsreader(self, news_reader_username,topic,sentiment):
        """add newsreader's topic and sentiment to the database"""
        print(self._db_subscribed_users)
        if str(news_reader_username) in self._subscribed_users:
            for news_reader in self._db_subscribed_users:
                if news_reader.username==str(news_reader_username):
                    if news_reader.topic!=topic or news_reader.
    sentiment!=sentiment:
                        news_reader.topic=topic
                        news_reader.sentiment=sentiment news_reader.
    save(update_fields=['topic','sentiment'])
                        return "topic or sentiment is updated"
        else:
            if  str(news_reader_username) in self.
    _authenticated_users:
                newsreader=UserPortal(username=str(
    news_reader_username),topic=topic,urls="",updatedate="",sentiment
```

```
=sentiment)
25              newsreader.save()
26              return " topic and sentiment is entered"
27
28  def remove_newsreader(self, username):
29      """remove subscribed users from database"""
30      userportal.objects.filter(username=news_reader_username).
    delete()
31      return "user is removed"
32
33  def notify(self,username):
34       result={}
35       from datetime import date
36       for news_reader in self._db_subscribed_users:
37           if news_reader.username==str(username):
38               url_list=news_reader.urls.strip("][").split(',')
39               urls=[]
40               for index in url_list:
41                   urls.append(index.strip(" '"))
42               news_reader.urls=urls
43               result={ 'topic': news_reader.topic, 'url' :
    news_reader.urls, 'sentiment': news_reader.sentiment, 'date':
    date.today()}
44               return result
```

Code Listing 4.1: NewsPublisher class program

- **CheckUpdate is a concrete subject class with monitor_url_status() method**.
  - **monitor_url_status()**: This is implemented using Django-background-task library. Similar to crontab on linux. Where a task is stored in a database and will be kept running. We used this property and in our

case, the task is monitoring every subscribed newsreader's topic for every 5 hours and if there is any change in URLs list, it will be saved in the database and updated in notify_dict

– **UserPortal**: It is a model saved with all newsreader's names, topic, sentiment and URLs. This model is used by the task to know the current URLs list and task will go on google search for the latest URLs list for the newsreader's saved topic. If the URL list is changed, it will be saved in the model and notified to the NewsPublisher class.

Algorithm 2 CheckUpdate

1: **Step 1**: method monitor_url_status()
2: /* method to monitor subscribed newsreader story */
3: **Step 2**: return notify_dict
4: /* returns story update result for all newsreader */
5:

```
class CheckUpdate(NewsPublisher):
        @background(schedule=10)
        def monitor_urls_status():
                # lookup user by id and send them a message
                notifobj=NewsPublisher()
                db=UserPortal.objects.all()
                for index in db:
                        topic=index.topic serach_obj=SearchWeb(topic
   ,sentiment=index.sentiment,num=3, stop=3, final_output={},
   sentiment_dict={})
                        search_result=serach_obj.thread_func()
                        url_list=search_result.keys()
                        if index.urls == str(list(url_list)):
                                notifobj.notify_dict[index.username
   ]=False
                        else:
```

```
                                    index.urls=str(list(url_list))
                                    index.save(update_fields=['urls'])
    notifobj.notify_dict[index.username]=str(list(url_list))
                print(notifobj.notify_dict)
                return notifobj.notify_dict
publisher_obj=CheckUpdate()
change=publisher_obj.monitor_urls_status(repeat=50)
```

Code Listing 4.2: CheckUpdate class program

- **UserPortalView is observer class with user_portal method**
  - User_portal method provides an interface for newsreaders to subscribe with the topic and sentiment. This will be saved in the userportal model.

Algorithm 3 UserPortal class

1: method userPortal()
2: **Step 1**:topic ← *newsreader_input*
3: **Step 2**: sentiment ← *newsreader_input*
4: **Step 3**: add_newsreader(newsreader_username, topic, sentiment)
5: /* this method provides an interface to newsreader to save topic and sentiment */
6: **Step 4**: return status
7:

```
class UserPortalView(View):
    def userPortal(request):
        if request.method=="POST":
                form3 = userPortalForms(request.POST)
                if form3.is_valid():
                    topic = form3.cleaned_data['topic']
                    sentiment = form3.cleaned_data['options']
                    subject=NewsPublisher() status=subject.
    add_newsreader(request.user,topic,sentiment)
                    messages.success(request, status)
```

```
        form3 = userPortalForms()
        return render(request,"userPortal.html",{'form': form3})
```

Code Listing 4.3: UserPortal class program

- **NewsReaderStoryUpdateView is a concrete observer class**
  - This class will be considered when the user wishes to see his updated result. Newsreader story updates will be displayed with the help view_your_update_result method. This method will display the notify dict invoked from the Check-Update class.

Algorithm 4 NewsReaderStoryUpdateView class

1: **Step 1**: userstate_dict ← {}
2: /* has all newsreader story current state */
3: **Step 2**: viewYourUpdatedResult()
4: /* this method invokes newsreader updated story */
5: **Step 3**: return result
6:

```
class NewsReaderStoryUpdateView(View):
    def __init__(self):
        self._userstatedict={}
        self._dbuserportal=UserPortal.objects.all()
        for index in self._dbuserportal:
             self._userstatedict[index.username]=index.urls

    def viewYourUpdatedResult(request):
        subjobj=NewsPublisher()
        result=subjobj.notify(request.user)
        result={'result':result}
        return render(request, 'viewYourUpdatedResults.html', result
    )
```

Code Listing 4.4: NewsReaderStoryUpdateView class program

Below is the class diagram of observer pattern that we incorporated in our project

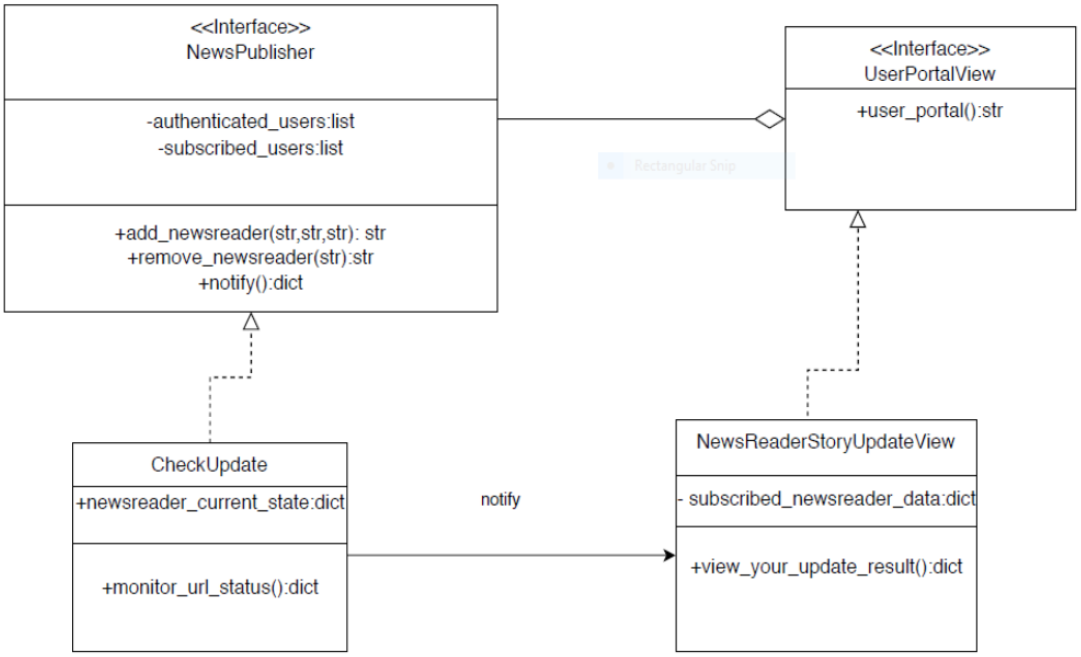

Figure 4.2: Class diagram of observer pattern

4.2.2 Factory Pattern

Reason for selecting factory pattern

URL to view mapping is a factory pattern as we have a set of view classes and depending on the URL we have to display the corresponding view. To implement this, we wrote each view in a separate class and every view class is inherited from Django's default view class. This Pattern consists of URL factory class, View class and set of view classes corresponding to URL.

- **URL factory**: It is a interface class which has view class objects for corresponding URL. Depending on the URL path view class method will be accessed. Get_view_function method keeps track of all view classes objects. It matches the URL with the object and can access the corresponding view. This method

returns a URL pattern list consisting URL and their corresponding view.

Algorithm 5 URLFactory class

1: method get_view_functions():
2: **Step 1**: urlpatterns $\leftarrow$ $path(url\_path, view\_function)$
3: /* urlpattern is a default django's url matching list */
4: /* path is a django method to match url and corresponding view function */
5: /* url_path is the url path and call corresponding view function object */
6: **Step 2**: repeat Step 1 for all url_path
7: **Step 3**: return urlpatterns
8:

```
class URLFactory:
    def get_view_functions():
        urlpatterns = [
            path('admin/', admin.site.urls),
            path('search_engine_app/',SearchResultView.search),
            path('search_engine_app/home.html', SearchResultView.
search),
            path('search_engine_app/ise.html', HomePageIntroView.ise
),
            path('search_engine_app/projectTeam.html',
HomePageIntroView.projectTeam),
            path('search_engine_app/login.html',
UsersAuthenticationView.loginPage, name ='login'),
            path('search_engine_app/register.html',
UsersAuthenticationView.registerPage),
            path("search_engine_app/logout", UsersAuthenticationView
.logoutUser, name="logout"),
            path("search_engine_app/userPortal.html",UserPortalView.
userPortal),
            path("search_engine_app/viewYourUpdatedResults.html",
NewsReaderStoryUpdateView.viewYourUpdatedResult),
            path("search_engine_app/feedBack.html",FeedbackView.
feedback_view),
```

```
15              path("search_engine_app/feedback_login.html",
    UsersAuthenticationView.feedback_login_view),
16          ]
17          return urlpatterns
18 view_factory_obj=URLFactory
19 urlpatterns=view_factory_obj.get_view_functions()
```

Code Listing 4.5: URLFactory class program

- **View**: an abstract class provided by Django. This class will take requests from the URL factory and invoke corresponding child classes. Algorithms for this class can be found in the link below.

Algorithm 6 view class

1: /* django's default class */
2: **Step 1**: Http_methods $\leftarrow$ *get*, *post*, *put*, *delete*, *patch*
3: /* all CRUD methods list */
4: **Step 2**: method as_view()
5: **Step 3**: method view()
6: **Step 3**: method setup()
7: **Step 3**: method dispatch()
8: **Step 3**: method options()
9: **Step 3**: method not_allowed()
10: **Step 3**: method http_methods_allowed()
11:

Program Source: https://github.com/AdarshKoppManjunath/Improved-Search-Engine-Based-on-Sentimental-Analysis/django/views/generic/base.py

- **SearchResultView**: inherited class from View. Search method will display the search results to the newsreader

Algorithm 7 SearchResultView class

1: method search(request):
2: **Step 1**: topic ← *newsreader_input*
3: /* topic to be searched, input by newsreader */
4: **Step 2**: sentiment ← *newsreader_input*
5: /* sentiment (positive, negative or neutral) by newsreader */
6: **Step 3**: search_web(topic, sentiment)
7: /* search web method gives search result */
8: **Step 4:** return result
9:

```
class SearchResultView(View):
    def search(request):
        """This function communicates between template and
    search_web files"""
        try:
            if request.method=="POST":
                form = GeneralForms(request.POST)
                if form.is_valid():
                    topic=form.cleaned_data['name']
                    sentiment = form.cleaned_data['options']
                   search_web_obj=SearchWeb(topic,final_output={},
    sentiment=sentiment,sentiment_dict={})
                    result=search_web_obj.thread_func()
                    if type(result)==str:
                        args = {'result': result}
                        return render(request, "alert.html", args)
                    else:
                        args = {'result':result,'sentiment':
    sentiment}
                        return render(request,"results.html", args)
            form = GeneralForms()
            return render(request,"home.html",{'form': form})
        except Exception as e:
            log.error('An exception occurred: {}'.format(e))
```

```
22                log.error(traceback.format_exc())
```

Code Listing 4.6: SearchResultView class program

- **HomePageIntoView**: Inherited class from View. Displays all header menu includes about us, userportal, ISE and home page.

Algorithm 8 HomePageIntroView class

1: **Step 1**: method ise()
2: /* display a page about project */
3: **Step 2**: method projectTeam()
4: /* display a page about project team */
5:

```
1 class HomePageIntroView(View):
2     def ise(request):
3         return render(request, "ise.html")
4 
5     def projectTeam(request):
6         return render(request, "projectTeam.html")
```

Code Listing 4.7: HomePageIntroView class program

- **UserAuthenticationView**: Inherited class from View. Displays login, logout and registration page to the newsreaders.

Algorithm 9 UsersAuthenticationView class

1: **Step 1**: method registerPage()
2: /* display newsreader registration page */
3: **Step 2**: method loginPage()
4: /* display newsreader login page */
5: **Step 3:** method logout()
6: /* display newsreader logout page */
7:

```
class UsersAuthenticationView(View):
    def registerPage(request):
        form = CreateUserForm()
        if request.method == 'POST':
            form = CreateUserForm(request.POST)
            if form.is_valid():
                form.save()
                uname = form.cleaned_data.get('username')
                messages.success(request, 'Account was created for '
    + uname )
                return redirect('login')
        context = {'form': form}
        return render(request, 'register.html', context)

    def loginPage(request):
        if request.method == 'POST':
            username = request.POST.get('username')
            password = request.POST.get('password')
            user = authenticate(request, username = username,
    password = password )
            if user is not None:
                login(request, user)
                print(request.path)
                return redirect('userPortal.html')
            else:
                messages.error(request,'Login failed!!! invalid
    credentials.')
        context = {}
        return render(request, 'login.html', context)

    def logoutUser(request):
        logout(request)
        return redirect('login')
```

```
31
32      def feedback_login_view(request):
33          if request.method == 'POST':
34              username = request.POST.get('username')
35              password = request.POST.get('password')
36              user = authenticate(request, username = username,
    password = password )
37              if user is not None:
38                  login(request, user)
39                  print(request.path)
40                  return redirect('feedBack.html')
41              else:
42                  messages.error(request,'Login failed!!! invalid
    credentials.')
43          context = {}
44          return render(request, 'login.html', context)
```

Code Listing 4.8: UsersAuthenticationView class program

- **UserPortalView**: Inherited class from View. Provide an Interface to the newsreaders to subscribe with the topic and sentiment.

Algorithm 10 UserPortal class

1: method userPortal():
2: **Step 1**: topic ← *newsreader_input*
3: **Step 2**: sentiment ← *newsreader_input*
4: **Step 3:** add_newsreader(newsreader_username, topic, sentiment)
5: /* this method provide an interface to newsreader to save topic and sentiment */
6: **Step 4:** return status
7:

```
1 class UserPortalView(View):
2
3     def userPortal(request):
4         if request.method=="POST":
```

```
                form3 = userPortalForms(request.POST)
                if form3.is_valid():
                    topic = form3.cleaned_data['topic']
                    sentiment = form3.cleaned_data['options']
                    subject=NewsPublisher()
                    status=subject.add_newsreader(request.user,topic
    ,sentiment)
                    messages.success(request, status)
        form3 = userPortalForms()
        return render(request,"userPortal.html",{'form': form3})
```

Code Listing 4.9: UserPortal class program

- **NewsReaderStoryUpdateView**: Inherited class from View. Provide an Interface to the newsreaders to see updated URLs.

Algorithm 11 NewsReaderStoryUpdateView class

1: **Step 1**: userstate_dict ← {}
2: /* has all newsreader story current state. */
3: **Step 2:** viewYourUpdatedResult()
4: /* this method invoke newsreader updated story */
5: **Step 3:** return result
6:

```
class NewsReaderStoryUpdateView(View):
    def __init__(self):
        self._userstatedict={}
        self._dbuserportal=UserPortal.objects.all()
        for index in self._dbuserportal:
             self._userstatedict[index.username]=index.urls

    def viewYourUpdatedResult(request):
        subjobj=NewsPublisher()
        result=subjobj.notify(request.user)
```

```
        result={'result':result}
        return render(request, 'viewYourUpdatedResults.html', result
    )
```

Code Listing 4.10: NewsReaderStoryUpdateView class program

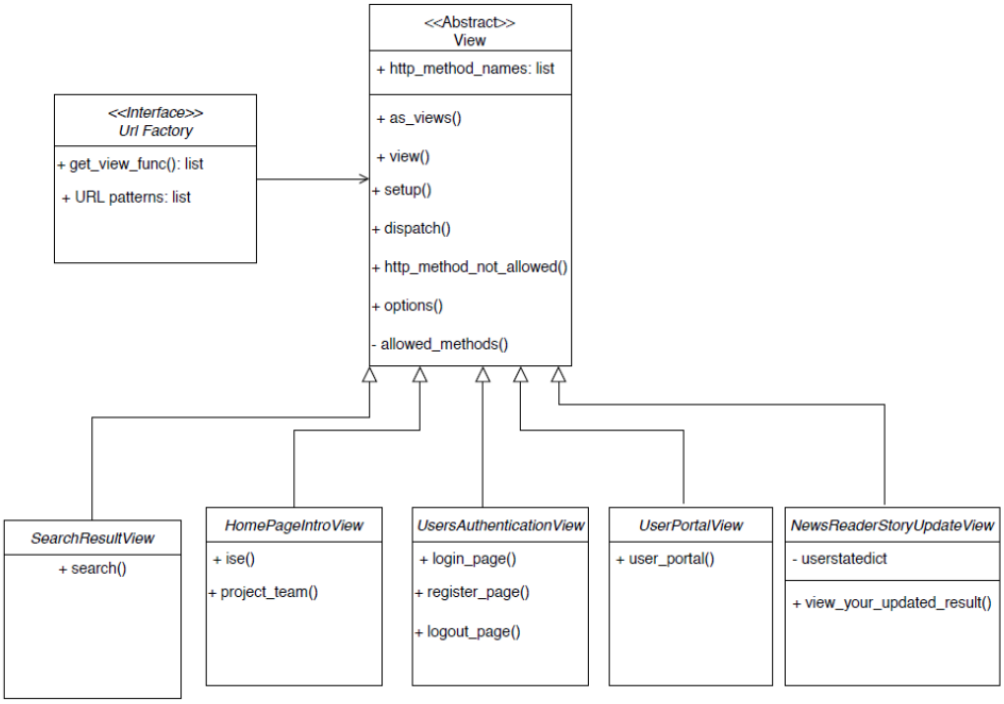

Figure 4.3: Class diagram of factory design pattern

Chapter 5

Programs

In this chapter, we will discuss the implementations details which include component diagram, deployment diagram, screenshots of all the table contents of the system data, and the link of our web-application.

5.1 Component diagram

Below diagram helps in understanding the organization of source code on a high level.

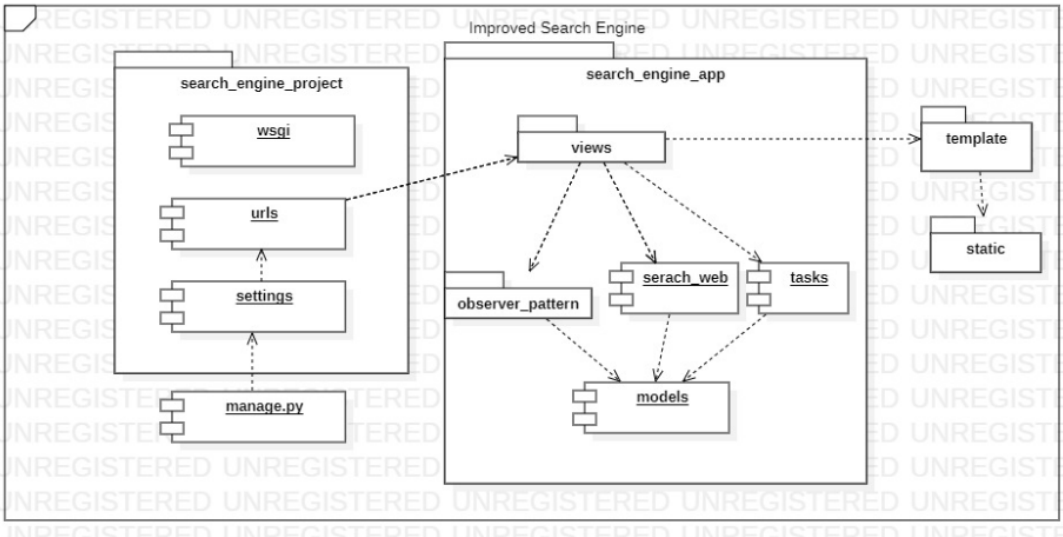

Figure 5.1: Componenent diagram regarding the organization of source code

- Source code is mainly composed of search_engine_project, search_engine_app and manage.py
- manage.py activates application by calling settings.py and settings.py in turn call urls.py
- Depending on the URL path, the corresponding view function will be invoked from views.
- These view function communicate with models to fetch and send data through observer_pattern, search_web.py and tasks.py
- View display result by rendering template files, which in turn communicates to static files. This generates a URL and will be displayed to the newsreader.

5.1.1 Source code structure

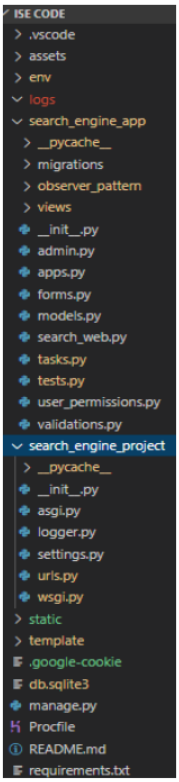

Figure 5.2: Source code structure

Below given is the source code repository link.

https://github.com/AdarshKoppManjunath/Improved-Search-Engine-Based-on-Sentimental-Analysis

5.2 Deployment diagram

- Improved Search Application is deployed on Heroku [4] as it provides both application and database hosting for free. So the application is deployed on a single server which comprises both web and data servers.
- Server is located in North Virginia, US

Below is the deployment diagram regarding the hardware configuration of the code

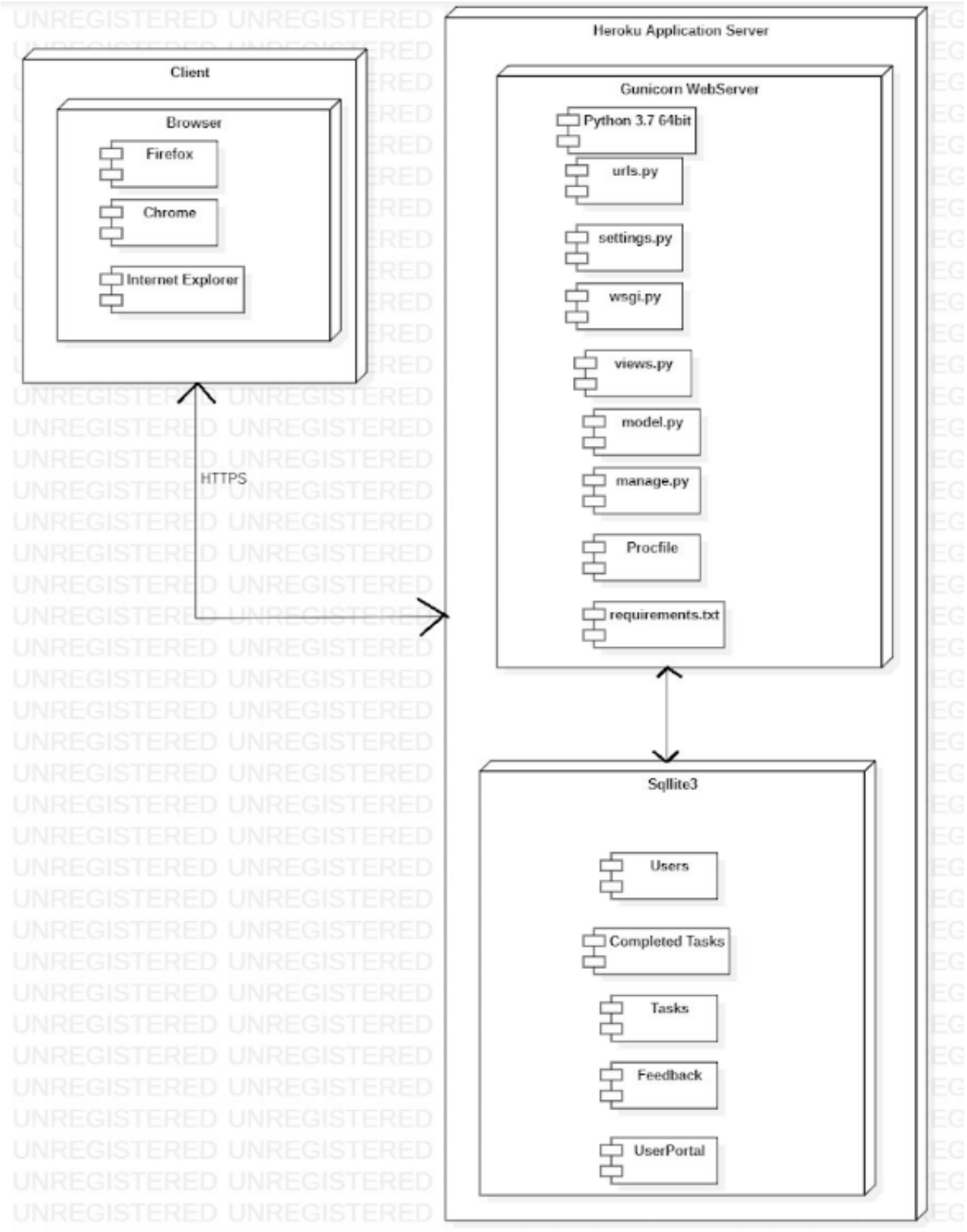

Figure 5.3: Deployment diagram regarding the hardware configuration of the code

Below shown is the deployment log generated while deploying our improved search engine application on Heroku. These logs help us in understanding the deployment process of our application.

```
-----> Need to update SQLite3, clearing cache
-----> Installing python-3.7
-----> Installing pip
-----> Installing SQLite3
Sqlite3 successfully installed.
-----> Installing requirements with pip
-----> $ python manage.py collectstatic --noinput
       152 static files copied to '/tmp/
   build_9adea680ae70e6759acb6eb196d755c5/staticfiles',
       478 post-processed.
-----> Discovering process types
       Procfile declares types -> web
-----> Compressing...
       Done: 301.5M
-----> Launching...
 !     Warning: Your slug size (301 MB) exceeds our soft limit (300
       MB) which may affect boot time.
       Released v20
       https://improvedsearchengine-se.herokuapp.com/ deployed to
       Heroku
```

Code Listing 5.1: Heroku deployment logs

5.3 System Data

In this chapter we focus on the application databases and their components. In this section we will show all the tables used for our application with screenshots.

5.3.1 Tables considered for the application

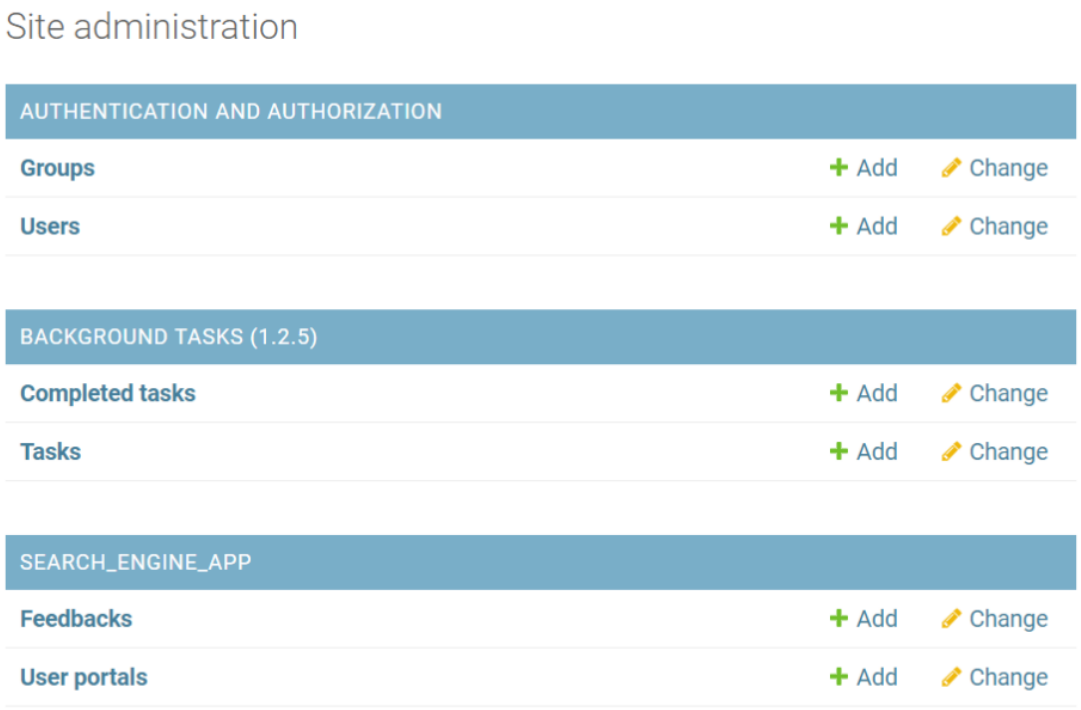

Figure 5.4: Screenshots of the database tables

We have categorised 5 tables into three types such as: Authentication and Authorization, Background Tasks, and Search Engine App administration.

- **Users**
  - This is Django's default users and authentication model. All users created will be saved in the Django's users and authentication model.
  - Users consists of both admin and website registered users in our case newsreaders. Staff status green checked is represented as admin.

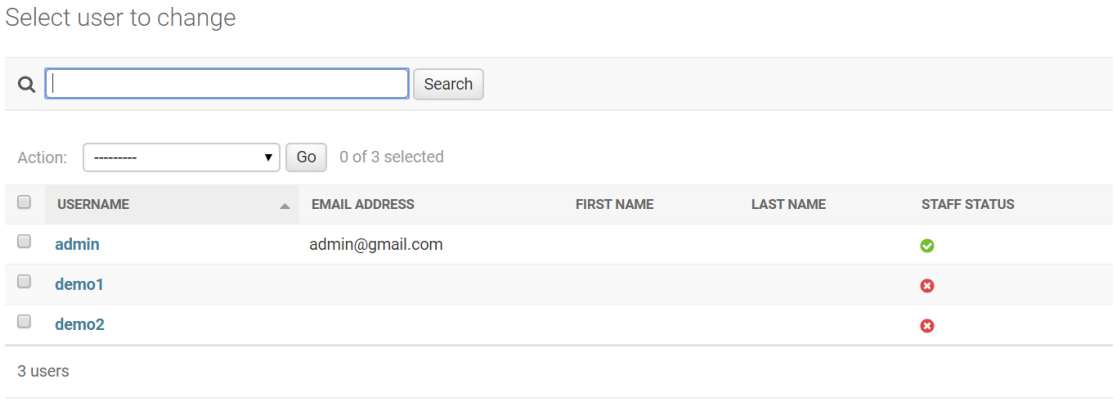

Figure 5.5: Screenshot of the users

- **Tasks and Background tasks**

  – This is created automatically by Django's background task library. To monitor URL status in observer pattern, we used a Django background task. Once a task is completed, the same task appears in the completed task.

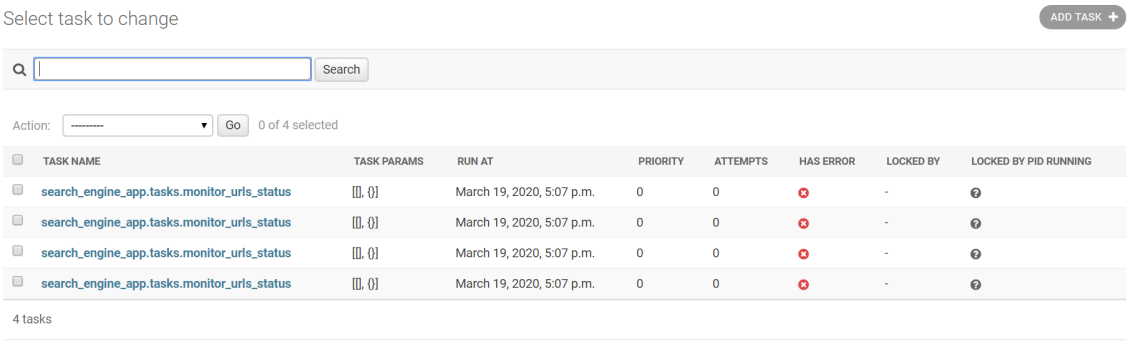

Figure 5.6: Screenshot of tasks

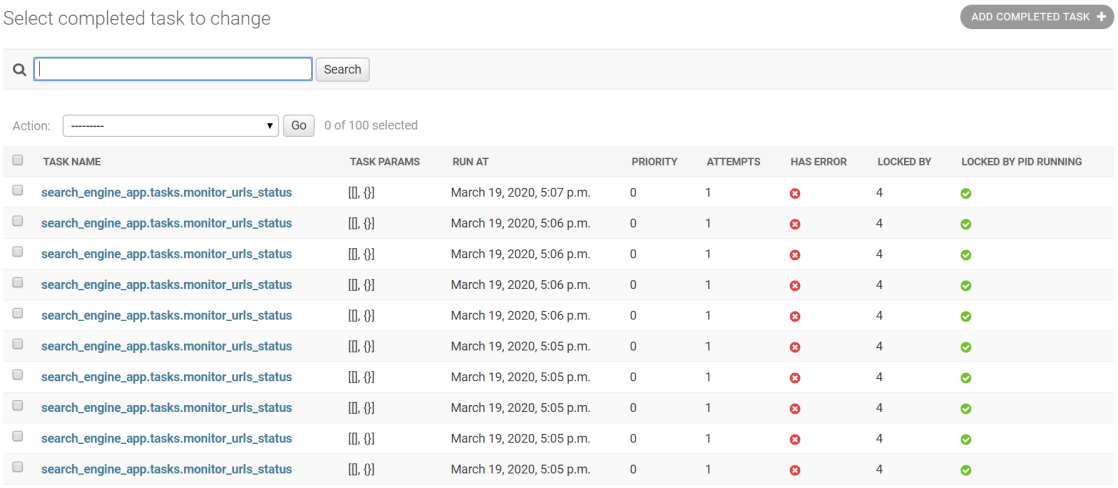

Figure 5.7: Screenshot of completed tasks

- **Feedbacks**
  - Used to store all the newsreader's feedback

Below is the screenshot of a list of newsreader's who have recently given feedback about the application.

Figure 5.8: Screenshot of the feedback given by recent newsreaders

As seen from the above screenshot demo1 and demo2 are newsreaders. Only the admin can see the list of newsreaders and their feedback. The admin can view and then edit the feedback according to their preference.

Shown below is the screenshot of the feedback given by the newsreader named **demo1**

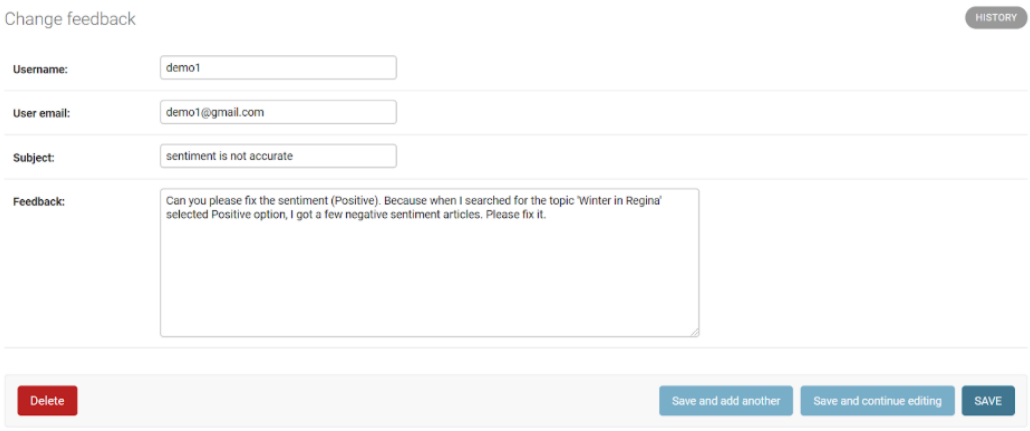

Figure 5.9: Screenshot of the newsreaders feedback

- **User Portals**

  – Used to store all the details of the article that the newsreader's want to save in their portal. This user portals table has fields as shown below:

Shown below is the screenshot of the topic details saved in the database given by the newsreader named **demo1**

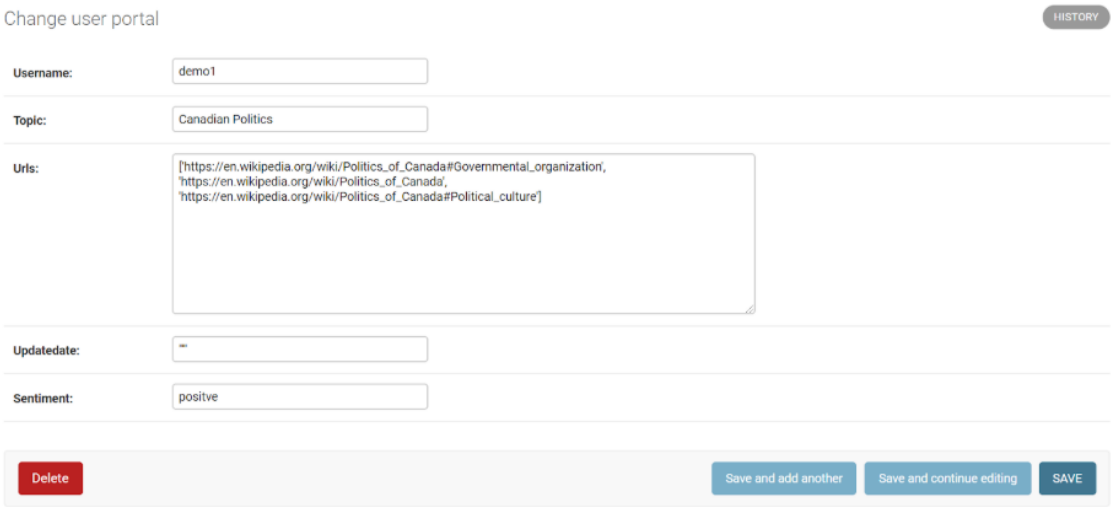

Figure 5.10: Screenshot of the newsreader's portal

5.4 Link to our web-based application

Below given is the link to our web-based application.

Link: https://improvedsearchengine-se.herokuapp.com/search_engine_app/

Chapter 6

Technical documentation

In this chapter, we will provide the technical documentation of the proposed application, which includes a list of programming languages, reused algorithms, software tools and environment.

6.1 List of programming languages

We have categorized the languages into three parts as programming, web-designing languages and Django's template language which can be found below:

6.1.1 Programming languages

- **Python:**
  - Since we all know that Python is one of the most commonly used programming languages, we coded the entire backend of our project using python.
  - All the views, forms, googlesearch, multi-threading, web scraping, cleaning text, sentiment analysis, and ordering the output based on the results and many more were coded in python.

Source	https://docs.python.org/3/
Version used	3.7
Filename examples	admin.py, forms.py, models.py, search_web.py, tasks.py, etc.

- **JavaScript**

  - As we know that JavaScript is a high-level programming language based on object orientation. We used it for expanding and collapsing the navigation bar when the user clicks on the icon which is used to enhance the user experience also known as responsiveness.

  - Also, we have used JavaScript code to increase the screen time delay in the messages (Success or failure). The JavaScript code can be found in the static/js folder. We also used JavaScript code to set the placeholder values for the forms.

Source	https://www.w3schools.com/js/
Version used	1.7
Filename examples	navBarJScript.js, messageScreenTime.js, registerPlaceholder.js

6.1.2 Web design language

- **HTML**

  - HTML as a markup language was very useful in building the front-end of the application.

  - The forms such as the search engine (text input form) and filtering the options (radio button forms) were all designed using HTML.

- All the HTML pages are located in the templates folder. Also, other pages such as home, project teams, about, login and signup, navigation bar all these were designed using HTML. Basically, all the front-end user interface was designed using HTML.

Source	https://www.w3schools.com/html/
Version used	5 (HTML5)
Filename examples	navBar.html, ise.html, login.html, home.html, etc.

- **CSS**

  - Although only using HTML doesn't make the application look colorful, we needed to incorporate different styles and layouts for the application. Hence, we used CSS (Cascading Style Sheets) to serve the purpose.

  - We have used CSS specifically for implementing the card views for displaying the information, grid view for the layouts, external fonts for the text and many more. The CSS files can be found in the static/css folder.

Source	https://www.w3.org/Style/CSS/
Version used	4 (CSS4)
Filename examples	navBarStyle.css, iseStyle.css, loginStyle.css, main.css, etc.

- **Bootstrap**

  - Last, we used bootstrap for making the application responsive. Every single page in the application is responsive meaning every page can and will perfectly fit into any screen irrespective of their aspect ratios.

  - There is no separate file written for the bootstrap code, we must place the bootstrap code inside the HTML code. To make the page responsive

all we did is used external bootstrap libraries and placed all the HTML code inside the div tag. By doing so automatically, the pages would be responsive.

Source	https://getbootstrap.com/
Version used	4.0.1
Filename examples	navBar.html, ise.html, login.html, home.html, etc.

- **Django Template Language**

  - The Django template language provides a platform to render python variables onto the HTML pages. Suppose in our case, when the newsreaders logged in to the application, we got their username, but the hardest part was displaying it in their portal (HTML) page. With the help of Django template language we could do this in one step by using {{% *request.uname* %%}}. We placed this code in the HTML page and then the username was rendered on the portal page.

Source	https://djangoproject.com/template-language-intro
Version used	3.0
Filename examples	home.html, results.html, userPortal.html, etc.

6.2 List of reused algorithms and programs

Below are the reused algorithms and programs we used in building our application. We have also provided their exact sources.

- **Extracting text from HTML file using python**: This program takes the URL as input and parses it to Beautiful soup library to extract the content.

Also, this program cleans the extracted text at the primary level with a basic blacklist filter.

Source: https://stackoverflow.com/questions/328356/extracting-text-from-html-file-using-python/2461818624618186

- **Django logging**: Logging level debug is used purely for debugging purposes while developing the software. Whenever we are debugging the programing rather than using the print statements to check the status of the code, we can use the Django logging module. There are 5 python log levels such as Debug, Info, Warning, Error, and Critical out of which we have used the Debug level which is a low-level system information used for debugging purposes

  Source: https://github.com/metakermit/fail-nicely-django

- **Navigation Bar**: This program will display a responsive navigation bar. This was coded in CSS and JavaScript. This navigation bar works very well even on resizing the browser window. This program has HTML, CSS and JavaScript codes.

  Source: https://www.w3schools.com/howto/tryit.asp?filename=tryhow_js_topnav

- **Application Launch Template:**This template was designed using CSS. We used this template to display more information about the improved search engine. We designed this template more like a "About" page for the application.

  Source: https://www.w3schools.com/w3css/tryw3css_templates_app_launch.htm

- **Team design card flipper:** This is a bootstrap card flipper, basically this is used to display team members and their details on cards

  Source: https://bootsnipp.com/snippets/92xNm

- **Login page:** This template was used to create a login form for our application. This program was designed using HTML5 and CSS3.

Source: https://bootsnipp.com/snippets/3522X

6.3 List of all the software tools and environment

Below are the software tools and environments that have benefited us during building the application.

- **Visual Studio Code**: This is the IDE where we wrote all the code both for the front-end and the backend. It is a lightweight, powerful code editor. The reason for choosing visual studio code is that of its extended features such as IntelliSense which provides auto code completion, code assist and code hinting. Also, with its built-in command prompt installing all the external libraries was like a cakewalk. With the help of its Git extensions we could easily learn, use and save our code on GitHub. We could easily document the code with its built-in features. Also, code navigation was helpful for us during the development.

Version used	1.4.2
Source	https://code.visualstudio.com/download

- **Star UML**: This is an open-source free tool which helps us to develop fast, extensible, featured UML diagrams. With the help of this, we could develop all the UML diagrams with no problem. The main benefit of this tool was that we got options to save the model as well in .PNG, .JPEG, or PDF, which was helpful, all this can be done without writing a single line of code. We used StarUML to draw **use case, activity, component, deployment, class diagram and MVT architecture**. StarUML tool made it very easy for us to draw all these diagrams

Version used	3.0.2
Source	http://staruml.io/

- **GitHub:** We uploaded all the code of the application on GitHub. In the future, if we or anyone wants to improve the application, then they can easily get the source code. This is one of the main reasons we wanted to make the code as open source such that this would be helpful for others. The main reason we used GitHub was that both of us parallelly wrote the code for front-end and back-end and every time we needed a place to store the code and commit all the changes. Also, deploying the code on Heroku was very easy, it directly pulled the code from GitHub.

Version used	2.3.1
Source	https://desktop.github.com/

- **Django:** Since Django is a python based web framework and also provides MVT architecture we decided to use this. Django provides authentication system, which was very useful for us to secure newsreaders accounts with minimal effort. Django provides class based view option which was useful in factory pattern. Django forms made our effort less while creating login, registration and feedback forms. Django features like logging system, messages, background task helped us to achieve our goal. Django greatly reduced our efforts by providing us the admin UI platform which helped us to access the database with its default UI.

Version used	3.0.3
Source	https://www.djangoproject.com or pip install Django

- **Django Background Task:** Database backend queue for Django. Used in our project to monitor URL status. Django background tasks can be scheduled or can keep the task in continuous running mode with an interval. Task is created as a CheckUpdate class, which monitors the userportal model to check URL status every five hours.

Version used	1.2.5
Source	https://django-background-tasks

- **Heroku:** Heroku is a platform as a service (Paas), that allows us to build and develop the application and then deploy in the cloud. This service was free for us to use. With the help of this service we were able to host both web-application and database (SQLite3) on the same server. Our application got default SSL certificate from DigiCert as we are using Heroku, so that the communication or the data sent through our application is secure. Deployment through GitHub repository was very helpful. Heroku deployment logs was very useful for debugging purpose.

Version used	7.39.0
Source	https://www.heroku.com/

- **Django Logger system**: Logger system comes with debug, info, error, warning and critical level. In this project only debug is used during software development. Django provides a default logger, but it has been modified using the tool below.

Version used	1.0
Source	https://github.com/metakermit/fail-nicely-django

- **DB SQlite3**: Lighter version of SQL comes with django by default. We used the same for the website development. Database consists of users, newsreader subscribers and feedback models.

Version used	3.8.3
Source	https://djangoproject.com/sqlite-notes

Chapter 7

Acceptance testing

In this chapter, we will conduct acceptance testing by performing functional testing, robustness testing, time-efficiency and security testing along with four inputs and output screenshots.

7.1 Functional testing

Requirements are the extended features mentioned in the proposed application section. In this section, we will be testing whether all the requirements are met or not.

Test case 1

When newsreader enters the topic and sentiment search result should be displayed relevant to the topic. Number of URLs should be less than 5 or 5 and description of each URL should be within 30 lines.

- **Input:** Newsreader enters the topic along with sentiment as input. Topic **Canada Winter**, sentiment **neutral**

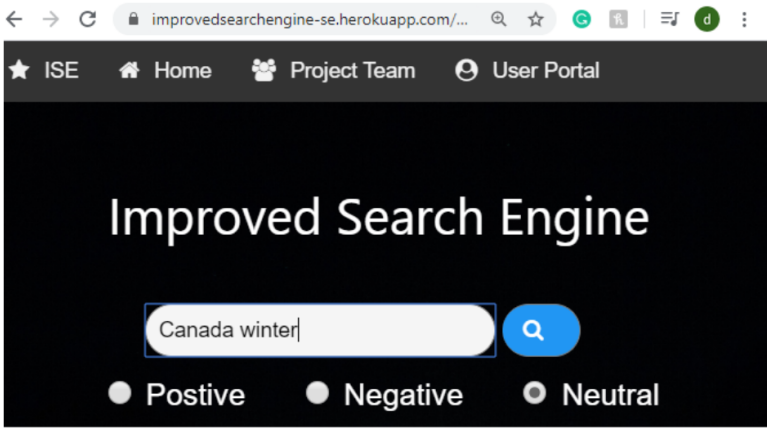

Figure 7.1: Search topic input along with sentiment

- **Output:** The description of the search results are displayed within 30 lines


https://www.tripsavvy.com/canada-in-winter-1481739

You will want to pack a winter jacket with a hood, ideally something that can also withstand wind and snowfall In addition, make sure to bring a hat, mitts and a scarf, sturdy winter boots, a vest, long-sleeved shirts and other items that can be layered to ensure warmth on the coldest days If you'll be doing any skiing or winter hiking, thermal underwear and thick socks are also a good idea Winter Events in Canada Just because it's cold outside, doesn't mean Canada slows down when it comes to fun things to do There are a wide range of festivals that occur throughout the winter season and here are some of the best Winterlude : Cold weather doesn't hold anyone back from having fun in the snow when it comes to Ottawa's annual Winterlude festivities Most festival activities are free and take place in the Ottawa-Gatineau region during the first three weekends of February You can expect ice sculptures, snow sculptures, skiing and snowboarding lessons, live performances and the chance to skate on the Rideau Canal Skateway – the world's largest naturally frozen ice skating rink Ice on Whyte : Edmonton hosts the Ice on Whyte festival each winter which centres on an ice carving competition featuring some of the best ice carvers from around the world But that's not all Guests can also participate in ice carving lessons, have a drink at the ice bar, cozy up at a fire pit, grab a bite to eat from a food truck and much more Carnaval de Quebec : Quebec City is home to one of the world's largest winter carnivals featuring everything from lively night parades and snow sculptures, to shows, ice skating, local food and more The fun takes place between the end of January until mid-February with activities and events happening throughout the city Aurora Winter Festival : Vancouver is where you'll find the annual Aurora Winter Festival , which takes place towards the end of November and runs until early January In addition to a festive atmosphere thanks to an abundance of twinkling lights, there are market stalls to browse, amusement rides, live entertainment, food huts and a skating pond to enjoy Frostival : This festival in Fredericton takes place over three weekends between January and February and offers visitors the chance to embrace the winter season Atlantic Canada's largest winter celebration includes more than 150 events, from theatre performances and family activities to a music festival, cultural experiences and sports competitions South Shore Lobster Crawl : The entire month of February is devoted to all things lobster during this Nova Scotia festival Come hungry to experience more than 150 lobster-infused menus, events, experiences and activities in ports between Barrington, 'The Lobster Capital of Canada' and Peggy's Cove Chow down on a lobster roll or two, go on a lobster tour and enjoy live performances throughout the fest Montréal en Lumière : This fun festival in Montreal combines fine dining, outdoor festivities, interactive lighting and a cultural program Some of the best local and international chefs serve up their best dishes, while festival-goers can also enjoy live music, activities for kids, warming stations and marshmallow roasting, bars and food trucks Everything culminates with La Nuit Blanche, an all-night arts and cultural event for all ages Winter Travel Tips Popular ski destinations like Banff and Lake Louise along the Rockies come to life this time of year, so they could actually be more expensive since it's high season for winter sports Other parts of Canada, however, may see lower prices on hotel rooms and flights making winter a more economical time to travel No matter where you're travelling in Canada in the winter, make sure to check the weather as you're packing so you have a more concrete idea of what to expect in terms of temperatures

https://www.tripsavvy.com/top-canada-winter-activities-1481749

The cathedral-like hotel is carved entirely of ice, including the furniture and icy candelabras hanging from the 18-foot ceilings What's even more special about this unique attraction is that the Quebec Ice Hotel is rebuilt each year, opening its doors from January to the beginning of April The walls are four feet thick and insulate the hotel to a crisp but comfortable 23 to 28 degrees Fahrenheit (-5 to -2 degrees Celsius) Visitors may choose to just pass through for a tour and a drink at the ice bar or settle in and stay overnight 03 of 18 Enjoy the Québec Winter Carnival Getty Images/Stuart Dee The residents of New France, now Quebec, had a rowdy tradition of getting together just before Lent to eat, drink, and be merry Today, the Québec Winter Carnival carries on this tradition with the biggest winter carnival in the world, celebrated annually at the end of January through mid-February The event is staged largely for families to enjoy, and they come out in droves to embrace and celebrate the cold all wearing the traditional red sash There are parades, pop-up events throughout the town, live music, and culinary offerings Aside from accommodations for the event, a trip to the Québec Winter Carnival costs relatively little 04 of 18 Skate the Rideau Canal (Ottawa) Preappy / Getty Images View Map Address Rideau Canal, Ottawa , ON , Canada Get directions Every winter, Ottawa's Rideau Canal becomes The Rideau Canal Skateway and at just under five miles (7.8 kilometers) long, it's the world's largest skating rink Locals and visitors alike make the most of this frozen roadway in winter, using it both as a means of transportation and form of recreation It's important to keep in mind when planning your trip that the skateway generally opens in January or February when the canal is sufficiently frozen and safe for skaters Skate rentals and sharpening, as well as boot checks, are also available, and you can even rent sleighs so children can sit while adults push them along the canal Continue to 5 of 18 below 05 of 18 Visit Winterlude Richard T Nowitz / Getty Images View Map Address Rideau Canal Eastern Pathway , Ottawa , ON K1P , Canada Get directions Phone +1 844-878-8333 Web Visit website Canadians celebrate sub-zero temperatures and waist-high snowdrifts by staging great winter festivals, such as Ottawa's Winterlude For the first three weekends every February, the nation's capital puts on a winter festival that features ice-skating on the world's longest rink, ice sculptures, a snow playground, concerts, and more 06 of 18 Go Dogsledding Kraig Becker Dogsledding is one of the more memorable Canadian adventures available in the winter months Whether you want to spend a few days or weeks in the outback or just try it for an afternoon, dog sledding is an activity available most anywhere in Canada that gets lots of snow The Call of the Wild is an award-winning, established adventure team that offers dogsledding excursions and year-round adventures 07 of 18 Take a Winter Rail Vacation Mike Grandmaison/Getty Images Sit back and soak up the majesty of the Canadian Rockies in the winter on a train trip across Canada Rocky Mountaineer offers spectacular Canadian winter train vacations that range from the leisurely to the downright luxurious Vacations include train travel to from Vancouver /Calgary, transportation to and between Banff, Jasper, and Lake Louise, plus opportunities to explore some of the country's most magnificent scenery and two great Canadian cities 08 of 18 Try Snowshoeing Getty Images/Nino H Photography Once used by trappers and traders traversing the snow-covered terrain, snowshoeing today is mostly a form of winter recreation Contemporary lightweight snowshoes make this traditional form of winter travel easier and more fun than ever Snowshoeing is both a wonderful way to explore the great outdoors and an effective and gentle form of exercise Many ski resorts and winterized lodges—such as the fairytale-like Fairmont Chateau Lake Louise —provide or rent snowshoes for casual outings Continue to 9 of 18 below 09 of 18 Go Ice Fishing Getty Images/Yves Marcoux/Design Pics Ice fishing is a magnificent way to enjoy the Canadian winter and commune with nature Not only does Canada offer the necessary climate, but it has a huge array of excellent resorts and lodges that range from basic to luxury offering chances to fish on the ice One of the favorite ice fishing outfitters is Andy Myer's Lodge on Eagle Lake in Ontario

https://traveltriangle.com/blog/best-places-to-visit-in-canada-in-winter/

Churchill: Enjoy The Beauty Of A Winter Wilderness Churchill is one of the best places to visit in Canada places to visit in Canada during winter for those who enjoy the peace of winters The highlight of this small town located on the shores of Hudson Bay is polar bears and Northern Lights Get a chance to see the powerful polar bears in the wild Tourist Attractions: Polar bears, seasonal bird watching and chance for wintertime Northern Lights visibility Best Place To Stay: Iceberg Inn Things To Do: Play with curious Beluga Whales Watch the Northern Lights dance View polar bears in the wild Suggested Read: 10 Vancouver Hotels For Luxury And Budget Travelers 4 Vancouver: Explore Local Seafood Popularly known for its physical beauty, Vancouver is one of the best winter places to visit in Canada The gorgeous city is famous for being one of the world's most liveable cities It is a great place to get away if you're looking for a break from the snow right here in Canada Have the best shopping and dining experience as well as visit renowned museum Tourist Attractions: Local seafood, shopping experience, outdoor sport and adventure Best Place To Stay: Fairmont Hotel Vancouver Things To Do In Vancouver : Try local seafood Indulge in outdoor recreational sports Build an igloo Suggested Read: 15 Incredibly Romantic Ways To Spend Valentines Day In Canada In 2020 5 Ontario: The Dazzling Niagara Falls Image Source While the humongous falls are an absolute delight in summers, the frozen falls in Canada winter look equally bewitching The sheer beauty of the frozen falls is difficult to express in the words A spectacular dazzler is as close as you can come to describe the mystical falls in the winter You can see the chilling mist rising above the falls looks so mystical It is definitely one of the best places to visit in Canada in winter Tourist Attractions : An ice-wine festival is organized every year in Niagara town which you ought to attend if you like sweet dessert-style wine Best Places To Stay: The Inn At Cobble Beach Things To Do: See the Capital Drink the wine Hike a trail Suggested Read:

Figure 7.2: Descriptions of search results

Test case 2

Search results should be ordered according to the newsreaders inputted sentiment. This test case has 3 scenarios which includes positive, negative and neutral.

- **Input:** Giving the input as positive sentiment along with the topic. Search result is ordered according to the sentiment inputted by the newsreader.

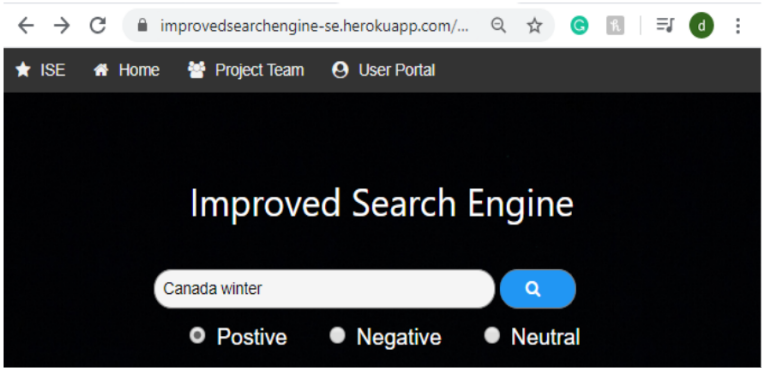

Figure 7.3: Positive sentiment along with the search topic

- **Output:** Search Results - Positive

Figure 7.4: Search results for positive sentiment

- **Input:** Giving the input as negative sentiment along with the topic

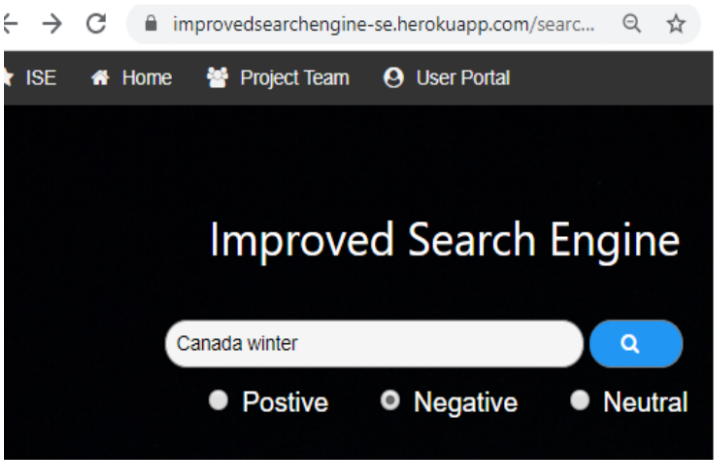

Figure 7.5: Negative sentiment along with sentiment topic

- **Output:** Search results - negative

Figure 7.6: Search results for negative sentiment

- **Input:** Giving the input as neutral sentiment along with the topic

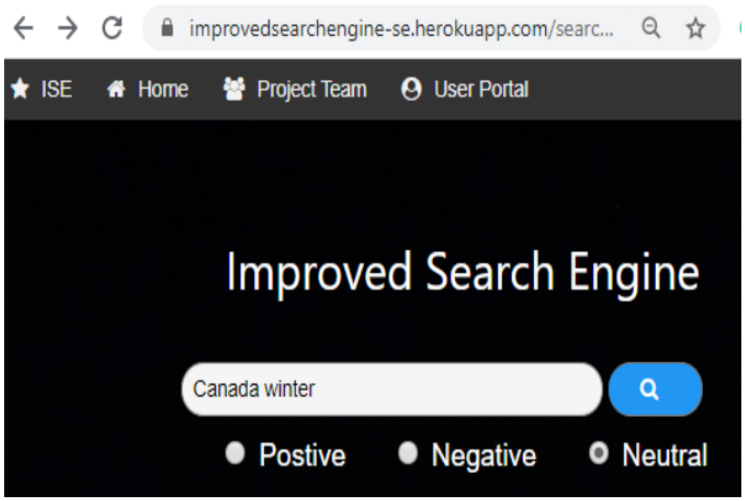

Figure 7.7: Neutral sentiment along with sentiment topic

- **Output:** Search results - neutral

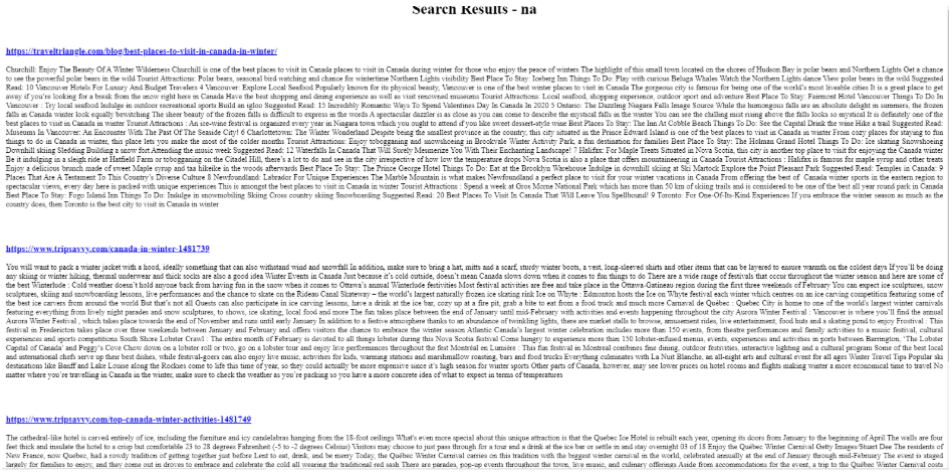

Figure 7.8: Search results for neutral sentiment

Test case 3

Newsreader should be able to save the topic and sentiment and can follow up with the topic

- **Input:** Newsreaders (demo1) can save topics along with sentiment, and follow up with the topic. Example, topic being entered is **Canadian Politics** along with the **positive** sentiment option is being saved

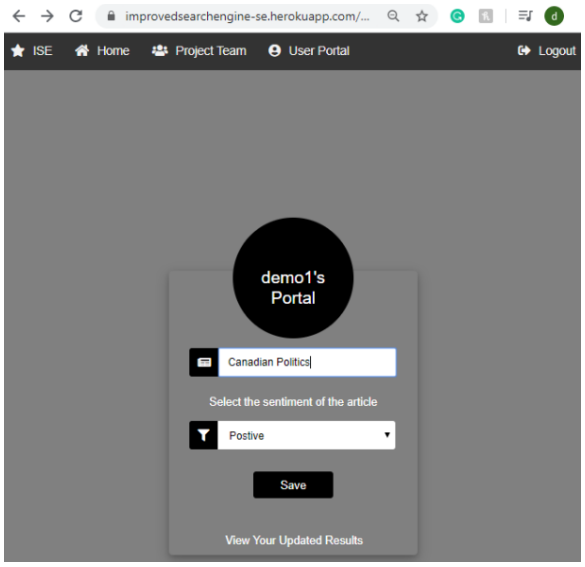

Figure 7.9: Search topic and sentiment to be followed up

- **Output:** All the URL links corresponding to the newsreader's given topic and sentiment would be saved and displayed.

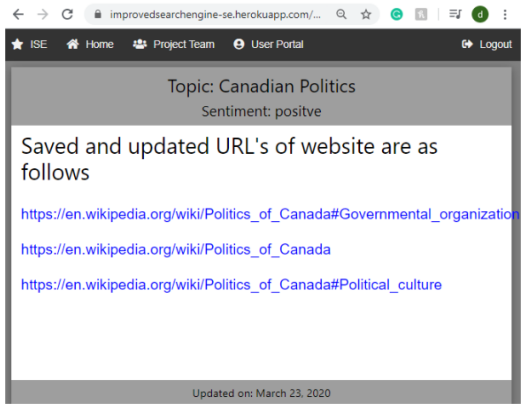

Figure 7.10: Updated URLs of the topic are displayed

Test case 4

Newsreader can send feedback to the admin if they are not happy with the result.

- **Input:** Newsreader can send feedback to the admin, if search result is not accurate. The newsreader clicks on the feedback button.

Figure 7.11: Newsreader clicks on feedback option

- **Output:** The newsreader gives feedback and submits it.

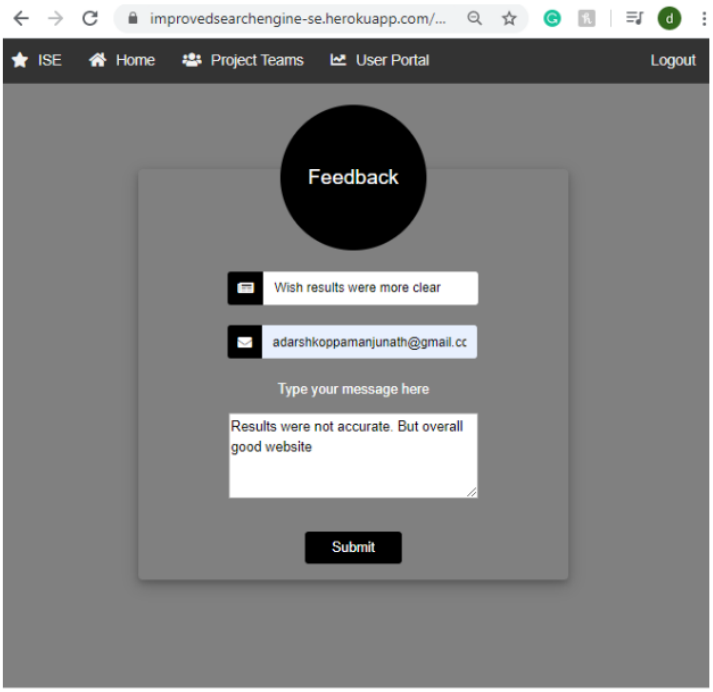

Figure 7.12: Submitting the feedback

7.2 Robustness testing

Below are the screenshots with four incorrect input and output data.

Test case 1

Entering the search topic other than English language

- **Input:** Entering a language (Kannada - Indian language) other than English in the search bar and selecting the sentiment option as neutral.

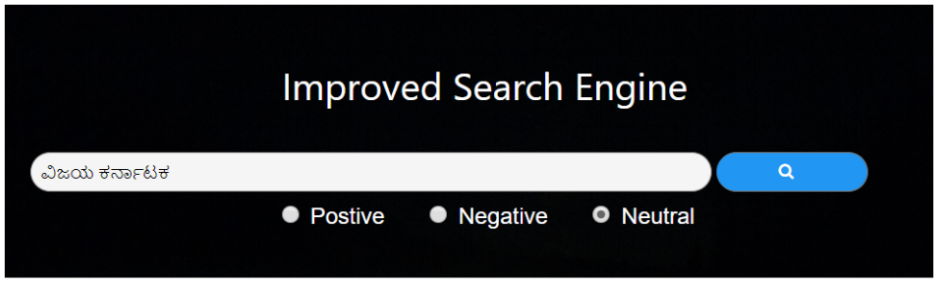

Figure 7.13: Entering a language other than English as input

- **Output:** An error message page would be prompted when the desired language is not English. Also, a link would be provided to go back to the search page.

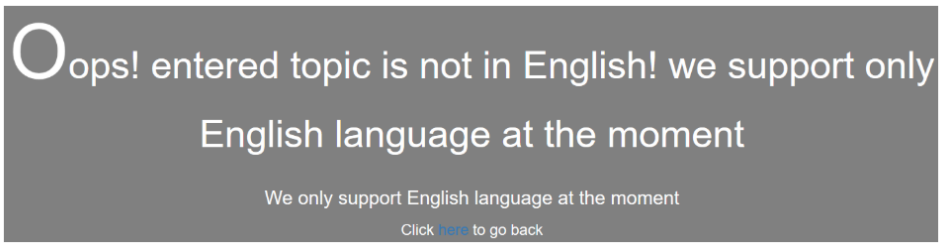

Figure 7.14: Error message as output for using incorrect language

Test case 2

Registration error if user is prior registered and creates a new account with the same username.

- **Input:** Newsreader tries to create a new account by giving the same username.

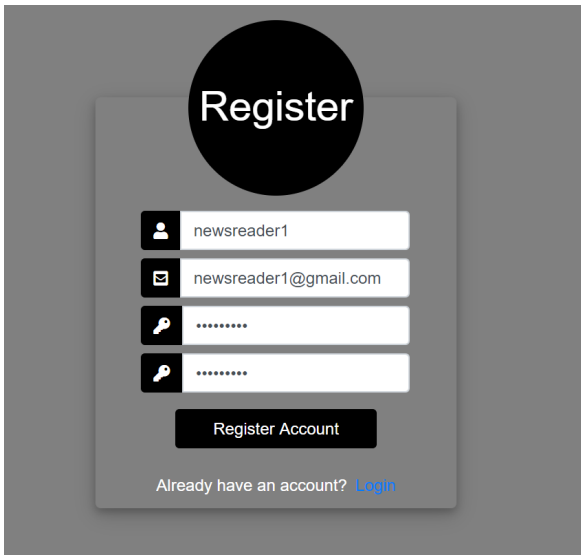

Figure 7.15: Existing newsreader creates a new account by inputting existing username

- **Output:** A validation error message as **"A user with that username already exists"** would be prompted.

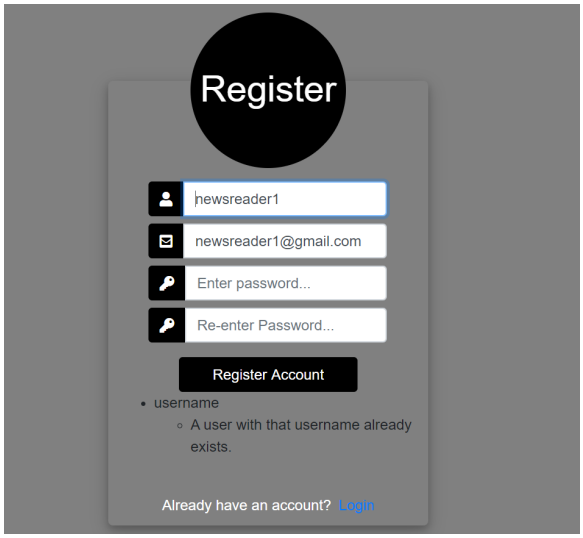

Figure 7.16: Error message, username already exists

Test case 3

Login error when a newsreader types incorrect login details such as username and password.

- **Input:** A existing newsreader tries to log in to the system by giving incorrect credentials.

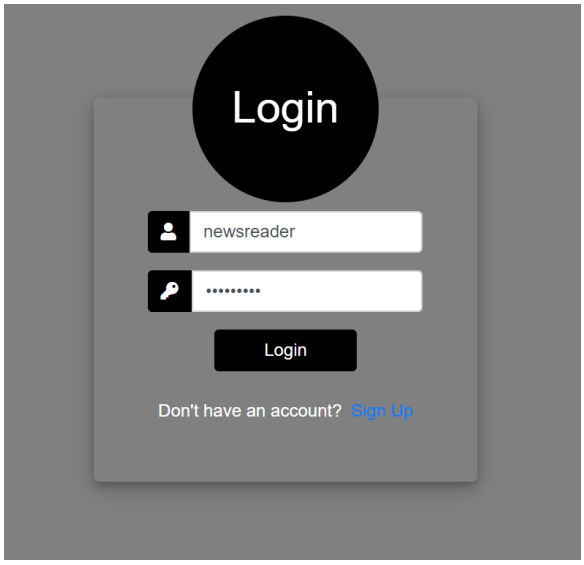

Figure 7.17: Existing newsreader inputs incorrect details while log in

- **Output:** A login error message will be prompted because of mismatch in the credentials.

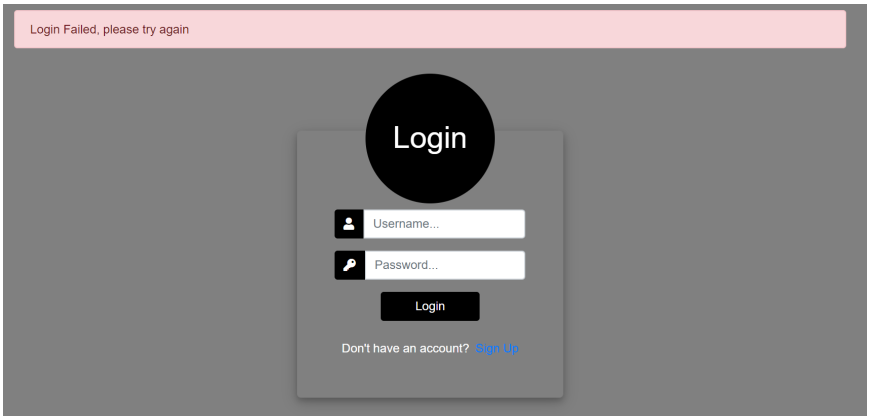

Figure 7.18: Login failed error messages would be prompted

Test case 4

If any form is not filled, and entered the submit button, please fill out this field warning will be displayed.

- **Input:** The newsreader inputs nothing and tries to submit the feedback form.

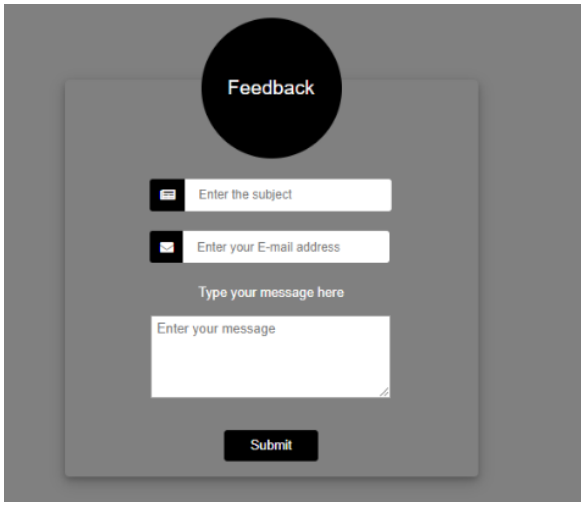

Figure 7.19: Newsreader tries to submit empty feedback form

- **Output:** The form warning would be displays which says **"Please fill out this field"**

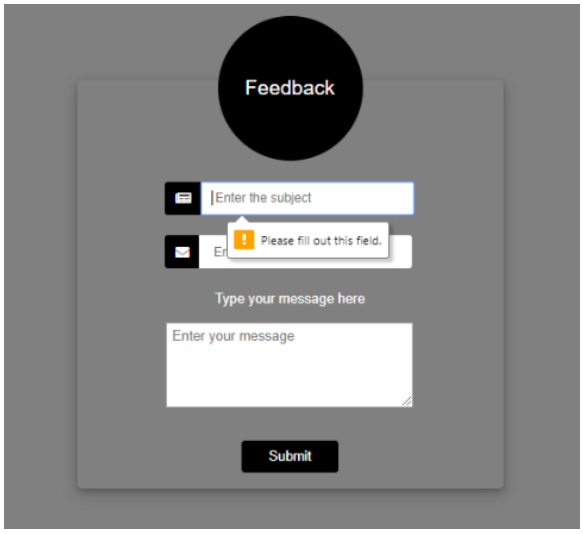

Figure 7.20: Feedback warning would be prompted, an newsreader can never submit an empty form

7.3 Time efficiency testing

Below are the screenshots for time efficiency testing of four functions.

Test case 1

- **Input:** Home Page

  https://improvedsearchengine-se.herokuapp.com/search_engine_app/

- **Output:** Report from pingdom tool

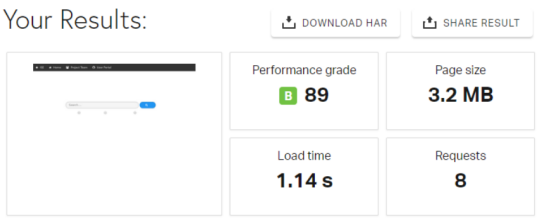

Figure 7.21: Home page report generated from pingdom tool

Test case 2

- **Input:** Login page

  https://improvedsearchengine-se.herokuapp.com/search_engine_app/login.html

- **Output:** Report from pingdom tool

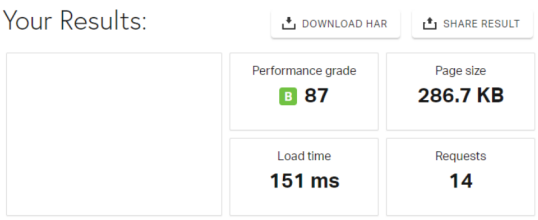

Figure 7.22: Login page report generated by pingdom tool

Test case 3

- **Input:** User portal page
  https://improvedsearchengine-se.herokuapp.com/search_engine_app/userPortal.html
- **Output:** Report from pingdom tool

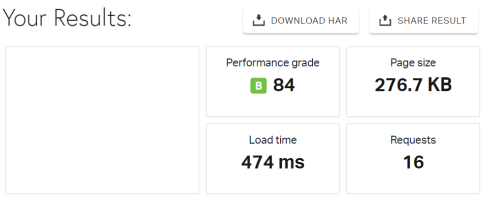

Figure 7.23: User portal page report generated by pingdom tool

Test case 4

- **Input:** Feedback page
  https://improvedsearchengine-se.herokuapp.com/search_engine_app/feedBack.html
- **Output:** Report from pingdom tool

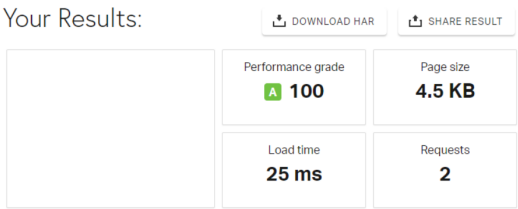

Figure 7.24: Feedback page report generated by pingdom tool

7.4 Security testing

Below are the screenshots for security testing with four test cases of input and output data.

Test case 1

HTTPs communication on URL navigation. Navigation from one URL to other should follow HTTPs protocol and not HTTP

- **Input:** URL navigation (Example feedback form URL)

Figure 7.25: URL navigation to feedback form

- **Output:** Secure connection

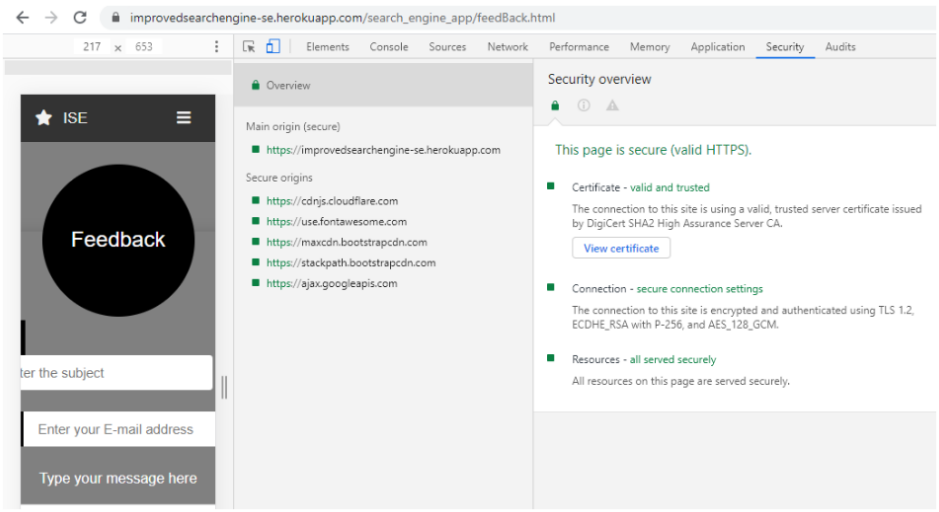

Figure 7.26: Connection is secure

Test case 2

TO show protected proxy. On clicking feedback a newsreader will be directed to feedback only if they are already logged in otherwise they will be redirected to the login page.

- **Input:** Newsreader request for feedback form

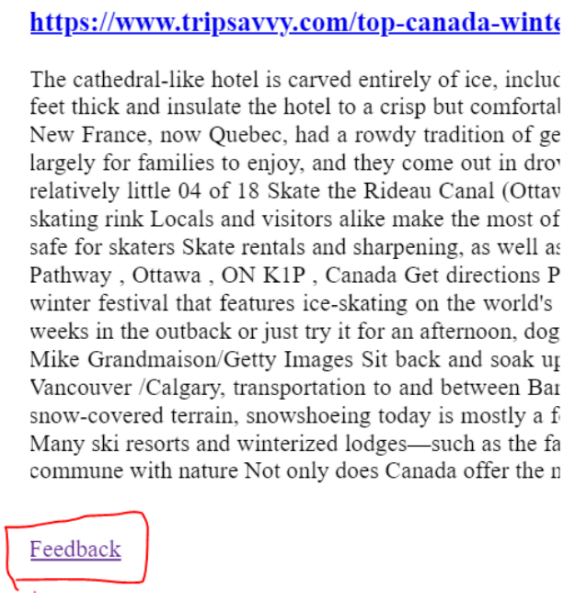

Figure 7.27: Newsreader requests for feedback form

- **Output:** Feedback page opens only if newsreader is already logged in and authorized

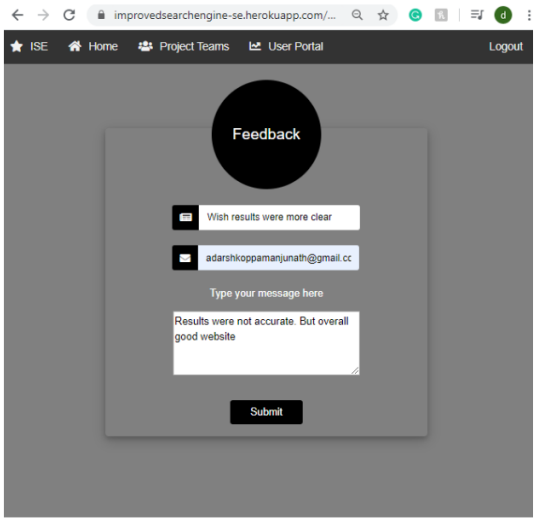

Figure 7.28: Newsreader is already logged in and authorized

If the newsreader is not logged in then the protected proxy invoke login page

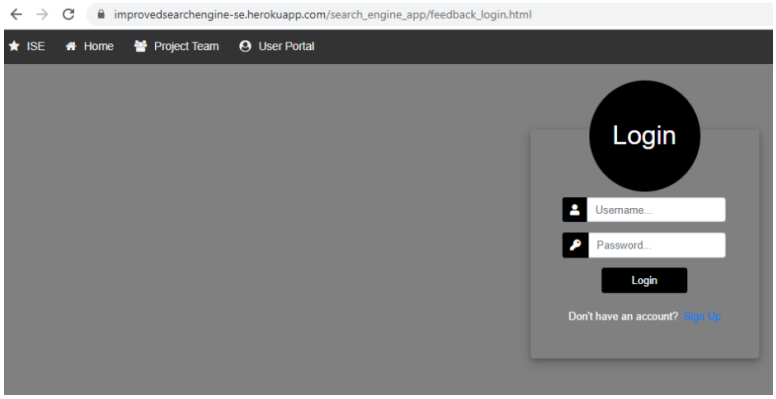

Figure 7.29: Login page

Test case 3

To show complex password pattern. Commonly used password, password same as user name, password only with digits, alphabets, or less than 8 characters will not be accepted.

- **Input:** Entering a weak password during user registration

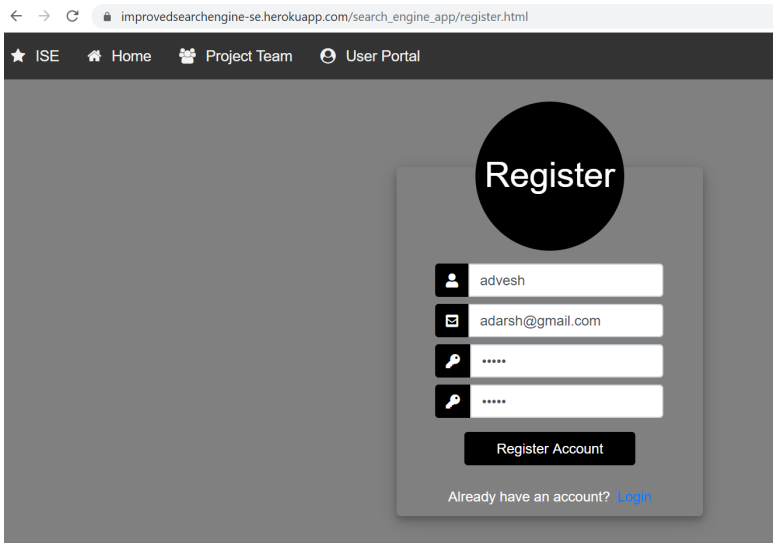

Figure 7.30: Entering a weak password during registration

- **Output:** Weak password error would be prompted if the password enter by the newsreader is weak such as the password entered must not be similar to the username, and it must be more that 8 characters and should comprise special characters.

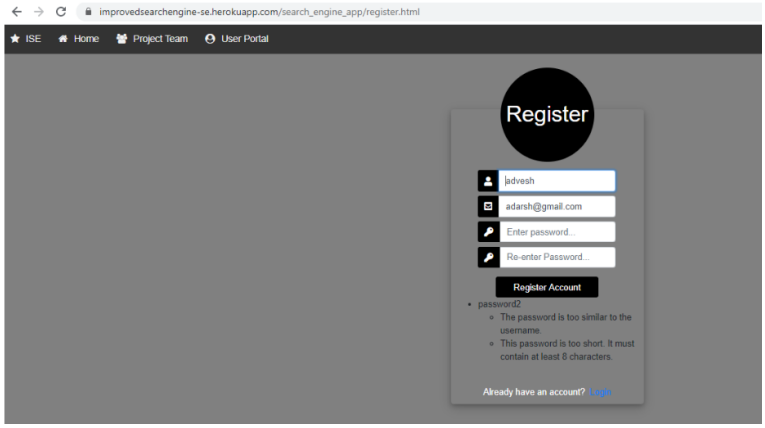

Figure 7.31: Weak password error

Test case 4

To show newsreader password encrypted, even admin cannot see newsreader's password.

- **Input:** List of newsreaders in admin portal.

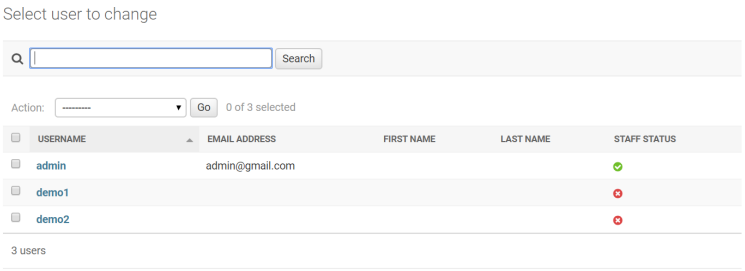

Figure 7.32: List of users in admin portal

- **Output:** The password of the newreader 1 called demo is encrypted even the admin cannot see the raw password.

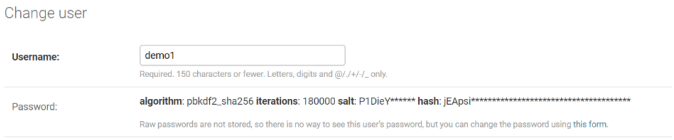

Figure 7.33: Encrypted password

Chapter 8

References

[1] Google. Retrieved 23 March 2020, from https://www.google.ca/

[2] Hseb, A. (2017). google-search. Retrieved 23 March 2020, from https://pypi.org/project/google-search/

[3] Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1994). Design patterns. Addison-Wesley.

[4] Cloud Application Platform—Heroku. Retrieved 24 March 2020, from https://www.heroku.com/

[5]PUTANO, B. (2019). The 5 Most Popular Programming Languages of 2019. Retrieved 24 March 2020, from https://stackify.com/popular-programming-languages-2018/