NEWS TRACKER APPLICATION

A PROJECT REPORT

Submitted by

S.DEEPAKCHANDHIRAN	(814619104004)
S.KARTHICK	(814619104009)
M.KRISHNARAJ	(814619104011)
S.VIGNESH	(814619104022)

In partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

TRICHY ENGINEERING COLLEGE, TRICHY

ANNA UNIVERSITY:CHENNAI600025

SEPTEMBER 2022

ABSTRACT

As our lives are very busy these days, we often feel we need more than 24 hrs. a day to cope up with everything we have in our schedule. Well, that's not possible but reducing the time by changing the conventional method of reading news can help. Just tell us what market news you're interested in and get a quick peek for the day. Only read what you feel is relevant and save your time.

This app helps you to query for all information about Indices, Commodities, Currencies, Future Rates, Bonds, etc.… as on official websites.

Topic tracking can help people to explore the process of topic development from the huge and complex network texts information. the authors analyzed the ability to use the blockchain method to allow users to track all oftheir uploaded news on social networks.

The world has witnessed anincredible amount of information relating to politics in the past few years which has people to believe the information all over the social media or any broadcasting platform is good but this comes the

Fake news, to increase chennal rating. Fake news is a false piece of information. In this day and age, with the increment in spread of phony news from web- based media and different sources it is getting vital tohave the option to classify between genuine news andphony news.

Due to the COVID-19 pandemic period. The issues of fake news have attained an increasing eminence in the diffusion ofshaping news stories. Many of them stop to depend on the newspapers, magazines,etc and started to rely on social media completely.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION AND PROJECT REVIEW

1.1 **Introduction:**

In the current scenario, there is no single platform (in application) present right now which provides cyber security information, E-Sport information, Science and Technology Information, etc. in one place. Cyber security users have to visit different websites together the news related to the cyber world. Many people do not have the time to visit different sites together information. Ultimately, this would be a waste of time and effort. Visiting different websites, the user might get the redundancy in the information

The System Manage the information of national breaking news. It tracks all the information of customization of content instant publishing, linking, articles etc. Shows the information and description of the entertainment news.

1.2 **project review:**

There are multiple news-sharing apps used by a single user and are often spammed with notifications. There is also a lot of fake news which gets shared. A news-sharing app wants to help users find relevant and important news easily every day and also understand explicitly that the news is not fake but from proper sources. These are useful to detect the news from the unnecessary public problems.

This solves the problem of the user having to reach out to other sources or to the Internet to verify/double-check if the news was real or fake that administers extra efforts on the user's part and also demolishes the main intent of the news app to provide a single-stop credible news platform and also causes irritation to the user.

The “like metric” solves the problem of the user not understanding whether it is worth spending his energy and time on this article. Also, it helps the user indicates that it is credible information.

1.3 **purpose:**

The purpose is to develop an android application, which will eliminate the problems faced in the current scenario. This application will provide all the information and news related to cyber security, E-sport, Science, and Technology or that are in trend at one place. So, it will save time and efforts of the users by making it more efficient. Using, this application will terminate the possibility of information redundancy.

The Motivations and scope behind this project are to connect people through this application and provide a medium to share their views on the topic/news/information. Then, People with the same interest can interact with each other. However, they can even share more information on the topic. Later on, we can publish this application on the Play Store.

CHAPTER 2

LITERATURE SURVEY

2.1 **Existing problem:**

we have examined and explored the performance of five algorithms that are dedicated to the detection of fake news.

The spread of fake news can adversely affect our surroundings. The purpose of experiment using a novel algorithm based on the detection and protection of nonmisleading information in all online and network environments. using a distributed and unchanging blockchain ledger we can find a source of information that helps determine whether the media supporting the news is trustworthy or false.

We have utilized RNN with LSTM units for neural organization. Also to check fact or headline, web scraped news is checked for authenticity.

As mentioned earlier the use of social media has been spread vastly, this research can be used as a skeleton for other investigators to interpret which models are precisely and accurately completing its mission in identifying fake news.

We have some mechanisms for the detection of fake news, or a way to aware people to know that everything is they read is not true, so we need critical thinking and evaluation. we can help people to make choices so that they won't be tricked or fooled into thinking what others want to guide or exploit into our thoughts.The LDA (latent Dirichlet allocation) model is used to extract the topic information from the news texts of different time windows. Then the improved Single-Pass algorithm is used for topic tracking, in which the time decay function and the JS divergence are used to measure the similarity between the topics. Finally, for the results of topic tracking, the content and strength of the topics are analyzed. In the experimental part, the topic discovery experiment is first carried out on the tagged corpus. It is found that the topics discovered by the LDA model are more reliable than the k-means clustering in topic recognition.

2.2 **References:**

1. Blockchain based News Application to combat Fake news. Mr Yashodhar Gulati, Mr. Gaurav Mistary, Mr. Swapnesh Tilekar, Ms Namrata Naikwade2022

2. Research on Topic Detection and Tracking for Online News Texts. Guixian xu, Yueting meng, Zhan chen , Xiaoyu qiu , Changzhi wang , and Haishen yao-2019

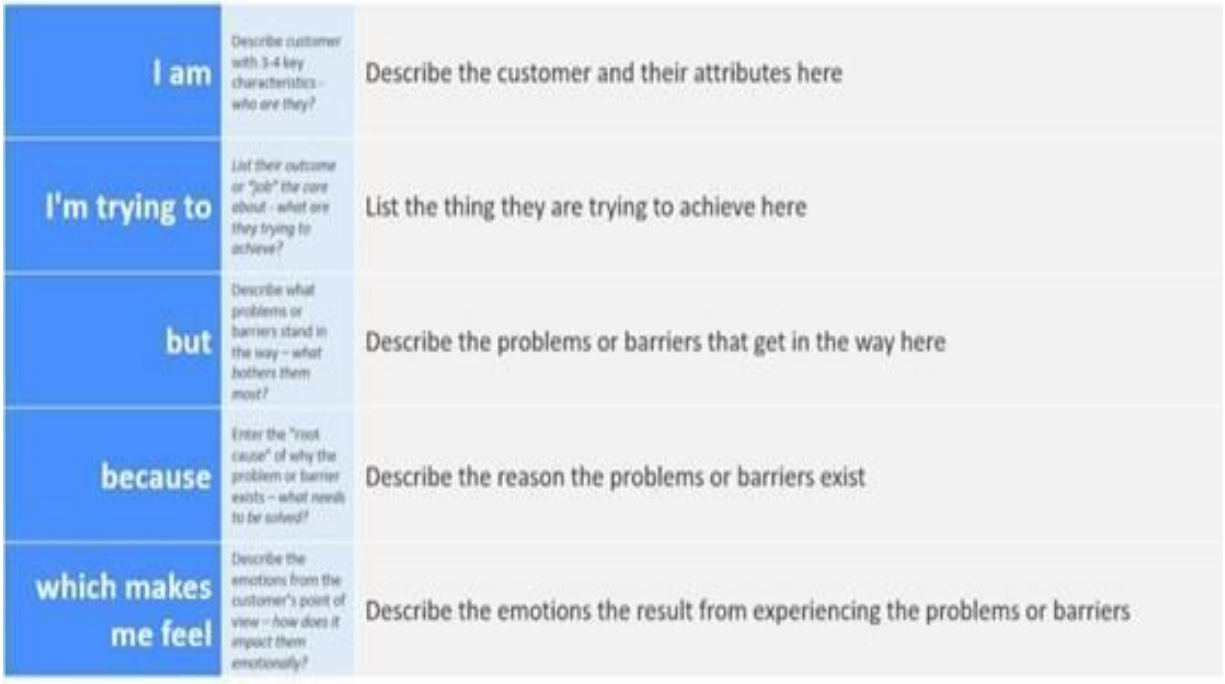
3. Fake News Detection and Tracker System. Aditi Raut , Aleena Marium, Ruchika Navandar, Shraddha Chitte, Harsha sonune, Kedarnath Dixit-2021

4. Analysis on Fake News Detection using Machine Learning. Sisira Joju, Mr. Praveen S Kammath-2021
5. Fake News Detection in Political Artefacts using Machine Leaarning. Swapnil Roy1, Sunny Khatri2, Merin Meleet-2020

2.3 **Problem Statement Definition:**

Create a problem statement to understand your customer's point of view. The Customer Problem Statement template helps you focus on what matters to create experiences people will love.

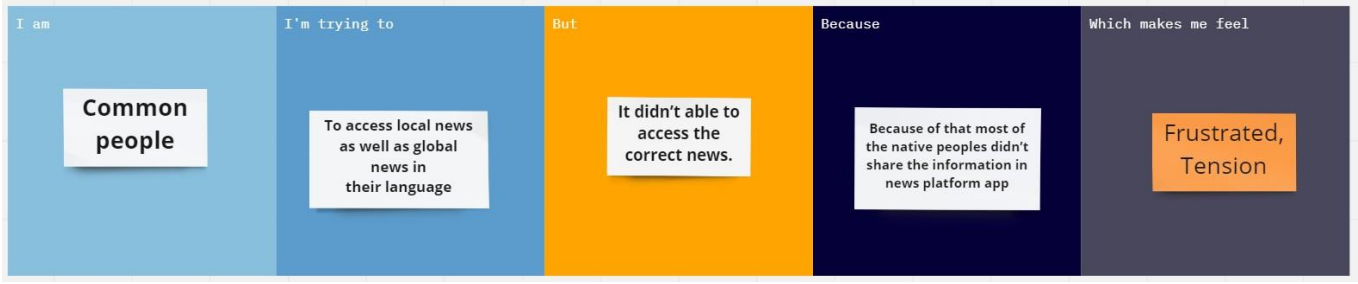
A well-articulated customer problem statement allows you and your team to find the ideal solution for the challenges your customers face. Throughout the process, you'll also be able to empathize with your customers, which helps you better understand how they perceive your product or service.



Problem Statement (PS)	**I am (Customer)**	**I'm trying to**	**But**	**Because**	**Which makes me feel**
PS-1	Busy worker	To get the news anywhere andanytime	It Shows fake and irrelevant news	It Didn't able to access correct NEWS Platform Resources	Confussed
PS-2	Common people	To access local news as well as globalnews in their language	It didn't able to access the correct news.	Because of that most of the native peoples didn't share the information in news platform app	Frustrated,Tension

Problem Statement-1:

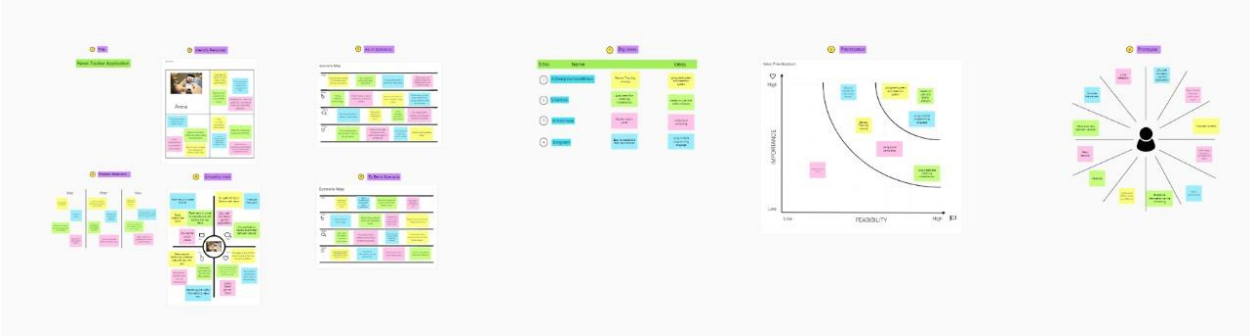
Problem Statement-2:

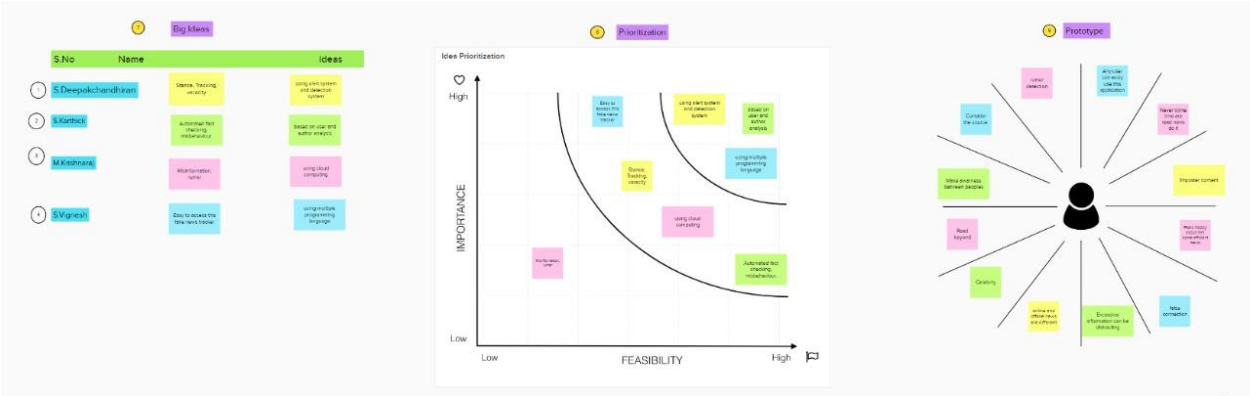


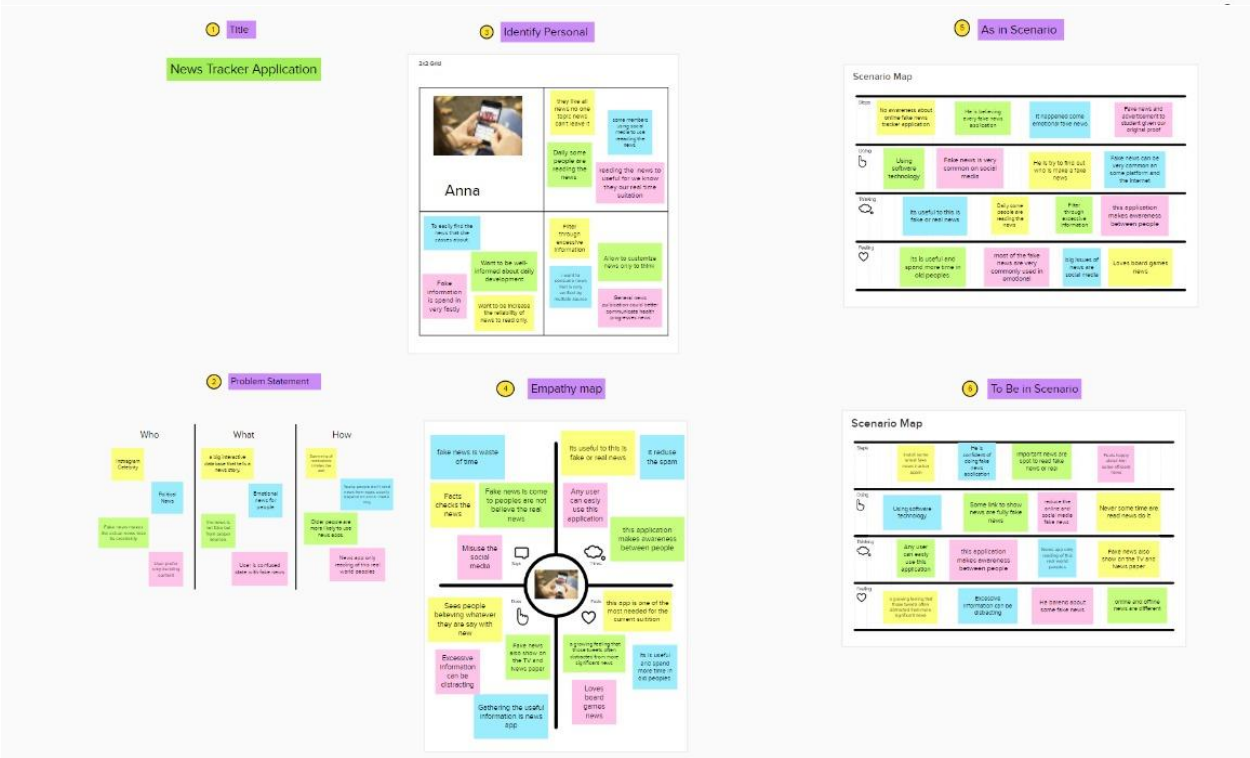
CHAPTER 3

IDEATION & PROPOSED SOLUTION

3.1 **Empathy Map Canvas:**







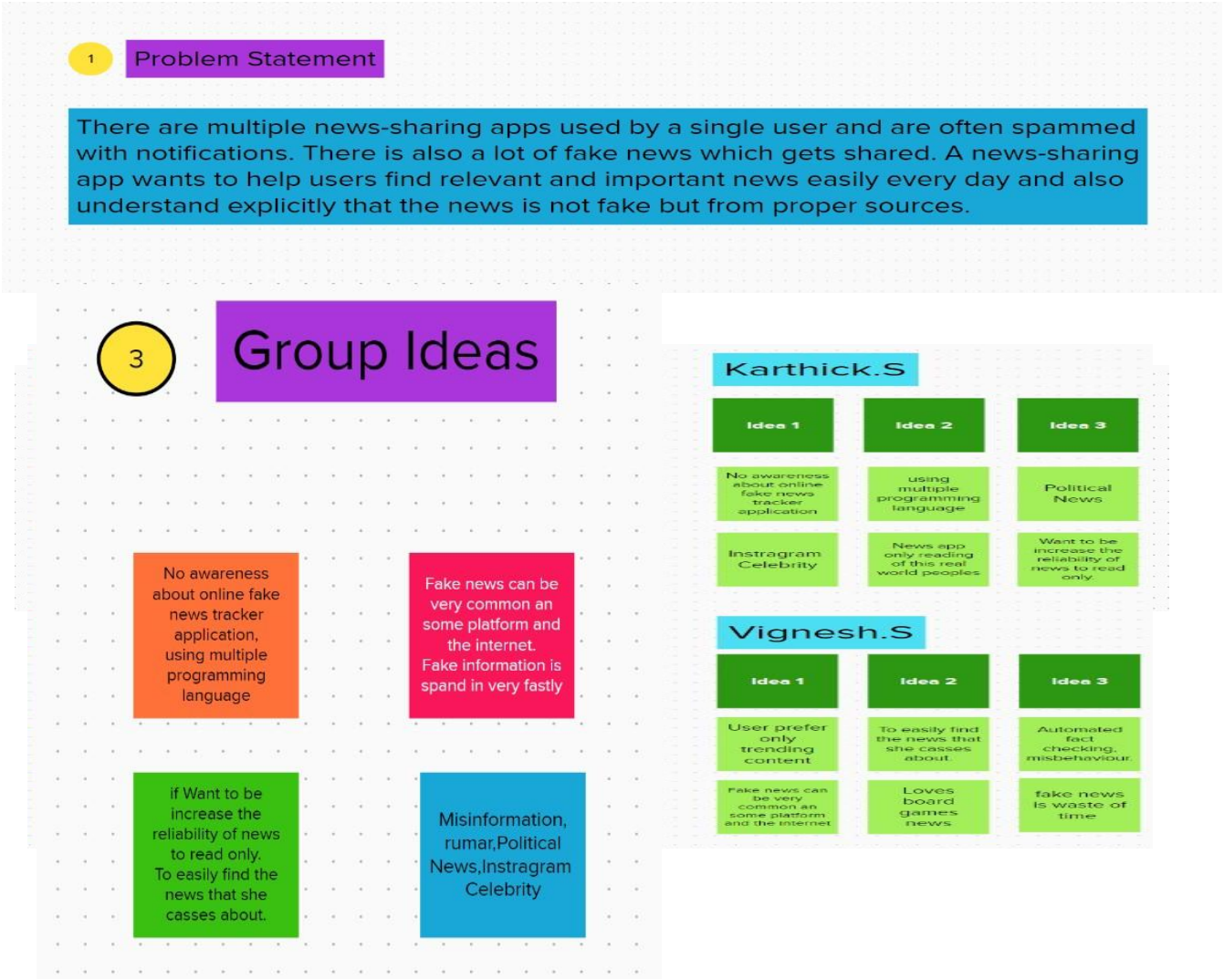
3.2 **Ideation & Brainstorming:**

Ideation:

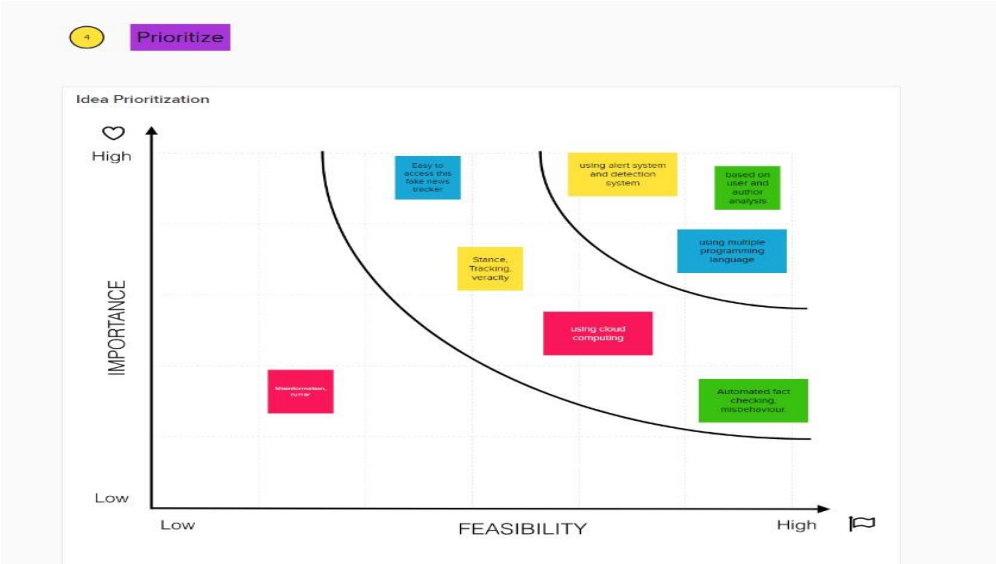
There are multiple news-sharing apps used by a single user and are often spammed with notifications. There is also a lot of fake news which gets shared. A news-sharing app wants to help users find relevant and important news easily every day and also understand explicitly that the news is not fake but from proper sources.

1. **Problem Statement**

2.Brainstorm, Idea Listing and Grouping



3.Idea Prioritization



3.3**Proposed Solution:**

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	There are multiple news sharing apps used by single user and are often spammed with notifications. There is also a lot of fake news which gets shared. A news sharing app wants to help users find relevant and important news easily every day and also understand explicitly that the news is not fake but from proper sources.
2.	Idea / Solution description	An application needs to be developed in which users can read news whenever they want and they will be able to customize their area of interest. So that they will be notified, if any new news is updated in their interested areas.
3.	Novelty / Uniqueness	1)Creating a user experience that is appealing to clients and delivering news headlines with more details. 2)User will have the option to select what varieties of news he would like to see. 3) Creating an attractive user interface for the customers and delivering important News Headlines with more details.
4.	Social Impact / Customer Satisfaction	Users held a positive attitude toward their mobile news reading experience and they described news they read on mobile apps with such words as “interesting”, “instant”, “positive”, “profound”, and “ironic”. Results of the second stage confirmed our hypothesis that content originality and user experience both had positive impacts on user satisfaction.
5.	Business Model (Revenue Model)	Advertising revenue, Online traffic, Digital content delivery, Continuous breaking news, Interactive networking
6.	Scalability of the Solution	Since the web application is deployed on IBM cloud, it can handle multiple users at a time. The user will go through a seamless experience and it enables them to view the news accordingto their interests and choices. Users from all age category can use the application and the news can also be filtered according to their age.

3.4 Problem Solution fit:

1.CUSTOMER SEGMENT:	6.CUSTOMER CONSTRAINT:	5.AVAILABLE SOLUTION:
• NEWS reader • Everyone who follows thenews daily	Knowledge to access the app. They need to have goodinternet connection. They need to have a PC ormobile phone.	Instesd of using conventionalmode of reading we can create anews app. In this app the many news inglobal level are covered and having a option of searching particular news.

2.JOBS TO BE DONE/PROBLEM:	9.PROBLEM ROOT CAUSE:	7.BEHAVIOUR:
• Reading unwanted and irrelevant and repeated content • Bad userinterface • Searching related news • Misleading Ads and unorganized contents • user unable to customize news content • Using internet for previously watched content • Forced notifications and ads • Providing dark mode	In a busy world, people do not havetime for reading newspaper and watching news channels.	The users have rights to access thenews they don't have permission to access personal details.

3.TRIGGERS:

We had suufcient features to reach the app. This app is a daily use application. It is safe and secure.

4.EMOTION BEFORE/AFTER:

Before: People did not use this app because they did not have anyawareness.

After: Now People are known about these app and usage of thisapp becomes high.

10.YOUR SOLUTION:

This application is used to show updated news on time to time.This willshow a news in reels format where usercan enjoy viewing the news in different manner. This application will avoid most of the advertisement and we sure it wont disturb our customer.user can also follow their interested news page in our application.The user interface in our application will be good and news feeds depend on user interest will be updated on time.

8.CHANNELS OF BEHAVIOUR:

Online

What kind of actions do customerstake online? Extract online channelsfrom #7

Continuous updations of news. Generate notification of list of news which the user need to follow user can stack the news.

Offline

What kind of actions do customers take offline? Extract offline channelsfrom #7 and use them for customer development.

user can see the downloaded news

CHAPTER 4

REQUIREMENT ANALYSIS

4.1 **Functional requirement:**

Following are the functional requirements of the proposed solution.

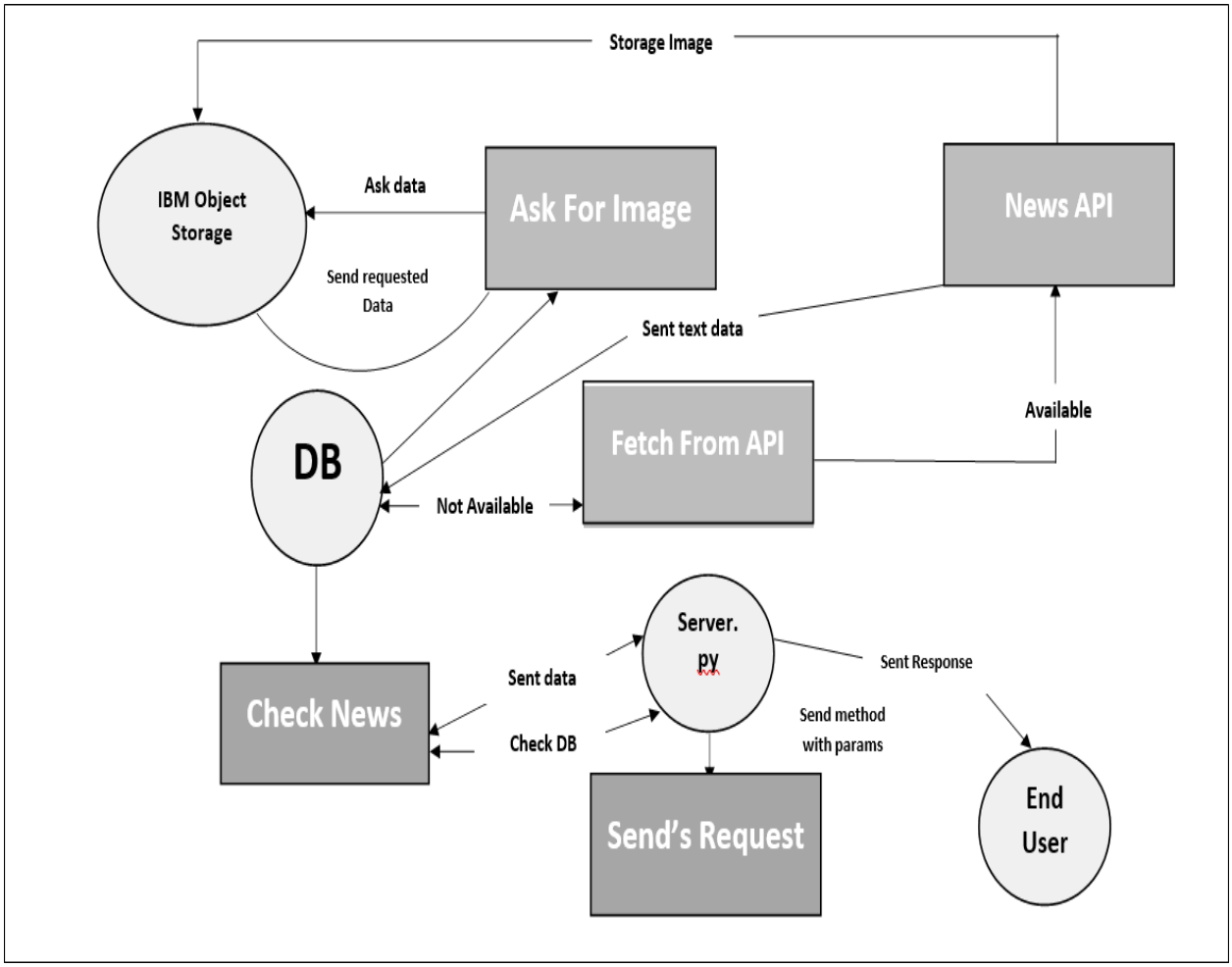
FR No.	**Functional Requirement (Epic)**	**Sub Requirement (Story / Sub-Task)**
FR-1	User Installation	User can install the app from google play store, App store and from website
FR-2	User Registration	Registration through Form Registration through Gmail Registration through LinkedIN
FR-3	User Confirmation	Confirmation via EmailConfirmation via OTP
FR-4	User login	User should login the app with the User's name or email and password.
FR-5	User Information	User Name User Email Id User Contact No User Address
FR-6	User interaction	Done through user interface between client and server View the related news by subscripted or requested page

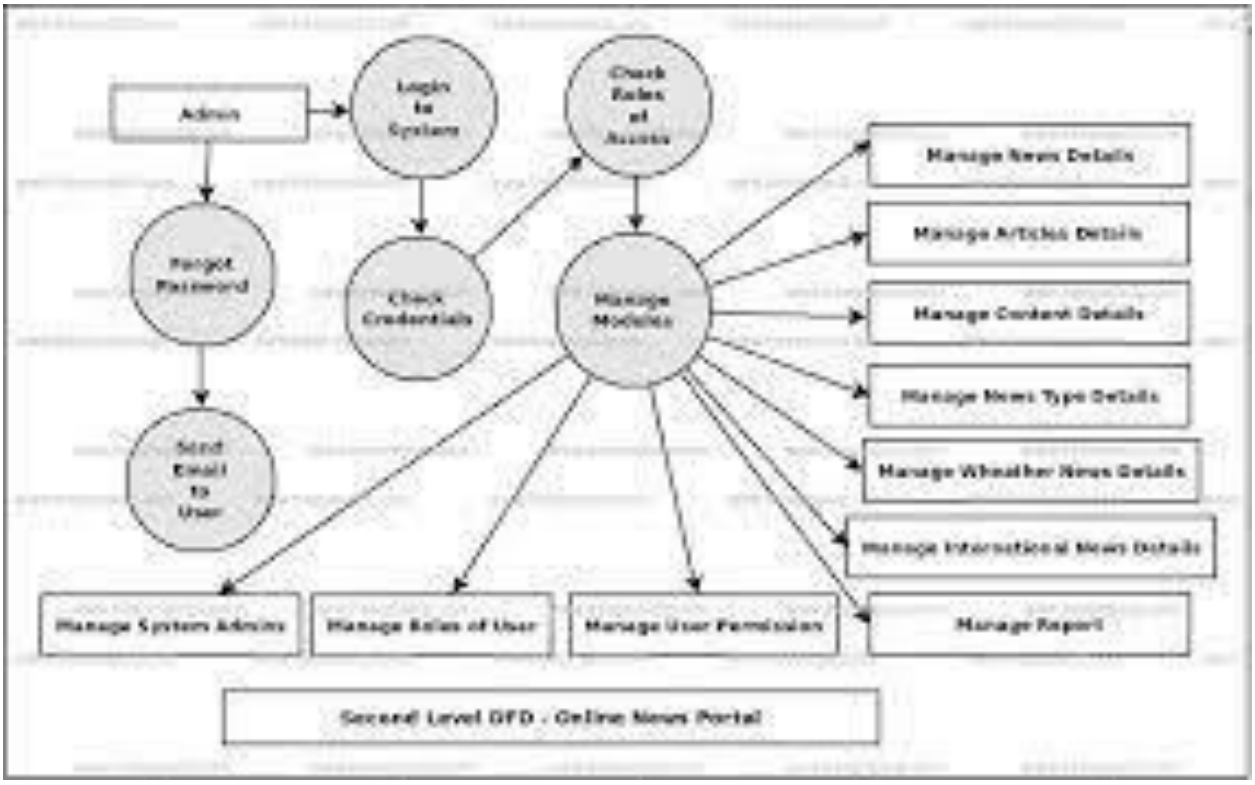
4.2 **Non-Functional requirements:**

FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	End users can receive push updates for new contenton a site by subscribing to the site's news feed
NFR-2	**Security**	This app is secured app, where users information is encrypted properly
NFR-3	**Reliability**	How often does the system experience critical failures? How much time does it take to fix the issue when it arises ?And how is user availability time compared to downtime?
NFR-4	**Performance**	Performance is the core non-functional requirements no system can do without.It defines how fast a software system or a particular piece of itresponds to certain users actions under a certain workload. In most cases, this metric explains how long a user must wait before the target operation

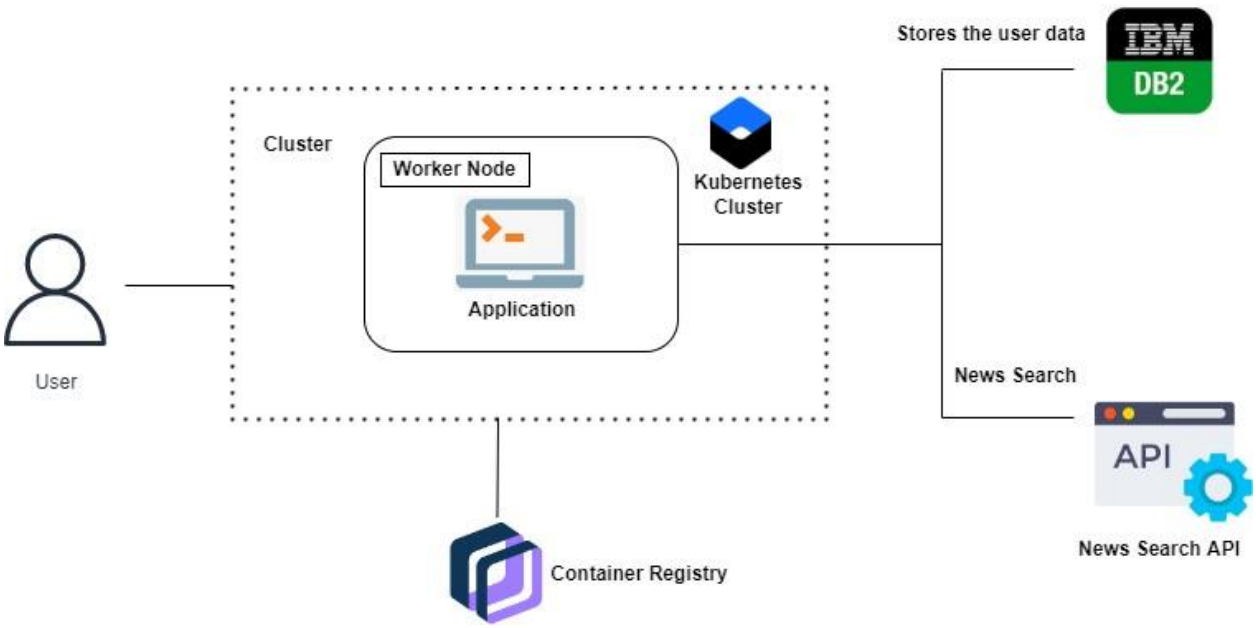
CHAPTER 5
PROJECT DESIGN

5.1 **Data Flow Diagrams:**





5.2 **Solution & Technical Architecture:**



5.3 **User Stories:**

Use the below template to list all the user stories for the product.

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming mypassword.	I can access my account / dashboard	High	Sprint-1
		USN-2	As a user, I will receive confirmation email once Ihave registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail	I can register & access the dashboard with Gmail Login	Medium	Sprint-2
	Login	USN-5	As a user, I can log into the application by entering email & password	I can view all types of information through this application	High	Sprint-1
	Dashboard	USN-1	As a user, I can log into the application and look into my dashboard	I can Look into My Dashboard After my login	Low	Sprint-1
		USN-2	As a user, I can log into the application and update my personal data	can View the personal data which can be updated by the user	Low	Sprint-2
Customer (W	Login	USN-1	As a user, I can register for the application by entering my email, password, and	I can access my account /	Hi	Sprint-1

eb user)			confirming my password through web applications such as Chrome, Firefox, brave, Operamini etc.	dashboard	gh	
	Search Bar	USN-2	User searches for news based on their own interest.	I can view the related news and can watch videos.	High	Sprint-2
		USN-3	The news can be viewed that is appearing onthe dashboard	On the dashboard. I shall click on the news wanted and can open it	Low	Sprint-2
Customer Care Executive	Dashboard	USN-1	As a user, I can Report to the customer service about the error or doubt of the application by calling to the customer service which is provided in the application help box	I can report to the customer service if I am facing an issue or I didn't know anything about the application	High	Sprint-1
Administrator	Server	USN-1	Provides correct news available from the database.	Provide the correct news from the database and fake news will be rejected	High	Sprint-1
		USN-2	Provide live news with video and audio contents.	I can get the news in which I am interested	Medium	Sprint-1

CHAPTER 6
PROJECT PLANNING & SCHEDULING

6.1 **Sprint Planning & Estimation:**

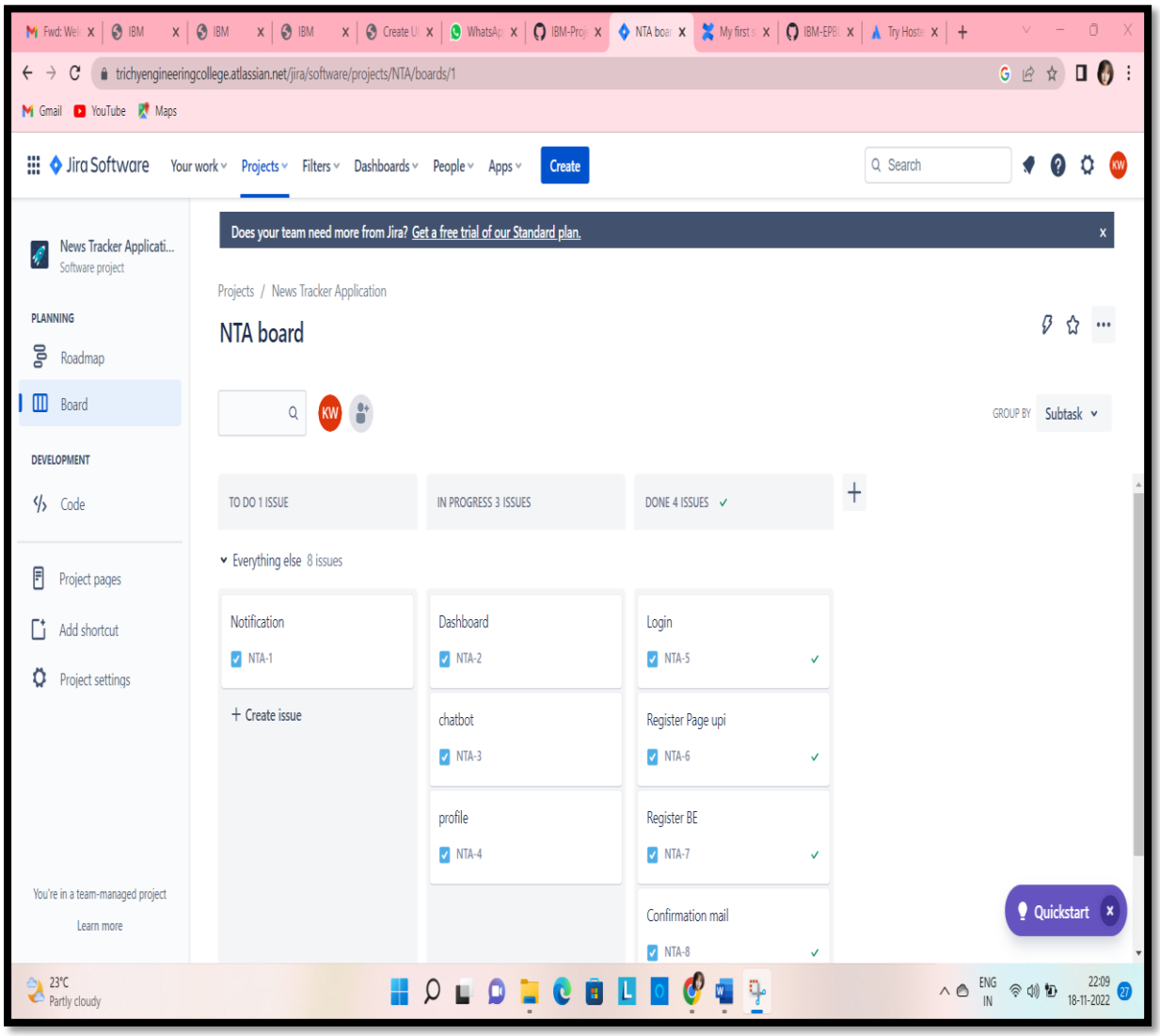
Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	2	High	Karthick S Krishnaraj M
Sprint-1		USN-2	As a user, I will receive confirmation emailonce I have registered for the application	1	High	DeepakchandhiranS Vignesh S
Sprint-2		USN-3	As a user, I can register for the applicationthrough Gmail	2	Low	Karthick SVignesh S
Sprint-1		USN-4	As a user, I can register for the applicationthrough Gmail	2	Medium	Krishnaraj MVignesh S
Sprint-1	Login	USN-5	As a user, I can log into the application byentering email & password	1	High	DeepakchandhiranS Vignesh S
	Dashboard	USN-6	As a user, I can enter into my Dashboardwhere I can navigate to different pages.	1	Medium	DeepakchandhiranS Karthick S
Sprint-3	Database	USN-7	As a user, I can store my login information so that whenever I visit the same page ,I will be redirected to the dashboard.	2	Low	Karthick S Krishnaraj M
Sprint-3		USN-7	As a user, I can log out my account in setting.	3	Medium	DeepakchandhiranS Vignesh S

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-4		USN-8	As a user, I can update my interests andchoice in account settings.	10	Medium	Vignesh S Karthick S
Sprint-4	Chat bot / Query	USN-9	Solve issues brought up by client	5	Medium	Karthick S Deepakchandhiran S
Sprint-4		USN-10	Roll out updates and bug fixes	5	High	Vignesh S Krishnaraj M

6.2 **Sprint Delivery Schedule:**

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	29 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	05 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	20	12 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	19 Nov 2022

6.3 **Reports from JIRA:**



CHAPTER 7
CODING & SOLUTIONING

(Explain the features added in the project along with code)

7.1 Feature 1

Login:

```
{% extends 'base.html' %}
{% block content %}
<div class="container">
   <h1 class="form display-3 text-center">Login</h1>
   <div class="text-center text-light">
      <p class="display-5 alert alert-success" id="login-q">Don't have an account? <a href="/register"
            class="alert-link">Register here</a></p>
      <div class="row justify-content-center">
         <div class="col-md-8">
            <form method="POST">
               {{ form.hidden_tag() }}
               {% for field in form if field.widget.input_type != 'hidden' %}
               <p>
               <div class="form-group">
                  {{field.label}}
                  {{field(class_="form-control")}}
                  {%for err in field.errors%}
                  <small class="form-text text-danger">
                     {{err}}
                  </small>
                  {% endfor %}
               </div>
               </p>
               {% endfor %}
               <button class="submit-b btn btn-success btn-large" type="Submit">Login</button>
            </form>
         </div>
      </div>
```

```
    </div>
</div>
{% endblock %}
```

Register:

```
{% extends 'base.html' %}
{% block content %}
<div class="container">
    <h1 class="display-3 text-center">Register</h1>
    <div class="text-center text-light">
        <div class="row justify-content-center ">
            <div class="col-md-8">
                <form method="POST" novalidate>
                    {{ form.hidden_tag() }}
                    {% for field in form
                    if field.widget.input_type != 'hidden' %}
                    <p>
                    <div class="form-group">
                        {{field.label}}
                        {{field(class_="form-control")}}
                        {%for err in field.errors%}
                        <small class="form-text text-danger">
                            {{err}}
                        </small>
                        {% endfor %}
                    </div>
                    </p>
                    {% endfor %}
                    <button class="submit-b btn btn-success btn-large" type="Submit">Register</button>
                </form>
            </div>
        </div>
    </div>
</div>
```

{% endblock %}

Base:

```
<!DOCTYPE html>
<html lang="en ">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.10.2/dist/umd/popper.min.js"
    integrity="sha384-
7+zCNj/IqJ95wo16oMtfsKbZ9ccEh31eOz1HGyDuCQ6wgnyJNSYdrPa03rtR1zdB"
    crossorigin="anonymous"></script>
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css"
    integrity="sha384-
TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M0jlfIDPvg6uqKI2xXr2"
crossorigin="anonymous">
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.min.js"
    integrity="sha384-
QJHtvGhmr9XOIpI6YVutG+2QOK9T+ZnN4kzFN1RtK3zEFEIsxhlmWl5/YESvpZ13"
    crossorigin="anonymous"></script>
  <link rel="stylesheet" href="../static/app.css">
 <!-- Added absolute path to fix rendering problems when in nested route. check to ensure works on
deployment -->
  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<title>NewsTracker</title>
  <link href="https://fonts.googleapis.com/css?family=Abel" rel="stylesheet">
</head>
<body>
  <nav class="navbar border-bottom border-secondary fixed-top navbar-expand navbar-dark bg-
dark">
    <a href="/" class="navbar-brand">NewsTracker</a>
    <div class="collapse navbar-collapse" id="navbarNav">
      <div class="navbar-nav">
        {% if g.user %}
```

```
      <a class="nav-item nav-link " href="/headlines">Headlines <span class="sr-
only">(current)</span></a>
      <a class="nav-item nav-link" href="/user/saved">My Stories</a>
      <a class="nav-item nav-link" href="/search">Detailed Search</a>
      <li class="nav-item dropdown">
       <a class="nav-link dropdown-toggle" href="#" id="navbarDarkDropdownMenuLink"
role="button"
         data-bs-toggle="dropdown" aria-expanded="false">
         My Queries
       </a>
       <ul class="dropdown-menu dropdown-menu-dark" aria-
labelledby="navbarDarkDropdownMenuLink" id="dropdown">
         {% if g.user.queries%}
         {%for query in g.user.queries%}
         <li class="dropdown-item" style="
         padding-right: 10px"><a class="query-link" href="/search/{{query.id}}">{{query.name}}
          </a>
          <span data-query="{{query.id}}" class="close">&times;</span>
          <!-- <button type="button" class="btn btn-outline-success btn-sm">Make Default</button> -
->
         </li>
         {%endfor%}
         {% else %}
         <span class="dropdown-item">You currently have no saved queries.</span>
         {% endif %}
       </ul>
      </li>
      <a class="nav-item nav-link" href="/logout">Logout</a>
      {% else %}
      <a class="nav-item nav-link" href="/headlines">Headlines</a>
      <a class="nav-item nav-link" href="/login">Login</a>
      <a class="nav-item nav-link" href="/register">Register</a>
      {% endif %}
    </div>
```

```
    </div>
    <!-- <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarNav">
        <span class="navbar-toggler-icon"></span>
      </button> -->
    <div>
      <form action="/search/simple" method="GET" class="form-inline" id="search">
        <input name="search" class="form-control mr-sm-2" type="search" placeholder="Search" aria-
label="Search">
        <button class="btn btn-outline-success badge-pill my-2 my-sm-0 badge-pill"
type="submit">Search</button>
      </form>
    </div>
  </nav>
  <div id="indentation">
  </div>
  {%with messages = get_flashed_messages(with_categories=true) %}
  {% if messages %}
  <section class="messages">
    {% for category, msg in get_flashed_messages(with_categories=true)%}
    <span class="{{category}}">{{msg}}</span>
    {% endfor %}
  </section>
  {% endif %}
  {% endwith %}
{% block content %}
  <!-- content goes here -->
  {% endblock %}
  <script src="/static/newstracker.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"
    integrity="sha384-
Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo"
    crossorigin="anonymous"></script>
  <script src="https://code.jquery.com/jquery-3.4.1.slim.min.js"
```

```
    integrity="sha384-
J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n"
crossorigin="anonymous">

 </script>

  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"

    integrity="sha384-
wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6"

    crossorigin="anonymous"></script>

</body>

</html>
```

7.2 Feature 2

Homepage:

```
{% extends 'base.html' %}

{% block content %}

<script src="/static/newstracker.js"></script>

<script>

  homeAdd(arg);

</script>

<div id="home" class="container">

  <div id="row" class="row">

 <div class="col-md-7 col-12">

     <h3 class="form display-5 text-center">What can I do with NewsTracker?</h3>

     <div id="info-container" class="container">

   <div class="row">

        <div class="heading">

          <h4 class="form display-5 text-center">Non-Users</h4>

          <p>You can click the "Login as Demo User" button on the right-hand column check out
NewsTracker's features with a dummy account, but you already have access to several features.
Here are some things you can do instantly, without even creating an account.</p>

        </div>

        <div class="announce col-lg">

          <div class="card">
```

```
          <div class="card-body">
            <h5 class="card-title">Get Headlines</h5>
            <p class="card-text">Simply click "Headlines" to browse the latest news of the day featuring
all
              categories.</p>
          </div>
        </div>
      </div>
      <div class="announce col-md col-lg">
        <div class="card" id="card2">
          <div class="card-body">
            <h5 class="card-title">Search by Keyword</h5>
            <p class="card-text">Use the nav-bar to search by keyword to get up-to-date results based
off of
              whatever you're looking for.</p>
          </div>
        </div>
      </div>
      <div class="announce col-lg">
        <div class="card">
          <div class="card-body">
            <h5 class="card-title">Browse by Category</h5>
            <p class="card-text">Click the category title of any category on the right to view headlines
from each
              category, or click on the story headline as it slides through to view each story directly.</p>
          </div>
        </div>
      </div>
    </div>
<div class="row">
      <div class="heading">
        <h4>Users</h4>
```

Through NewsTracker, Registered Users have access to a wide variety of tools that may assist them search

for and organize various information.

<h5 class="card-title">Detailed Search</h5>

Use our Detailed Search feature to further customize your search query by source, date-range, language, and more! You can save this search query to be your default news feed each time

you log in.

<h5 class="card-title">Sentiment Analysis Tools</h5>

NewsTracker uses Python’s NLTK Library and Web-Scraping techniques to perform Sentimental Analysis on each article that we receive from out news-api at your request! Polarity and Subjectivity scores may be extracted from each article, or used to filter search results.

<h5 class="card-title">Saved Stories</h5>

Never lose track of the information you read by using our “Saved Stories” feature to keep all of your most important stories in one place.

```
      </div>
     </div>
    </div>
    <div class="row">
     <div class="heading">
      <h4>Upcoming Features</h4>
      <p>NewsTracker is an ongoing project that we will continuously be updating in order to give myself and other users ability to access and organize our world’s latest trends, insights, and information. Here are some upcoming features from our roadmap.</p>
     </div>
     <div class="announce col">
      <div class="card">
       <div class="card-body">
        <h5 class="card-title">Amplified User Features</h5>
        <p class="card-text">Users will be able to better organize their saved content by being able to categorize them into separate folders named by their choosing, and will be able to write notes for each individual saved story.</p>
       </div>
      </div>
     </div>
     <div class="announce col">
      <div class="card">
       <div class="card-body">
        <h5 class="card-title">Integration of Twitter API</h5>
        <p class="card-text">Not only will the integration of the Twitter API be useful for those who wish to digest their information in more bite-sized pieces, it will also allow users to receive more precise and recent Sentiment Analysis Data. Automatic, visual representations of such insights should be useful to investors, marketers, etc.</p>
       </div>
      </div>
     </div>
    </div>
   </div>
```

```
<div id="slideshows" class="col-md-5 col-sm">
    <div class="container" id="slide-counter">
      <div class="row" style="justify-content: center;">
        {% if no_user%}
        <div class="demo-user">
          <label for="demo">
            <h5 class="card-title">Try NewsTracker with our dummy account:</h5>
          </label>
          <form id="save" class="sa-button col-sm my-auto col-6" action='/login/demo' , method="POST">
            <input type="submit" value="Login as Demo User" id="demo" class="btn btn-dark btn-md badge-pill">
          </form>
        </div>
        {% endif %}
        {%for obj in data%}
        <div class="container">
          <h3 class="form display-5 text-center"><a href="/headlines/{{obj.name}}" class="category">{{obj.name}}</a>
          </h3>
          <div class="row">
            <div class="col-10" id="slideshow">
              <div class="carousel slide" data-ride="carousel">
                <!-- <ol class="carousel-indicators">
                  <li data-target="#carouselExampleIndicators" data-slide-to="0" class="active"></li>
                  <li data-target="#carouselExampleIndicators" data-slide-to="1"></li>
                  <li data-target="#carouselExampleIndicators" data-slide-to="2"></li>
                </ol> -->
                <!-- https://getbootstrap.com/docs/4.0/components/carousel/ -->
                <div class="carousel-inner">
                  <a href="{{obj.top_story.url}}">
                    <div class="carousel-item active" data-interval="3500">
```

```
            <img class="slide d-block w-100" src="{{obj.top_story.image}}" alt="First slide" ;>
            <div class="carousel-caption d-md-block">
              <span class="caption">{{obj.top_story.headline}}</span>
            </div>
          </a>
        </div>
        {%for story in obj.results%}
        <div class="carousel-item" data-interval="4000">
          <a href="{{story.url}}">
            <img class="d-block w-100" src="{{story.image}}" alt="Third slide">
            <div class="carousel-caption d-md-block">
              <span class="caption">{{story.headline}}</span>
            </div>
          </a>
        </div>
        {%endfor%}
      </div>
      <!-- <a class="carousel-control-prev" role="button" data-slide="prev">
        <span class="carousel-control-prev-icon" aria-hidden="true"></span>
        <span class="sr-only">Previous</span>
      </a>
      <a class="carousel-control-next" role="button" data-slide="next">
        <span class="carousel-control-next-icon" aria-hidden="true"></span>
        <span class="sr-only">Next</span>
      </a> -->
    </div>
  </div>
 </div>
 {%endfor%}
</div>
```

```
  </div>
 </div>
</div>
</div>
{% endblock %}
```

Search:

```
{% extends 'base.html' %}
{% block content %}
<div class="container">
  <h1 class="display-3 text-center">Search</h1>
  <div class="text-center text-light">
    <div class="row justify-content-center">
      <div class="col-md-8">
        <form method="POST" novalidate>
          {{ form.hidden_tag() }}
          {% for field in form
          if field.widget.input_type != 'hidden' %}
          <p>
          <div class="form-group">
            {{field.label}}
            {{field(class_="form-control")}}
            {%for err in field.errors%}
            <small class="form-text text-danger">
              {{err}}
            </small>
            {% endfor %}
          </div>
          </p>
          {% endfor %}
          <button class="btn btn-success badge-pill btn-large" type="Submit">Search</button>
```

```
            </form>
          </div>
        </div>
      </div>
    </div>
</div>
{% endblock %}
```

Show_stories:

```
{% extends 'base.html' %}
{% block content %}
{% if not results%}
<span class="no-stories">We weren't able to find anything. Try checking your spelling or widening your search query.
</span>
{% endif %}
<section id="stories">
   {% for story in results%}
   <div class="container">
      <ul>
         <div class="row justify-content-center">
            <div class="card my-4">
               <div class="card-body text-center">
                  <div class="row media">
                     {% if story.pol%}
                     <form data-story="{{story.id}}" class="sa-button col-sm my-auto order1 col-6 " id="left-b"
                        action='/feed/{{story.id}}/polarity' , method="POST">
                        <input type="submit" value="{{story.pol}}" class="btn btn-dark btn-sm badge-pill">
                     </form>
                     {% else %}
                     <form data-story="{{story.id}}" class="sa-button col-sm my-auto order1 col-6 " id="left-b"
```

```
                    action='/story/{{story.id}}/polarity' , method="POST">
                    <input type="submit" value="Get Polarity" class="btn btn-dark btn-sm badge-pill">
                </form>
                {% endif %}
<img class="text-center col-md-6 col-12" id="image" src="{{story.image}}"
                    alt="{{story.headline}}" style="width: 300px; height: 225px; object-fit: contain;" />
{% if story.sub%}
                <form data-story="{{story.id}}" class="sa-button col-sm my-auto col-6" id="right-b"
                    action='/feed/{{story.id}}/subjectivity' action='/sacalls/{{story.id}}/subjectivity' ,
                    method="POST">
                    <input type="submit" value="{{story.sub}}" class="btn btn-dark btn-sm badge-
pill">
                </form>
                {% else %}
                <form data-story="{{story.id}}" class="sa-button col-sm my-auto col-6" id="right-b"
                    action='/story/{{story.id}}/subjectivity' ,method="POST">
                    <input type="submit" value="Get Subjectivity" class="btn btn-dark btn-sm badge-
pill">
                </form>
                {% endif %}
            </div>
 <h4 class="card-title text-center"> <a class="headline" href="/story/{{story.id}}/open">
                    {{story.headline}}</a></h4>
            <h5 class="card-subtitle mb-2 text-center blockquote-footer"
id="date">{{story.published_at}}
                </li>
                <p class="card-text text-center" id="content">{{story.content}}</p>
                <form action='/story/{{story.id}}/save_story' , method="POST" id="save">
                    <input type="submit" value="Save Story" class="btn btn-dark btn-sm badge-pill">
                </form>
            </div>
        </div>
```

```
        </div>
      </ul>
    </div>
    {%endfor%}
</section>
{% endblock %}
```

User:

```
{% extends 'base.html' %}
{% block content %}
<!-- <script src="/static/newstracker.js"></script> -->
{% if is_empty == True%}
<span class="no-stories">It looks like you don't have any stories saved. To find a story to save, you
    can either click the search button ontop of the navigation bar to enter a specific search,
    or click the home page to see today's top headlines!
</span>
{% endif %}
{% for story in user.saved_stories%}
<section id="stories">
    <div class="container">
        <ul>
            <div class="row justify-content-center">
                <div class="card my-4 ">
                    <div class="card-body text-center">
                        <div class="row media">
                            {% if story.pol%}
                            <form data-story="{{story.id}}" class="sa-button col-sm my-auto order1 col-6"
id="left-b"
                                action='/{{story.id}}/polarity' , method="POST">
                                <input type="submit" value="{{story.pol}}" class="btn btn-dark btn-sm badge-pill">
                            </form>
```

```
 {% else %}
                    <form data-story="{{story.id}}" class="sa-button col-sm my-auto order1 col-6"
id="left-b"
                      action='/story/{{story.id}}/polarity' , method="POST">
                      <input type="submit" value="Get Polarity" class="btn btn-dark btn-sm badge-pill">
                    </form>
                    {% endif %}
<img class="text-center col-md-6 col-12" id="image" src="{{story.image}}"
                      alt="{{story.headline}}" style="width: 300px; height: 225px; object-fit: contain;" />

                    {% if story.sub%}
                    <form data-story="{{story.id}}" class="sa-button col-sm my-auto col-6" id="right-b"
                      action='/{{story.id}}/subjectivity' action='/sacalls/{{story.id}}/subjectivity' ,
                      method="POST">
                      <input type="submit" value="{{story.sub}}" class="btn btn-dark btn-sm badge-
pill">
                    </form>
{% else %}
                    <form data-story="{{story.id}}" class="sa-button col-sm my-auto col-6" id="right-b"
                      action='/story/{{story.id}}/subjectivity' ,method="POST">
                  <input type="submit" value="Get Subjectivity" class="btn btn-dark btn-sm badge-pill">
                    </form>
                    {% endif %}
                  </div>
<h4 class="card-title text-center">
                    <a class="headline" href="story/{{story.id}}/open"> {{story.headline}}</a>
                  </h4>
                  <li class="card-subtitle mb-2 text-center blockquote-footer"
id="date">{{story.published_at}}
                  </li>
                  <p class="card-text text-center" id="content">{{story.content}}</p>
                  <form action='/story/{{story.id}}/remove' , method="POST" id="save">
```

```
                    <input type="submit" value="Remove Story" class="btn btn-danger btn-sm badge-
pill">
                </form>
              </div>
            </div>
          </div>
        </ul>
    </div>
</section>
{%endfor%}
{% endblock %}
```

CHAPTER 8

TESTING

8.1 Test Cases:

Test case	feature	component	Test scenario	Expected result	Actual result	status	comments	bug
Sign in	**Functional**	**Login page**	**Verify user can see the sign in option**	**can visible**	**Yes visible**	**pass**	**successful**	-
Sign up	Functional	Login page	Verify user has the option to sign up	Can visible	Yes visible	pass	Successful	-

News feeds	Functional	Home page	Verify user can get the news	News will be feed to the app	Feeds are appearing	yes	successful	-
Types of news available in the categories options	Functional	**Dashboard**	Types of news available	Weather, Sport, Economy.	Hover buttons will be shown.	Yes	successful	-

8.2 User Acceptance Testing:

1.Purpose of Document

The purpose of this document is to briefly explain the test coverage and open issues of the NEWS TRACKER APPLICATION project at the time of the release to User Acceptance Testing (UAT).

2.Defect Analysis

This report shows the number of resolved or closed bugs at each severity level, and how theywere resolved

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	10	4	2	3	19
Duplicate	1	0	3	0	4
External	2	3	0	1	6
Fixed	11	2	4	20	37
Not Reproduced	0	0	1	0	1
Skipped	0	0	1	1	2
Won't Fix	0	5	2	1	8
Totals	24	14	13	26	77

3.Test Case Analysis

This report shows the number of test cases that have passed, failed, and untested

Section	Total Cases	Not Tested	Fail	Pass
Print Engine	6	0	0	6
Client Application	25	0	0	20
Security	2	0	0	2
Outsource Shipping	3	0	0	3
Exception Reporting	7	0	0	7
Final Report Output	4	0	0	4
Version Control	2	0	0	2

CHAPTER 9
RESULT

News tracker application using cloud is developed and this project is executed successfully at the level of completed.

9.1 Performance Metrics:

- Apdex Scores
- Average Response Time
- Garbage Collection
- UX and performance metrics
- Engagement KPI metrics
- Revenue metrics

CHAPTER 10

ADVANTAGES & DISADVANTAGES

Advantages:-

1. In this app news is already categorization.
2. Easily accessible and portable
3. Better user experience.
4. Apps help you convert visitors to loyal readers.
5. Minute by minute updates of news
6. This app helps you to get local news updates instantly.
7. To explore and discover trending news and topics
8. News feeds for you based on your interest.

Disadvantages:-

1.Some apps demand premium subscription from user.

2. Occurance of Advertisment disturb the user.

3. Sometimes the news gives brief information.

4. Prevailance of fake and uncertain news can confuse the user lead toϖ misconception.

5. Fake news may mislead the readers.

CHAPTER 11

CONCLUSION:

We explored the feasibility of recognising patterns of news reading interactions and evaluated three adaptive interface designs for different news reader types. We show that from their interaction log, a specific user can be recognised as one of three kinds. The reader types emerging from the online survey are well defined and distinct. The evaluation of the three variant interfaces suggests that different news reader types need different user interfaces. We have demonstrated a method for monitoring users' news reading behaviour and inferring news reader type from it. In the future we will further explore the design of adaptive interfaces, in order to be in a position to demonstrate a complete adaptive mobile news framework providing automatic personalisation of news apps.

CHAPTER 12

FUTURE SCOPE:

In the future we will further explore the design of adaptive interfaces, in order to be in a position to demonstrate a complete adaptive mobile news framework providing automatic personalisation of news apps.

CHAPTER 13

APPENDIX

13.1 Source Code:

App.py:

```
# flask, config, and env imports
from dotenv import load_dotenv
load_dotenv()
from flask import Flask
app = Flask(__name__)
if app.config["ENV"] == "production":
    app.config.from_object('config.ProductionConfig')
elif app.config["ENV"] == "development":
    app.config.from_object('config.DevelopmentConfig')
else:
    app.config.from_object('config.TestingConfig')
#server-side session
CURR_USER_KEY = "curr_user"
from flask_session import Session
server_session = Session(app)
#db set-up
from models import connect_db, db
connect_db(app)
db.create_all()
#view imports
from views import sa_views, site_views, api
#todo: write app.before_request middleware for security, include here
```

Config.py:

```
import os
import redis
```

```
class Config(object):
    DEBUG = False
    TESTING = False
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    SQLALCHEMY_ECHO = True
    API_KEY = os.getenv("API_KEY")
    SECRET_KEY = os.getenv("SECRET_KEY")
    SESSION_TYPE = 'redis'
    SESSION_USE_SIGNER = True
    SESSION_PERMANENT = False
class DevelopmentConfig(Config):
    DEBUG = True
    SQLALCHEMY_DATABASE_URI = 'postgresql:///news-tracker7'
    SESSION_REDIS = redis.from_url('redis://localhost:6379')
class ProductionConfig(Config):
    SQLALCHEMY_DATABASE_URI = os.getenv("DATABASE_URL")
    SESSION_REDIS = redis.from_url(os.getenv('REDIS_URL', 'redis://localhost:6379')) #2nd argument for local db is necessary for some reason because the production version keeps getting called in development.
class TestingConfig(Config):
    TESTING = True
    SQLALCHEMY_DATABASE_URI = 'postgresql:///newstracker-test'
    SQLALCHEMY_ECHO = False
    DEBUG_TB_HOSTS= ['dont-show-debug-toolbar']
    WTF_CSRF_ENABLED = False
```

Forms.py:

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, IntegerField, SelectField, DateField, BooleanField, DateTimeField
from wtforms.validators import InputRequired, Optional, NumberRange
import datetime as dt
class RegisterForm(FlaskForm):
```

```
    '''form to add a new user. Username, password, and email required'''
    username = StringField("Username", validators=[InputRequired()])
    password = PasswordField("Password", validators=[InputRequired()])
    email = StringField("Email")
    first_name = StringField("First Name")
    last_name = StringField("Last Name")
class LoginForm(FlaskForm):
    '''form to login user with username and password'''
    username = StringField("Username", validators=[
                        InputRequired(message="Please enter a username")])
    password = PasswordField("Password", validators=[
                         InputRequired(message="Please enter a password")])
class SearchForm(FlaskForm):
    keyword = StringField("Enter a search term:", validators=[Optional()])
    source = SelectField("Choose source:", choices=[
        ('', 'All'), ('abc-news', 'ABC News'), ('al-jazeera-english',
                                    'Al Jazeera'), ('associated-press', 'Associated Press'), ('axios', 'Axios'),
        ('bbc-news', 'BBC News'), ('bloomberg', 'Bloomberg'), ('cbc-news',
         'CBC News'), ('cbs-news', 'CBS News'), ('cnn', 'CNN'), ('fox-news', 'Fox News'),
        ('google-news', 'Google News'), ('independent', 'Independent'), ('msnbc',
                  'MSNBC'), ('nbc-news', 'NBC News'), ('politico', 'Politico'),
        ('reuters', 'Reuters'), ('the-hill', 'The Hill'), ('the-huffington-post',
                          'The Huffington Post'), ('the-wall-street-journal', 'The Wall Street Journal'),
        ('the-washington-post', "The Washington Post"), ('time', 'Time'), ('usa-today', 'USA Today'),
('vice-news', 'Vice News')],
        validators=[Optional()])
quantity = IntegerField("Enter # of many articles you want returned (max 15):",
     default=10, validators=[
    NumberRange(min=1, max=15, # this is represented by the "pageSize" parameter in the API where
default and max are both set to 100
    message="Please enter a number between 1 and 10")])
date_from = DateField("Select a starting date:", validators=[Optional()])
    date_to = DateField("Up until which date?",  # API default if will be newest to oldest with no limit
```

```
    default=dt.datetime.today, validators=[Optional()])
    language = SelectField("Choose which language you want to see results from:", choices=[ # API default is all languages
        ('en', 'English'), ('es', 'Spanish'), ('fr', 'French'),
        ('ar', 'Arabic'), ('he', 'Hebrew'), ('he', 'Hebrew'), ('it', 'Italian'),
        ('nl', 'Dutch'), ('no', 'Norwegian'), ('pt', 'Portuguese'), ('ru', 'Russian'),
        ('zh', 'Chinese'), ('se', 'Swedish')],
        validators=[Optional()]) # can't find 'ud' language code as specified by api
    sort_by = SelectField("Search by:", choices=[('relevancy', 'Relevant'), ('popularity', 'Popular'), ('publishedAt', 'Recent'), # API default: publishedAt
                                        ('polarity', 'Polarity (results ordered from positive to negative)'),
                                        ('subjectivity', 'Objectivity (results ordered from objective to subjective)')], validators=[Optional()])
    default = BooleanField(
        "Make this your default search settings for your home page feed?")
    saved_query = BooleanField(
        "Would you like to add this to your saved search queries?")
    name = StringField("Give your saved search query a name (if applicable)", validators=[Optional()])
```

Helpers.py:

```
from models import db, User, Query, Story
from sent_analysis import subjectize, polarize, parse_async
from flask import session
from sqlalchemy import exc
from psycopg2.errors import UniqueViolation
"""Helper Functions"""
def order_pol():
    """Loops over session results, filters out stories with no SA results,
    then orders by polarity"""
    results = parse_async(session['results'])
    for story in results:
        score = polarize(story, parsed=True)
        if not score:
```

```
            story['pol'] = None
        else:
            story['pol'] = score['article_res']['result']
    session['results'] = [story for story in results if story['pol']]
    results = session['results']
    not_negative = [
        story for story in results if story['pol'][0] is not "-"]
    negative = [
        story for story in results if story['pol'][0] is "-"]
    ordered_neg = sorted(negative,
                         key=lambda story: story['pol'])
    ordered_not_neg = sorted(not_negative,
                             key=lambda story: story['pol'], reverse=True)
    ordered = ordered_not_neg + ordered_neg
    return ordered
def order_sub():
    """Loops over session results, filters out stories with no SA results,
    then orders by subjectivity"""
    results = parse_async(session['results'])
    for story in results:
        score = subjectize(story, parsed=True)
        if not score:
            story['sub'] = None
        else:
            story['sub'] = str(score['measure'])
    session['results'] = [story for story in results if story['sub']]
    results = session['results']
    ordered = sorted(results,
                     key=lambda story: story['sub'], reverse=True)
    return ordered
"""Conversions form SqlAlchemy object to dictionary"""
def db_query_to_dict(query):
```

```
    """Converts Query SqlAlchemy object to dict"""
    dict = {}
    dict['name'] = query.name
    dict['id'] = query.id
    dict['user_id'] = query.user_id
    dict['keyword'] = query.keyword
    dict['source'] = query.source
    dict['quantity'] = query.quantity
    dict['date_from'] = query.date_from
    dict['date_to'] = query.date_to
    dict['language'] = query.language
    dict['sa'] = query.sa
    dict['sort_by'] = query.sort_by
    if "query" in session:
        session.pop("query")
    session["query"] = dict
    return dict
def db_story_to_dict(story):
    """Converts Story SqlAlchemy object to dict"""
    dict = {}
    dict['id'] = story.id
    dict['user_id'] = story.user_id
    dict['headline'] = story.headline
    dict['source'] = story.source
    dict['content'] = story.content
    dict['author'] = story.author
    dict['description'] = story.description
    dict['url'] = story.url
    dict['image'] = story.image
    dict['published_at'] = story.published_at
    return dict
def db_user_to_dict(user):
```

```
    """Converts User SqlAlchemy object to dict"""
    dict = {}
    dict['id'] = user.id
    dict['username'] = user.username
    dict['password'] = user.password
    dict['email'] = user.email
    dict['first_name'] = user.first_name
    dict['last_name'] = user.last_name
    return dict
"""Conversions from dictionary to SqlAlchemy object"""
def dict_query_to_db(user_id, dict):
    """Adds saved query to associated user in db"""
    if dict['sort_by'] == "subjectivity" or dict['sort_by'] == "polarity":
        dict['sa'] = dict['sort_by']
        dict['sort_by'] = 'relevancy'
    dict['type'] = "Detailed Search"
    query = Query(user_id = user_id,
        name = dict['name'],
        source = dict['source'],
        quantity = dict['quantity'],
        date_from = dict['date_from'],
        date_to = dict['date_to'],
        language = dict['language'],
        sort_by = dict['sort_by'],
        sa = dict['sa'],
        type = dict['type']
        )
    return query
def dict_story_to_db(user_id, dict):
    """Adds saved story to associated user in db"""
    story = Story(user_id = user_id,
    headline = dict['headline'],
```

```
    source = dict['source'],
    content = dict['content'],
    author = dict['author'],
    description = dict['description'],
    url = dict['url'],
    image = dict['image'],
    published_at = dict['published_at'],
    sub = dict['sub'],
    pol = dict['pol']
    )
    return story
"""Conversions from form data to dict"""
def form_query_to_dict(form):
    """Converts form data to dict"""
    query = {}
    query['keyword'] = form.keyword.data
    query['source'] = form.source.data
    query['quantity'] = form.quantity.data
    query['date_from'] = form.date_from.data
    query['date_to'] = form.date_to.data
    query['language'] = form.language.data
    if form.sort_by.data == "subjectivity" or form.sort_by.data == "polarity":
        query['sa'] = form.sort_by.data
        query['sort_by'] = 'relevancy'
    else:
        query['sort_by'] = form.sort_by.data
        query['sa'] = None
      if form.default.data:
        query['name'] = form.name.data
        query['default'] = True
    elif form.saved_query.data:
        query['name'] = form.name.data
```

```
    session['query'] = query
    return query
```

Models.py:

```
# import flask_sqlalchemy and create db instance
from flask_sqlalchemy import SQLAlchemy
db = SQLAlchemy()
# hashing
from flask_bcrypt import Bcrypt
bcrypt = Bcrypt()
from datetime import datetime
def connect_db(app):
    """Connect to database."""
    db.app = app
    db.init_app(app)
class User(db.Model):
    __tablename__ = "users"
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    username = db.Column(db.String(20), nullable=False,unique=True)
    password = db.Column(db.Text, nullable=False)
    email = db.Column(db.String(50), nullable=False, unique=True)
    first_name = db.Column(db.String(30), nullable=False)
    last_name = db.Column(db.String(30), nullable=False)
    saved_stories = db.relationship(
        'Story', cascade="all, delete-orphan")
    queries = db.relationship(
        'Query', cascade="all, delete-orphan")
   @classmethod
    def register(cls, username, pwd, email, first_name, last_name):
        hashed = bcrypt.generate_password_hash(pwd)
        hashed_utf8 = hashed.decode("utf8")
        new_user = cls(username=username, password=hashed_utf8,
```

```
                    email=email, first_name=first_name, last_name=last_name)
        db.session.add(new_user)
        return new_user
@classmethod
    def authenticate(cls, username, pwd):
        u = User.query.filter_by(username=username).first()
        if u and bcrypt.check_password_hash(u.password, pwd):
            return u
        else:
            return False
 def __repr__(self):
        return f"<ID: {self.id}, Username:{self.username}>"
class Story(db.Model):
    __tablename__ = "stories"
    id = db.Column(db.String, nullable = False, unique = True, primary_key = True)
    user_id = db.Column(db.Integer, db.ForeignKey('users.id'), nullable=False)
    """information taken from newsapi request"""
    headline = db.Column(db.String, nullable=False)
    source = db.Column(db.String, nullable=False)
    content = db.Column(db.String)
    author = db.Column(db.String)
    description = db.Column(db.String)
    url = db.Column(db.Text)
    image = db.Column(db.Text)
    published_at = db.Column(db.DateTime)
    """information related to its interaction with app"""
    sub = db.Column(db.Text)
    pol = db.Column(db.Text)
    def __repr__(self):
        return f"<ID: {self.id}, User_ID: {self.user_id} H:{self.headline}, S:{self.source}>"
class Query(db.Model):
    __tablename__ = "queries"
```

```
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    user_id = db.Column(db.Integer, db.ForeignKey('users.id'), nullable=False)
    name = db.Column(db.String, nullable = False)
    source = db.Column(db.String, nullable = True)
    default = db.Column(db.Boolean, nullable = True)
    keyword = db.Column(db.String, nullable = True)
    quantity = db.Column(db.Integer, default = 10, nullable = True)
    date_from = db.Column(db.String, nullable = True)
    date_to = db.Column(db.String, nullable = True)
    language = db.Column(db.String, nullable = True)
    sort_by = db.Column(db.String, nullable = True)
    type = db.Column(db.String, nullable = True)
    sa = db.Column(db.String, default = "", nullable = True)
def __repr__(self):
        return f"<ID: {self.id}, User ID#:{self.user_id}, Name:{self.name}, Default:{self.default}"
```

news_api_calls.py:

```
#general imports
import os
from flask import session
import uuid
from dateutil import parser
# newsApi imports
from newsapi.newsapi_client import NewsApiClient
my_api_key = os.environ.get("API_KEY")
newsapi = NewsApiClient(api_key=my_api_key)
#async imports
from multiprocessing.dummy import Pool as ThreadPool
"""Individual functions for handling API response data"""
def collect_results(articles):
    results = []
```

```
    for article in articles:
        headline = article['title']
        source = article["source"]["name"]
        if not article["content"]:  # sometimes Newsapi is unable to provide content for each story
            content = "No content preview found. Click the link above to access the full story."
        else:
            content = article['content']
        author = article['author']
        description = article['description']
        url = article['url']
        image = article['urlToImage']
        api_date = article['publishedAt']
        published_at = parser.parse(api_date)
        id = uuid.uuid4().hex[:10]
        story = {'headline':headline, 'source':source, 'content':content,
        'author':author, 'description':description, 'url':url,
        'image':image, 'published_at':published_at, 'id': id}
        results.append(story)
    return results
def save_to_session(articles):
    """Saves results from api calls as a list of session objects"""
    if "results" in session: # clears previous session results if they exist
        session.pop('results')
    results  = collect_results(articles)
    session["results"] = results
    return results
"""Individual functions for separate types of API Calls"""
def cat_calls(query, slideshow = True):
    """API call to get generalized headlines for a specific catagory"""
    data = newsapi.get_top_headlines(language="en", category=f"{query}")
    articles = data['articles']
    if slideshow: #if api request is being made for homepage, transfer data directly rather than save to
session
```

```
        data = collect_results(articles)
        return data
    saved = save_to_session(articles)
    return saved
def top_headlines_call():
    """API call to get top headlines for all categories"""
    data = newsapi.get_top_headlines(language="en")
    articles = data['articles']
    saved = save_to_session(articles)
    return saved
def simple_search_call(query):
    """API call to get results from single search query"""
    data = newsapi.get_everything(q=f"{query}")
    articles = data['articles']
    spliced = articles[:10]
    saved = save_to_session(spliced)
    return saved
def advanced_search_call(query):
    from_ = str(query['date_from'])
    to = str(query['date_to'])
if to == 'None' and from_ == 'None':
        data = newsapi.get_everything(q=f"{query['keyword']}", sources=f"{query['source']}",
language=f"{query['language']}", sort_by=f"{query['sort_by']}"
                                      )
    elif to == 'None' and from_ != 'None':
        data = newsapi.get_everything(q=f"{query['keyword']}", sources=f"{query['source']}",
language=f"{query['language']}", sort_by=f"{query['sort_by']}", from_param=f"{from_}"
                                      )
    elif to != 'None' and from_ == 'None':
        data = newsapi.get_everything(q=f"{query['keyword']}", sources=f"{query['source']}",
language=f"{query['language']}", sort_by=f"{query['sort_by']}", to=f"{to}"
                                      )
 else:
```

```
        data = newsapi.get_everything(q=f"{query['keyword']}", sources=f"{query['source']}",
language=f"{query['language']}", sort_by=f"{query['sort_by']}", from_param=f"{from_}", to=f"{to}"
                                      )
    # api seems to not want to allow dates to be optional if specified
    # I ran into trouble with the pagesize parameter from news-api, however a
    # temporary solution to this is to extract that number from the query dict,
    # and then splice the resulting list of articles.
    quantity = int(query['quantity'])
    articles = data['articles']
    spliced = articles[:quantity]
    saved = save_to_session(spliced)
    return saved
"""Executes Asyncronous API requests for cat_calls"""
def async_reqs(query):
    pool = ThreadPool(10)
    results = pool.map(cat_calls, query)
    pool.close()
    pool.join()
    return results
```

Sent_analysis.py:

```
# libraries for parsing and sentiment analysis
import nltk
# nltk corupus downloads
nltk.download('vader_lexicon')
nltk.download('stopwords')
nltk.download('punkt')
from nltk.sentiment.vader import SentimentIntensityAnalyzer as SIA
sia = SIA()
from textblob import TextBlob
from nltk.corpus import stopwords
from newspaper import Article
# utility
```

```
from multiprocessing.dummy import Pool as ThreadPool
import re
def parse(story, text_only=False):
    try:
        article = Article(story['url'])
        article.download()
        article.parse()
        parsed = article.text
        if text_only:
            return parsed
        story['text'] = parsed
        return story
    except:
        return ""
def parse_async(stories):
    pool = ThreadPool(10)
    results = pool.map(parse, stories)
    pool.close()
    pool.join()
    return results
def polarize(headline, parsed=False):
    # function returns an object of two sepearate polarity scores; one based off the text of the article and the other
    # from just the headline alone. Each of these are represented in their own respective objects. Currently, headline_res
    # isn't being used publically.
    pol_obj = {}
    headline_res = {}
    article_res = {}
    """Extracting polarity from article text"""
    try:
        """Story might be a dictionary that's already been parsed
```

```
        to be used in order_pol() by async_parse (in the case of multiple articles). Alternatively, story could be passed from

        our SA api still needing to be parsed either as a sqlalchemy instance in a user's saved stories or a dictionary in the

        session result's list. Regardless, calling this function through the SA directly sends result back to form (button) that

        initiated the call, so there is no need to keep track of id and other information. Only text."""

    if parsed:

            parsed = headline # parses story text

            sentenced = nltk.tokenize.sent_tokenize(parsed['text']) # tokenizes story text by sentence

            headline = sia.polarity_scores(parsed['headline']) # nltk analysis is performed (this is where the magic happens)

        else: # same logic as above

            parsed = parse(headline, text_only=True)

            sentenced = nltk.tokenize.sent_tokenize(parsed)

            headline = sia.polarity_scores(headline['headline'])

    except:

        return None

coms = []

    pos = []

    negs = []

    neus = []

for sentence in sentenced:

        res = sia.polarity_scores(sentence) # nltk analysis is performed

        pos.append(res["pos"])

        negs.append(res["neg"])

        neus.append(res["neu"])

        if res['compound']:

            # sometimes the composite will be zero for certain sentences. We don't want to include that data.

            coms.append(res['compound'])

    if len(coms) == 0:

        return None

    avg_com = round((sum(coms) / len(coms)), 2)
```

```
    avg_pos = round((sum(pos) / len(pos)), 2)
    avg_neu = round((sum(neus) / len(neus)), 2)
    avg_neg = round((sum(negs) / len(negs)), 2)
    article_res["avg_com"] = round(avg_com, 2)
    article_res["avg_pos"] = round(avg_pos, 2)
    article_res["avg_neg"] = round(avg_neg, 2)
    article_res["avg_neu"] = round(avg_neu, 2)
    if avg_com >= 0.4:
        article_res['result'] = f"{avg_com} (Very Positive)"
    elif avg_com >= 0.2:
        article_res['result'] = f"{avg_com} (Positive)"
    elif avg_com <= - 0.2:
        article_res['result'] = f"{avg_com} (Negative)"
    elif avg_com <= - 0.2:
        article_res['result'] = f"{avg_com} (Very Negative)"
    else:
        article_res['result'] = f"{avg_com} (Neutral)"
    article_res["message"] = f"{article_res['result']}. {avg_neg *100}% Negative, {avg_neu *100}%
Neutral, and {avg_pos *100}% Positive"
    """Extracting polarity from headline text"""
    headline_res["com"] = headline['compound']
    headline_res["pos"] = headline['pos']
    headline_res["neg"] = headline['neg']
    headline_res["neu"] = headline['neu']
    if headline_res['com'] >= 0.2:
        headline_res['result'] = "Positive"
    elif headline_res['com'] >= 0.4:
        headline_res['result'] = "Very Positive"
    elif headline_res['com'] <= - 0.2:
        headline_res['result'] = "Negative"
    elif headline_res['com'] <= - 0.4:
        headline_res['result'] = "Very Negative"
    else:
```

```
        headline_res['result'] = "Neutral"
    pol_obj['headline_res'] = headline_res
    pol_obj['article_res'] = article_res
    return pol_obj
def subjectize(headline, parsed=False):
    try:
        if parsed:
            parsed = headline['text']
        else:
            parsed = parse(headline, text_only=True)
    except:
        return None
    tblobbed = TextBlob(parsed)
    subjectivity = round(tblobbed.sentiment.subjectivity, 2)
    subjectivity = str(subjectivity)
    subjectivity = subjectivity[-2:]
    if subjectivity == ".":
        subjectivity = f"{subjectivity}0"
    if subjectivity == ".0":
        return None
    subjectivity = round(float(subjectivity))
    sub_obj = {}
    if subjectivity > 80:
        sub_obj['measure'] = f"{subjectivity}% (Very Objective)"
    elif subjectivity > 60:
        sub_obj['measure'] = f"{subjectivity}% (Objective)"
    elif subjectivity > 40:
        sub_obj['measure'] = f"{subjectivity}% (Neutral)"
    elif subjectivity > 20:
        sub_obj['measure'] = f"{subjectivity}% (Subjective)"
    else:
        sub_obj['measure'] = f"{subjectivity}% (Very Subjective)"
```

```
    sub_obj['score'] = subjectivity
if sub_obj['score'] == 0.0:
        return None
    return sub_obj
```

api.py:

```
"""Restful Server-Side API"""
from flask import request, jsonify
from flask_bcrypt import Bcrypt
bcrypt = Bcrypt()
from helpers import db_query_to_dict, db_user_to_dict, dict_query_to_db, db_story_to_dict,
dict_story_to_db
from app import app
# sqlalchemy imports
from sqlalchemy import exc
from psycopg2.errors import UniqueViolation
from models import db, User, Story, Query
"""USERS"""
@app.route('/users', methods=["GET"])
def get_all_users():
    """Gets all users"""
    users = User.query.all()
    dict_list = [db_user_to_dict(user) for user in users]
    return jsonify(users = dict_list)
@app.route('/users/delete', methods=["DELETE"])
def delete_all_users():
    """Deletes all users"""
    try:
        User.query.delete()
        db.session.commit()
    except exc.SQLAlchemyError as e:
        response = {"Unable to delete all users": f"{e.origin}"}
```

```
        return response
    return {"message":"Success! All users were deleted."}
@app.route('/users/<int:user_id>', methods=["GET"])
def get_user(user_id):
    """Gets a user by user_id"""
    user = User.query.get_or_404(user_id)
    dict = db_user_to_dict(user)
    return jsonify(user = dict)
@app.route('/users/<int:user_id>', methods=["DELETE"])
def delete_user(user_id):
    """Deletes a user by user_id"""
    try:
        User.query.filter_by(id=user_id).delete()
        db.session.commit()
        return {"message":"Success! User was deleted."}
    except exc.SQLAlchemyError as e:
        response = {"Unable to delete user": f"{e.origin}"}
        return response
@app.route('/users/new', methods=["POST"])
def new_user():
        """Creates a new user"""
        data = request.json['user']
        new_user = User.register(
            data['username'], data['password'], data['email'], data['first_name'], data['last_name'])
        try:
            db.session.add(new_user)
            db.session.commit()
            response = {"message": f"User {data['username']} added successfully."}
        except exc.SQLAlchemyError as e:
            if isinstance(e.orig, UniqueViolation):
                response = {"message": f"{e.origin}"}
            else:
```

```
            response = {"message": "Sorry, something went wrong. Unable to add new user to database."}
        return response
@app.route('/users/<int:user_id>/edit', methods=["PUT"])
def edit_user(user_id):
        """Edits a user's information by user_id"""
        user = User.query.get(user_id)
        data = request.json['user']
        user.username = data['username']
        hashed = bcrypt.generate_password_hash(data['password'])
        hashed_utf8 = hashed.decode("utf8")
        user.password = hashed_utf8
        user.email = data['email']
        user.first_name = data['first_name']
        user.last_name = data['last_name']
        try:
            db.session.add(user)
            db.session.commit()
            dict = db_user_to_dict(user)
            return jsonify(updated_user = dict)
        except exc.SQLAlchemyError as e:
            if isinstance(e.orig, UniqueViolation):
                response = {"message": f"{e.origin}"}
            else:
                response = {"message": "Sorry, something went wrong. Unable to update user."}
            return response
"""USER QUERIES"""
@app.route('/users/<int:user_id>/queries/new', methods=["POST"])
def new_query(user_id):
    """Creates a new query and associates it with user_id"""
    data = request.json['query']
    try:
        query = dict_query_to_db(user_id, data)
```

```
        db.session.add(query)
        db.session.commit()
    except exc.SQLAlchemyError as e:
        response = {"message": f"{e.origin}"}
        return response
dict = db_query_to_dict(query)
    return jsonify(new_query = dict)
@app.route('/users/<int:user_id>/queries/', methods=["GET"])
def get_all_queries_by_user(user_id):
    """Gets all queries matching user_id"""
    user = User.query.get_or_404(user_id)
    queries = user.queries
    dict_list = [db_query_to_dict(query) for query in queries]
    return jsonify(queries = dict_list)
@app.route('/users/<int:user_id>/queries/<int:query_id>/edit', methods=["PUT"])
def edit_query(user_id, query_id):
    """Edits a User Query by User and Query ids"""
    data = request.json['query']
    query = Query.query.get(query_id)
    try:
        if user_id != query.user_id:
            raise ValueError("The user id is not associated with the query id.")
    except ValueError:
        response = {"Value Error": "The user id is not associated with the query id."}
        return response
    try:
        query = dict_query_to_db(user_id, data)
        db.session.add(query)
        db.session.commit()
    except exc.SQLAlchemyError as e:
        response = {"message": f"{e.origin}"}
        return response
```

```
    dict = db_query_to_dict(query)
    return jsonify(updated_query = dict)
@app.route('/users/<int:user_id>/queries/<int:query_id>/delete', methods=["DELETE"])
def delete_query(user_id, query_id):
    """Deletes a User Query by User and Query ids"""
    query = Query.query.get(query_id)
    try:
        if user_id != query.user_id:
            raise ValueError("The user id is not associated with the query id.")
    except ValueError:
        response = {"Value Error": "The user id is not associated with the query id."}
        return response
    try:
        Query.query.filter_by(id=query_id).delete()
        db.session.commit()
        return {"message":"Success! Query was deleted."}
    except exc.SQLAlchemyError as e:
        response = {"Unable to delete query": f"{e.origin}"}
        return response
"""USER STORIES"""
@app.route('/users/<int:user_id>/stories/new', methods=["POST"])
def new_story(user_id):
    """Creates a new story and associates it with user_id"""
    data = request.json['story']
    try:
        story = dict_story_to_db(user_id, data)
        db.session.add(story)
        db.session.commit()
    except exc.SQLAlchemyError as e:
        response = {"message": f"{e.origin}"}
        return response
        dict = db_query_to_dict(story)
```

```
    return jsonify(new_story = dict)
@app.route('/users/<int:user_id>/stories/', methods=["GET"])
def get_all_stories_by_user(user_id):
    """Gets all queries matching user_id"""
    user = User.query.get_or_404(user_id)
    stories = user.stories
    dict_list = [db_story_to_dict(story) for story in stories]
    return jsonify(stories = dict_list)
@app.route('/users/<int:user_id>/stories/<int:story_id>/edit', methods=["PUT"])
def edit_story(user_id, story_id):
    """Edits a User Story by User and Query ids"""
    data = request.json['query']
    story = Story.query.get_or_404(story_id)
    try:
        if user_id != story.user_id:
            raise ValueError("The user id is not associated with the story id.")
    except ValueError:
        response = {"Value Error": "The user id is not associated with the story id."}
        return response
    try:
        story = dict_story_to_db(user_id, data)
        db.session.add(story)
        db.session.commit()
        dict = db_query_to_dict(story)
        return jsonify(updated_story = dict)
    except exc.SQLAlchemyError as e:
        response = {"message": f"{e.origin}"}
        return response
@app.route('/users/<int:user_id>/queries/<int:story_id>/delete', methods=["DELETE"])
def delete_story(user_id, story_id):
    """Deletes a User Story by User and Query ids"""
    story = Query.query.get_or_404(story_id)
```

```
    try:
        if user_id != story.user_id:
            raise ValueError("The user id is not associated with the story id.")
    except ValueError:
        response = {"Value Error": "The user id is not associated with the story id."}
        return response
    try:
        Query.query.filter_by(id=story_id).delete()
        db.session.commit()
        return {"message":"Success! Story was deleted."}
    except exc.SQLAlchemyError as e:
        response = {"Unable to delete story": f"{e.origin}"}
        return response
"""QUERIES"""
@app.route('/queries', methods=["GET"])
def get_all_queries():
    """Gets all queries"""
    queries = Query.query.all()
    dict_list = [db_query_to_dict(query) for query in queries]
    return jsonify(queries = dict_list)

@app.route('/queries/<int:query_id>', methods=["GET"])
def get_query(query_id):
    """Gets a query by query_id"""
    query = Query.query.get_or_404(query_id)
    dict = db_query_to_dict(query)
    return jsonify(query= dict)
"""STORIES"""
@app.route('/stories', methods=["GET"])
def get_all_stories():
    """Gets all stories"""
    stories = Story.query.all()
```

```
    dict_list = [db_story_to_dict(story) for story in stories]
    return jsonify(stories = dict_list)
@app.route('/stories/<story_id>', methods=["GET"]) #story ids are non-numerical as they are inherited from sessions randomly generated uuid() ids
def get_story(story_id):
    """Gets story by story id"""
    story = Story.query.get_or_404(story_id)
    dict = db_story_to_dict(story)
    return jsonify(story = dict)
#sample query:
#{"query": {"name":"test222", "source":"fox news", "quantity":10, "date_from":"", "date_to":"", "language":"en", "sort_by":"subjectivity", "sa":"", "type":"Detailed Search"}}
#sample user:
# {"user": {"username":"coolguy41", "password":"hmmm", "email":"eee@mail.com", "first_name":"first", "last_name":"last"}}
```

Sa_views.py:

```
"""Sentiment Analysis API for individual stories"""
from app import app
from sent_analysis import subjectize, polarize
from models import Story
from flask import jsonify, session
from helpers import *
@app.route('/story/<id>/polarity', methods=['POST'])
def show_pol_calls(id):
    try:
        # check to see if id represents a sqlalchemy object that needs converted to dict to be fed to SA functions
        # write logic to save score to db if story saved
        db_story = Story.query.get(id)
        story = db_story_to_dict(db_story)
    except:
        results = session['results']
        story = [story for story in results if story['id'] == id][0]
```

```
    score = polarize(story)
    if not score:
        story['pol'] = "No Data"
    else:
        score = score['article_res']['result']
        story['pol'] = str(score)
    return jsonify({'response': story['pol']})
@app.route('/story/<id>/subjectivity', methods=['POST'])
def show_sub_calls(id):
    try:
        # check to see if id represents a sqlalchemy object that needs converted to dict to be fed to SA functions
        db_story = Story.query.get(id)
        story = db_story_to_dict(db_story)
    except:
        results = session['results']
        story = [story for story in results if story['id'] == id][0]
    score = subjectize(story)
    if not score:
        story['sub'] = "No Data"
    else:
        score = score['measure']
        story['sub']= str(score)
    return jsonify({'response': story['sub']})
```

site_views.py:

```
from app import app, CURR_USER_KEY
from flask import request, render_template, flash, redirect, render_template, session, g, jsonify
from models import User, Story, Query
from helpers import *
from news_api_calls import *
from forms import RegisterForm, LoginForm, SearchForm
```

```
from sqlalchemy import exc
from psycopg2.errors import UniqueViolation
#TODONOW:
#-rewrite code and distinguish between session['saved] and session['results]
@app.before_request
def add_user_to_g():
    """If we're logged in, add curr user to Flask global."""
    if CURR_USER_KEY in session:
        g.user = User.query.get(session[CURR_USER_KEY])
    else:
        g.user = None
def do_login(user):
    """Log in user."""
    session[CURR_USER_KEY] = user.id
def do_logout(user):
    """Logout user."""
    if CURR_USER_KEY in session:
        del session[CURR_USER_KEY]
"""View functions for application"""
with app.app_context():
    @app.route('/')
    def homepage():
        # return redirect("/login")
        no_user = True
        if g.user:
            no_user = False
        """NewsTracker Homepage"""
        categories = ['business', 'entertainment',
                'health', 'science', 'sports', 'technology']
        data = []
        results = async_reqs(categories)
        for index, result in enumerate(results):
```

```
            obj = {}
            obj['results'] = result
            obj['top_story'] = result[0]
            obj['name'] = categories[index].capitalize()
            data.append(obj)
        return render_template('/homepage.html', data=data, no_user=no_user)
@app.route('/headlines', methods=['GET', 'POST'])
def headlines():
    """Default Search Query if user logged in with a saved query as default"""
    if CURR_USER_KEY in session:
        user = User.query.get(g.user.id)
        queries = user.queries
        default = [query for query in queries if query.default]
        if default:
            return redirect(f"search/{default[0].id}")
        else:
            """Otherwise, Generic Top Headlines"""
            results = top_headlines_call()
            return render_template('/show_stories.html', results=results)
    results = top_headlines_call()
    return render_template('/show_stories.html', results=results)
@app.route(f'/headlines/<category>')
def show_for_category(category):
    """Display top headlines for given category based off link clicked from homepage"""
    category = category.lower()
    results = cat_calls(category, slideshow=False)
    return render_template('show_stories.html', results=results)
@app.route('/search', methods=['GET', 'POST'])
def search_form():
    """This function creates a dictionary extracting data from the search form to be sent to the news-
api"""
    if CURR_USER_KEY in session:
        form = SearchForm()
```

```
        if form.validate_on_submit():
            try:
                dict_query = form_query_to_dict(form)
                if form.saved_query.data or form.default.data:
                    db_query = dict_query_to_db(g.user.id, session['query'])
                    db.session.add(db_query)
                    db.session.commit()
                advanced_search_call(dict_query)
                return redirect('/search/results')
            except:
                return render_template('/search.html', form=form)
        else:
            return render_template('/search.html', form=form)
    else:
        flash("You do not have permission to view this page. Please log-in and try again.", "danger")
        return redirect("/login")
@app.route('/search/<int:query_id>')
def search_user_queries(query_id):
    """Makes advanced search call based off of pre-saved query"""
    if CURR_USER_KEY in session:
        query_obj = Query.query.get(query_id)
        query_dict = db_query_to_dict(query_obj)
        advanced_search_call(query_dict)
        return redirect('/search/results')
    else:
        flash("You do not have permission to make this action. Please log-in and try again.", "danger")
        return redirect("/login")
@app.route('/search/results')
def handle_results():
    if CURR_USER_KEY in session:
        query = session['query']
        results = session['results']
```

```
        if query['sa'] == 'polarity':
            results = order_pol()
        elif query['sa'] == 'subjectivity':
            results = order_sub()
        else:
            return render_template('/show_stories.html', results=results)
    else:
        flash("You do not have permission to view this page. Please log-in and try again.", "danger")
        return redirect("/login")
    return render_template('/show_stories.html', results=results)
@app.route('/search/simple', methods=['GET'])
def search_simple():
    """API Call and Results for Simple Search"""
    keyword = request.args.get("search")
    results = simple_search_call(keyword)
    return render_template('/show_stories.html', results=results)
@app.route('/user/saved')
def user_saved():
    if CURR_USER_KEY in session:
        user = User.query.get(g.user.id)
        is_empty = False
        if len(user.saved_stories) == 0:
            is_empty = True
        if "saved" in session:
            session.pop("saved")
        session["saved"] = [story for story in user.saved_stories]
        return render_template("/user.html", user=user, is_empty=is_empty)
    else:
        flash("You do not have permission to view this page. Please log-in and try again.", "danger")
        return redirect("/login")
@app.route('/story/<id>/open')
def open_story_link(id):
```

```
    """Opens url associated with story when link is clicked"""
    try:
        story = Story.query.get(id)
        return redirect(f"{story.url}")
    except:
        results = session['results']
        story = [story for story in results if story['id'] == id][0]
        return redirect(f"{story['url']}")
@app.route('/story/<id>/save_story', methods=["POST"])
def save_story(id):
    if CURR_USER_KEY in session:
        results = session["results"]
        session_story = [story for story in results if story['id'] == id][0]
        story = Story(headline=session_story['headline'], source=session_story['source'],
content=session_story['content'],
                    author=session_story['author'], description=session_story['description'],
url=session_story['url'], image=session_story['image'],
                    published_at=session_story['published_at'], id=session_story['id'], user_id = g.user.id)
        try:
            db.session.add(story)
            db.session.commit()
        except exc.SQLAlchemyError as e:
            if isinstance(e.orig, UniqueViolation):
                flash(f'The story "{story.headline}" already exists in your saved stories.', 'danger')
                return redirect("/user/saved")
        return redirect("/user/saved")
    else:
        flash("You do not have permission to make this action. Please log-in and try again.", "danger")
        return redirect("/login")
@app.route('/story/<id>/remove', methods=["POST"])
def remove_story(id):
    if CURR_USER_KEY in session:
        Story.query.filter_by(id=id).delete()
```

```
        db.session.commit()
        return redirect("/user/saved")
    else:
        flash("You do not have permission to make this action. Please log-in and try again.", "danger")
        return redirect("/login")
@app.route('/register', methods=['GET', 'POST'])
def register_user():
    form = RegisterForm(prefix='form-register-')
    if form.validate_on_submit():
        username = form.username.data
        password = form.password.data
        email = form.email.data
        first_name = form.first_name.data
        last_name = form.last_name.data
        new_user = User.register(
            username, password, email, first_name, last_name)
        try:
            db.session.add(new_user)
            db.session.commit()
            flash("Congratulations! You have successfully created an account.", "success")
            do_login(new_user)
            return redirect('/headlines')
        except exc.SQLAlchemyError as e:
            if isinstance(e.orig, UniqueViolation):
                form.username.errors = [
                    "The username you entered is already taken. Please pick another one."]
            #
https://www.youtube.com/embed/iBYCoLhziX4?showinfo=0&controls=1&rel=0&autoplay=1
    else:
        print("Form Not Validated")
    return render_template('register.html', form=form)
@app.route('/login', methods=['GET', 'POST'])
def login_user():
```

```
    form = LoginForm()
    if form.validate_on_submit():
        username = form.username.data
        password = form.password.data
        user = User.authenticate(username, password)
        if user:
            do_login(user)
            flash("Credentials verified. You are now logged in.", "success")
            return redirect('/headlines')
        else:
            form.username.errors = [
                "Invalid username or password. Please try again."]
    return render_template('login.html', form=form)
@app.route('/login/demo', methods=['POST'])
def login_demo_user():
    random = uuid.uuid4().hex[:8]
    username = f"demo-user{random}"
    password = 'demouser'
    first_name = "first"
    last_name = "name"
    email = f"demo{random}@user.com"
    demo_user = User.register(
            username, password, email, first_name, last_name)
    db.session.commit()
    db.session.add(demo_user)
    db.session.commit()
    do_login(demo_user)
    return redirect('/headlines')
@app.route('/logout')
def logout():
    """Handle logout of user."""
    if CURR_USER_KEY in session:
```

```
        flash(f"You have successfully logged out.", "success")
        user = User.query.get(g.user.id)
        do_logout(user)
        return redirect("/headlines")
    else:
        return redirect("/login")
@app.route('/user/<int:query_id>/delete', methods=['POST'])
def remove_query(query_id):
    if CURR_USER_KEY in session:
        Query.query.filter_by(id=query_id).delete()
        db.session.commit()
        return jsonify({'response': f"Query deleted!"})
    else:
        flash("You do not have permission to make this action. Please log-in and try again.", "danger")
        return redirect("/login")
```

tests_database.py

```
import sys
sys.path.append("..")
from unittest import TestCase
from app import app
from models import db, User, Story, Query
app.config['SQLALCHEMY_ECHO'] = False
app.config['TESTING'] = True
app.config['SQLALCHEMY_ECHO'] = False
app.config['DEBUG_TB_HOSTS'] = ['dont-show-debug-toolbar']
app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql:///newstracker-test'
db.drop_all()
db.create_all()
class User_Model(TestCase):
    def setUp(self):
        """Cleans up any existing users"""
```

```
        User.query.delete()
        user = User.register(
            "dummy", "dummypassword", "dumb@y.com", "dummy", "account")
        db.session.add(user)
        db.session.commit()
        self.user = user
    def tearDown(self):
        db.session.rollback()
    def test_register(self):
        user = User.register(
            "testuser", "test4444", "test@test.com", "test", "user")
        """Tests columns id and username as written in __repr__"""
        self.assertEqual(str(user), f"<ID: {user.id}, Username:{user.username}>")
        # self.assertEquals(str(user), f"<ID: 1, Username:testuser>")
        """Tests that columns are accessible as written"""
        # todo: make sure model is autonmatically assigning id, right now the following test fails
        # self.assertEquals(user.id, 1)
        self.assertEqual(user.username, "testuser")
        self.assertEqual(user.email, "test@test.com")
        self.assertEqual(user.first_name, "test")
        self.assertEqual(user.last_name, "user")
        """Tests that password is hashed and not accessible"""
        self.assertNotEqual(user.password, "test4444")
    def test_passes_authenticate(self):
        """Tests that authentication can be passed based off of pre-existing data"""
        user = User.authenticate(self.user.username, "dummypassword") #password is hashed on
registration and needs to be passed as string to pass test
        self.assertNotEqual(user, False) # authenticate method returns false if fails, user object if passes
     def test_fails_authenticate(self):
        user = User.authenticate("incorrect", "dummypassword")
        self.assertEqual(user, False)
    def test_fails_authenticate_hashed(self):
        user = User.authenticate(self.user.username, self.user.password)
```

```
        self.assertEqual(user, False)
```

tests_helpers.py:

```
import sys
sys.path.append("..")
from unittest import TestCase
from app import app
app.config['SQLALCHEMY_ECHO'] = False
app.config['TESTING'] = True
app.config['DEBUG_TB_HOSTS'] = ['dont-show-debug-toolbar']
```

app.css:

```
/* general foundational styles */
body {
    padding: 0;
    margin: 0;
    background: black;
    font-family: 'Abel', sans-serif !important;
}
ul {
    padding: 0;
    list-style-type: none;
    margin: auto;
}
span {
    color:whitesmoke
}
.messages {
    padding: 1rem;
    border-radius: 1rem;
    display:flex;
    justify-content: center;
```

```
    align-items: center;
    text-align:center;
}
.messages .success {
    background-color: rgba(0, 225, 0, 0.1);
    font-size:2rem;
    border-radius:1rem;
    padding-right:1rem;
    padding-left:1rem;
}
.messages .danger {
    background-color: rgba(255, 0, 0, 0.1);
    font-size:2rem;
    border-radius:1rem;
    padding-right:1rem;
    padding-left:1rem;
}

.home-body {
    display: flex;
    justify-content: center;
}
#indentation {
    margin-bottom: 100px;
    margin-right:0px;
}
@media only screen and (max-width: 576px) {
    ul {
    list-style-type: none;
    }
    #stories {
    margin-left:40px;
```

```
    }
}
/* card styles for each news story*/
.no-stories {
    /* padding:2rem; */
    margin-left:2rem;
    margin-right:2rem;
    text-align:center;
    display:flex;
}
.row {
    margin: auto;
}
.container {
    background: #242424;
    border-radius: 30px;
    margin-bottom:2rem;
}
.card {
    justify-content: center;
    margin: auto;
    border-radius: 10px;
    transition-duration: 0.5s;
}
.card:hover {
    border-width: 1px;
    background: #ade7ff;
    margin: auto;
    box-shadow: 1px 1px 10px 10px #95eeee;
}
.card-body {
    background: #3d3d3d;
```

```
    border-radius: 10px;
    justify-content: center;
    transition-duration: 0.5s;
}
.card-body:hover {
    background: rgb(65, 65, 65)
}
.results {
    color:whitesmoke;
    margin:auto;
    }
#date {
    color:gainsboro;
}
#image {
    margin-bottom: 10px;
}
/* more specific card styles to adapt to the behavior of mobile devices*/
@media only screen and (max-width: 768px) {
    #left-b{
        margin-bottom: 10px;
        order: 1;
        padding-right: 0px;
    }
    #image {
        order: 0;
        width: 100%;
    }
    #right-b{
        margin-bottom: 10px;
        order: 1;
        padding-left:0px;
```

```
    }
    }
@media only screen and (max-width: 576px) {
    .container {
        justify-content: center;
    }
    #image {
        margin-top: -20px;
        order:0;
    }
    #search{
        display:none;
    }
}
/* form styling */
h1 {
    color:whitesmoke;
    font-family: 'Times New Roman', Times, serif;
}
.form {
    color: whitesmoke;
}
.form-alert {
        margin: 20px;
}
.submit-b {
    margin-bottom:20px;
}
.demo-user {
    text-align:center;
    margin-bottom:20px;
}
```

```
#content {
    color:white;
    margin-top: 5px;
}
.headline {
    margin-top: 12px;
    color:gainsboro;
}
.headline:hover {
    text-decoration: none;
    color: #a3a3a3;
    border-radius: 40px;
}


/* navbar styles*/
navbar {
    background-color: black;
    color:rgb(65, 65, 65);
}
.close {
    display: inline-block;
    font-size: 30px;
    font-weight: 600;
  }
.navbar-toggler {
    margin-right:10px;
}
.nav-item {
    margin:auto;
    color: #f2f2f2;
    font-family: 'Times New Roman', Times, serif;
```

```
    padding-left: 0px;
    text-decoration: none;
    font-size: 17px;
    transition-duration: 0.25s;
    font-size:16px;
}
.nav-item:hover {
    background-color:rgb(119, 119, 119) ;
    color: black;
    text-decoration: none;
}
.navbar-nav{
    margin:0px;
}
#dropdown-item {
    display: inline;
    text-decoration: none;
    padding-right:10px;
}
.query-link {
    display: inline;
    text-decoration: none;
    color:#3d3d3d;
}
#search{
    margin-right: 20px;
}
@media only screen and (max-width: 768px) {
    #search{
        display:none;
        order:0;
    }
```

```
}
#nav-msg-spc {
margin-top: 100px;
}
/* Home-Page */
#home {
    margin: auto;
    margin-top: 100px;
    padding-right: 0px;
    padding-left: 0px;
    padding-top:2rem;
    padding-bottom:2rem;
}
h4 {
    /*used throughout app as headlines*/
    color:white;
    margin:auto
}
h5 {
    color:white;
    margin:auto
}
p {
    color:whitesmoke
}
/* Right (category-slideshow)*/
#slideshows {
    padding:0px;
}
#slideshow {
    margin:auto;
    margin-bottom:15px;
```

```
    padding:0px;
}
.d-block {
    /*image*/
    width: 250px;
    height: 175px;
    object-fit: contain
}
#slide-counter {
    padding:0px;
 }
h3 a{
    text-decoration: none;
color:white
}
/*Left*/
.col-md-7 {
    /* column left for overall container, contains info */
    padding:0px;
}
.announce {
    /*div for columns in cards*/
    padding:3px;
    margin:3px;
}
.heading {
    /*larger header div that includes descriptive text*/
    margin: 10px;
    text-align: center;
}
.heading h4 {
    /*eg, non-user, user, upcoming*/
```

```
    font-size: 2rem;
    padding-top: 1rem;
    padding-bottom:.5rem;
}
.carousel-caption {
            text-align:justify;
            padding:0px;
            background:rgba(0,0,0,0.4);
    font-size: 1.25vw;
    margin-bottom:2rem;
}
.category {
    padding:0.5rem;
    color:white;
}
.category:hover {
    text-decoration: none;
    color: #a3a3a3;
}
@media only screen and (max-width: 768px) {
    #slideshows{
        padding-top: 1rem;
    }
    .carousel-caption {
        text-align:justify;
        padding:0px;
        background:rgba(0,0,0,0.4);
        font-size: 2vw;
        margin-bottom:2rem;
    }
    .caption {
        text-align:justify;
```

```
    }
}
```

Newstracker.js

```
const body = document.querySelector("body");
const homeAdd = function (arg) {
    arg.classList.add("home-body");
}
//axios requests that reach our server to run SA on said story and show result inside button
const saButtons = document.querySelectorAll(".sa-button");
for (let button of saButtons) {
    button.addEventListener("click", async function (evt) {
        evt.preventDefault();
        const text = button.firstChild;
        const input = text.nextSibling;
        const value = input.value;
        const storyID = button.getAttribute('data-story')
        if (value === "Get Polarity") {
            try {
                const req = await axios.post(`/story/${storyID}/polarity`);
                const resp = req.data.response;
                input.value = resp;
            } catch (error) {
                console.error(error.response.data)
            }
        }
        else if (value === "Get Subjectivity") {
            const req = await axios.post(`/story/${storyID}/subjectivity`);
            const resp = req.data.response;
            input.value = resp;
        }
    })
```

```
}
const deleteQueryButtons = document.querySelectorAll(".close")
for (let button of deleteQueryButtons) {
    button.addEventListener("click", async function (evt) {
        console.log("CLICKEDDDD")
        const queryID = button.getAttribute('data-query')
        console.log(queryID)
        const req = await axios.post(`/user/${queryID}/delete`);
        const resp = req.data.response;
        button.value = resp;
    })
}
//Replace broken images with default image
const default_avatar = 'https://secure.gravatar.com/avatar?d=wavatar';
window.addEventListener("load", event => {
    let images = document.querySelectorAll('img');
    for (let image of images) {
        let isLoaded = image.complete && image.naturalHeight !== 0;
        if (!isLoaded) {
            image.src = default_avatar;
        }
    }
});
```

test_routes.py:

```
import sys
sys.path.append("..")
from unittest import TestCase
from app import app
from models import db, User, Story, Query
from forms import RegisterForm, LoginForm, SearchForm
from sqlalchemy import exc
from psycopg2.errors import UniqueViolation
```

```
app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql:///newstracker-test'
app.config['SQLALCHEMY_ECHO'] = False
app.config['TESTING'] = True
app.config['DEBUG_TB_HOSTS'] = ['dont-show-debug-toolbar']
app.config['WTF_CSRF_ENABLED'] = False
db.drop_all()
db.create_all()
registration_data = {
'form-register-username':'user1',
'form-register-password':'pass1',
'form-register-email':'test@email.com',
'form-register-first_name':'test',
'form-register-last_name':'user'
}
from flask import session
from flask_session import Session
app.config['SESSION_TYPE'] = 'redis'
app.config['SESSION_USE_SIGNER'] = True
app.config['SESSION_PERMANENT'] = False
CURR_USER_KEY = "curr_user"
server_session = Session(app)
class No_User(TestCase):
    def setUp(self):
        """Add sample user."""
        User.query.delete()
        user = User.register(
            "testuser", "test4444", "test@test.com", "test", "user")
        db.session.add(user)
        db.session.commit()
        self.user_id = user.id
 def tearDown(self):
        db.session.rollback()
```

```
    def test_forbidden_routes(self):
        """Tests Redirects"""
        with app.test_client() as client:
            res = client.get("/search")
            self.assertEqual(res.status_code, 302)
            self.assertEqual(res.location, "http://localhost/login")
            res2 = client.get("/search/results")
            self.assertEqual(res2.status_code, 302)
            self.assertEqual(res2.location, "http://localhost/login")
            res3 = client.get("/user/saved")
            self.assertEqual(res3.status_code, 302)
            self.assertEqual(res3.location, "http://localhost/login")
            res4 = client.get("/logout")
            self.assertEqual(res4.status_code, 302)
            self.assertEqual(res4.location, "http://localhost/login")
    def test_redirection_followed(self):
        """Test that redirects are followed"""
        with app.test_client() as client:
            res = client.get("/search", follow_redirects=True)
            self.assertNotEqual(res.status_code, 302)
            html = res.get_data(as_text=True)
            self.assertIn('You do not have permission', html)
            res2 = client.get("/search/results", follow_redirects=True)
            self.assertNotEqual(res2.status_code, 302)
            html2 = res2.get_data(as_text=True)
            self.assertIn('You do not have permission', html2)
            res3 = client.get("user/saved", follow_redirects=True)
            self.assertNotEqual(res3.status_code, 302)
            html3 = res3.get_data(as_text=True)
            self.assertIn('You do not have permission', html3)
def test_404(self):
        with app.test_client() as client:
```

```
        resp = client.get("/this/route/does/not/exist")
        self.assertEqual(resp.status_code, 404)
    def test_get_login(self):
        """Simulates GET Request to check for status code and HTML"""
        with app.test_client() as client:
            res = client.get("/login")
            html = res.get_data(as_text=True)
            self.assertEqual(res.status_code, 200)
            self.assertIn('<h1 class="form display-3 text-center">Login</h1>', html)
    def test_get_register(self):
        """Simulates GET Request to check for status code and HTML"""
        with app.test_client() as client:
            res = client.get("/register")
            html = res.get_data(as_text=True)
            self.assertEqual(res.status_code, 200)
            self.assertIn('<h1 class="display-3 text-center">Register</h1>', html)
    def test_simple_search(self):
        """Simulates simple search"""
        with app.test_client() as client:
            #tests results when expected
            res = client.get("/search/simple?search=china")
            html = res.get_data(as_text=True)
            self.assertEqual(res.status_code, 200)
            self.assertNotIn("We weren't able to find anything. Try checking your spelling or widening
your search query.", html)
         #tests no results when expected
            res2 = client.get("/search/simple?search=12345567890qwertyuio")
            html2 = res2.get_data(as_text=True)
            self.assertIn("We weren't able to find anything. Try checking your spelling or widening your
search query.", html2)
 def test_headlines(self):
        """Simulates non-user selecting headlines tab"""
        with app.test_client() as client:
```

```
        res = client.get("/headlines")
        self.assertEqual(res.status_code, 200)
        html = res.get_data(as_text=True)
        #tests button values on stories
        self.assertIn("Get Polarity", html)
        self.assertIn("Get Subjectivity", html)
        self.assertIn("Save Story", html)
        #tests html elements and classes
        self.assertIn('<div class="container">', html)
        self.assertIn('<section id="stories">', html)
# def test_home_page(self):
    #     """Simulates non-user selecting headlines tab"""
    #     with app.test_client() as client:
    #         res = client.get("/")
    #         self.assertEqual(res.status_code, 200)
    #         html = res.get_data(as_text=True)
    #         #tests all categories are appearing
    #         self.assertIn("Business", html)
    #         self.assertIn("Science", html)
    #         self.assertIn("Health", html)
    #         self.assertIn("Entertainment", html)
    #         self.assertIn("Sports", html)
    #         self.assertIn("Technology", html)
    #         #tests html elements and classes
    #         self.assertIn('What can I do with NewsTracker?</h3>', html)
    #         self.assertIn('Non-Users</h4>', html)
    #         self.assertIn('Get Headlines</h5>', html)
    #         self.assertIn('Search by Keyword</h5>', html)
    #         self.assertIn('Browse by Category</h5>', html)
    #         self.assertIn('Users</h4>', html)
    #         self.assertIn('Detailed Search</h5>', html)
    #         self.assertIn('Sentiment Analysis Tools</h5>', html)
```

```
#         self.assertIn('Saved Stories</h5>', html)
#         self.assertIn('Upcoming Features</h4>', html)
#         self.assertIn('Amplified User Features</h5>', html)
#         self.assertIn('Integration of Twitter API', html)
#         self.assertIn('<div id="slideshows"', html)
#         self.assertIn('Try NewsTracker with our dummy account', html)
#         self.assertIn('<div class="carousel slide" data-ride="carousel">', html)
def test_create_user(self):
    """Simulates POST Request to create a new user"""
    with app.test_client() as client:
        res = client.post("/register", follow_redirects=True, data=registration_data)
        html = res.get_data(as_text=True)
        self.assertEqual(res.status_code, 200)
        self.assertIn('Congratulations! You have successfully created an account.', html)
def test_login_user(self):
    """Simulates POST Request to login a user"""
    with app.test_client() as client:
        user_creds = User.query.get(self.user_id)
        res = client.post("/login", follow_redirects=True, data={'username': user_creds.username,
        'password': "test4444"})
        html = res.get_data(as_text=True)
        self.assertEqual(res.status_code, 200)
        self.assertIn('Credentials verified. You are now logged in.', html)
def test_login_user_invalid(self):
    """Simulates an invalid POST Request to register a user"""
    with app.test_client() as client:
        res = client.post("/login", follow_redirects=True, data={'username':'testuser',
        'password':'pass1',
        'email':'test@email.com',
        'first_name':'test',
        'last_name':'user'})
        html = res.get_data(as_text=True)
```

```
            self.assertIn('Invalid username or password. Please try again', html)
class With_User_Without_Data(TestCase):
    def setUp(self):
        """Register sample user. (No Query or Story data)"""
        #clear previous
        User.query.delete()
        #register user
        user = User.register(
            "testuser", "test4444", "test@test.com", "test", "user")
        db.session.add(user)
        db.session.commit()
        self.user_id = user.id
    def tearDown(self):
        db.session.rollback()
  # def test_home_page(self):
    #     """Simulates non-user selecting headlines tab"""
    #     with app.test_client() as client:
    #         with client.session_transaction() as change_session:
    #             change_session[CURR_USER_KEY] = self.user_id
    #         res = client.get("/")
    #         self.assertEqual(res.status_code, 200)
    #         html = res.get_data(as_text=True)
    #         #tests all categories are appearing
    #         self.assertIn("Business", html)
    #         self.assertIn("Science", html)
    #         self.assertIn("Health", html)
    #         self.assertIn("Entertainment", html)
    #         self.assertIn("Sports", html)
    #         self.assertIn("Technology", html)
    #         #tests html elements and classes
    #         self.assertIn('What can I do with NewsTracker?</h3>', html)
    #         self.assertIn('Non-Users</h4>', html)
```

```
        #         self.assertIn('Get Headlines</h5>', html)
        #         self.assertIn('Search by Keyword</h5>', html)
        #         self.assertIn('Browse by Category</h5>', html)
        #         self.assertIn('Users</h4>', html)
        #         self.assertIn('Detailed Search</h5>', html)
        #         self.assertIn('Sentiment Analysis Tools</h5>', html)
        #         self.assertIn('Saved Stories</h5>', html)
        #         self.assertIn('Upcoming Features</h4>', html)
        #         self.assertIn('Amplified User Features</h5>', html)
        #         self.assertIn('Integration of Twitter API', html)
        #         self.assertIn('<div id="slideshows"', html)
        #         #Asserts that option for dummy account is NOT available
        #         self.assertNotIn('Try NewsTracker with our dummy account"', html)
        #         self.assertIn('<div class="carousel slide" data-ride="carousel">', html)
    def test_nav_bar(self):
        """Asserts that user features are now present in navbar"""
        with app.test_client() as client:
            with client.session_transaction() as change_session:
                change_session[CURR_USER_KEY] = self.user_id
            res = client.get("/headlines")
            self.assertEqual(res.status_code, 200)
            html = res.get_data(as_text=True)
            self.assertIn('My Queries', html)
            self.assertIn('My Stories', html)
            self.assertIn('Detailed Search', html)
            self.assertIn('Logout', html)
            #asserts that non user navigation items are not present
            self.assertNotIn('Login', html)
            self.assertNotIn('Register', html)
    def test_no_user_data(self):
        """Asserts that no stories message is present"""
        with app.test_client() as client:
```

```
            with client.session_transaction() as change_session:
                change_session[CURR_USER_KEY] = self.user_id
            res = client.get("/user/saved")
            self.assertEqual(res.status_code, 200)
            html = res.get_data(as_text=True)
            self.assertIn('<span class="no-stories">', html)
            """Asserts that no queries message are present"""
            self.assertIn('You currently have no saved queries', html)
    def test_logout(self):
        with app.test_client() as client:
            with client.session_transaction() as change_session:
                change_session[CURR_USER_KEY] = self.user_id
            res = client.get("/logout", follow_redirects=True)
            self.assertEqual(res.status_code, 200)
            html = res.get_data(as_text=True)
            self.assertIn('You have successfully logged out', html)
class With_User_With_Data(TestCase):
    def setUp(self):
        """Register sample user and attach data"""
        Query.query.delete()
        Story.query.delete()
        User.query.delete()
        user = User.register(
            "testuser", "test4444", "test@test.com", "test", "user")
        db.session.add(user)
        db.session.commit()

        story = Story(
        headline= 'Leaked documents show the hoops Roblox jumped through to do business in China',
        source= 'Engadget',
        author = 'Stevy Hawks',
        content = "In late June, Blizzard delayed",
```

```
        url= 'https://www.engadget.com/roblox-china-documents-172350532.html',
        image= 'https://s.yimg.com/os/creatr-uploaded-images/2022-07/804e2750-0c3c-11ed-97f7-998944b9b6f4',
        published_at= "2022-05-08",
        id= '75f3410055',
        user_id = user.id)
    db.session.add(story)
        db.session.commit()
        query = Query(
                    name = "United Kingdom",
                    user_id = user.id,
                    source = 'bbc-news',
                    keyword="UK",
                    quantity=12,
                    language='en',
                    default=True,
                    type="detailed_search",
                    sa='polarity',
                    sort_by='relevancy',
                    date_from="2022-07-22",
                    date_to="2022-09-08")
        #todo: change the date_from to automatically be one month out from present day, newstracker free tier only allows for one month in the past
        db.session.add(query)
        db.session.commit()
        self.user_id = user.id
        self.story_id = story.id
    def tearDown(self):
        db.session.rollback()
    def test_saved_stories(self):
        """Simulates user/saved route to ensure demo story in present"""
        with app.test_client() as client:
            with client.session_transaction() as change_session:
```

```
            change_session[CURR_USER_KEY] = self.user_id

        story = Story.query.get(self.story_id)

        res = client.get("/user/saved")

        html = res.get_data(as_text=True)

        self.assertEqual(res.status_code, 200)

        self.assertIn('Leaked documents', html) # portion of headline from sample story attached to class and saved in db

        self.assertIn(story.headline, html)

        self.assertIn(story.content, html)

    def test_default_query_redirect(self):

        """Simulates headlines route to ensure default query works"""

        with app.test_client() as client:

            with client.session_transaction() as change_session:

                change_session[CURR_USER_KEY] = self.user_id

            res = client.get("/headlines")

            self.assertEqual(res.status_code, 302)

    def test_default_query_results(self):

        """Simulates headlines redirect successfull response with stories"""

        with app.test_client() as client:

            with client.session_transaction() as change_session:

                change_session[CURR_USER_KEY] = self.user_id

        res = client.get("/headlines", follow_redirects=True)

        html = res.get_data(as_text=True)

        self.assertEqual(res.status_code, 200)

        self.assertIn('UK', html) # replace this with flash message confirming default q has been selected

    def test_get_detailed_search(self):

        """Simulates search route"""

        with app.test_client() as client:

     with client.session_transaction() as change_session:

                change_session[CURR_USER_KEY] = self.user_id

            res = client.get("/search")

            html = res.get_data(as_text=True)

            self.assertEqual(res.status_code, 200)
```

```
            self.assertIn('<form method="POST" novalidate>', html)
    def test_query_name(self):
        with app.test_client() as client:
            with client.session_transaction() as change_session:
                change_session[CURR_USER_KEY] = self.user_id
            res = client.get("/search") #route is arbitrary since we are checking navbar elements
            html = res.get_data(as_text=True)
            self.assertIn("United Kingdom", html)
    def test_cant_save_dups(self):
        #FIGURE OUT HOW TO TEST ERRORS CORRECTLY
        with app.test_client() as client:
            with client.session_transaction() as change_session:
                change_session[CURR_USER_KEY] = self.user_id
                story = {
                'headline': 'Leaked documents show the hoops Roblox jumped through to do business in
China',
                'source': 'Engadget',
                "description": "something, something",
                'author':  'Stevy Hawks',
                'content':  "In late June, Blizzard delayed",
                'url': 'https://www.engadget.com/roblox-china-documents-172350532.html',
                'image':'https://s.yimg.com/os/creatr-uploaded-images/2022-07/804e2750-0c3c-11ed-97f7-
998944b9b6f4',
                'published_at': "2022-05-08",
                'id': '75f3410055',
                'user_id': self.user_id}
                change_session['results'] = [story]

            test = Story.query.get(self.user_id)
            self.assertRaises(exc.SQLAlchemyError, client.post, f"/story/{self.story_id}/save_story")
            with self.assertRaises(exc.SQLAlchemyError):
                client.post(f"/story/{self.story_id}/save_story")
```

13.2 GitHub & Project Demo Link

GitHub Link:

https://github.com/IBM-EPBL/IBM-Project-39241-1660402222

Project Demo Link:

https://youtu.be/_0IIl7iz9z0