

COSC 4P02: Software Engineering II

Group 9: Final Report

Scrum Master: Jaden Kuhn (JK21PF@BROCKU.CA - 7249683)

Product Owner: Dea Kukuqani (DK16QS@BROCKU.CA – 6196018)

Developers: Shijie Tong (ST20AZ@BROCKU.CA - 7081201)

Nicholas Caruso (YQ20OZ@BROCKU.CA - 7189749)

Thomas Semenak (TS19CP@BROCKU.CA - 6745038)

Dalton Morris (DM20BQ@BROCKU.CA - 7053184)

Chidera Nwana (CN20RQ@BROCKU.CA - 7078124)

Instructor: Naser Ezzati-Jivan

Teaching Assistant: Madeline Janecek

April 27th, 2025

Table of Contents

Table of Contents	2
1. Introduction	5
2. Updated Requirements	5
3. Test Documentation and Results	5
3.1. Web Aggregator (Component)	5
3.1.1. Test Results	5
3.1.2. Test Execution Commands	5
3.1.3. Method Test Cases and Descriptions	6
3.2. HTML Web Scraper (Component).....	8
3.2.1. Test Results	8
3.2.3. Method Test Cases and Descriptions	8
3.3. Website Navigation.....	10
3.3.1. Test Results	10
3.3.3. Method Test Cases and Descriptions	10
3.4. Meta Tags	11
3.4.1. Test Results	11
3.4.2. Test Execution Commands	12
3.4.3. Method Test Cases and Descriptions	12
3.5. Social Media Share Buttons.....	13
3.5.1. Test Results	13
3.5.3. Method Test Cases and Descriptions	13
3.6. Post Editing.....	15
3.6.1. Test Results	15
3.6.3. Method Test Cases and Descriptions	15
3.7. Content Generation (Unit)	17
3.7.1. Test Results	17
3.7.2. Test Execution Commands	17
3.7.3. Method Test Cases and Descriptions	17

3.8 Registration	18
3.8.1. Test Execution Commands	18
3.8.2. Method Test Cases and Descriptions	18
Result Quality:	19
3.9. Login	19
3.9.1. Test Execution Commands	19
3.9.2. Method Test Cases and Descriptions	19
3.10 Scalability	20
3.10.1. Test Results	20
3.10.2. Test Execution Commands	22
3.10.3. Method Test Cases and Descriptions	22
4. User Manual.....	23
4.1. Accessing the Website.....	23
4.2. Registering an Account / Logging In	23
4.3. Generating Content	24
4.4. Saving and Managing Posts	25
4.5. Logging Out.....	28
5. Project Contributions.....	28
5.1. Jaden Kuhn	28
5.1.1. Contributions.....	28
5.1.2. GitHub Commits	29
5.2. Dea Kukuqani.....	29
5.2.1. Contributions.....	29
5.2.2. GitHub Commits	30
5.3. Nicholas Caruso.....	30
5.3.1. Contributions.....	30
5.3.2. GitHub Commits	31
5.4. Shijie Tong	31
5.4.1. Contributions.....	31

5.4.2. GitHub Commits	32
5.5. Thomas Semenak	32
5.5.1. Contributions.....	32
5.5.2. GitHub Commits	33
5.6. Dalton Morris	33
5.6.1. Contributions.....	33
5.6.2. GitHub Commits	34
5.7. Chidera Nwana	34
5.7.1. Contributions.....	34
5.7.2. GitHub Commits	34
Appendix A. Website and GitHub Repository Links	35
Appendix B. Meeting Minutes	35
Appendix D. Sprint Burndown Charts (Jira)	39
Appendix E. GitHub Screenshots	39

1. Introduction

The AI Powered Newsletter and Social Media Content Generator project aims to make content creation easier and more efficient for newsletters and social media. The goal is to automate tasks like gathering relevant content, summarizing it, and formatting it in a way that is useful for users. By using machine learning, data scraping, and a user-friendly interface, the platform will allow users to generate high-quality content with minimal effort.

This report outlines the final set of requirements for the software, test documentation and results, a user manual for the software, and the contributions of each group member throughout the project.

2. Updated Requirements

Throughout the project, we successfully developed the majority of the features outlined in our initial release plan. This included building dynamic article aggregation from external sources, implementing an automated keyword-based tagging system, providing user authentication (signup/login), and creating a manual categorization dashboard to enhance user control over article organization. However, we were unable to complete the scheduling functionality for automatic content generation due to time limitations and technical complexity. By focusing on delivering a stable, user-friendly, and fully operational system, we ensured that the essential goals of the platform were met while postponing secondary features that could be added in future development stages.

3. Test Documentation and Results

3.1. Web Aggregator (Component)

3.1.1. Test Results

```
test.py::test_index PASSED [ 14%]
test.py::test_search PASSED [ 28%]
test.py::test_MissingQuery PASSED [ 42%]
test.py::test_InvalidSearch PASSED [ 57%]
test.py::test_history PASSED [ 71%]
test.py::test_SearchHistoryFileHandling PASSED [ 85%]
test.py::test_APIFailure PASSED [100%]
===== 7 passed in 9.64s =====
```

3.1.2. Test Execution Commands

```
python -m pytest test.py -v
```

```
python -m pytest test.py -s -v (this will display the messages and content printed in the tests when run)
```

```
python -m pytest test.py -k "test_index" -s -v
python -m pytest test.py -k "test_search" -s -v
python -m pytest test.py -k "test_MissingQuery" -s -v
python -m pytest test.py -k "test_InvalidSearch" -s -v
python -m pytest test.py -k "test_history" -s -v
python -m pytest test.py -k "test_SearchHistoryFileHandling" -s -v
python -m pytest test.py -k "test_APIFailure" -s -v
```

3.1.3. Method Test Cases and Descriptions

test_index:

Test Case 1: Testing the index of the aggregator. This will confirm that the index page is reachable and correct. This method will test the index page of the Flask app. It sends a GET request to the root URL ('/') and checks if the response status code is 200 (OK) and if the response data contains the text "Keyword Search Aggregator".

Expected Result: Pass. The page should load successfully and display the "Keyword Search Aggregator" text.

test_search:

Test Case 2: Testing the search functionality of the `/search` route with a query parameter. This method verifies that the search functionality works when the user submits a query. The test sends a GET request to the `/search` endpoint with the query parameter, checks that the status code is 200 (OK), and prints the first 10 search result URLs from both Google Search and Google News. It also ensures that the response contains the expected "organic_results" and "news_results" for Google Search and Google News respectively.

Expected Result: Pass. The search functionality should return valid search results, and the results should contain the appropriate URLs.

test_MissingQuery:

Test Case 3: Testing the aggregator when the query parameter is missing from the `/search` route. This method verifies that the `/search` route returns a 400-status code and the appropriate error message when no query parameter is provided. The test sends a GET request to the `/search` endpoint without the query parameter and checks that the response status code is 400 (Bad Request). It also ensures that the response contains the correct error message indicating that the query parameter 'q' is missing.

Expected Result: Pass. The route should return a 400-status code and an error message stating that the 'q' parameter is missing.

test_InvalidSearch:

Test Case 4: Testing the `/search` route with an invalid keyword ("!!!"). This method verifies that when an invalid keyword "!!!" is provided, the API returns no search results. It checks that both the Google Search and Google News sections are empty in the response.

Expected Result: Pass. The search results should be empty for both Google Search and Google News.

test_history:

Test Case 5: Testing the history functionality of the `/history` route to retrieve stored search history. This method verifies that the history functionality works as expected. It sends a GET request to the `/history` route, checks that the response status code is 200 (OK), and prints the search history entries along with the associated URLs and sources. It also ensures that the response is a list, even if it is empty.

Expected Result: Pass. The history functionality should return valid search history entries, and the response should be a list of records with keywords and results.

test_SearchHistoryFileHandling:

Test Case 6: Testing the creation and updating of "search_history.json" file. This test ensures that when a search is performed and "search_history.json" does not exist, it is created. Additionally, it checks that the file is updated properly when new searches are performed.

Expected Result: Pass. The file should be created if it doesn't exist and updated with the new search history when new searches are added.

test_APIFailure:

Test Case 7: Testing the `/search` route when the SerpAPI fails to respond. This method verifies that when the SerpAPI service fails, the `/search` endpoint handles the failure gracefully by returning a 500-status code and an appropriate error message. It uses a mock request to simulate a failure when contacting the SerpAPI service.

Expected Result: Pass. The `/search` route should return a 500-status code and an error message indicating the failure to contact the SerpAPI or another internal error.

3.2. HTML Web Scraper (Component)

3.2.1. Test Results

```

web_scrape_clean_html_test.py::test_clean_text PASSED [ 12%]
web_scrape_clean_html_test.py::test_scrape_and_clean PASSED [ 25%]
web_scrape_clean_html_test.py::test_ValidURL1 PASSED [ 37%]
web_scrape_clean_html_test.py::test_ValidURL2 PASSED [ 50%]
web_scrape_clean_html_test.py::test_InvalidURL PASSED [ 62%]
web_scrape_clean_html_test.py::test_NoMainContent PASSED [ 75%]
web_scrape_clean_html_test.py::test_NonEnglishContent PASSED [ 87%]
web_scrape_clean_html_test.py::test_TagsInContent PASSED [100%]
===== 8 passed in 6.86s =====

```

3.2.2. Test Execution Commands

```
python -m pytest web_scrape_clean_html_test.py -v
```

python -m pytest web_scrape_clean_html_test.py -s -v (this will display the messages and content printed in the tests when run)

```
python -m pytest web_scrape_clean_html_test.py -k "test_clean_text" -s -v
```

```
python -m pytest web_scrape_clean_html_test.py -k "test_scrape_and_clean" -s -v
```

```
python -m pytest web_scrape_clean_html_test.py -k "test_ValidURL1" -s -v
```

```
python -m pytest web_scrape_clean_html_test.py -k "test_ValidURL2" -s -v
```

```
python -m pytest web_scrape_clean_html_test.py -k "test_InvalidURL" -s -v
```

```
python -m pytest web_scrape_clean_html_test.py -k "test_NoMainContent" -s -v
```

```
python -m pytest web_scrape_clean_html_test.py -k "test_NonEnglishContent" -s -v
```

```
python -m pytest web_scrape_clean_html_test.py -k "test_TagsInContent" -s -v
```

3.2.3. Method Test Cases and Descriptions

test_clean_text:

Test Case 1: Testing the “clean_text” method of the web scraper independently. This will confirm base functionality of the “clean_text” method. The method will replace newline and carriage return characters with spaces, replace quotes with pipe characters, and reduce multiple spaces to a single space.

Expected Result: Pass. The cleaned text should match the expected output.

test_scrape_and_clean:

Test Case 2: Testing the “scrape_and_clean” method with a valid URL independently. For this test, not a Wikipedia page, but a CNN page. This method will test the “scrape_and_clean” method on a valid URL. The method should scrape the page, clean the content, and return the page content.

Expected Result: Pass. The content scraped from the URL is a valid string with only permitted characters.

test_ValidURL1:

Test Case 3: Testing the “get_clean_content” method with a valid URL. For this test, Wikipedia. This method will test the “get_clean_content” method of the web scraper with a valid Wikipedia URL. The URL will be scraped, and the content will be cleaned. The content will be checked to ensure that it is a string and that it is not empty.

Expected Result: Pass. The content scraped from the URL is a valid string and is significantly long.

test_ValidURL2:

Test Case 4: Testing the “get_clean_content” method with a valid URL. For this test, CBS Sports. This method will test the “get_clean_content” method of the web scraper with a valid CBS Sports URL. The URL will be scraped, and the content will be cleaned. The content will be checked to ensure that it is a string and that it is not empty.

Expected Result: Pass. The content scraped from the URL is a valid string and is significantly long.

test_InvalidURL:

Test Case 5: Testing the “get_clean_content” method with an invalid URL. For this test, www.cosc4p02group9fakeurl.com. This method will test the “get_clean_content” method of the web scraper with a custom invalid URL. The web scraper returns "Error:" if the URL is invalid, so the test will check for this string in the content. If the string is found, the test will pass, the URL was invalid.

Expected Result: Pass. "Error:" will exist in the content because the URL is invalid.

test_NoMainContent:

Test Case 6: Testing the “get_clean_content” method with a valid URL but no main content. For this test, <https://www.example.com/>. This method will test the “get_clean_content” method on a valid URL that doesn't have any main content. The page may be blank, or there may be no main content. The web scraper returns "Main content not found." if the main content is not found, so the test will check for this string in the content. If the string is found, the test will pass, the URL had no main content.

Expected Result: Pass. "Main content not found." will exist in the content because the URL has no main content.

test_NonEnglishContent:

Test Case 7: Testing the “get_clean_content” method with a valid URL that has content not in English. This method will test the “get_clean_content” method on a page with content in a language other than English. The web scraper should not include non-English lines and return only English text if present.

Expected Result: Pass. The content should not contain any lines that contain other languages.

test_TagsInContent:

Test Case 8: Testing the “get_clean_content” method with a valid URL that contains HTML tags embedded in the content. Citations, Footnotes, etc. This method will test the “get_clean_content” method on a page with HTML tags within the content. The web scraper should remove the HTML tags and return only readable text. Wikipedia articles contain many HTML tags, so this test could be performed on other Wikipedia articles to showcase the removal of embedded tags. Other sites also have tags, but Wikipedia is a good example.

Expected Result: Pass. The content should only contain clean text.

3.3. Website Navigation

3.3.1. Test Results

```
header_navigation_test.py::test_header_present
DevTools listening on ws://127.0.0.1:57895/devtools/browser/cd3f4e58-d97f-4135-8979-0c0555a006b7
PASSED [ 25%]
header_navigation_test.py::test_header_present_after_login PASSED [ 50%]
header_navigation_test.py::test_header_present_after_logout PASSED [ 75%]
header_navigation_test.py::test_header_functionality PASSED [100%]
===== 4 passed in 31.72s =====
```

3.3.2. Test Execution Commands

```
python -m pytest header_navigation_test.py -v
```

```
python -m pytest header_navigation_test.py -v -s (this will display the messages and content printed in the tests when run)
```

```
python -m pytest header_navigation_test.py -k "test_header_present" -s -v
```

```
python -m pytest header_navigation_test.py -k "test_header_present_after_login" -s -v
```

```
python -m pytest header_navigation_test.py -k "test_header_present_after_logout" -s -v
```

```
python -m pytest header_navigation_test.py -k "test_header_functionality" -s -v
```

3.3.3. Method Test Cases and Descriptions

test_header_present:

Test Case 1: Testing the presence of the header. This will confirm that the header is present on the page for navigation. This method will check if the header is present on the

page. It uses the Selenium WebDriver (predefined as a fixture above) to navigate to our URL and check for the presence of the buttons by the text of the buttons.

Expected Result: Pass. The header should be present on the page.

test_header_present_after_login:

Test Case 2: Testing the presence of the header after login. This will confirm that the header is present on the page for navigation after login, and the proper pages have appeared/disappeared. This method will check if the header is present on the page after login, and check that the proper pages have appeared/disappeared. It uses the Selenium WebDriver (predefined as a fixture above) to navigate to our URL and check for the presence of the buttons by the text of the buttons.

Expected Result: Pass. The header should be present on the page after login with the correct pages and buttons.

test_header_present_after_logout:

Test Case 3: Testing the presence of the header after logout. This will confirm that the header is present on the page for navigation after logout, and the proper pages have appeared/disappeared. This method will check if the header is present on the page after logout, and check that the proper pages have appeared/disappeared. It uses the Selenium WebDriver (predefined as a fixture above) to navigate to our URL and check for the presence of the buttons by the text of the buttons.

Expected Result: Pass. The header should be present on the page after logout with the correct pages and buttons.

test_header_functionality:

Test Case 4: Testing the functionality of the header buttons. This will confirm that the header buttons are functional and navigate to the correct pages. This method will check if the header buttons are functional and navigate to the correct pages. It uses the Selenium WebDriver (predefined as a fixture above) to navigate to our URL and check for the presence of elements relevant to the different pages of our site.

Expected Result: Pass. The header buttons should be functional and navigate to the correct pages.

3.4. Meta Tags

3.4.1. Test Results

```
meta_tags_test.py::test_open_graph_meta_tags PASSED [100%]
1 passed in 0.79s
```

When and how we last scraped the URL

Time Scraped	3 seconds ago Scrape Again
Response Code	200
Fetches URL	https://group9website-gth5dkhfajb4d4g9.canadaeast-01.azurewebsites.net/
Canonical URL	https://group9website-gth5dkhfajb4d4g9.canadaeast-01.azurewebsites.net/index.php 0 likes, shares and comments (More Info)
Redirect Path	Input URL → https://group9website-gth5dkhfajb4d4g9.canadaeast-01.azurewebsites.net/ og:url Meta Tag → https://group9website-gth5dkhfajb4d4g9.canadaeast-01.azurewebsites.net/index.php
Link Preview	<div> GROUP9WEBSITE-GTH5DKHFJAB4D4G9.CANADAEAST-01.AZUREWEBSITES.NET SmartSummaries: AI-Powered Newsletter & Social Media Content Generator Create engaging newsletters and social media posts effortlessly with AI-driven... </div>

This screenshot was obtained from <https://developers.facebook.com/tools/debug/>

3.4.2. Test Execution Commands

```
python -m pytest meta_tags_test.py -v
```

```
python -m pytest meta_tags_test.py -v -s (this will display the messages and content printed in the tests when run)
```

```
python -m pytest meta_tags_test.py -k "test_open_graph_meta_tags" -s -v
```

3.4.3. Method Test Cases and Descriptions

test_open_graph_meta_tags:

Test Case 1: Testing the meta tags are present on the hosted website. This will confirm the meta tags are present and correct. The method will scrape the page source of the website and check for the presence of Open Graph meta tags.

Expected Result: Pass. The meta tags are present and correct.

3.5. Social Media Share Buttons

3.5.1. Test Results

```
social_media_sharing_test.py::test_share_buttons_present
DevTools listening on ws://127.0.0.1:55619/devtools/browser/eef65374-89a7-42d5-867f-a2d71b42597f
PASSED [ 16%]
social_media_sharing_test.py::test_share_buttons_clickable PASSED [ 33%]
social_media_sharing_test.py::test_facebook_button_url PASSED [ 50%]
social_media_sharing_test.py::test_x_button_url PASSED [ 66%]
social_media_sharing_test.py::test_email_button_url Created TensorFlow Lite XNNPACK delegate for CPU.
Attempting to use a delegate that only supports static-sized tensors with a graph that has dynamic-sized tensors (tensor#1 is a dynamic-sized tensor).
PASSED [ 83%]
social_media_sharing_test.py::test_x_template PASSED [100%]
6 passed in 18.08s
```

3.5.2. Test Execution Commands

```
python -m pytest social_media_sharing_test.py -v
```

```
python -m pytest social_media_sharing_test.py -v -s (this will display the messages and
content printed in the tests when run)
```

```
python -m pytest social_media_sharing_test.py -k "test_share_buttons_present" -s -v
```

```
python -m pytest social_media_sharing_test.py -k "test_share_buttons_clickable" -s -v
```

```
python -m pytest social_media_sharing_test.py -k "test_facebook_button_url" -s -v
```

```
python -m pytest social_media_sharing_test.py -k "test_x_button_url" -s -v
```

```
python -m pytest social_media_sharing_test.py -k "test_email_button_url" -s -v
```

```
python -m pytest social_media_sharing_test.py -k "test_x_template" -s -v
```

3.5.3. Method Test Cases and Descriptions

test_share_buttons_present:

Test Case 1: Testing the presence of the social media share buttons. This will confirm that the buttons are present on the page. This method will check if the share buttons for Facebook, Twitter, and Email are present on the page. It uses the Selenium WebDriver (predefined as a fixture above) to navigate to our URL and check for the presence of the share buttons by their class names.

Expected Result: Pass. The share buttons should be present on the page.

test_share_buttons_clickable:

Test Case 2: Testing the clickability of the social media share buttons. This will confirm that the buttons are clickable while on the page. This method will check if the share buttons for Facebook, Twitter, and Email are clickable on the page. It uses the Selenium WebDriver (predefined as a fixture above) to navigate to our URL and check for the enabled share buttons by their class names.

Expected Result: Pass. The share buttons should be clickable on the page.

test_facebook_button_url:

Test Case 3: Testing that the Facebook share button redirects to the correct Facebook Post URL. This will confirm that the opened URL is correct when the Facebook share button is clicked. This method will check if the Facebook share button redirects to the correct Facebook Post URL when clicked. It uses the Selenium WebDriver (predefined as a fixture above) to navigate to our URL and click the Facebook share button. It then checks if the current (redirect) URL contains "facebook.com" to confirm that the redirect was successful.

Expected Result: Pass. The Facebook share button should redirect to the correct Facebook Post URL.

test_x_button_url:

Test Case 4: Testing that the X share button redirects to the correct X Post URL. This will confirm that the opened URL is correct when the X share button is clicked. This method will check if the X share button redirects to the correct X Post URL when clicked. It uses the Selenium WebDriver (predefined as a fixture above) to navigate to our URL and click the X share button. It then checks if the current (redirect) URL contains "x.com" to confirm that the redirect was successful.

Expected Result: Pass. The X share button should redirect to the correct X Post URL.

test_email_button_url:

Test Case 5: Testing that the email share button redirects to the correct blank URL, and then closes this URL. This will confirm that the redirect window opens and closes correctly when the email share button is clicked. This method will check if the email share button redirects to the about:blank URL when clicked. It uses the Selenium WebDriver (predefined as a fixture above) to navigate to our URL and click the email share button. It then checks if the current (redirect) URL contains "about:blank" to confirm that a new blank window opens and closes correctly. It also checks if the number of windows is the same as before clicking the email button, which would confirm that the redirect was successful, since Selenium does not check for non-browser windows. This is a workaround to test this functionality.

Expected Result: Pass. The email share button should redirect to the correct about:blank URL and then close this URL.

test_x_template:

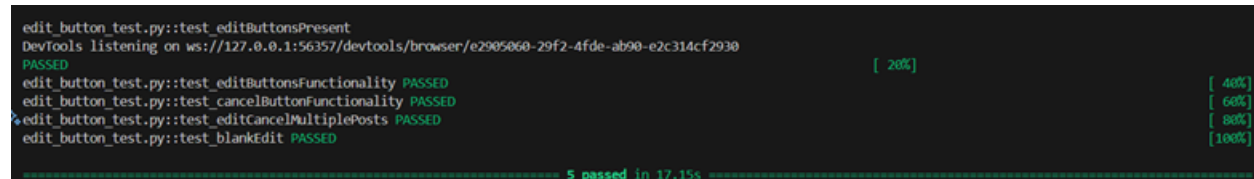
Test Case 6: Testing the pre-populated X template when the share button is selected. This will confirm that the X template is correct. This method will check if the X template is pre-

populated with the correct text when the X share button is clicked. It uses the Selenium WebDriver (predefined as a fixture above) to navigate to our URL and check for the pre-populated X template by executing JavaScript to retrieve the text of the X template.

Expected Result: Pass. The X template should be pre-populated with the correct text.

3.6. Post Editing

3.6.1. Test Results



```
edit_button_test.py::test_editButtonsPresent
DevTools listening on ws://127.0.0.1:56357/devtools/browser/e2905060-29f2-4fde-ab90-e2c314cf2930
PASSED [ 20%]
edit_button_test.py::test_editButtonsFunctionality PASSED [ 40%]
edit_button_test.py::test_cancelButtonFunctionality PASSED [ 60%]
edit_button_test.py::test_editCancelMultiplePosts PASSED [ 80%]
edit_button_test.py::test_blankEdit PASSED [100%]
===== 5 passed in 17.15s =====
```

3.6.2. Test Execution Commands

```
python -m pytest edit_button_test.py -v
```

```
python -m pytest edit_button_test.py -v-s (this will display the messages and content
printed in the tests when run)
```

```
python -m pytest edit_button_test.py -k "test_editButtonsPresent" -s -v
```

```
python -m pytest edit_button_test.py -k "test_editButtonsFunctionality" -s -v
```

```
python -m pytest edit_button_test.py -k "test_cancelButtonFunctionality" -s -v
```

```
python -m pytest edit_button_test.py -k "test_editCancelMultiplePosts" -s -v
```

```
python -m pytest edit_button_test.py -k "test_blankEdit" -s -v
```

3.6.3. Method Test Cases and Descriptions

test_editButtonsPresent:

Test Case 1: Testing the presence of the edit button(s). This will confirm that the button(s) are present on the page. This method will check if the edit button(s) are present on the page. It uses the Selenium WebDriver (predefined as a fixture above) to navigate to our URL and check for the presence of the edit button(s) by their class names.

Expected Result: Pass. The edit button(s) should be present on the page.

test_editButtonsFunctionality:

Test Case 2: Testing the functionality of the edit button(s). This will confirm that the button(s) work as intended. This method will check if the edit button(s) work properly. It uses the Selenium WebDriver (predefined as a fixture above) to navigate to our URL and

check that the edit button(s) function as expected when clicked. It will click the first edit button, edit the post content, and then save the changes.

Expected Result: Pass. The edit button(s) should work properly.

test_cancelButtonFunctionality:

Test Case 3: Testing the functionality of the cancel button. This will confirm that the cancel button works as intended. This method will check if the cancel button works properly. It uses the Selenium WebDriver (predefined as a fixture above) to navigate to our URL and check that the cancel button functions as expected when clicked. It will click the first edit button, edit the post content, and then cancel the changes.

Expected Result: Pass. The cancel button should work properly. The content should not be changed after canceling.

test_editCancelMultiplePosts:

Test Case 4: Testing the functionality of editing multiple posts. This will confirm that only the edited post is updated, while the other remains unchanged. This method will check if the edit button(s) work properly when editing multiple posts. It uses the Selenium WebDriver (predefined as a fixture above) to navigate to our URL and check that the edit button(s) function as expected when clicked. It will click the first edit button, edit the post content, and then save the changes. It will also click the second edit button but not make any changes to the post content.

Expected Result: Pass. The edited post should be updated, while the other post should remain unchanged.

test_blankEdit:

Test Case 5: Testing the functionality of the save button when the text area is blank. This will confirm that the button does not save when the text area is blank. This method will check if the save button works properly when the text area is blank. It uses the Selenium WebDriver (predefined as a fixture above) to navigate to our URL and check that the save button functions as expected when clicked. It will click the first edit button, clear the post content, and then save the changes.

Expected Result: Pass. The save button should not save the changes when the textarea is blank.

3.7. Content Generation (Unit)

3.7.1. Test Results

```
PS C:\xampp\htdocs\Ap02\Ap02GroupProject\tests\Generation Test> python -m pytest test.py -s -v
===== test session starts =====
platform win32 -- Python 3.10.9, pytest-7.3.1, pluggy-1.0.0 -- C:\Users\lions\AppData\Local\Programs\Python\Python310\python.exe
cachedir: .pytest_cache
rootdir: C:\xampp\htdocs\Ap02\Ap02GroupProject\tests\Generation Test
plugins: anyio-4.4.0, hydra-core-1.0.7, mock-3.10.0
collected 4 items

test.py::test_basic_generation Generation result: (A did you know Canada is more than just maple syrup and hockey? Spanning from the Atlantic to the Pacific and up into the Arctic, it's the world's second-largest country with the longest coastline! 🇨🇦 Home to over 41 million people, Canada's story starts with Indigenous communities who've thrived here for millennia, followed by French and British settlers. Today, it's a vibrant mosaic of cultures, languages, and traditions—officially bilingual (hello, English and French!) and celebrated for its inclusivity.

From the rocky peaks of Newfoundland to the boreal forests of the Yukon, Canada's landscapes are as diverse as its people. Fun fact: The maple leaf isn't just a symbol—it's a national obsession! 🍁 And while peace, order, and good governance are its bedrock, Canadians also know how to embrace the cold (hello, ice hockey and Northern lights!).

Proud of its universal healthcare, high quality of life, and role as a global peacekeeper, Canada keeps making waves—whether through groundbreaking tech, iconic artists, or welcoming newcomers. Next time you think Canada, think BIG: big skies, big hearts, and a big commitment to diversity. 🌟)
PASSED
test.py::test_invalid_input PASSED
test.py::test_result_quality Generated content length: 1892 characters
PASSED
test.py::test_concurrent_generation Concurrent test results: {'Canada': True, 'Germany': True, 'USA': True, 'France': True, 'UK': True}
PASSED

===== 4 passed in 71.59s (0:01:11) =====
```

3.7.2. Test Execution Commands

`python -m pytest test.py -s -v`

3.7.3. Method Test Cases and Descriptions

Base function:

Case 1: The basic functionality of the system was tested through the normal process of the website. By providing the website with three keywords, the API searches for articles and generates the ones that the user needs.

Expected result: Pass. Generation functionality should work properly and return valid results.

Invalid input:

Case 2: This part is tested by inputting or using special characters or topics/keywords that are too long.

Expected result: Pass. Generation functionality should handle invalid inputs properly.

Result quality:

Case 3: This part is carried out by checking the completeness, coherence and length of the produced content. The main inspection content is to check whether the length is reasonable.

Expected result: Pass. Generation results should meet post format requirements.

Concurrent:

Case 4: This part of the test examines whether the generator can accommodate multiple users to use it simultaneously. Conducted the test by generating multiple pieces of content simultaneously.

Expected result: Pass. System should handle concurrent requests properly.

3.8 Registration

3.8.1. Test Execution Commands

php tests/Register Test/RegisterTest.php

3.8.2. Method Test Cases and Descriptions

Base function:

Case 1:

The basic functionality of the registration page was tested by navigating to /register.php and verifying the presence of the registration form, including the input fields for username, password, and confirm password.

Expected result: Pass.

The registration page loads correctly with all required input fields.

Valid input:

Case 2:

A new user registration was tested by submitting a randomly generated valid username along with a strong password.

Expected result: Pass.

The system successfully registers the new user and displays a "Registration successful" message.

Invalid input:

Case 3:

(Not covered in this version but recommended for future testing.)

Testing with mismatched passwords or empty fields should return error messages.

Expected result: Pass.

The system should handle invalid submissions gracefully with clear error prompts.

Result Quality:

All registration tests passed successfully.

The system accurately handled user registration with valid data.

3.9. Login

3.9.1. Test Execution Commands

php tests/Login Test/LoginTest.php

3.9.2. Method Test Cases and Descriptions

Base function:

Case 1:

The basic functionality of the login page was verified by navigating to /login.php and ensuring that the login form displayed fields for username and password.

Expected result: Pass.

The login page loads properly with the expected input fields.

Invalid input:

Case 2:

An invalid login attempt was made using a random, non-existent username and incorrect password.

Expected result: Pass.

The system correctly shows an "Invalid username or password" error message.

Valid input:

Case 3:

(Not covered in this version but recommended for future testing.)

Logging in with correct credentials should lead to a successful login and dashboard access.

Expected result: Pass.

The user should be authenticated and redirected appropriately.

Result Quality:

All login tests passed successfully.

The system correctly handled failed login attempts and protected against unauthorized access.

3.10 Scalability

3.10.1. Test Results

```

PS C:\Users\tseme> pytest test_concurrent_users.py -s -v
===== test session starts =====
platform win32 -- Python 3.12.1, pytest-8.0.0, pluggy-1.4.0 -- C:\Users\tseme\AppData\Local\Programs\Python\Python312\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\tseme
collected 1 item

test_concurrent_users.py::test_concurrent_100_users [admin-2] Saved test post.
[admin5-13] Saved test post.
[admin-14] Saved test post.
[admin5-64] Saved test post.
[admin-35] Saved test post.
[admin-53] Saved test post.
[tom-42] Saved test post.
[admin5-46] Saved test post.
[admin5-16] Saved test post.
[admin-5] Saved test post.
[admin5-52] Saved test post.
[admin5-4] Saved test post.
[tom-33] Saved test post.
[admin5-10] Saved test post.
[tom-54] Saved test post.
[admin5-25] Saved test post.
[admin5-1] Saved test post.
[admin-20] Saved test post.
[tom-39] Saved test post.
[tom-51] Saved test post.
[tom-9] Saved test post.
[tom-18] Saved test post.
[admin-29] Saved test post.
[admin-62] Saved test post.
[tom-66] Saved test post.
[admin-50] Saved test post.
[admin-32] Saved test post.
[admin5-22] Saved test post.
[tom-21] Saved test post.
[tom-6] Saved test post.
[tom-45] Saved test post.
[admin-65] Saved test post.
[admin5-34] Saved test post.
[admin5-73] Saved test post.
[admin5-31] Saved test post.
[tom-3] Saved test post.
[admin-8] Saved test post.
[admin5-55] Saved test post.
[admin5-37] Saved test post.
[tom-30] Saved test post.
[admin5-58] Saved test post.
[admin5-43] Saved test post.
[tom-57] Saved test post.
[admin5-7] Saved test post.
[admin-74] Saved test post.

```

```
[admin-95] Saved test post.  
[admin-83] Saved test post.  
[tom-15] Saved test post.  
[admin-92] Saved test post.  
[admin5-67] Saved test post.  
[admin-80] Saved test post.  
[tom-75] Saved test post.  
[admin5-88] Saved test post.  
[tom-72] Saved test post.  
[admin5-91] Saved test post.  
[tom-84] Saved test post.  
[tom-96] Saved test post.  
[tom-87] Saved test post.  
[tom-99] Saved test post.
```

All 100 users completed in 108.62 seconds

```
User 1: Success  
User 2: Success  
User 3: Success  
User 4: Success  
User 5: Success  
User 6: Success  
User 7: Success  
User 8: Success  
User 9: Success  
User 10: Success  
User 11: Success  
User 12: Success  
User 13: Success  
User 14: Success  
User 15: Success  
User 16: Success  
User 17: Success  
User 18: Success  
User 19: Success  
User 20: Success  
User 21: Success  
User 22: Success  
User 23: Success  
User 24: Success  
User 25: Success  
User 26: Success  
User 27: Success  
User 28: Success  
User 29: Success  
User 30: Success  
User 31: Success  
User 32: Success  
User 33: Success  
User 34: Success  
User 35: Success  
User 36: Success  
User 37: Success
```

```
User 59: Success
User 60: Success
User 61: Success
User 62: Success
User 63: Success
User 64: Success
User 65: Success
User 66: Success
User 67: Success
User 68: Success
User 69: Success
User 70: Success
User 71: Success
User 72: Success
User 73: Success
User 74: Success
User 75: Success
User 76: Success
User 77: Success
User 78: Success
User 79: Success
User 80: Success
User 81: Success
User 82: Success
User 83: Success
User 84: Success
User 85: Success
User 86: Success
User 87: Success
User 88: Success
User 89: Success
User 90: Success
User 91: Success
User 92: Success
User 93: Success
User 94: Success
User 95: Success
User 96: Success
User 97: Success
User 98: Success
User 99: Success
User 100: Success
PASSED
===== 1 passed in 109.02s (0:01:49) =====
PS C:\Users\tseme>
```

3.10.2. Test Execution Commands

`pytest test_concurrent_users.py -s -v`

3.10.3. Method Test Cases and Descriptions

Case 1 tests whether a user can successfully log into the portal using a valid username and password without errors.

Expected result, pass a user can login so long as the username and password are valid

Case 2 verifies that after logging in, a user can access the dashboard page without encountering any server errors or session issues.

Expected result, pass the logged in user can access their own personal dashboard

Case 3 simulates the user submitting a “generate” request, checking if the server accepts input without full content generation delays.

Expected result, pass the user can submit a generate request.

Case 4 tests the ability to save a generated post ensuring that the post is correctly processed and saved.

Expected result, pass the user can save a generated post.

Case 5 validates that the saved post properly appears in the user’s dashboard, confirming that session handling and database interactions are functioning correctly.

Expected result, pass the saved post appears on the users personalized dashboard.

This test focuses on validating backend stability, session management correctness, and API endpoint reliability under concurrent usage by simulating multiple user workflows in parallel. It does not simulate realistic network latency or backend generation times.

4. User Manual

4.1. Accessing the Website

The website can be found at group9portal-eehbdbxbhcgftezez.canadaeast-01.azurewebsites.net.

Browser Requirements: N/A

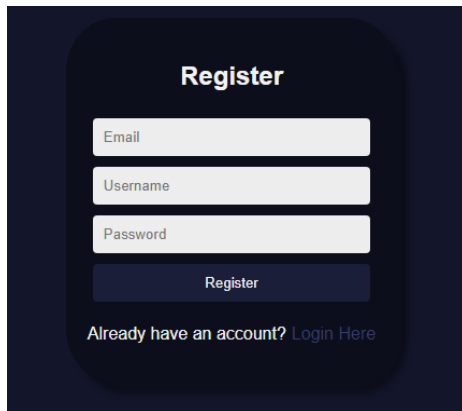
4.2. Registering an Account / Logging In

To register for an account, click on the Register page using the navigation bar.



On both the Dashboard and Login pages there are also redirects to the Register page (if one is not logged in already).

On the Register page, a valid email address, a unique username and a password are needed to register an account.

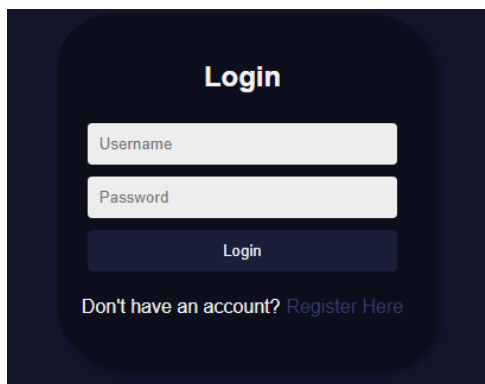
A dark-themed registration form titled "Register". It contains three input fields for "Email", "Username", and "Password", followed by a "Register" button. At the bottom, there is a link: "Already have an account? [Login Here](#)".

Once registered, to login, click on the Login page using the navigation bar.



On both the Dashboard and Register pages there are also redirects to the Login page (if one is not logged in already).

On the Login page, enter a registered Username and Password to login.

A dark-themed login form titled "Login". It contains two input fields for "Username" and "Password", followed by a "Login" button. At the bottom, there is a link: "Don't have an account? [Register Here](#)".

After logging in, one will be redirected to the Dashboard page and the navigation bar will show Profile and Logout buttons indicating the login was successful.

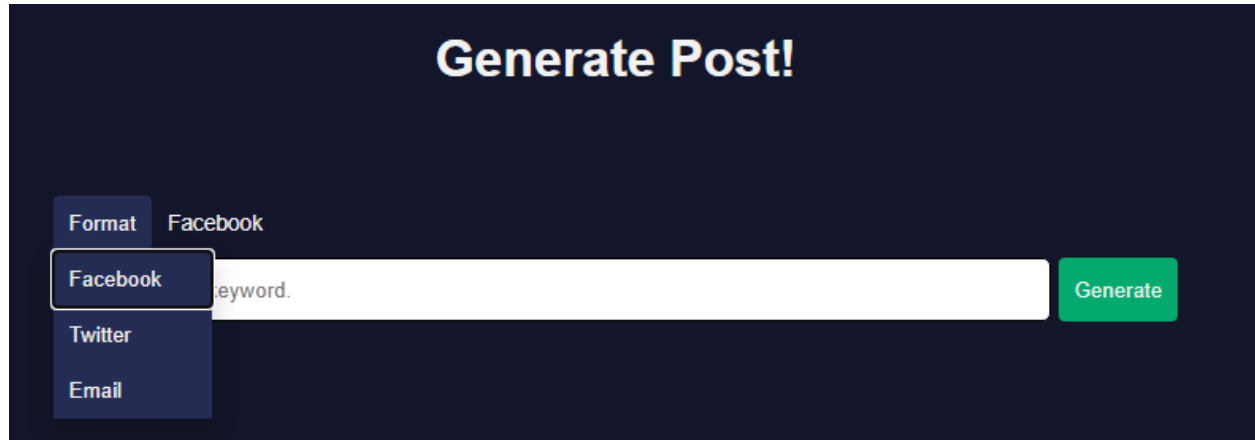


4.3. Generating Content

To generate a post, first go to the Generate page using the navigation bar.



To select a specific post format (Facebook, Twitter or Email), hover over (or click) the Format button and select an option from the dropdown menu.

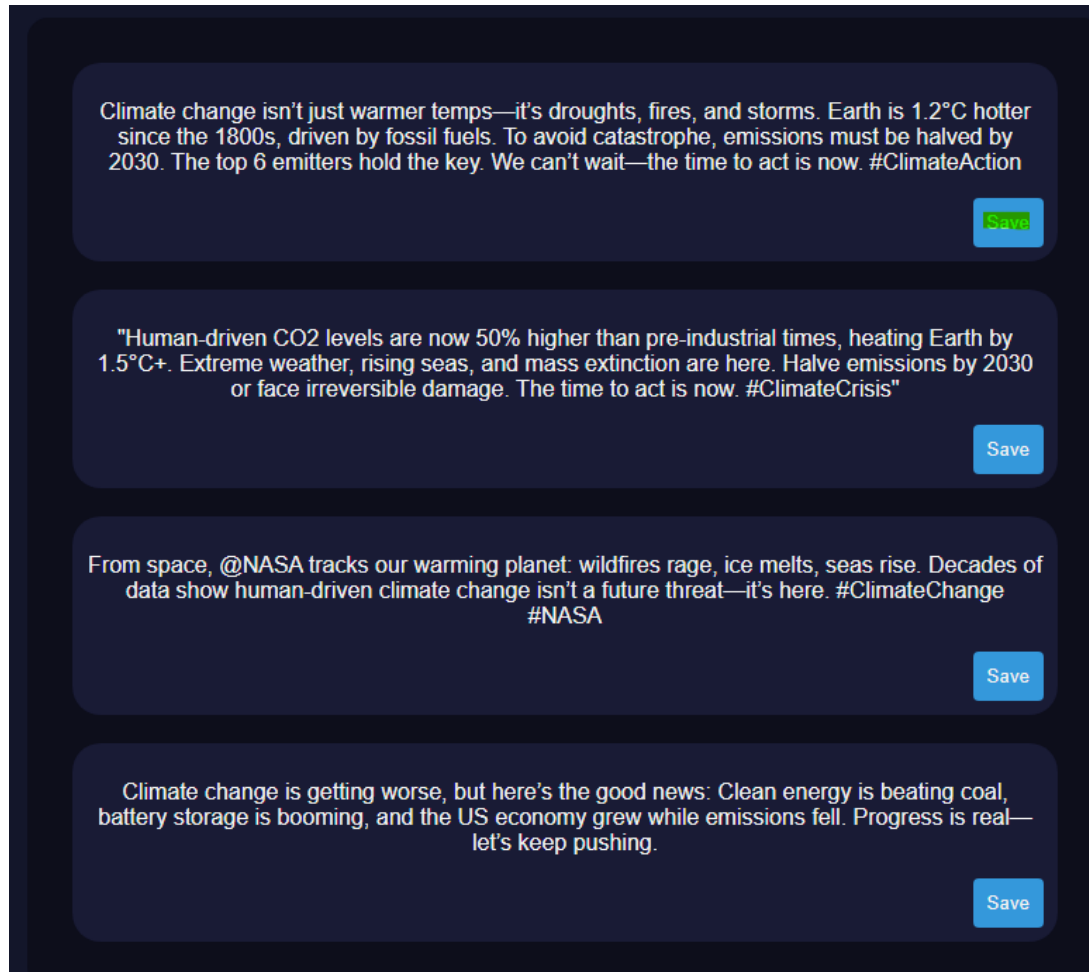


After a format is selected, simply enter a topic or keyword in the input box, then click the Generate button to generate the post(s) (this takes roughly 30 seconds per post generated – up to 5 posts are generated after clicking the Generate button).



4.4. Saving and Managing Posts

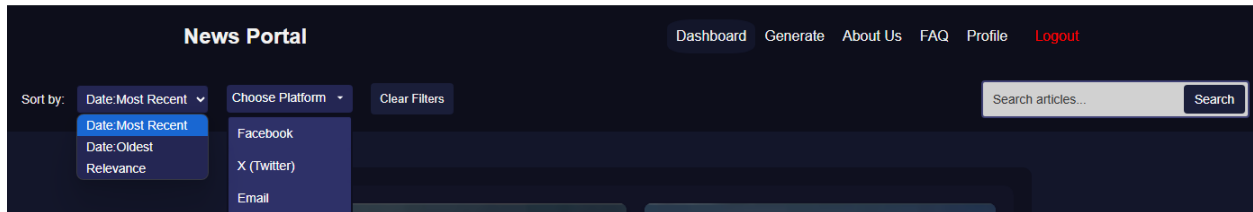
On the Generate page, after clicking the Generate button (only after entering a topic or keyword), up to 5 posts will be generated. To save a post, simply click the Save button located at the bottom right of each generated post.



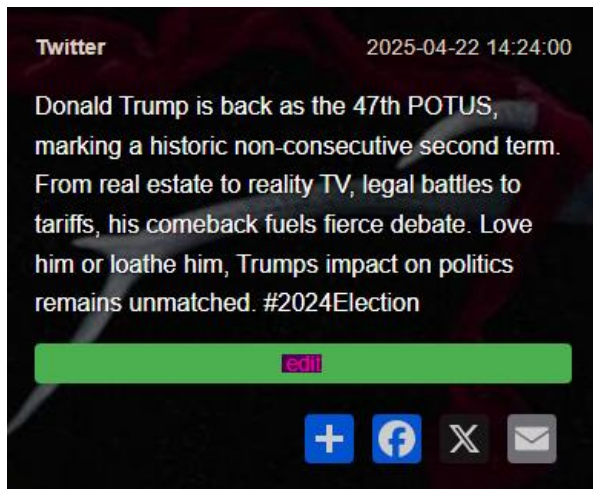
Posts will be saved to the users account and saved posts will be visible on the Dashboard page. Using the navigation bar, click on the Dashboard button to go to the dashboard page.



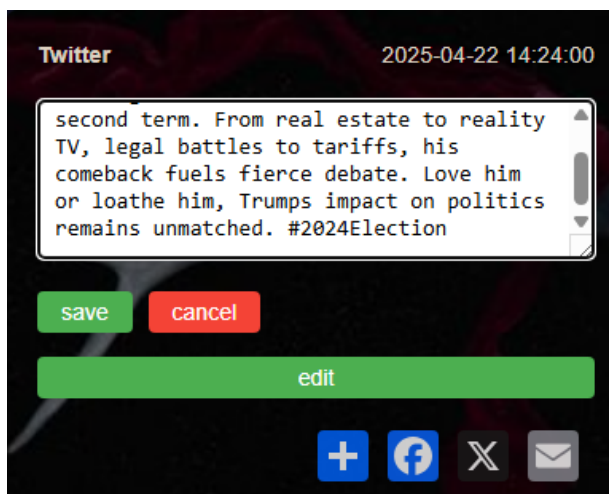
On the Dashboard page, one can filter through their posts using the filter selections at the top of the page. Filter by date/relevance and platform are both supported. A search bar in the same location can also be used to search for specific posts.



Located on every saved post is an “edit” button which can be used to edit the post.



Simply edit any part of the post and click the save button to save the changes or cancel button to discard the changes.



At the bottom of each saved post also includes buttons to share the post to different platforms.

4.5. Logging Out

To logout, click on the logout button on the right of the navigation bar. After doing so, one will be logged out and redirected to the Dashboard page.



5. Project Contributions

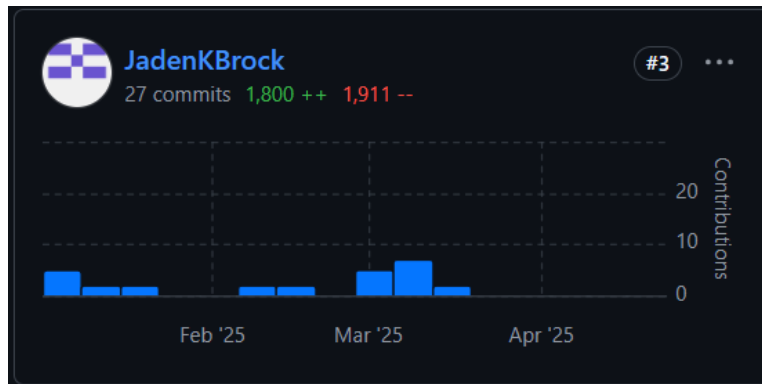
5.1. Jaden Kuhn

5.1.1. Contributions

- Researched multiple different possible LLMs and datasets for the next sprint.
- Ran tests with the LLM we picked from Sprint 1 and discovered that both the LLM and datasets were probably not going to work.
- Ran some more tests with the LLM Deepseek (which we ended up choosing for our LLM).
- After we chose to stick with Deepseek, I experimented with changing the model parameters and prompt and eventually got good results from the model.
- Created an Azure web app along with a resource group to host everything in our project.
- Connected the GitHub repository with Azure and got the site working (hosted and running).
- Created a web scraping algorithm which takes a URL and scrapes the content of the website.
- Overhauled our website and connected the base files for the website. I changed all the HTML files to PHP and organized the file structure for our website. I also made it, so all our styling is the same and implemented a common theme among the pages.
- Implemented the LLM on Azure using Azure AI Foundry.
- Integrated our functions using Azure Functions for the web aggregator as well as the html scraper.
- Created another function to call the LLM from an endpoint and get a response.
- Connected all the functions with each other and integrated with the website so now generating a post works.
- Helped Thomas to be able to call the DB on Azure.
- Added the save button the on generated posts (and the functionality of the button)
- Fixed an issue where the user was not being recognized on the Dashboard page.

- Added format selection for Twitter and Email (and the differences in the LLM generation based on format type).
- Made it so that up to 5 posts are generated instead of just 1.
- Updated the navigation bar to make it more responsive.
- Changed the style of the filter section and moved it to the top of the page.

5.1.2. GitHub Commits



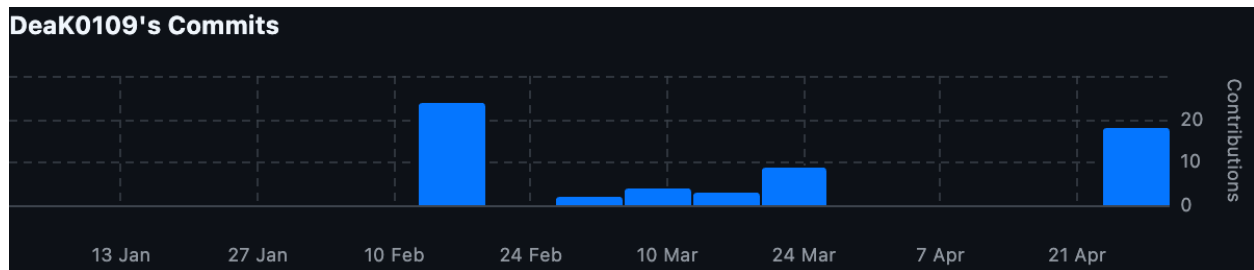
5.2. Dea Kukuqani

5.2.1. Contributions

- Focused on researching possible LLMs and Datasets.
- Continued testing different mixes of the LLMs and Data sets before making a final decision with the rest of the group.
- Tested out different prompts to be used as well in order to achieve the goal of summarizing content while maintaining the integrity of the article as well as making sure it meets the criteria of different social media platforms.
- Collaborated with Chidera to get the main landing page working and focused on UX/UI research to make sure that users can achieve their goals on our site and get access to different features through the use of different active buttons and simultaneously making sure the design and aesthetic of our site creates a good experience for the user.
- Implemented an automated tagging system to categorize aggregated articles based on predefined keywords.
- Integrated a dashboard feature allowing users to manually assign categories to uncategorized articles.
- Established backend functionality to update and store category changes, enhancing content management efficiency.
- Designed and implemented automated tests for the registration and login functionalities.

- Developed PHP test scripts to validate page loading, input field presence, successful user registration, and proper error handling during login attempts.
- Prepared detailed test plans, test cases, and execution reports to ensure that the website's authentication features functioned reliably and securely.

5.2.2. GitHub Commits

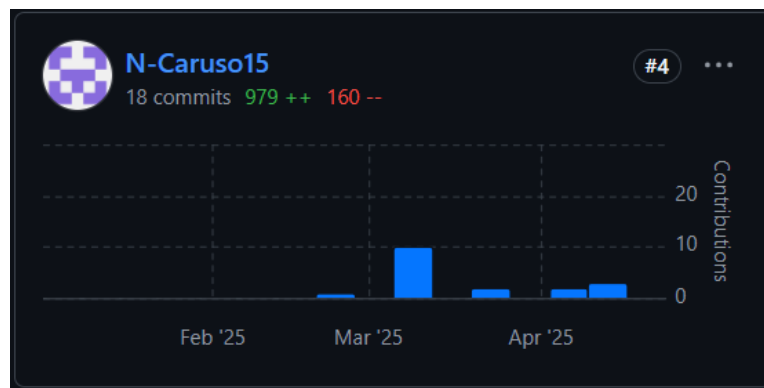


5.3. Nicholas Caruso

5.3.1. Contributions

- Researched various LLMs and datasets for the project, including models like Pegasus, GPT, BLOOM, and datasets like Newsroom, Reddit, XSum.
- Ran initial tests with the T5 model locally.
- Ran tests with the Deepseek model locally, which became our model.
- Experimented with Deepseek locally, and using different platforms such as Colab and Hugging Face Spaces.
- Generated automated test scripts using Pytest to test the web aggregator and data scraper before integration.
- Hosted the second Azure subscription for the project and configured it with Jaden.
- Researched and experimented with Meta Developer Accounts and APIs.
- Implemented the core functionality of the AddToAny tool to allow for exporting to social media.
- Refined the core functionality of AddToAny to pre-populate the X window when the button is clicked. Due to platform restrictions, Facebook does not allow this.
- Created additional automated test scripts using Pytest for many of the core features, including website navigation, Meta tags, social media share buttons, and post editing.

5.3.2. GitHub Commits



5.4. Shijie Tong

5.4.1. Contributions

- Finished the login and registration web page (connect with SQL, most of code, logic part)
- Helped work with News API and wrote an algorithm using this API.
- Experimented with Serp API and wrote an algorithm for storing history using this API.
- Fixed SQL code and linked apps and login and registration to each other.
- Added new features and tested related features and links of existing work.
- Continued to fix the issue where aggregator couldn't be used with multiple users.
- Improvements were researched and several alternate versions were written for potential use – for the web scraper and aggregator.
- Redesigned and improved the database structure.
- Made it so that one can view their posts on the dashboard and added animations to the posts when filtering.
- Fixed filtering issues on the dashboard.
- Fixed an issue where the profile didn't show information.
- Added generating post animations for User experience for the Generate page.
- Implemented the content generation tests. (pytest)

5.4.2. GitHub Commits



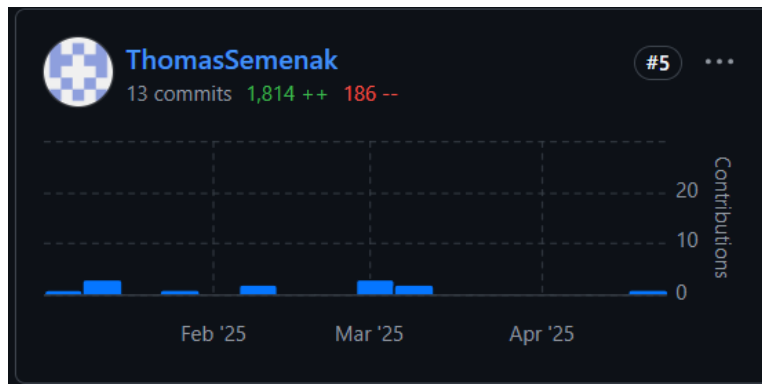
5.5. Thomas Semenak

5.5.1. Contributions

- Developed a local database to support user registration and login.
- Experimented with a custom web scraping solution using the Beautiful soup library as an alternative to the News API.
- Collaborated with Shijie to evaluate alternative APIs, ultimately identifying SerpApi as the best option.
- Coded the Web Aggregator algorithm with the help of Shijie. Allowing users to input a keyword and then retrieves relevant links from the web.
- Combined keywords from both google_search and google_news into a single result set, offering users enhanced flexibility with what appeared to be the optimal parameters.
- Built a template HTML file to prototype how search results would be displayed on a webpage.
- Modified the data aggregator by compiling all links into a list instead of converting them to JSON, which simplifies and allows integration with components such as the web scraper or LLM.
- Created the database in azure and connected necessary tables to support backend operations
- Developed the login and registration system, securely storing user credentials in the database for future logins. Integrated into the app with enhanced styling, ensuring that users are seamlessly redirected to the dashboard homepage upon login.
- Spent extensive time troubleshooting PHP file issues related to connecting to the Azure database and more troubleshooting ensuring that each user's profile and personalized dashboard displayed correctly after login.

- Created the About Us and FAQ pages to provide users with additional context
- Built a concurrent testing script to simulate multiple users accessing the platform at once for performance evaluation
- Responsive styling throughout the app to make sure the app is viewable and functional on mobile devices.

5.5.2. GitHub Commits

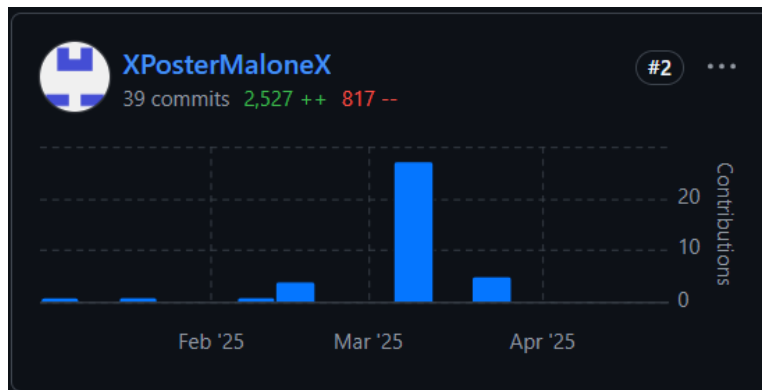


5.6. Dalton Morris

5.6.1. Contributions

- Helped with implementing login and registration page and tidying it up in terms of the backend code and UI.
- Added parsing error checking for articles for the original web scraping API, so when an article object was created it would check for bad/incorrect URLs and other errors.
- Added more exception handling for the newest web scraping API.
- Touched up the history storing and display for the newest web scraping API.
- Got core functionality sorted out for automated scheduler, very basic for now, allows the user to choose daily, weekly, monthly, or a custom interval to have a newsletter generated.

5.6.2. GitHub Commits

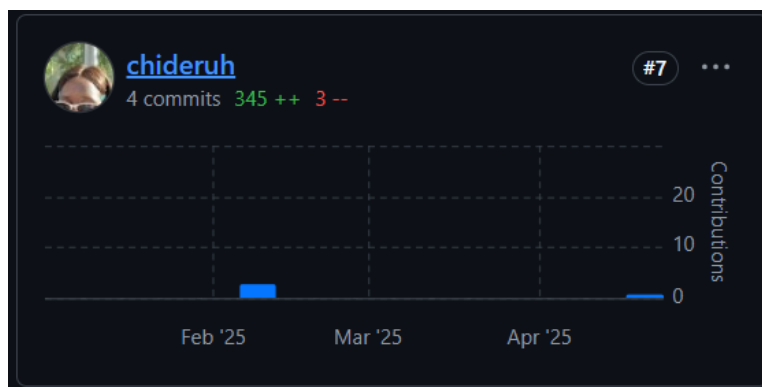


5.7. Chidera Nwana

5.7.1. Contributions

- Developed tables and queries to be used for database management for the website using MySQL.
- Created the initial draft of the dashboard for the website.
- Implemented initial sorting functionality to allow content to be sorted by popularity, date added and tags.
- Collaborated with Dea to cleanup initial issues with the dashboard, implement better button arrangement and reorganize the UI for enhanced use and easier navigation.
- Designed and created tables for login and registering to save user details for easy and efficient access into the website using saved credentials.
- Created queries to be used to access information from the tables ensuring efficient access to any information needed.

5.7.2. GitHub Commits



Appendix A. Website and GitHub Repository Links

Website Link: <https://group9portal-eehbdbxbhcgftezez.canadaeast-01.azurewebsites.net/index.php>

GitHub Repository Link: <https://github.com/JadenKBrock/4P02GroupProject>

Appendix B. Meeting Minutes

January 7, 2025 – Meeting 1

- This was the formation of the group in lecture and was not a formal meeting.

January 14, 2025 – Meeting 2

Attendees: Dea, Shijie, Dalton, Jaden, Nicholas, Thomas, Chidera.

- The group went over the project description and worked together to break the project into several themes and epics. The project was broken down into 6 initial themes:
 - Six group members were allocated a theme and were tasked with breaking down the theme into user stories, and then into further tasks (Shijie, Dalton, Jaden, Nicholas, Thomas, Chidera). A single group member was allocated to reviewing and modifying these themes, stories, and tasks to ensure they were clear and concise (Dea).
- The group also discussed the Release Planning Meeting with the TA. Thursday (1/16/2025) at 11:40 was the timeslot selected, where the most members could attend (6/7 were able to attend, as the meeting is outside of the course hours).

January 16, 2025 – Meeting 3 - Release Planning Meeting

Attendees: Dea, Dalton, Jaden, Nicholas, Thomas, Chidera.

Absent: Shijie (schedule conflict)

- The group presented the themes and user stories to the TA.
 - Also requested clarification on the format of the report.
 - Suggestions were also made for additional contents in the report. For example, researching a similar market application and including pros and cons to the features it already has.

- After the Release Planning Meeting was finished, the group met to debrief the meeting.
 - Now that the user stories were presented, we would begin breaking down the stories into tasks.
 - The group also discussed GitHub Projects, and it was determined that Jira may be the best software to implement our backlog, as it has many options and customizations that are not available within GitHub.
 - The group agreed, and the switch was made. All members of the group were added to the Jira Project.

Meeting 4 – January 21, 2025

Attendees: Dea, Dalton, Jaden, Nicholas, Thomas, Chidera, Shijie

- Discussed Sprint 1 task distribution.
- Teams assigned for research, UI development, and database setup.
- Communication was set to take place via Microsoft Teams.

Meeting 5 – January 28, 2025

Attendees: Dea, Dalton, Jaden, Nicholas, Thomas, Chidera

Absent: Shijie

- Sprint 1 review confirmed LLM research completion, UI implementation, and database setup.
- Sprint 2 tasks were allocated to teams.

Meeting 6 – February 4, 2025

Attendees: Dea, Dalton, Jaden, Nicholas, Thomas, Chidera

Absent: Shijie

- Mid-Sprint 2 progress review.
- LLM narrowed to GPT and DeepSeek, with training initiated.
- Scraper prototype uploaded to GitHub.

Meeting 7 – February 11, 2025

Attendees: Dea, Jaden, Nicholas, Thomas, Shijie

Absent: Dalton, Chidera

- DeepSeek selected as the LLM model.

- Landing page uploaded to GitHub.
- Identified issues with NewsApi and initiated the transition to SERPAPI integration

Meeting 8 – February 17, 2025

Attendees: Dea, Dalton, Jaden, Nicholas, Thomas, Chidera, Shijie

- Confirmed all teams were on track for Sprint 2 completion.
- Data scraper demonstrated improved results.
- Dashboard redesign finalized and uploaded.

Meeting 9 – February 18, 2025

Attendees: Dea, Dalton, Jaden, Nicholas, Thomas, Chidera, Shijie, Madeline, Naser.

- Presented the current state of our project to Madeline for feedback and verification.

Meeting 10 – February 25, 2025

Attendees: Jaden, Nicholas, Thomas, Chidera, Shijie.

Absent: Dea, Dalton.

- Discussed the results of the third sprint, challenges encountered, and improvements for the next sprints.
- Discussed remaining tasks in the product backlog and divided them into the remaining sprints.
- Modified the product backlog to exclude features that are no longer part of the project.

Meeting 11 – March 4, 2025

Attendees: Dea, Dalton, Jaden, Nicholas, Thomas, Chidera.

Absent: Shijie.

- Discussed setbacks during this current sprint and decided that additional work is required as tasks are behind schedule for this sprint.
- Discussed what we need to do going forward for this extended sprint and possible tasks for future sprints in general.

Meeting 12 – March 11, 2025

Attendees: Dea, Dalton, Jaden, Nicholas, Thomas, Chidera, Shijie.

- Discussed how the remaining work for the sprint is going and any challenges associated with that.

- Talked about the next steps for the remainder of the sprint and what we wanted to accomplish before the March 18 meeting.

Meeting 13 – March 18, 2025

Attendees: Dea, Dalton, Jaden, Nicholas, Thomas, Chidera, Shijie, Madeline.

- Presented the current state of our project to Madeline for feedback and verification.

Meeting 14 - March 25th

Attendees: Nicholas, Jaden, Thomas, Dea, Chidera, Shijie, Dalton.

- Discussed the results of the fifth sprint. Discussed challenges and possible improvements.
- Discussed task allocation for the sixth sprint.

Meeting 15 - April 1st

Attendees: Nicholas, Jaden, Thomas, Dea, Chidera, Shijie, Dalton.

- Discussed the progress of the sixth sprint. This sprint will last another week to achieve the goals for this sprint.

Meeting 16 - April 8th

Attendees: Nicholas, Jaden, Thomas, Chidera, Shijie, Dalton.

Absent: Dea.

- Discussed the results of the sixth sprint. Discussed challenges and possible improvements for future projects or in the industry.
- Discussed finishing touches that will be the focus for the remainder of the project.

Meeting 17 - April 22nd

Attendees: Jaden, Nicholas, Thomas, Chidera.

Absent: Dea, Shijie, Dalton.

- Discussed final touches on the project. Adding accessibility on other platforms. "Delete Post" button on the dashboard. Adding a "Copy" button on the dashboard.
- Discussed final testing that needs to be completed.

- Discussed features that will not make the final project due to time constraints.
- Discussed cleanup for GitHub.

Appendix D. Sprint Burndown Charts (Jira)



Appendix E. GitHub Screenshots

Note: All of our updated code is through the deploy branch:

<https://github.com/JadenKBrock/4P02GroupProject/tree/deploy>

The screenshot shows the GitHub repository page for **4P02GroupProject**. The repository is public and has 1 watch, 1 fork, and 1 star. It is currently on the **main** branch, which is 220 commits ahead of the **origin/main** branch. The repository has 7 branches and 0 tags.

The file list shows the following files and their commit history:

File	Commit Message	Time Ago
<code>.github/workflows</code>	Add or update the Azure App Service build and deployment workflow	3 weeks ago
<code>Generate_Post</code>	Added linked LLM and webscraper	last month
<code>reports</code>	Add files via upload	2 months ago
<code>scripts</code>	filter and some others	2 weeks ago
<code>src</code>	comment	2 days ago
<code>styles</code>	Fixed responsive navbar	4 days ago
<code>tests</code>	generation test	5 hours ago
<code>views</code>	Fixed responsive navbar	4 days ago
<code>LICENSE</code>	Update LICENSE	3 months ago
<code>README.md</code>	Added member names to README	3 months ago
<code>auto_generate_post.php</code>	Create auto_generate_post.php	4 days ago
<code>default-profile-pic.png</code>	changed some stuff	last month
<code>index.php</code>	comment	2 days ago
<code>logout.php</code>	ldk	3 weeks ago
<code>update_post.php</code>	sry wrong file, dont forget to change header.php	3 weeks ago

The repository also includes a **README** file and a **License** file.

The right sidebar shows the following sections:

- About**: No description, website, or topics provided.
- Releases**: No releases published. [Create a new release](#)
- Packages**: No packages published. [Publish your first package](#)
- Contributors**: 7 contributors.
- Deployments**: 390 deployments. [Production](#). [+ 389 deployments](#)
- Languages**: Python 64.0%, PHP 15.8%, CSS 7.3%, HTML 7.0%.