Department of Computer Science
COSC 4P02 - Software Engineering II
WINTER 2025

AI-Powered Newsletter and Social Media Content Generator

Final Report

Professor: Naser Ezzati Jivan

Prepared by: Group 14

Arifuzzaman (Scrum Master)	(7088214)
Hridoy Rahman	(7340334)
Jeffin Sam Joji	(7344526)
Kabir Sethi (Product Owner)	(6933782)
Sahil Rashid	(6954739)
Yuanhan Huang	(7642440)
Jayant Saini	(7331556)

Date: April 28, 2025

Contents

1 Overview

Throughout the development of this project, our team has made consistent and strategic progress by adhering closely to Agile and Scrum principles. We focused on iterative development, frequent feedback loops, and continuous improvement to deliver a user-centric and scalable platform.

In the early phases, we established a strong foundation by implementing core functionalities such as user authentication, registration, dashboard management, and content aggregation powered by large language models (LLMs). This initial groundwork allowed us to build a stable and extensible system to support more advanced features in later stages.

As the project evolved, we expanded the platform to include dynamic social media integration, content scheduling, and email newsletter sharing using customizable templates. We introduced interactive preview capabilities for Instagram and Reddit, enhancing user engagement and improving content visibility. The user interface was continuously refined through feedback received during sprint reviews, focusing on intuitiveness, responsiveness, and a consistent visual experience.

In the final sprint, we delivered several major enhancements, including multilingual support, content archiving, and external API integration. These features further extended the platform's functionality and accessibility. To better visualize system processes and interactions, we introduced a set of diagrams; activity diagrams, state diagrams, and sequence diagrams; providing clear representations of the workflows behind key features.

To support future users and developers, we created a comprehensive Software Design Specification (SDS) Installation Guide. This document offers detailed, step-by-step instructions for setting up the development environment, installing necessary dependencies, and running the platform, ensuring a smooth onboarding experience.

Throughout the project, Agile Scrum methodologies were rigorously applied, with structured sprint planning, daily standups, sprint reviews, and retrospectives. This approach allowed us to remain flexible, quickly respond to feedback, and prioritize features that enhanced user experience and system performance. Moving forward, the project is well-positioned for further refinement, scaling, and real-world deployment.

2 Installation Guideline

The project's installation is straightforward. Follow these steps:

Base Instructions

- Open **Visual Studio Code**.
- Open a folder where you want to run the project.

- Install Python 3.13.1 for dependencies sake.

Clone the Repository

- Open the terminal in Visual Studio Code.
- Run the following command:
  - `git clone https://github.com/iaminhri/COSC-4P02.git`

Navigate to the Project Directory

- Run:
  - `cd COSC-4P02`

Set Up a Virtual Environment

- Create and activate a virtual environment:
  - `python3 -m venv venv`
  - `source venv/bin/activate`

Install Project Dependencies

- Run:
  - `pip install -r current_requirements.txt`

Navigate to the App Directory

- Run:
  - `cd trendTailor/trendApp`

Run the Django Development Server

- Run:
  - `python manage.py runserver`

Access the Website

- Open your Chrome browser.

- Go to: http://127.0.0.1:8000

3 Design & Implementation

3.1 Sprint 1

3.1.1 SCRUM - 9

In our first sprint, we focused on laying the foundation of our platform while following Agile Scrum methodologies to ensure timely progress and adaptability. The primary goal was to implement user authentication, allowing users to register, log in, and securely access their accounts. We validated inputs, stored user data in the database, and ensured proper authentication mechanisms were in place, with rigorous testing to guarantee security and reliability.

Task	Associated User Stories	Priority
Create a registration and login web interface	Scrum-9	High
Create form fields for login and registration	Scrum-9	High
Validate inputs for the forms	Scrum-9	High
Create database to store users data	Scrum-9	High
Create authentication system for users to login	Scrum-9	High

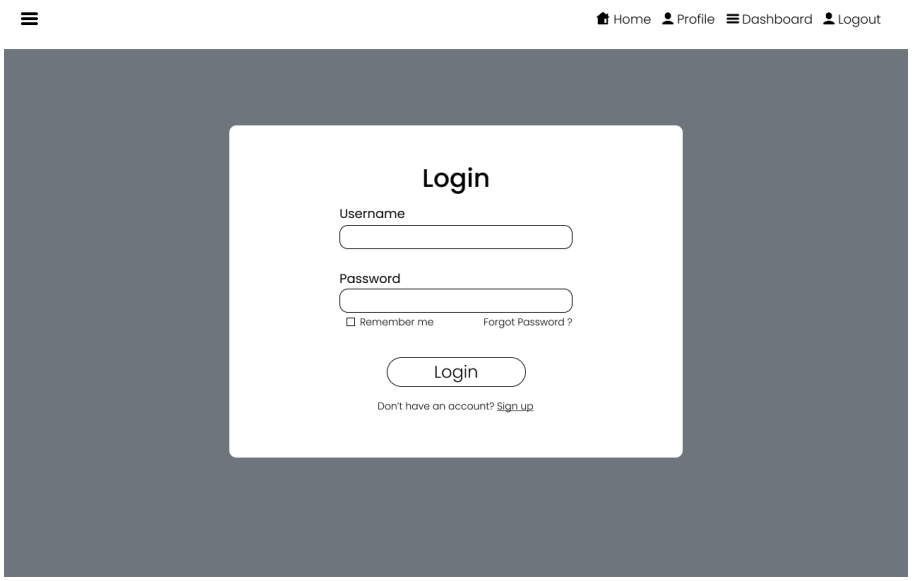


Figure 3.1: Login Page

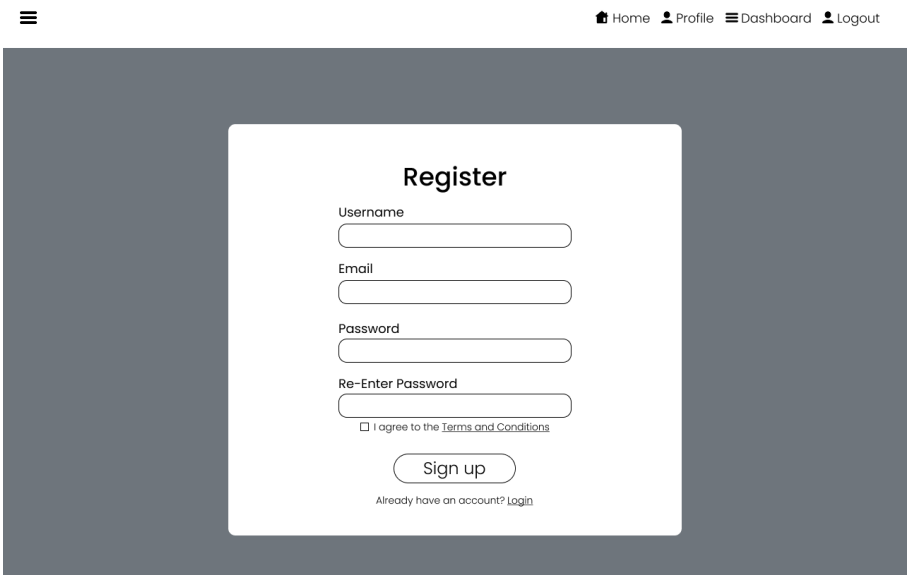


Figure 3.2: Registration Page

3.1.2 SCRUM - 1

Next, we developed the dashboard and profile management system, enabling users to view and manage generated content, track performance, and update preferences. We aimed to give the interface an aesthetic, modern, and futuristic look, ensuring a visually appealing and user-friendly experience. We stored user preferences, including selected topics and sources in the database, making it easy for users to access and modify their settings seamlessly.

Task	Associated User Stories	Priority
Create Profile page, Home page, Dashboard view, Archive News (HTML Pages)	Scrum-1	High
Create backend logic for managing generated content	Scrum-1	High
Create backend logic for updating preferences	Scrum-1	High
Store preferences to the database	Scrum-1	High
Store generated content to database for later use	Scrum-1	Medium

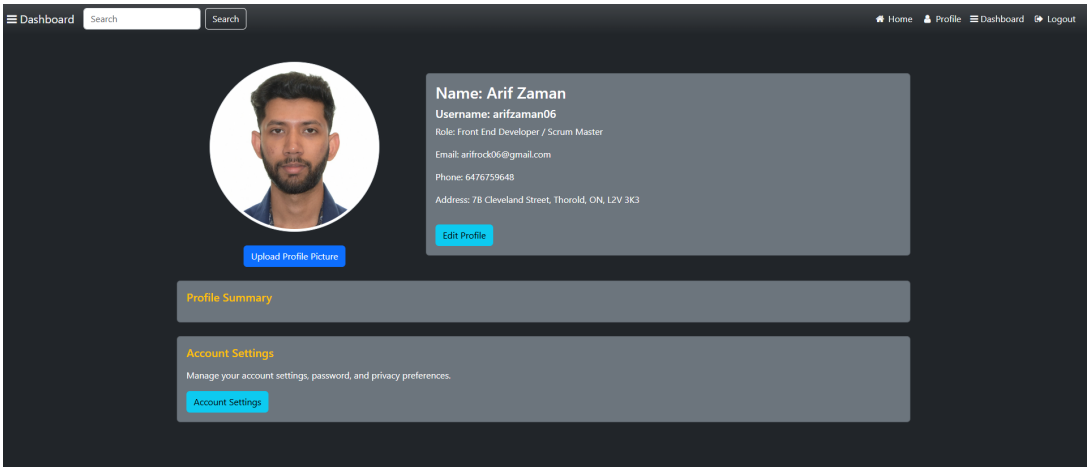


Figure 3.3: Profile Page

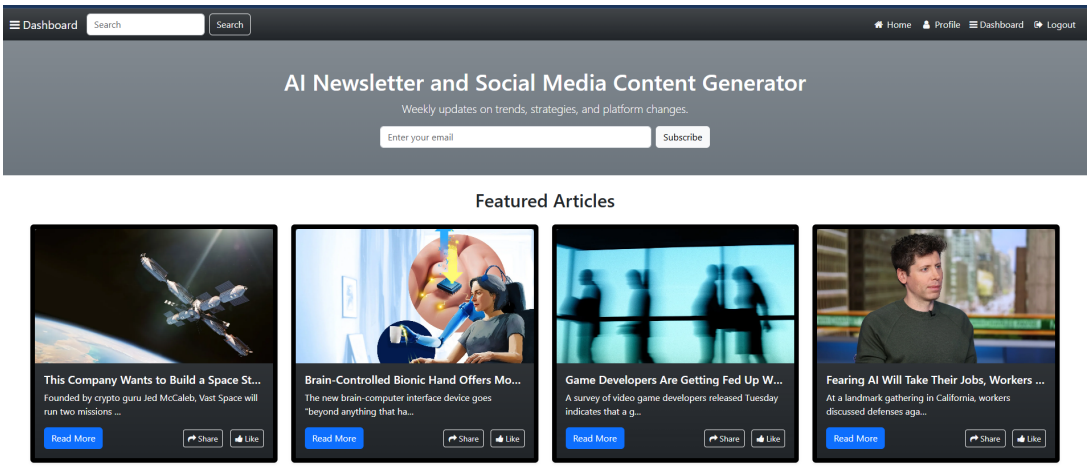


Figure 3.4: Home Page

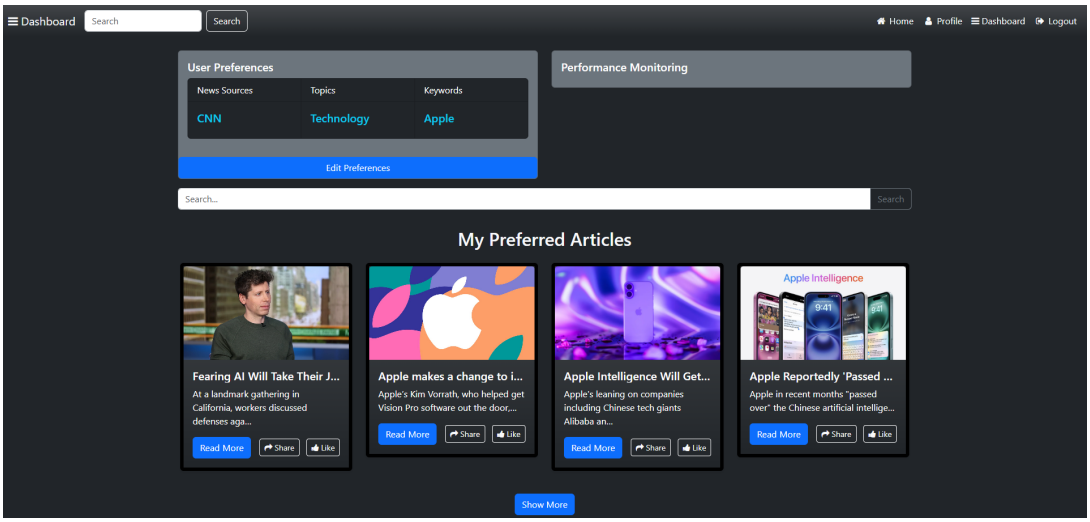


Figure 3.5: Dashboard Page

3.1.3 SCRUM - 2

Lastly, we implemented content aggregation, allowing users to input preferred news sources, topics, and keywords. These preferences were stored and utilized by our backend to scrape relevant content dynamically, ensuring personalized and meaningful results. By the end of Sprint 1, we had successfully built the foundation of our application, keeping development aligned with Agile principles through iterative improvements and structured sprint planning.

Task	Associated User Stories	Priority
Create input fields for sources, topics, and keywords	Scrum-2	High
Scrap relevant keywords and topics-based news from the news sources or default sources	Scrum-2	High
Develop backend logic for content aggregation	Scrum-2	High
Implement a database to store user preferences	Scrum-2	Medium

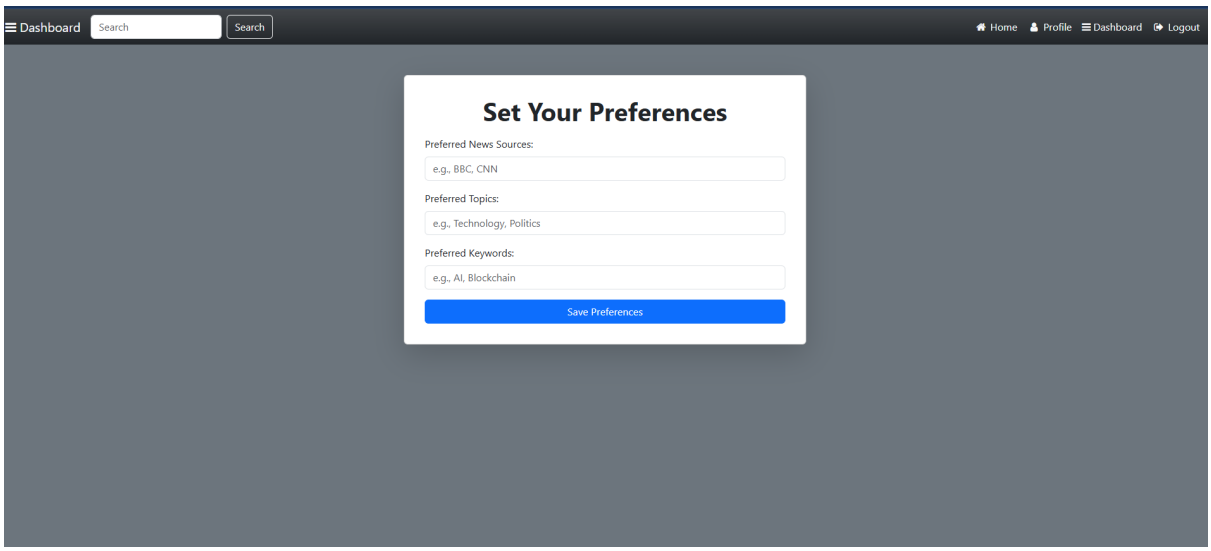


Figure 3.6: Preferences Page

3.1.4 Diagrams for Sprint 1

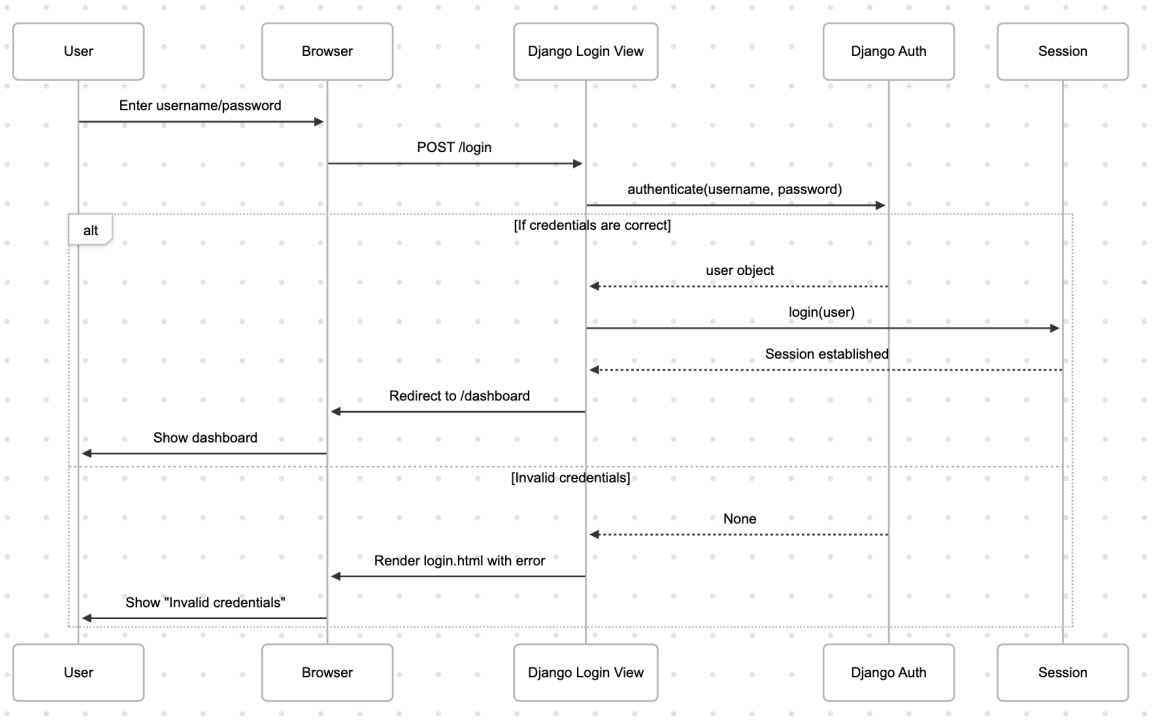


Figure 3.7: Sequence Diagram - Login

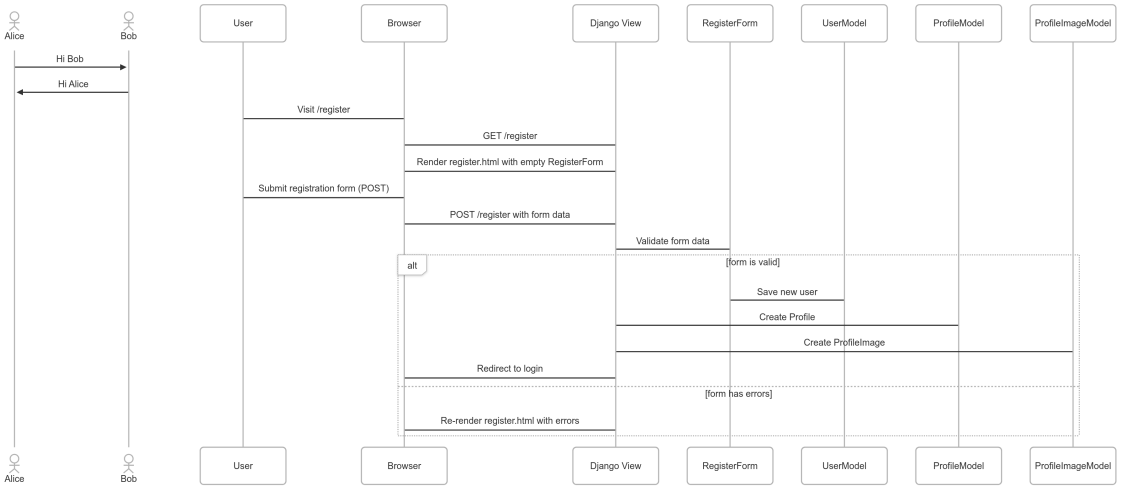


Figure 3.8: Sequence Diagram - Registration

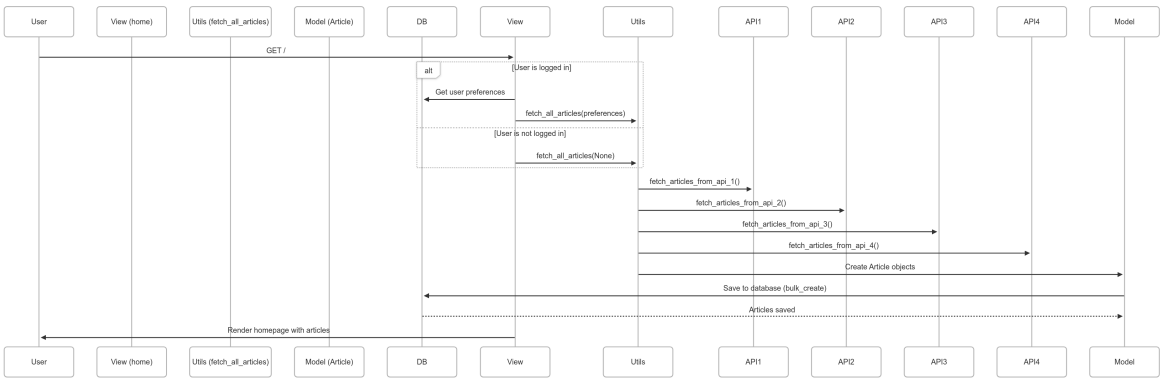


Figure 3.9: Sequence Diagram - Homepage

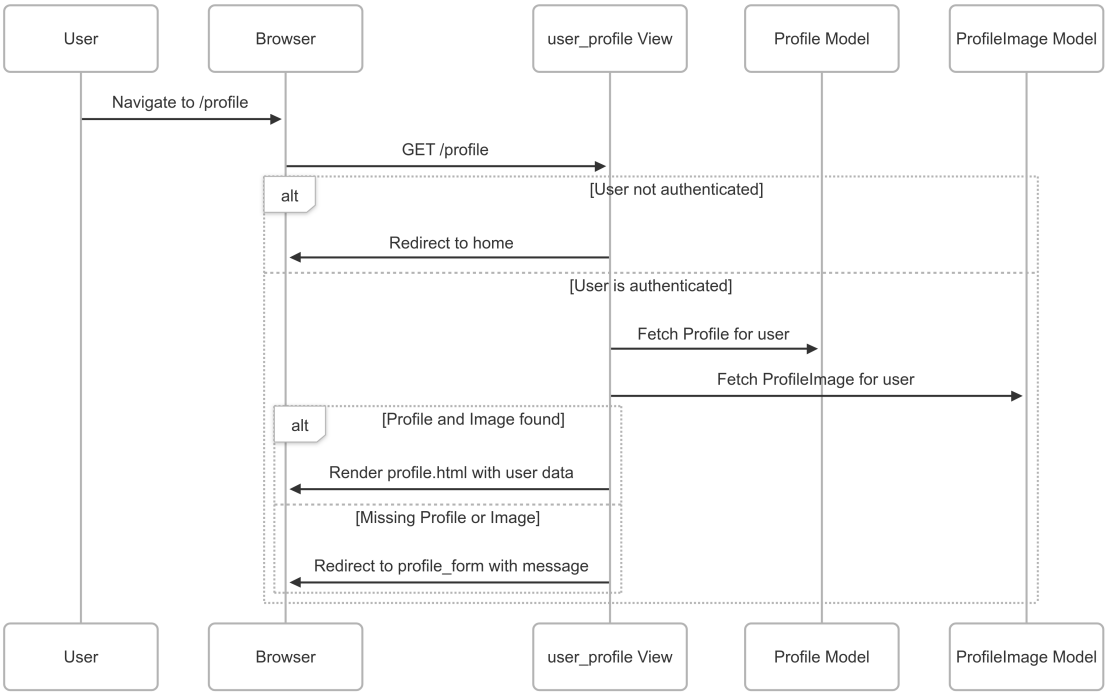


Figure 3.10: Sequence Diagram - Profile Page

Figure 3.11: Sequence Diagram - User Preference

3.2 Sprint 2

3.2.1 SCRUM - 3

In Sprint Two, we focused on personalization and content optimization while maintaining Agile Scrum principles. We introduced a customizable email newsletter system, giving users access to a set of predesigned email templates. These templates come in different styles and designs, allowing users to select one that best suits their preferences. The templates dynamically fetch content based on user preferences, ensuring each newsletter is personalized and relevant. Users could also save and reuse templates for consistency. Testing was conducted to ensure that the templates displayed content correctly and functioned smoothly.

Task	Associated User Stories	Priority
Design and develop the template selection interface (grid/list view with preview options)	Scrum-3	High
Allow users to select content preferences, including topics, tone, and layout styles	Scrum-3	Medium
Implement functionality to apply selected templates to generated content dynamically	Scrum-3	Medium
Create an option to save and reuse custom templates	Scrum-3	Low
Conduct testing to ensure templates are correctly applied and formatted	Scrum-3	High
Display personalized content within the templates	Scrum-3	Medium

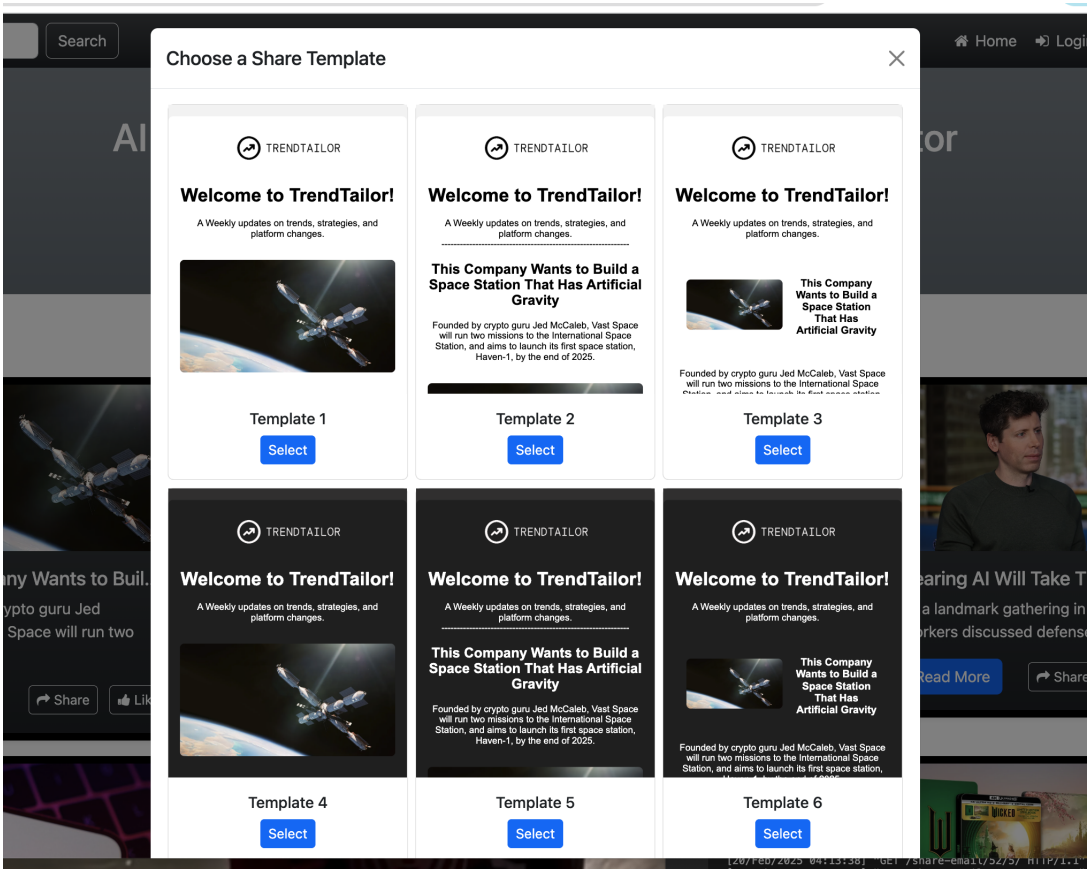


Figure 3.12: Templates Overview Diagram

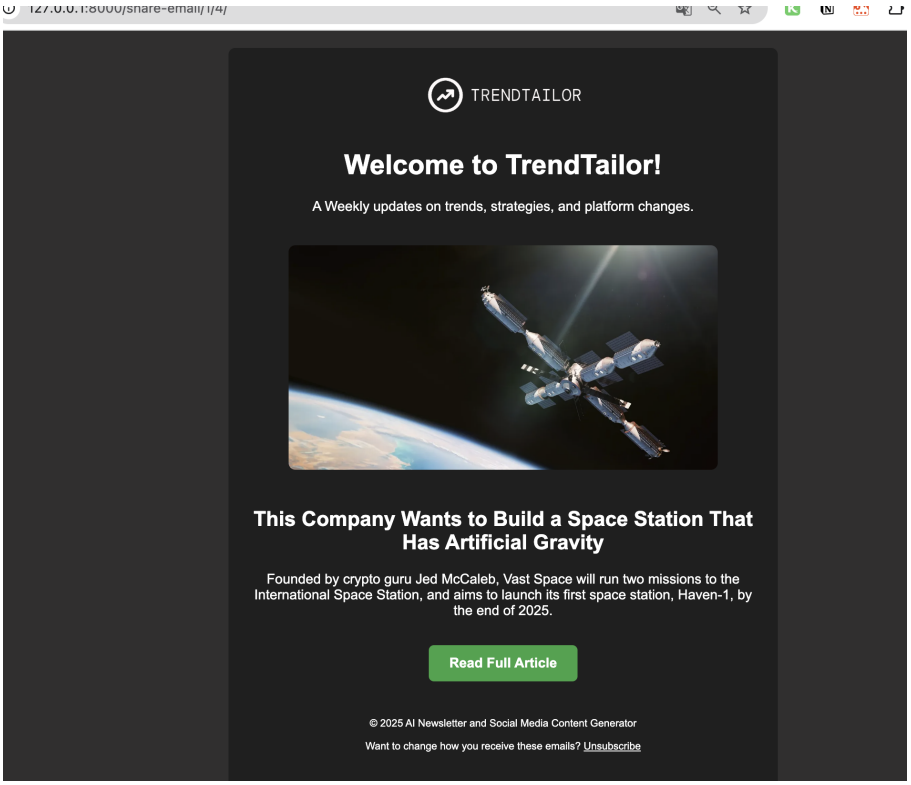


Figure 3.13: Template 1

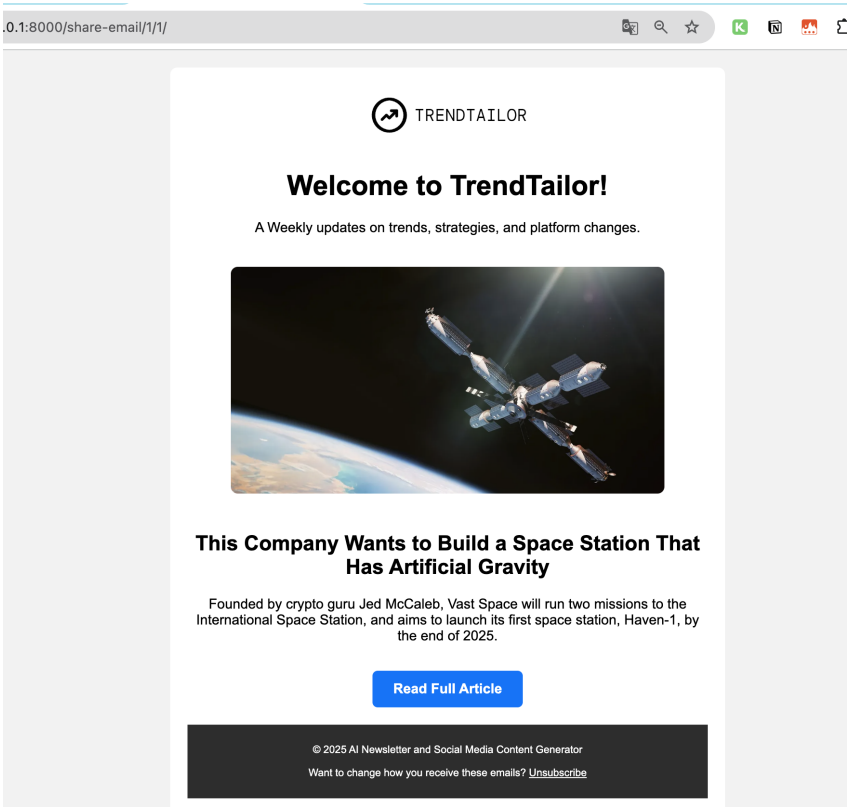


Figure 3.14: Template 2

3.3 Sprint-3

3.3.1 SCRUM - 5

In this sprint, we implemented LLM-based summarization, allowing users to upload content via a text box or a file. The uploaded text is then processed and summarized using a large language model. We leveraged the Transformers library's pipeline functionality with Facebook's BART-Large-CNN model to generate summaries within our Trend Tailor app. The following demonstration provides the implementation details:

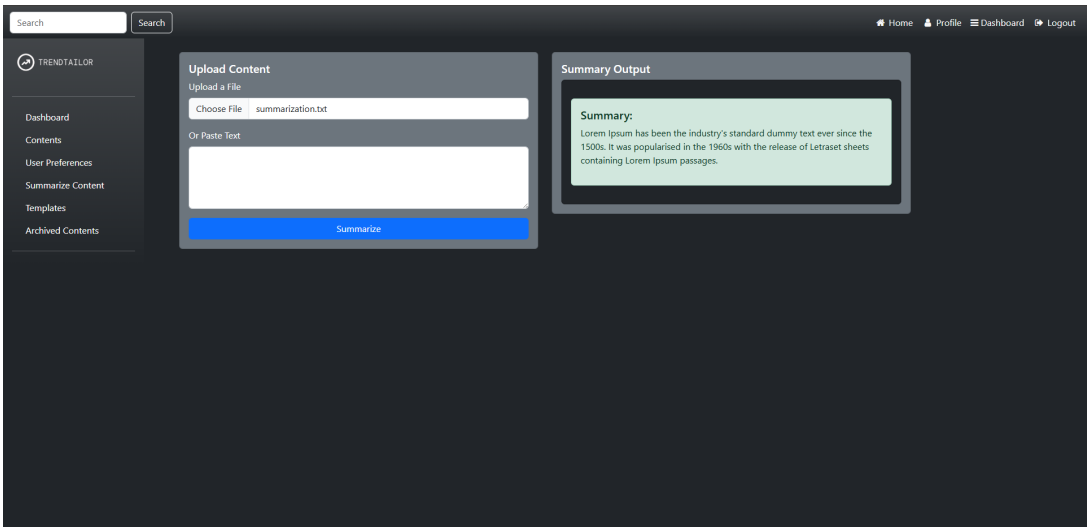


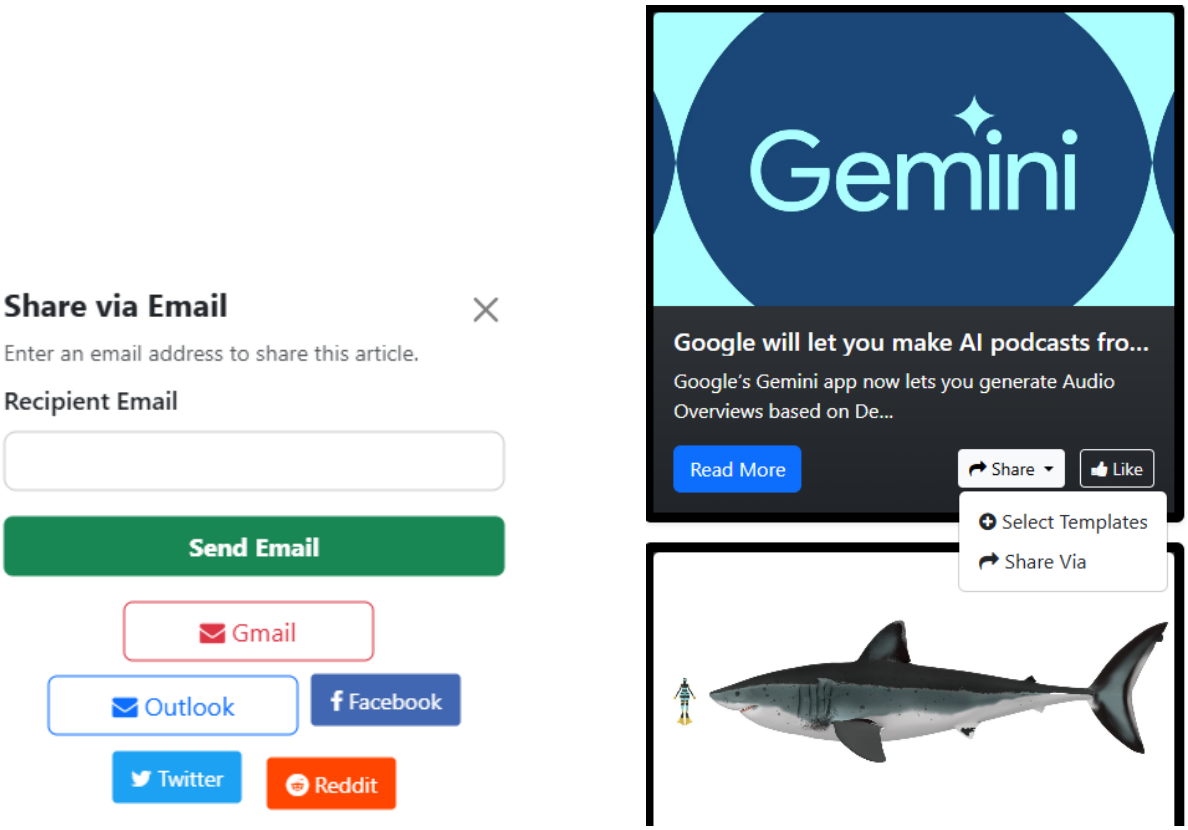
Figure 3.15: Summarization User Interface

Figure 3.15 demonstrates the Summarization UI, where the user can upload a file or text to summarize the contents. The system takes one of the inputs from the user and summarizes the content. Furthermore, the UI was developed using HTML and Bootstrap, while Django handles the backend form and parses the user inputs, either from the file or the text input box, and summarizes them. Additionally, this functionality was tested using various sample inputs. We will conduct a proper unit testing in our next sprint.

Task	Associated User Stories	Priority
Implement article summarization using LLM by prompt	Scrum-5	High
Integrate summarization LLM API into the backend	Scrum-5	High
Test summarization for samples	Scrum-5	Medium
Develop UI for summarization	Scrum-5	Medium

3.3.2 SCRUM - 7

This section discusses the design and implementation of SCRUM - 7. In this phase, we focused on enhancing content distribution by integrating email and social media sharing features into the front-end.



(a) Sharing via Email Module

(b) Sharing Option UI

Figure 3.16: Sharing Workflow Screens

We implemented the email-sending feature by utilizing a "Share Via" button that opens a Bootstrap modal. Inside the modal, we created a simple form prompting the user to input an email address to share the article, as shown in Figure 3.16(a). The sharing options include email and social media platforms such as Outlook, Gmail, Facebook, Reddit, and Twitter. When the "Share Via" button is clicked, the modal opens, allowing the user to select a sharing option, as seen in Figure 3.16(a). Upon selecting a social media or email platform, the system automatically pulls articles from the database and shares the content directly to the respective platform. This feature enables users to quickly and easily share content, increasing visibility and engagement, while ensuring that the appropriate URLs and metadata are formatted correctly for each platform. The system is not limited to sharing default content. We created several template options from which the user can select, and then share the respective template via email, as shown in Figure 3.16(b). After selecting a template, an input form appears, prompting the user for an email address, see Figure 3.19(a). Once the user provides the email and clicks "Send Email," the content, along with the selected template, is shared with the provided email address. The email template along with the content is now in the user's inbox. See Figure 3.18(a), 3.18(b) and 3.19 for reference.

Task	Associated User Stories	Priority
Create a form to send the newsletter content using a list of emails	Scrum-7	High
Create options for users to modify emails, such as add or remove	Scrum-7	High
Create an option to share it with social media	Scrum-7	High
Develop backend logic to send emails and to share it with social media	Scrum-7	High

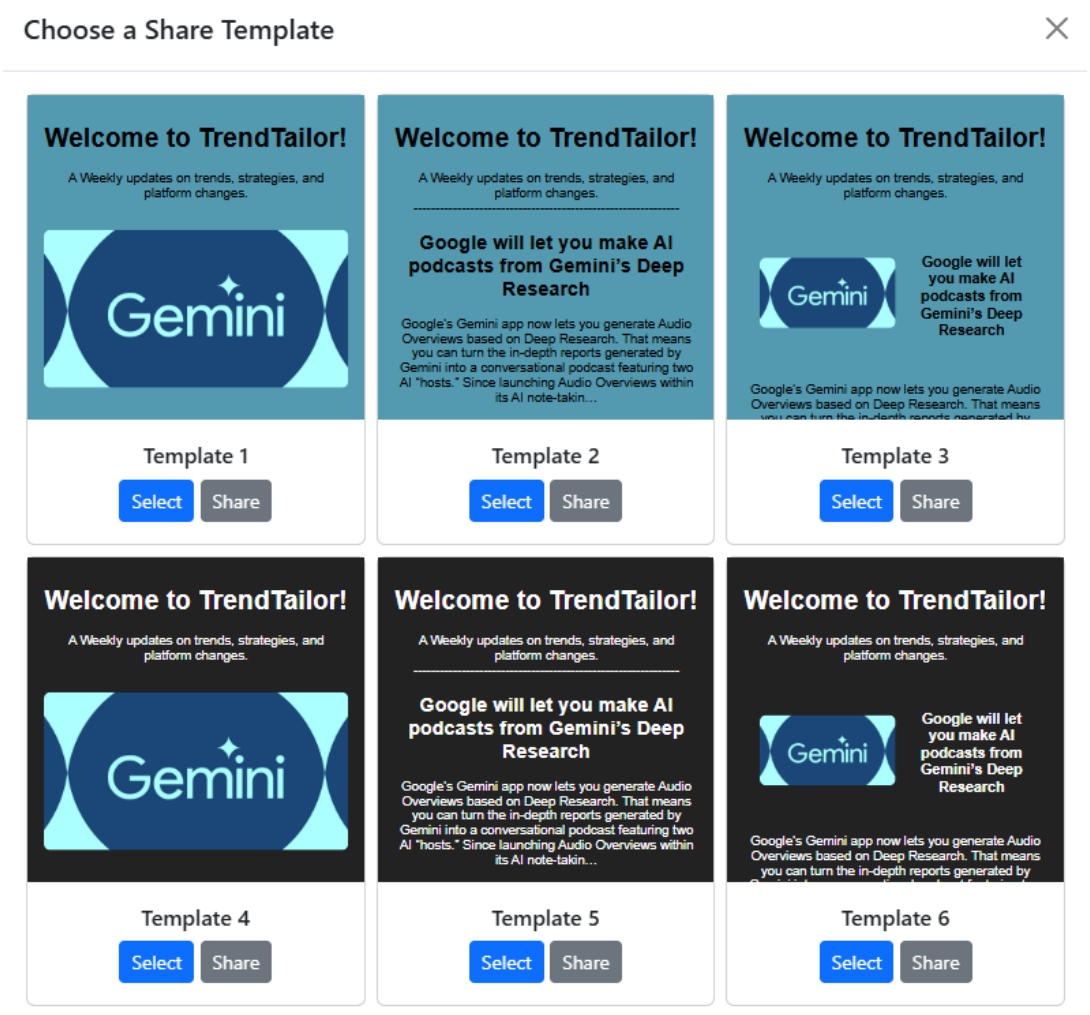
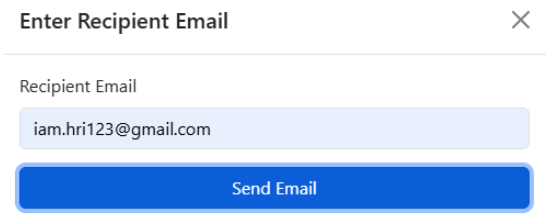
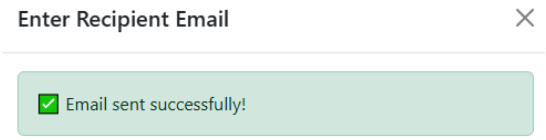


Figure 3.17: Email Templates



(a) Upon selecting Share from above, this email prompt is shown



(b) Success Email Sent

Figure 3.18: Email Sent and Sharing UI

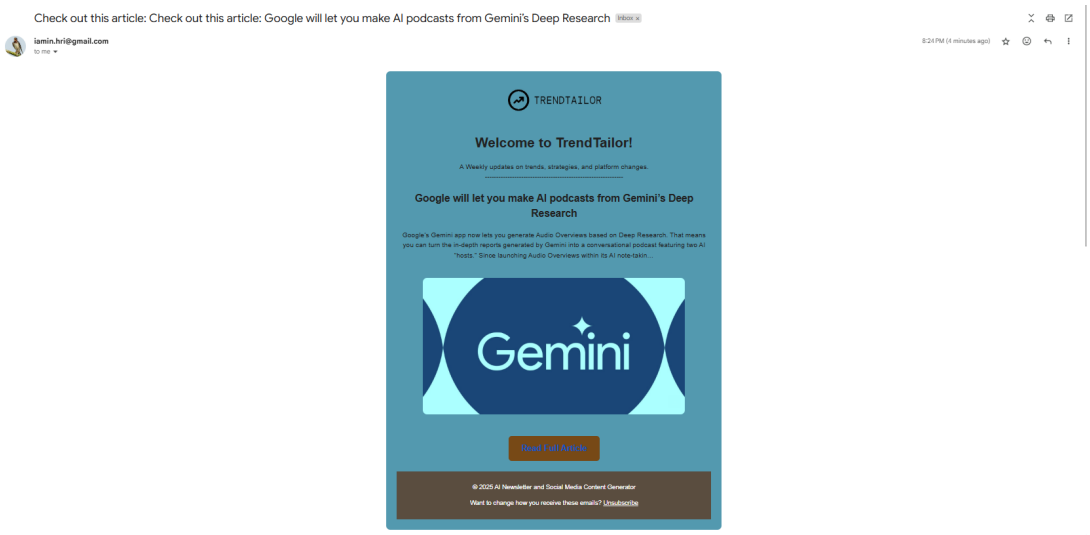


Figure 3.19: Shared Using Email Templates

3.3.3 Diagrams for Sprint 3

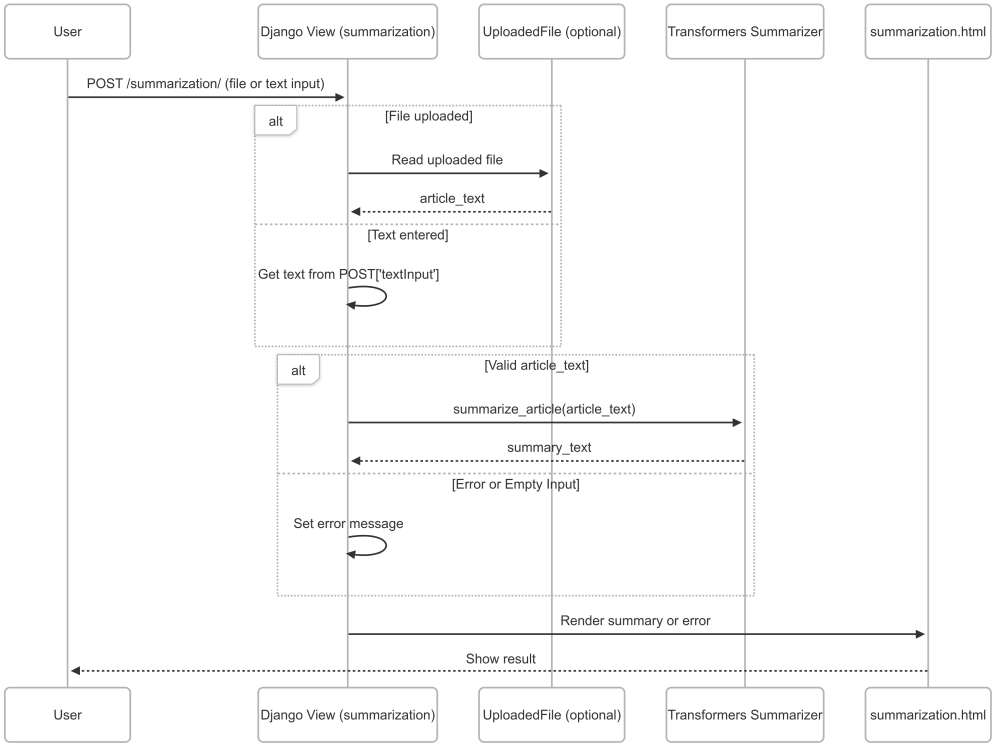


Figure 3.20: Sequence Diagram - Content Summarization

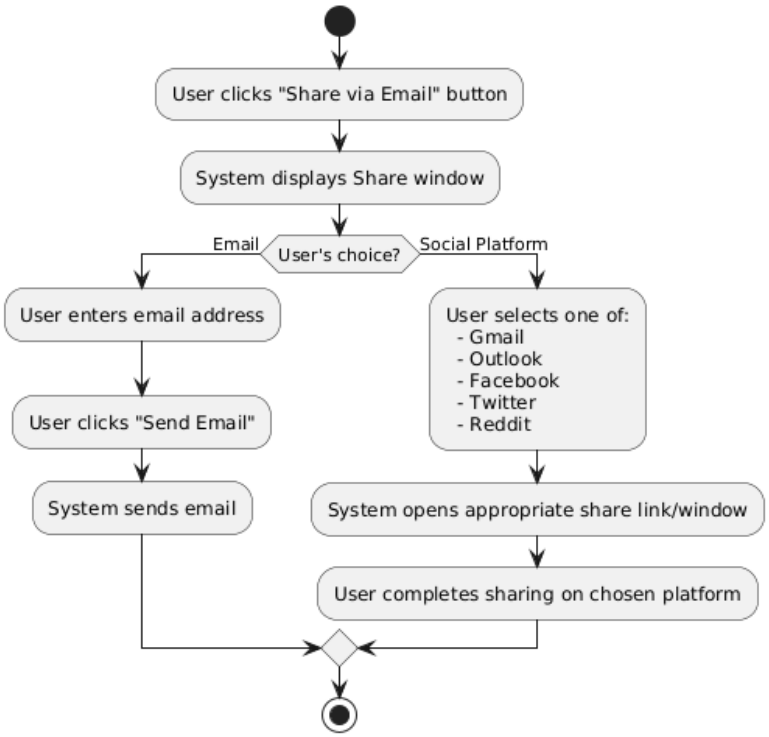
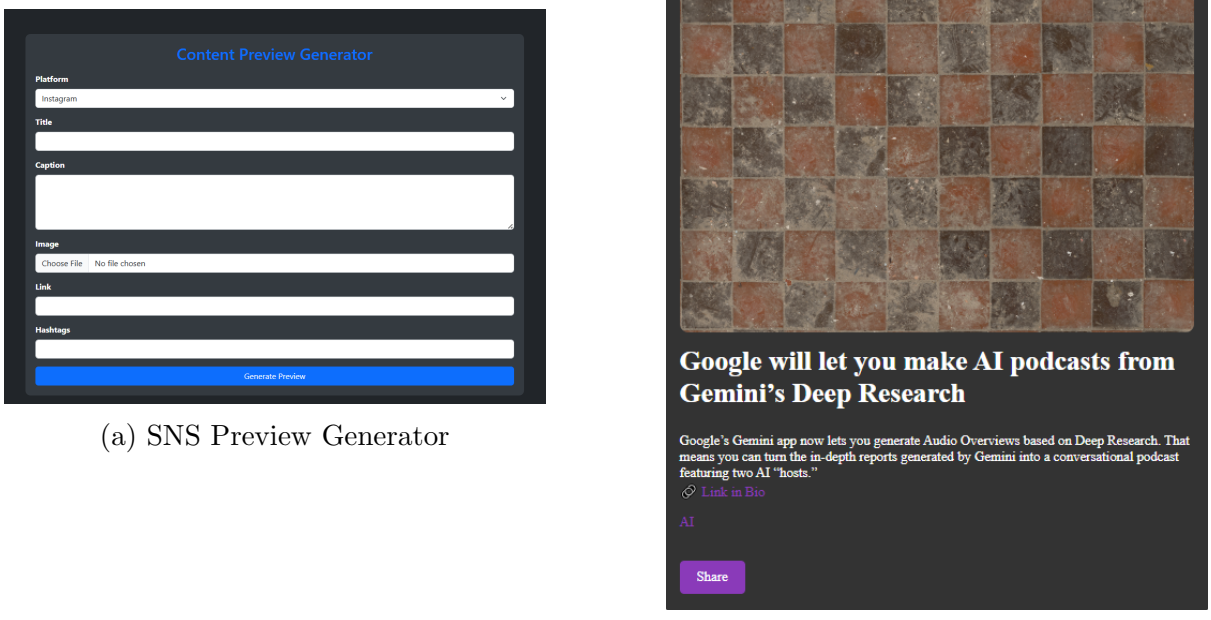


Figure 3.21: State Diagram - Share Via Email

3.4 Sprint - 4

3.4.1 Scrum - 11

In this section, we focused on creating SNS templates and sharing functionalities. We developed a dynamic preview function that allows users to input content for SNS platforms like Instagram and Reddit and immediately see a live preview. The following functionality was implemented using a Django view, which used prebuilt templates to capture details such as titles, captions, and images, then render a preview using platform-specific HTML. Additionally, we implemented a file upload mechanism that saves user images to a designated media folder, ensuring that the images are properly displayed. One major challenge was handling local image uploads. Initially, we used Base64 encoding, which cluttered the HTML and resulted in inconsistent image displays. This issue was resolved by switching to Django's File System Storage to save uploaded files directly into the media/SNS content folder, retrieving a clean URL for each image, which provided a reliable solution for displaying user-uploaded images. Furthermore, we implemented the "read out loud" functionality, a newly added sub-task to our Trend Tailor platform. Although the individual functionality has been implemented, it is not yet integrated into our system. The final implementation will be shown in our final progress report.



(a) SNS Preview Generator

(b) Success Email Sent

Figure 3.22: SNS preview template for Instagram

In the Figure above, the share button automatically allows the user to share content directly to their respective SNS platform.

Task	Associated User Stories	Priority
Investigate SNS formatting requirements	Scrum-11	Low
Create content templates for different SNS	Scrum-11	Medium
Implement preview function for contents	Scrum-11	Medium
Create content sharing functionality for SNS using templates	Scrum-11	Medium
Implement read out loud functionality (newly added tasks)	Scrum-11	Medium

3.4.2 Scrum - 12

In this sprint, we focused on developing the content scheduling functionality. We designed the content scheduling user interface by adding a "Schedule" button to the dashboard, along with a "Scheduled Content" section to store scheduled items. When the "Schedule" button is clicked, a Bootstrap modal appears, allowing users to choose the desired date, time, frequency, and article for scheduling. Once the information is submitted, it is displayed in the "Scheduled Content" section, showing the article, time, date, and frequency, along with edit and delete buttons. The edit button opens a modal that allows users to modify the time, date, and frequency, updating the scheduled content accordingly. Finally, the delete button allows users to remove the scheduled content if they no longer wish to post the article.

Please see Figures 3.23, 3.24, and 3.25 for reference.

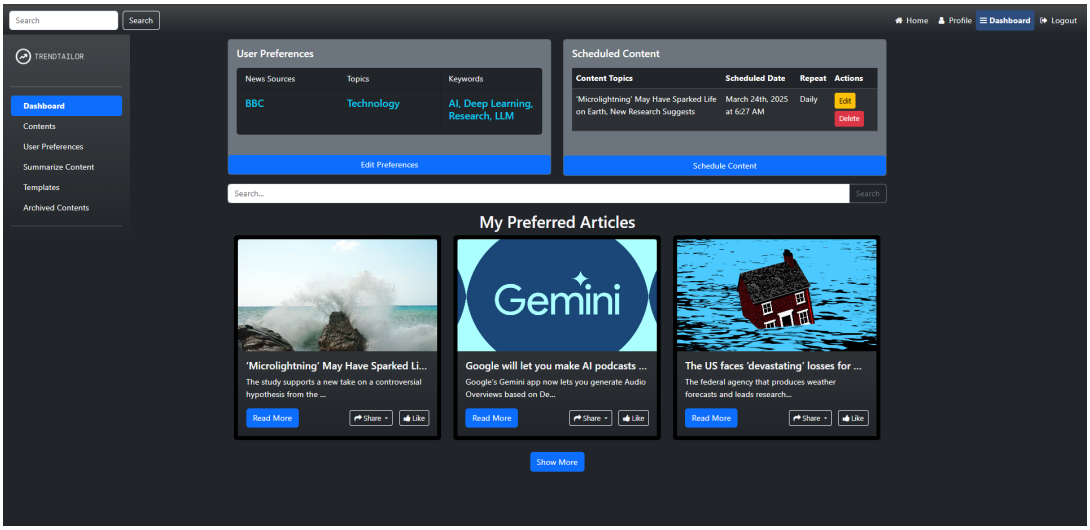
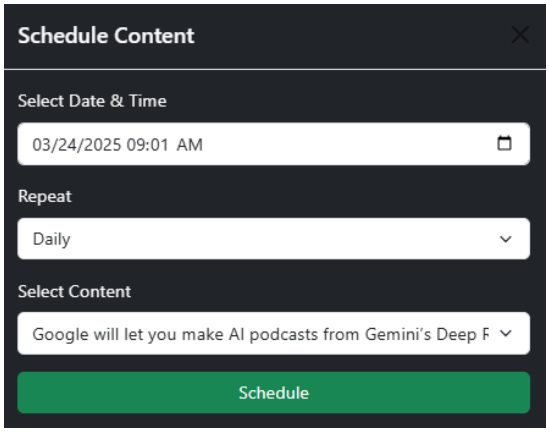
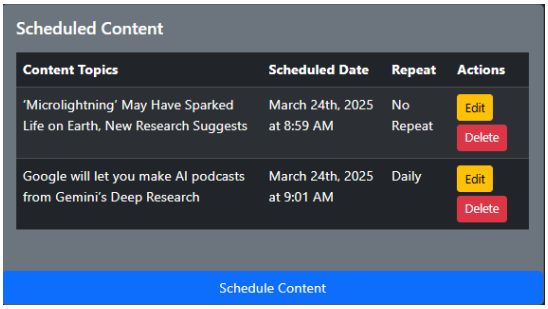


Figure 3.23: Dashboard with Schedule Module

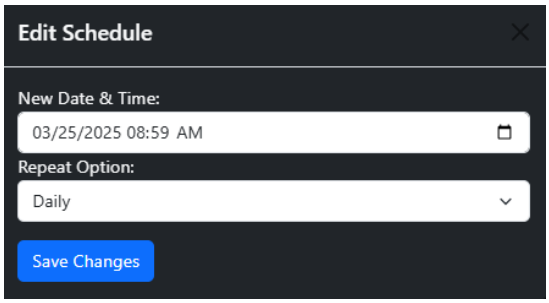


(a) Scheduling Content Modal

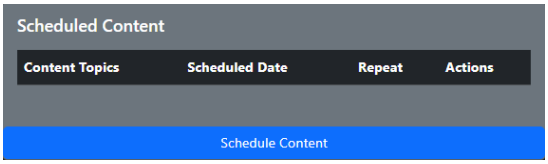


(b) New Schedule Added

Figure 3.24: Add and View Scheduling Content



(a) Edit Existing Schedule



(b) Schedules deleted

Figure 3.25: Scheduling Content: Edit & Delete

Figure 3.26: Schedule Content Sample

Task	Associated User Stories	Priority
Design the scheduling interface for content generation settings	Scrum-12	High
Implement UI to display, edit, and delete scheduled intervals	Scrum-12	High
Implement backend logic to store and manage user-defined scheduling intervals	Scrum-12	High
Develop automation logic to generate content at the specified intervals	Scrum-12	Medium
Conduct testing to ensure proper scheduling and content generation	Scrum-12	Medium

3.4.3 Diagrams for Sprint 4

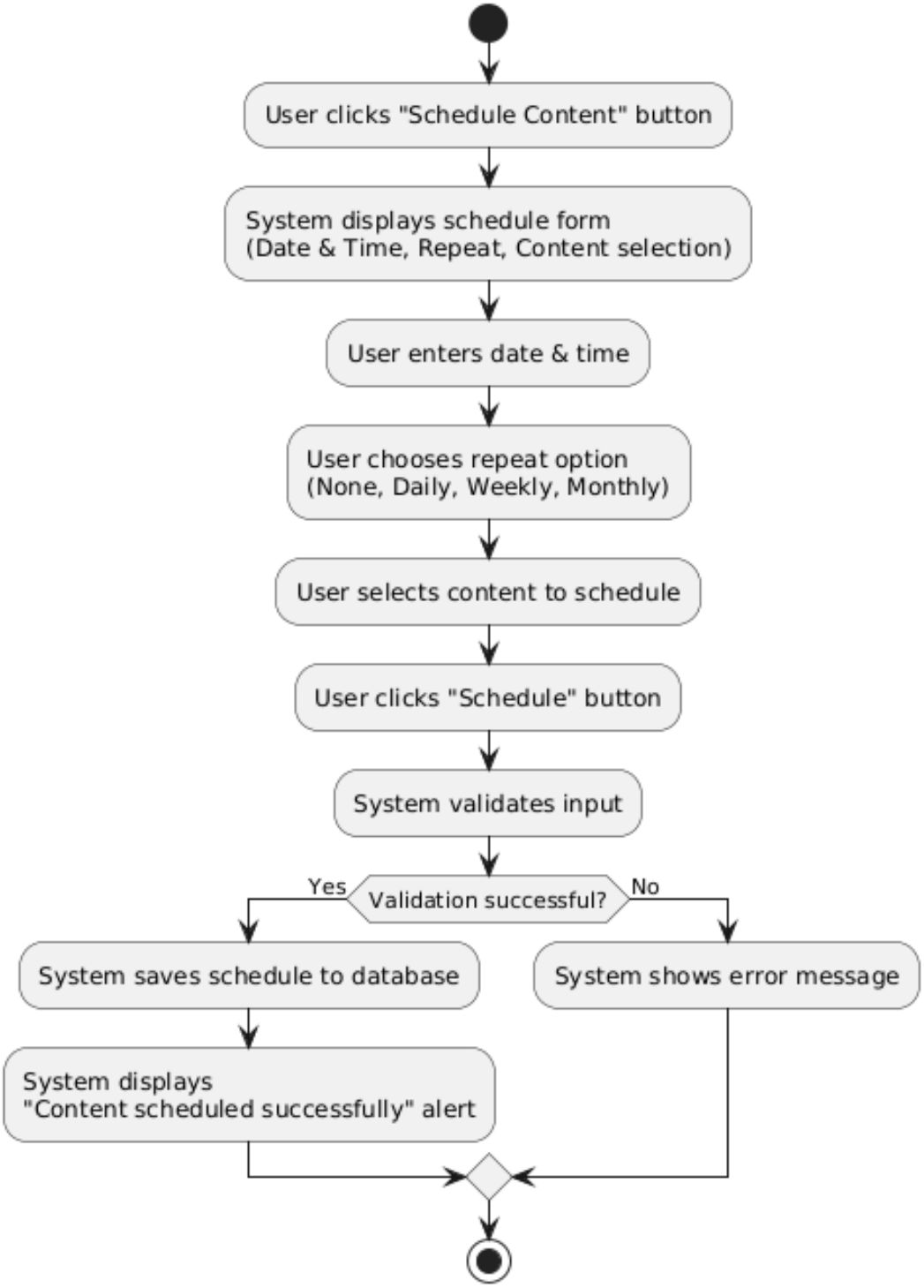


Figure 3.27: Activity Diagram - Content Scheduling

3.5 Sprint - 5

3.5.1 Scrum - 4

In this section, we focused on implementing the multi-language content generation functionality as part of our Pro Features. We began by creating a language selection menu that allows users to easily choose their preferred language before generating content. The menu was integrated into the user interface to provide a smooth and intuitive experience for users targeting a global audience. Next, we completed the functionality where the system dynamically generates content based on the language selected by the user. This was achieved by passing the selected language parameter through the Django backend, ensuring that the correct language model or translation pipeline is used during content creation. A major part of this work involved testing the language selection system to verify that the correct language output is consistently produced. We tested multiple language selections to ensure proper content rendering and identified and resolved edge cases where fallback defaults were needed. This ensures that users receive a reliable and high-quality multilingual content generation experience. The completed functionality lays the foundation for further enhancements, including potential automatic language detection and user profile-based language preferences. The subscription functionality and like option works as well.

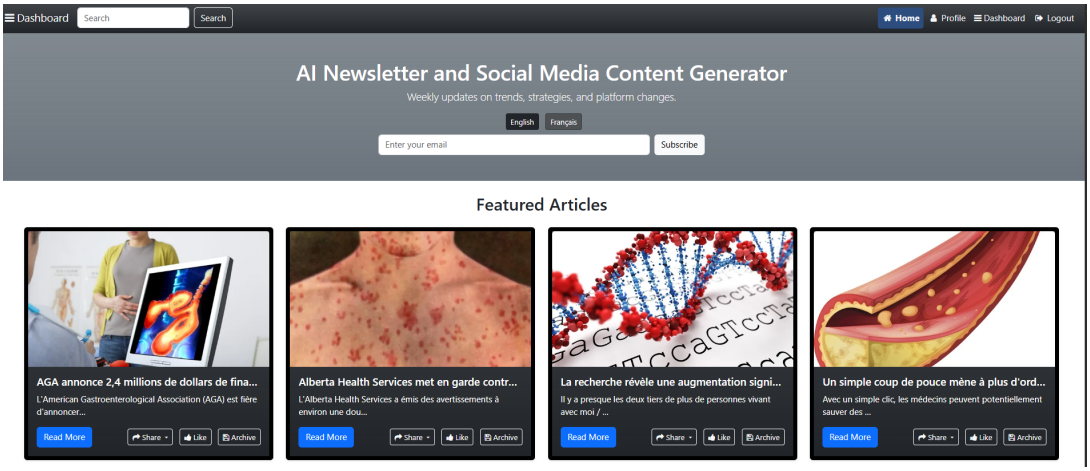


Figure 3.28: Preview of Multi language / French Displayed

Task	Associated User Stories	Priority
As a user, I want to be able to generate content in multiple languages (Pro Feature), so that I can cater to a global audience.	Scrum-4	Low
Create a menu section to select language.	Scrum-4	Low
Users selects languages from selection menu to generate content in selected language	Scrum-4	Low
Test language selection working properly	Scrum-4	Low

3.5.2 Scrum - 10

In this section, we focused on providing users with the ability to manage and export archived content through the dashboard. First, we created an Archive section within the user dashboard where all archived content is organized and easily accessible. This section was designed with a clear layout to improve user navigation and visibility of saved content. Next, we implemented search and filter functionality, allowing users to quickly find specific archived items based on keywords, dates, or content type. This made managing large volumes of archived material much more efficient. Finally, we added an export option that enables users to download selected archived content in a structured format, such as PDF or CSV, depending on the content type. This export functionality ensures users can back up their important content or reuse it across other platforms easily. Together, these features significantly improve the content lifecycle management within the platform, offering users more control, flexibility, and accessibility over their stored content.

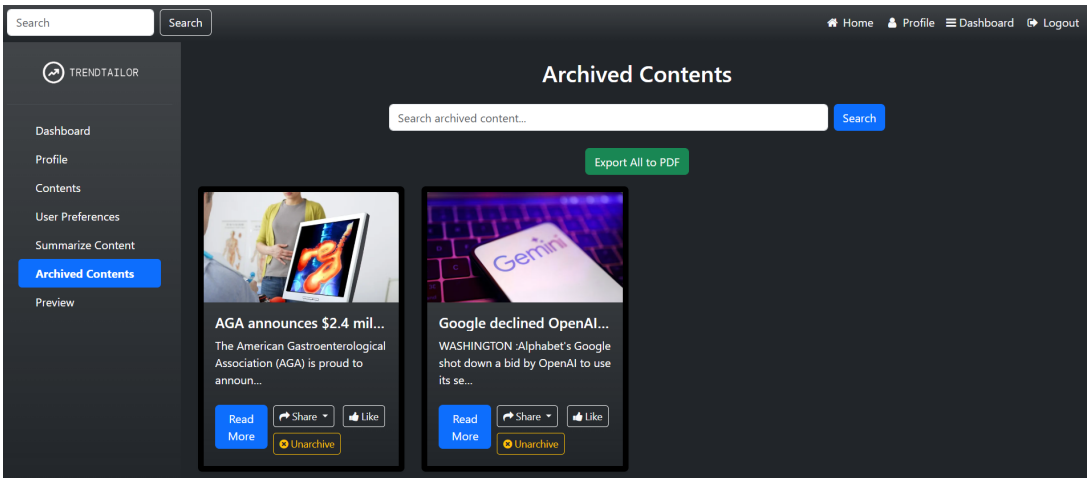


Figure 3.29: Preview of Archieved Content

Task	Associated User Stories	Priority
Provide an option to export archived content.	Scrum-10	Low
Create an archive section in the dashboard.	Scrum-10	Low
Allow users to search and filter archived content.	Scrum-10	Low
Provide an option to export archived content.	Scrum-10	Low

3.5.3 Scrum - 6

In this section, we focused on building API access for Trend Tailor to allow users to integrate its content generation and management features into their own applications. First, we designed and documented a set of API endpoints, clearly outlining how users can create, retrieve, and manage articles directly from the Trend Tailor database. This documentation included details about request structures, required parameters, and expected responses. Next, we developed a secure authentication mechanism using sign in-based

access, ensuring that only authorized users could interact with the API and access stored articles. We then implemented the backend logic for the API, connecting each endpoint to Trend Tailor's news content database to allow users to save newly generated articles and fetch previously saved articles as needed. Finally, we tested and validated the API functionality by simulating different user actions, verifying correct authentication, ensuring that articles are properly saved and retrieved from the database, and confirming consistent system behavior. With this implementation, Trend Tailor now enables users to fully manage their content programmatically, making the platform more extensible and developer-friendly. Please refer to the images below for having a close look for how the API is working in Trendtailor.

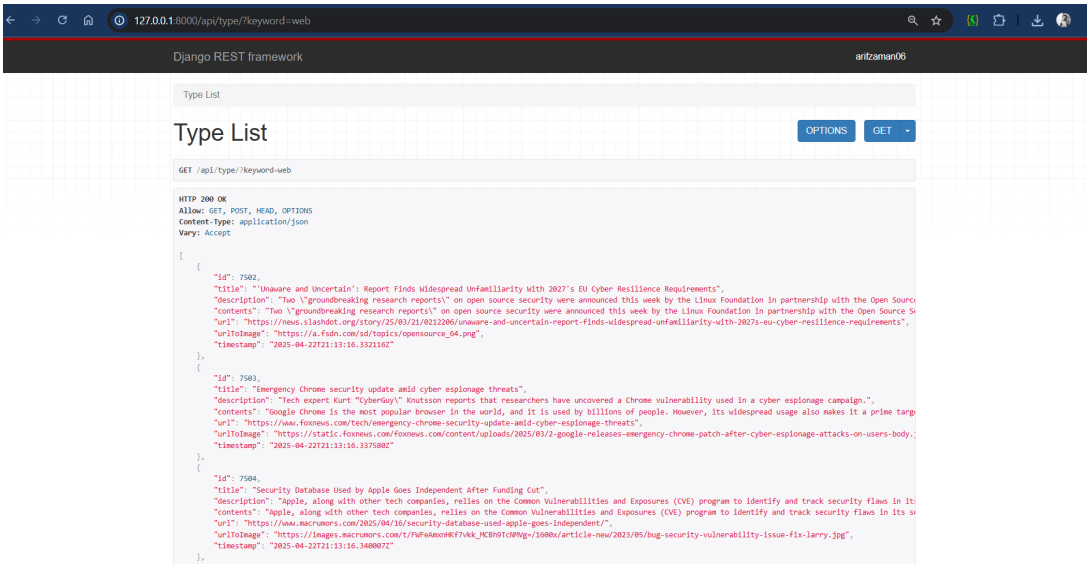


Figure 3.30: API search result via Keywords - Web(see URL)

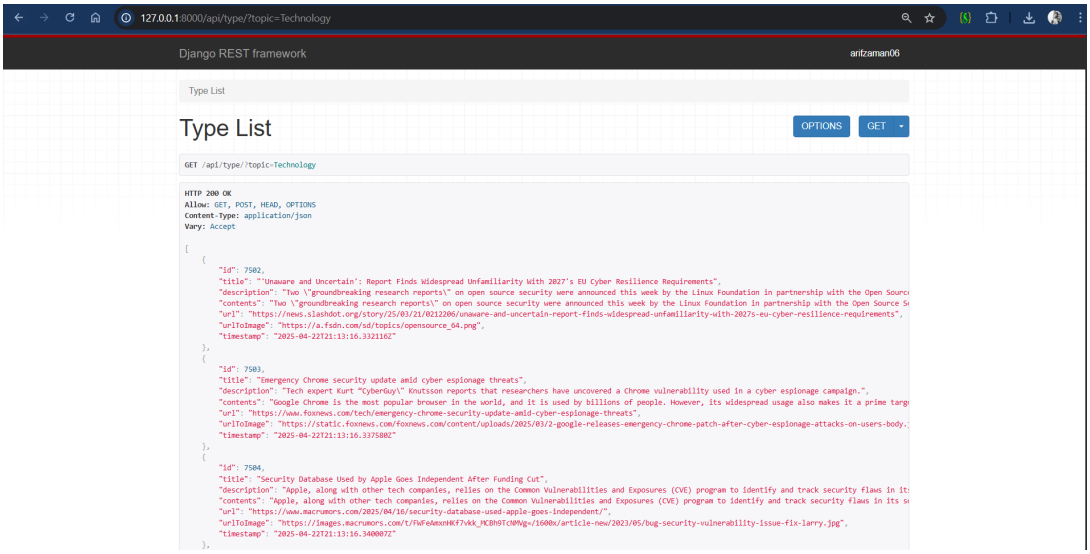


Figure 3.31: API search result via Topic - Technology(see URL)

Furthermore, in TrendTailor, APIs are used to fetch real-time news content, ensuring that other services or platforms can use our news content. Additionally, if we ever develop a mobile version of this, it will enable us to access the database using the API endpoint

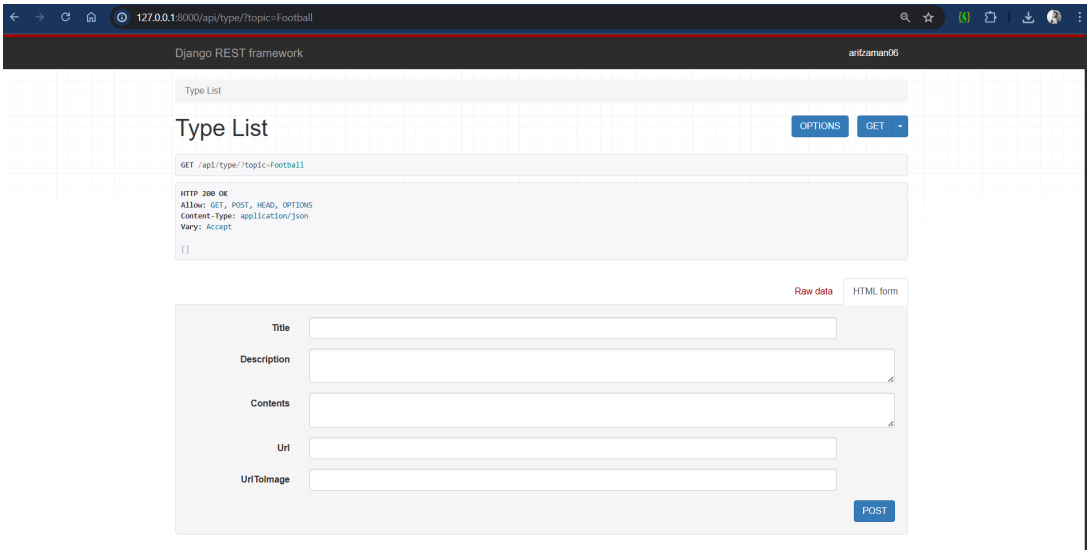


Figure 3.32: API search result via Topic - Football not available(see URL)

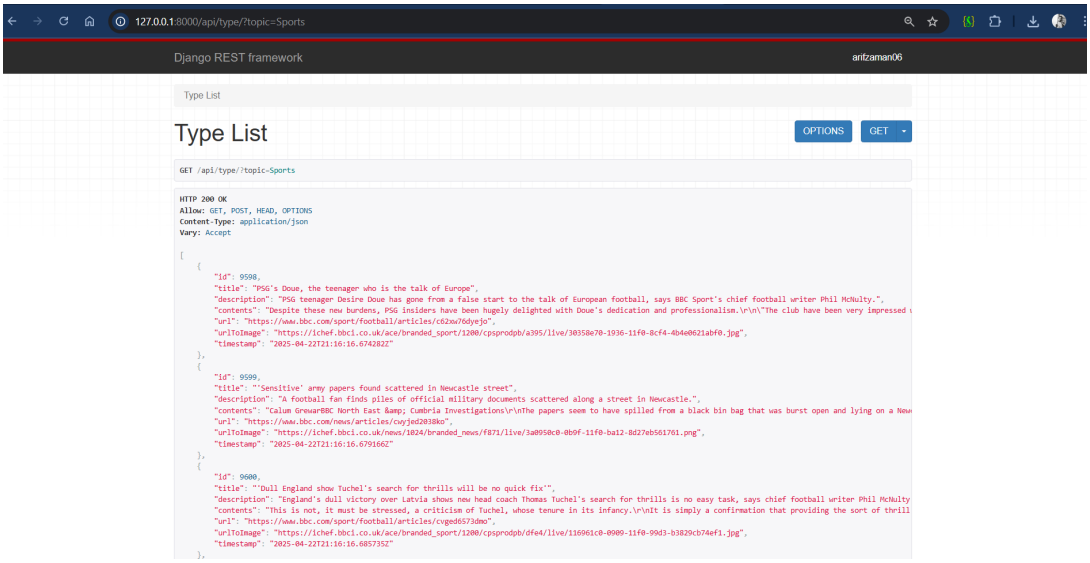


Figure 3.33: API search result via Topic - Sports available(see URL)

we created for this app. The API supports content searching by keywords and topics, allowing for the efficient retrieval of content.

Task	Associated User Stories	Priority
As a user, I want to be able to access an API to integrate the service into my own application so that I can extend its functionality.	Scrum-6	Medium
Design and Document API endpoints.	Scrum-6	Medium
Develop an authentication mechanism for authorized use only	Scrum-6	Medium
Develop API backend logic.	Scrum-6	Medium
Test and validate API functionality.	Scrum-6	Medium

4 Testing

4.1 Testing Methodology

To ensure the quality and stability of the TrendTailor application, a combination of testing strategies was applied across all five sprints:

Unit Testing:

Individual functions, classes, and modules were tested using Django's TestCase framework to validate isolated components such as user preferences, form submissions, article handling, and SNS content classes.

Integration Testing:

Features that involved interaction between multiple modules (e.g., article sharing with email templates, SNS content previews) were tested to ensure smooth data flow and integration.

End-to-End (E2E) Testing:

Full user journeys, from registration to content sharing, were simulated to ensure all functionalities worked as expected when combined.

Manual UI Testing:

Core user interface components like modals, dropdowns, email forms, and share buttons were manually tested across browsers for responsiveness and usability.

Regression Testing:

As new features were added in later sprints, existing functionality from earlier sprints was re-tested to ensure no regressions were introduced.

Code Coverage Analysis:

Python's coverage.py tool was used in combination with pytest to measure test coverage, identify untested code blocks, and track improvements across sprints.

4.2 Testing in Depth

4.2.1 Sprint 1: User Authentication and Profile Management

- `test_register_user` – Tests successful registration of a new user.
- `test_login_user` – Tests user login with valid credentials.
- `test_dashboard_access_with_login` – Verifies dashboard access for authenticated users.

- `test_dashboard_access_without_login` – Verifies that unauthenticated users are redirected.

- `test_logout_user` – Tests user logout functionality.

- `test_profile_update` – Tests updating user profile information through the edit form.

- `test_profile_image_upload` – Tests uploading a new profile image.

4.2.2 Sprint 2: User Preferences and Article Management

- `test_update_preferences` – Tests saving user-selected news topics, sources, and keywords.

- `test_articles_saved_in_database` – Verifies correct storage and retrieval of articles in the database.

- `test_articles_show_on_homepage` – Tests that stored articles appear properly on the homepage.

4.2.3 Sprint 3: Homepage, Pagination, and Subscription

- `test_homepage_access` – Tests successful access to the homepage.

- `test_homepage_pagination` – Tests homepage pagination across multiple pages of articles.

- `test_subscribe_valid_email` – Tests subscribing to the newsletter with a valid email.

- `test_subscribe_duplicate_email` – Tests error handling when trying to subscribe with an already subscribed email.

- `test_subscribe_invalid_email` – Tests validation error for invalid email formats.

4.2.4 Sprint 4: SNS Preview and Sharing

- `test_instagram_preview_post` – Tests generation of Instagram preview content using user input and uploaded image.

- `test_reddit_preview_post` – Tests Reddit preview rendering using form data and image upload.

- `test_preview_invalid_platform` – Ensures proper error message and status code for unsupported SNS platforms.

- `test_preview_get_render` – Verifies that the preview page loads properly for GET

requests.

4.2.5 Sprint 5: Content Scheduling and Email Sharing

- `test_schedule_content_success` – Tests scheduling an article for future sharing with repeat options.

- `test_schedule_content_invalid_data` – Tests error handling when scheduling data is missing or invalid.

- `test_edit_schedule` – Tests editing an existing scheduled post (date and repeat options).

- `test_delete_schedule` – Tests deleting an existing scheduled post.

- `test_get_template_content` – Tests retrieval of email template content for article sharing.

- `test_send_email_success` – Tests sending an email with selected template content.

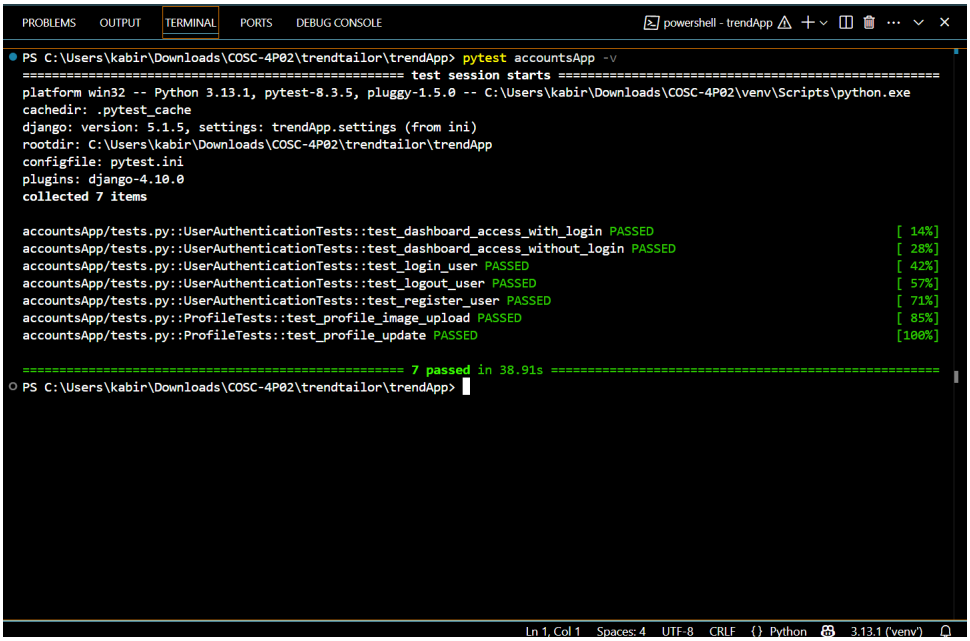


Figure 4.1: Test 1

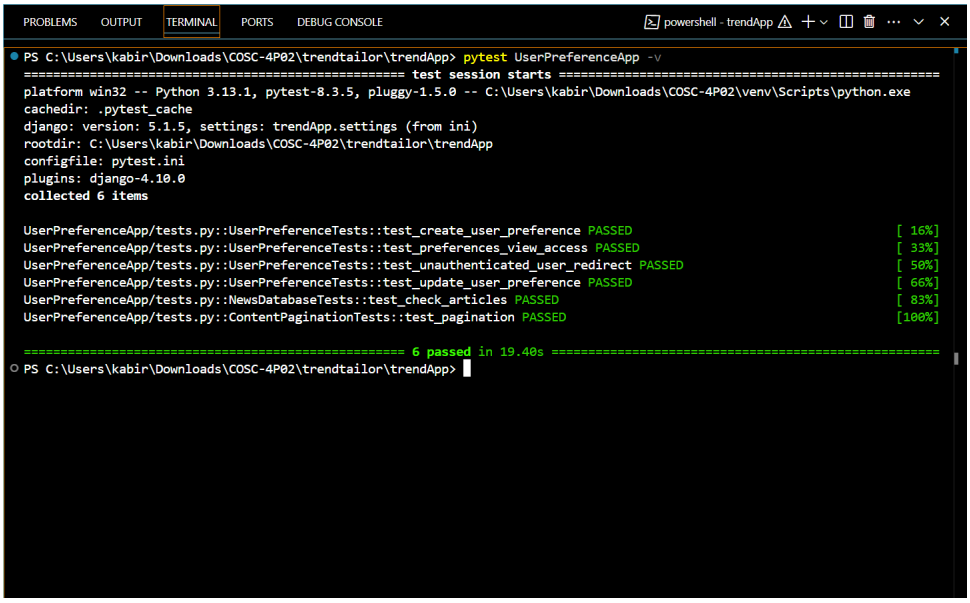


Figure 4.2: Test 2

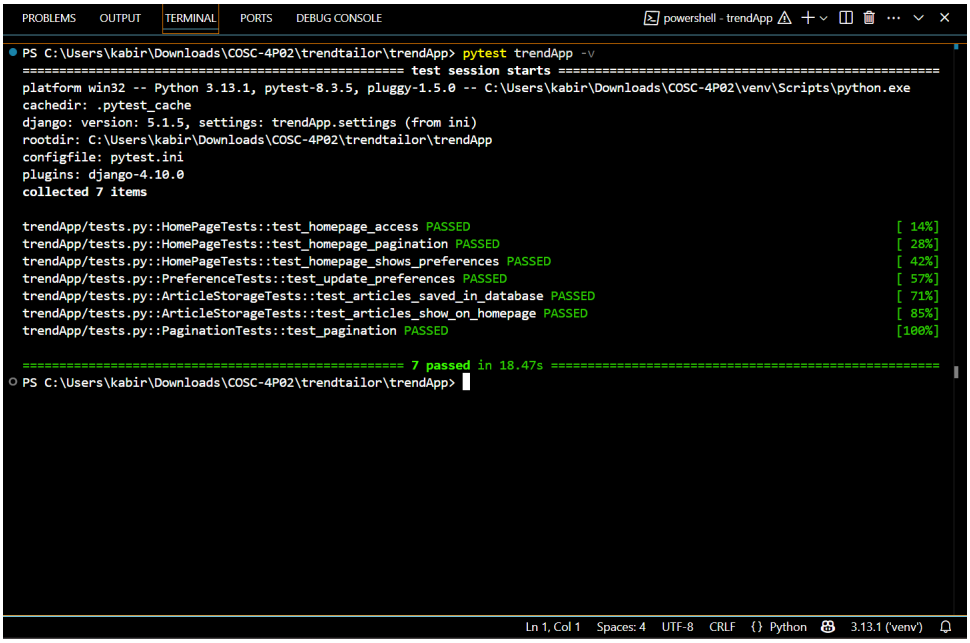


Figure 4.3: Test 3

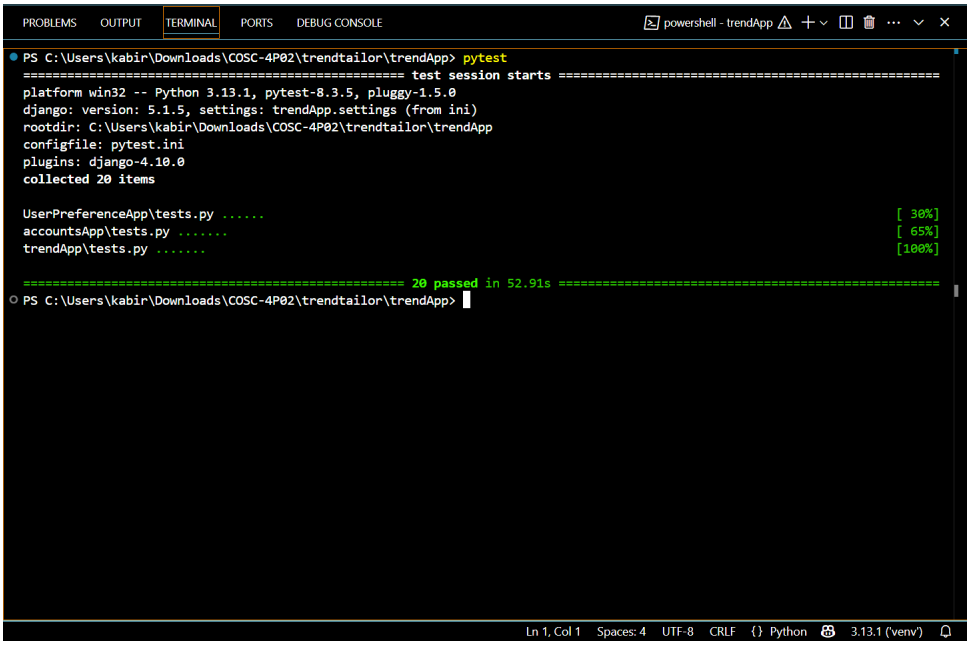


Figure 4.4: Test 4

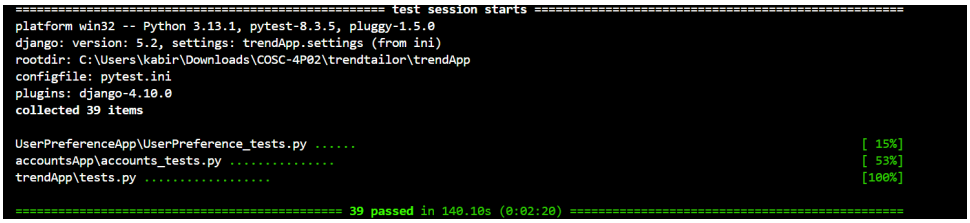


Figure 4.5: Test 5

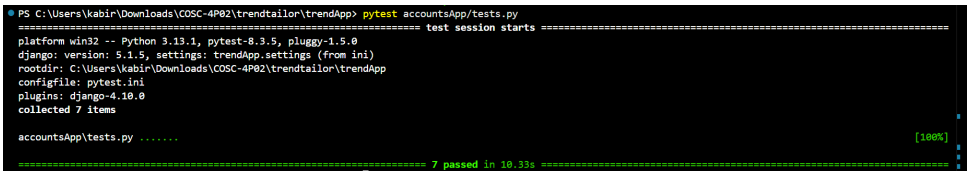


Figure 4.6: Test 6

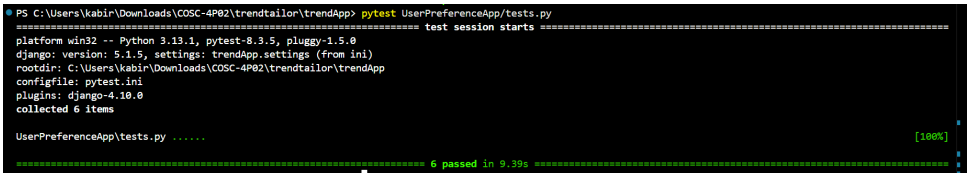


Figure 4.7: Test 7

4.3 Code Coverage Analysis

Testing coverage was tracked using the coverage.py tool. The final code coverage report after Sprint 5 shows: Line Coverage: 78

Modules Covered: All critical Django views, forms, and models.

Untested Sections: Minor utility methods and error branches not directly triggered through standard workflows.

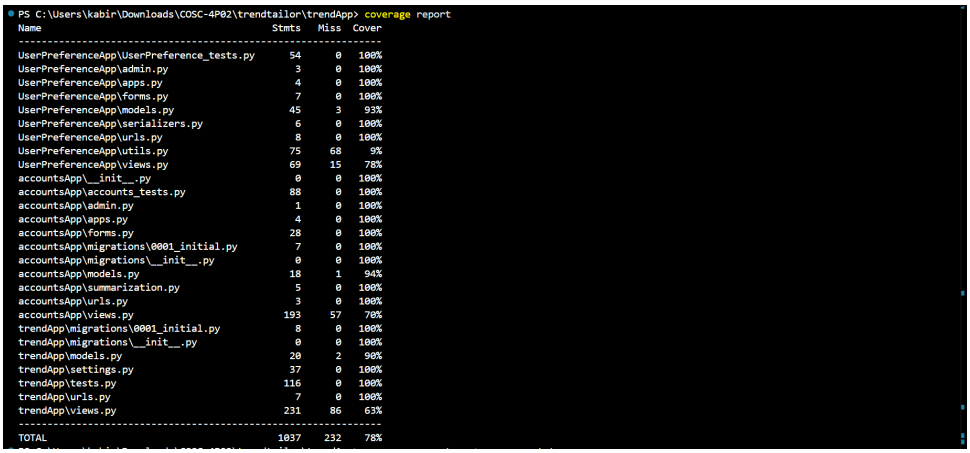


Figure 4.8: Code Coverage

4.4 Security/Penetration Testing 1

4.4.1 Objective

To assess the security of the Django-based web application running on `localhost:8000`, focusing on:

- Authentication security (login page)
- CSRF protection validation

4.4.2 Testing Methodology

Tools Used

- Hydra: For brute-force attack simulation.
- Burp Suite: For HTTP request interception and manual testing.
- Python Requests: For automated CSRF-protected login testing.
- Browser DevTools: For inspecting responses and debugging.

Test Cases & Findings: Login Page Brute-Force Attack (Hydra) Target: http://127.0.0.1:8000/accounts/login/

Test Case:

- Attempted brute-forcing with known username (`jeffin@`) and password (`qwertyuiop123@`).

Command Used:

```
hydra -l jeffin@ -p qwertyuiop123@ 127.0.0.1 http-port-form "/accounts/login:
username=^USER^&password=^PASS^:F=incorrect" -V -s 8000
```

Result: Failed (Connection errors due to CSRF protection). Server Response: `403 Forbidden (CSRF token missing)`.

Conclusion: The login page is protected against Hydra brute-forcing due to CSRF tokens.

CSRF Token Validation (Burp Suite) Test Case:

- Intercepted a valid login request to analyze CSRF protection.

Findings:

- The login form requires a `csrfmiddlewaretoken` in POST data.
- A matching `csrftoken` cookie.
- Missing/invalid tokens result in `403 Forbidden`.

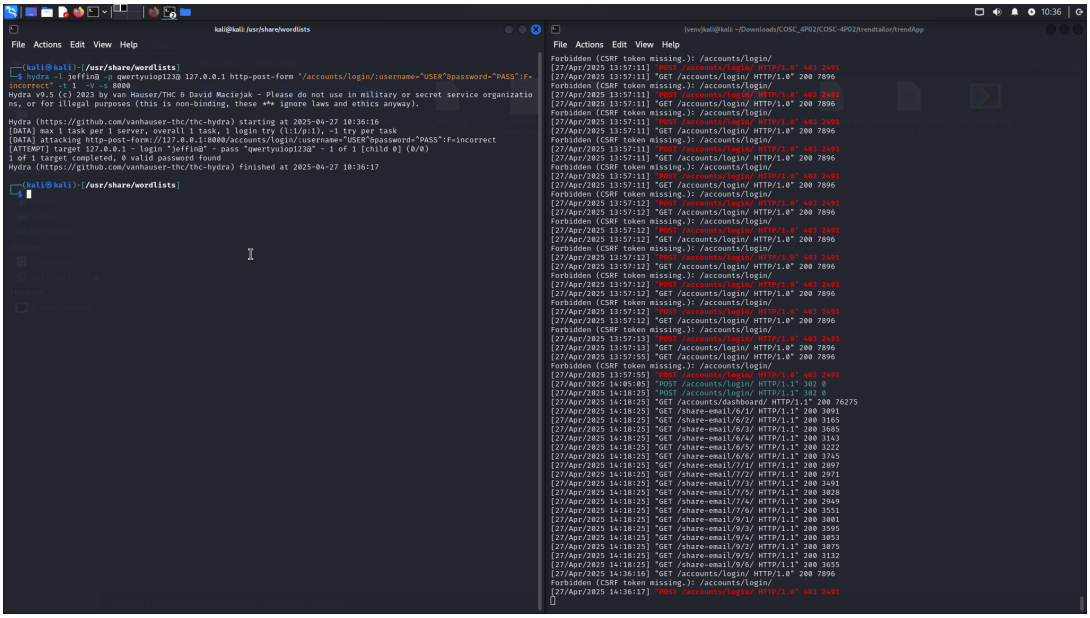


Figure 4.9: Brute-Force Attack Using Hydra

Conclusion: CSRF protection is properly implemented, preventing automated attacks without token handling.

4.4.3 Conclusion

The login system is secure against automated brute-forcing due to CSRF checks. Deployment inconsistencies (template paths) need resolution. No critical vulnerabilities found, but rate limiting should be added for defense in depth.

RISK	Severity	STATUS	Remediation
CSRF Bypass (Hydra)	Low	Mitigated	CSRF tokens block brute-forcing.
No Rate Limiting	Medium	Open (if applicable)	Implement login attempt throttling.

Table 1: Summary of Risks Identified

4.5 Security Testing 2

4.5.1 Objective

A security assessment was conducted on the login page (`/accounts/login/`) to test for SQL injection vulnerabilities. The test was performed using `sqlmap`, an automated SQL injection detection tool. The findings indicate that the login form does not appear to be vulnerable to basic SQL injection attacks. However, the server responded with `403 Forbidden` errors, suggesting the presence of security mechanisms such as CSRF protection or rate limiting.

4.5.2 Testing Methodology

Target Analysis

- URL: http://127.0.0.1:8000/accounts/login/
- HTTP Method: POST
- Parameters Tested: `username`, `password`

4.5.3 Tools Commands Used

```
sqlmap -u "http://127.0.0.1:8000/accounts/login/"
       --data="username=test&password=test"
       --method POST
```

4.5.4 Testing Scope

- Boolean-based blind SQLi
- Time-based blind SQLi
- Error-based SQLi
- UNION query-based SQLi
- Stacked queries (where applicable)

Parameter	Injection Type Tested	Result	Notes
username	Boolean-based	Not Vulnerable	No dynamic behavior detected
username	Time-based	Not Vulnerable	No delayed responses
username	UNION-based	Not Vulnerable	No UNION query success
password	Boolean-based	Not Vulnerable	No dynamic behavior detected
password	Time-based	Not Vulnerable	No delayed responses
password	UNION-based	Not Vulnerable	No UNION query success

Table 2: SQL Injection Test Results

4.5.5 Security Observations

- **403 Forbidden Errors (246 occurrences)**
  - Indicates possible CSRF protection (common in Django).
  - Could also be due to rate limiting or security middleware.
- **No SQL Injection Detected**
  - The login form appears to be secure against basic SQLi attacks.
  - Likely due to proper use of Django ORM or parameterized queries.

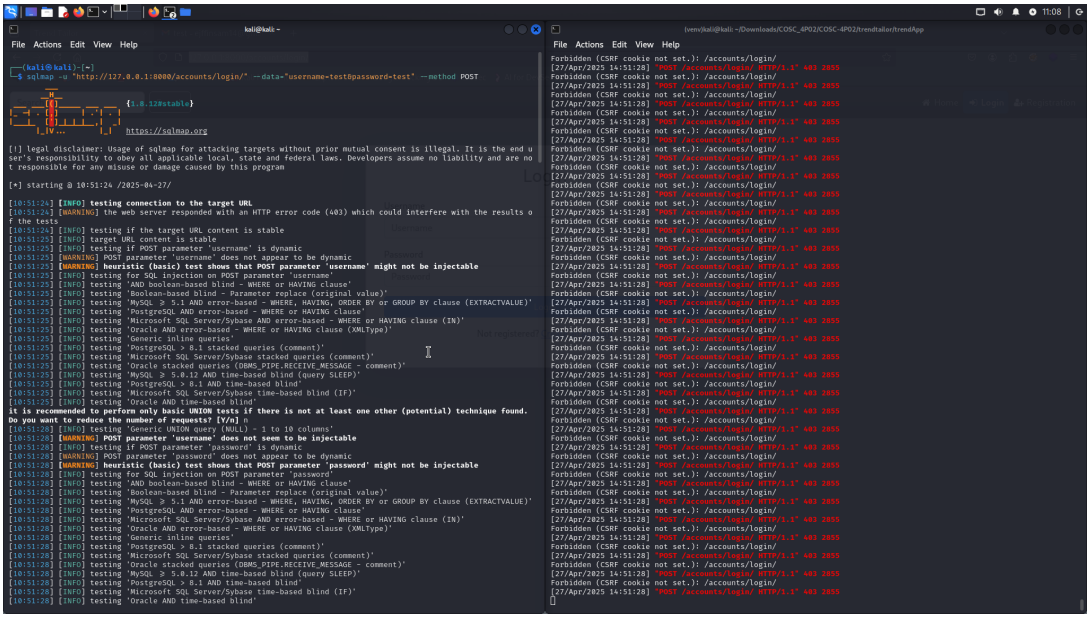


Figure 4.10: SQL Injection

4.5.6 Conclusion

The login page does not appear vulnerable to SQL injection based on automated testing. However, the `403 Forbidden` errors suggest additional security layers are in place. Further manual review and advanced testing (with CSRF handling) are recommended to ensure complete security.

5 Sprints & Meetings

5.1 Weekly Scrum Meetings

Throughout the development of TrendTailor, weekly Scrum meetings were a cornerstone of our Agile workflow. Led by Arifuzzaman (Scrum Master), we met every Friday to provide updates on completed tasks, discuss ongoing work, and collaboratively solve challenges. These structured meetings allowed us to track progress effectively, plan upcoming tasks, and ensure alignment with our sprint goals.

Alongside scheduled meetings, we maintained constant collaboration through pair programming sessions. These sessions enhanced code quality, allowed knowledge sharing across the team, and ensured consistent development standards. We also frequently held impromptu one-on-one discussions on Discord to quickly resolve issues and maintain momentum.

At the end of each sprint, we conducted Sprint Retrospectives to reflect on what went well, identify areas for improvement, and adjust our workflows accordingly. From the initial Sprint Planning Meeting—where we broke down user stories into manageable tasks and organized our product backlog—to ongoing evaluations of our Agile-Scrum practices, our focus on continuous improvement kept us aligned and motivated.

Through open communication, regular feedback loops, and collaborative efforts, we steadily transformed our project into a functional, scalable, and high-quality web application.

5.2 Sprint Burndown Charts

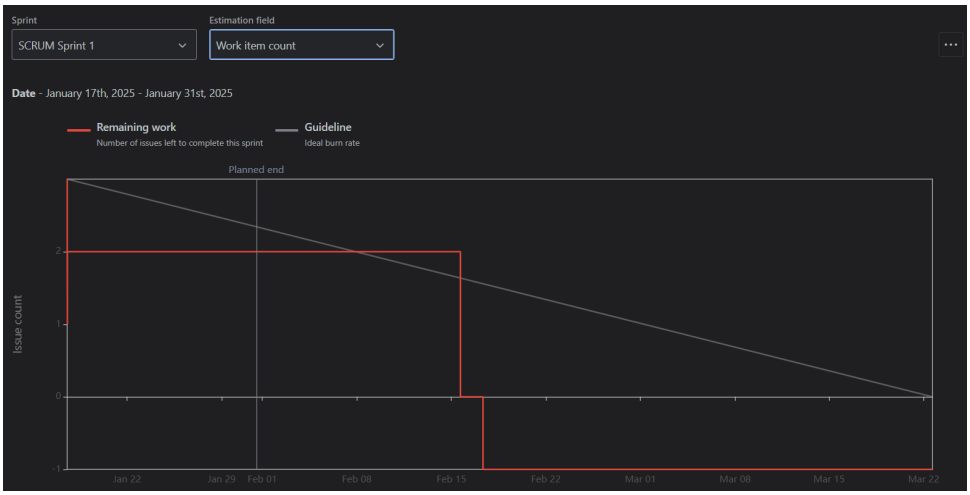


Figure 5.1: Sprint Burndown Chart - Sprint 1

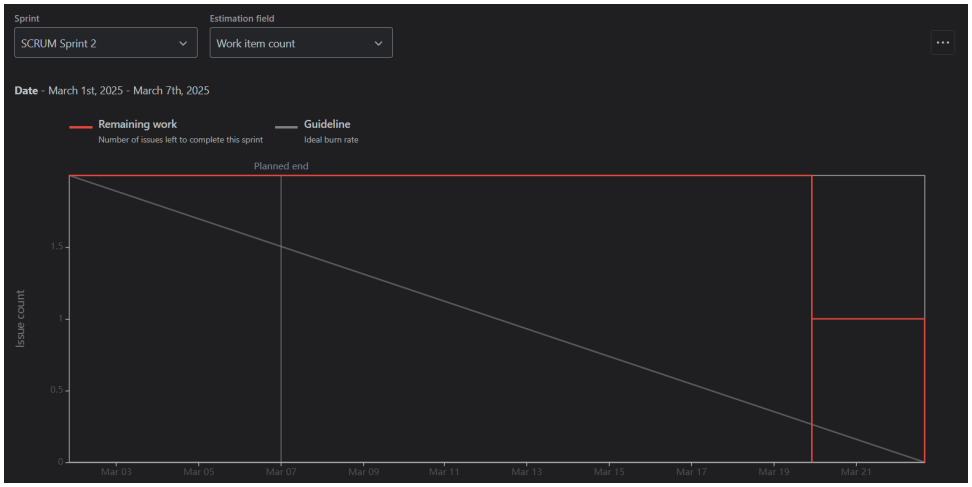


Figure 5.2: Sprint Burndown Chart - Sprint 2

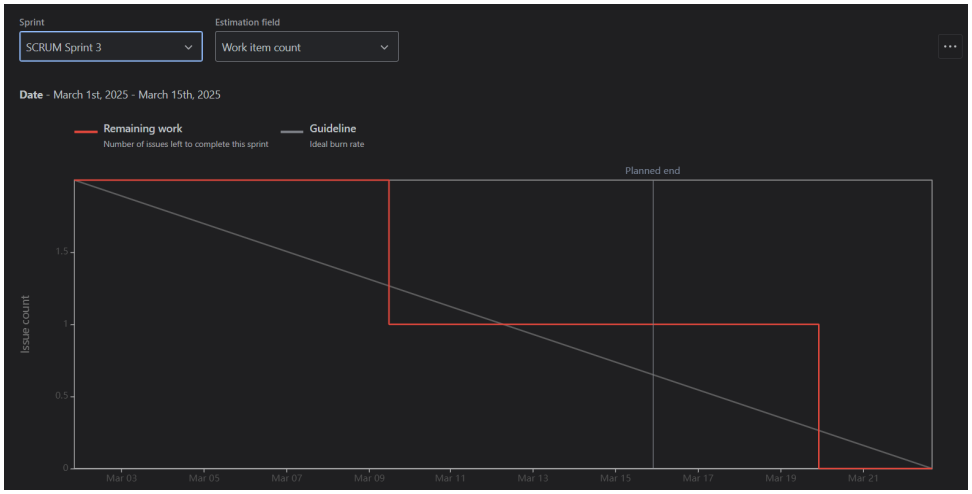


Figure 5.3: Sprint Burndown Chart - Sprint 3

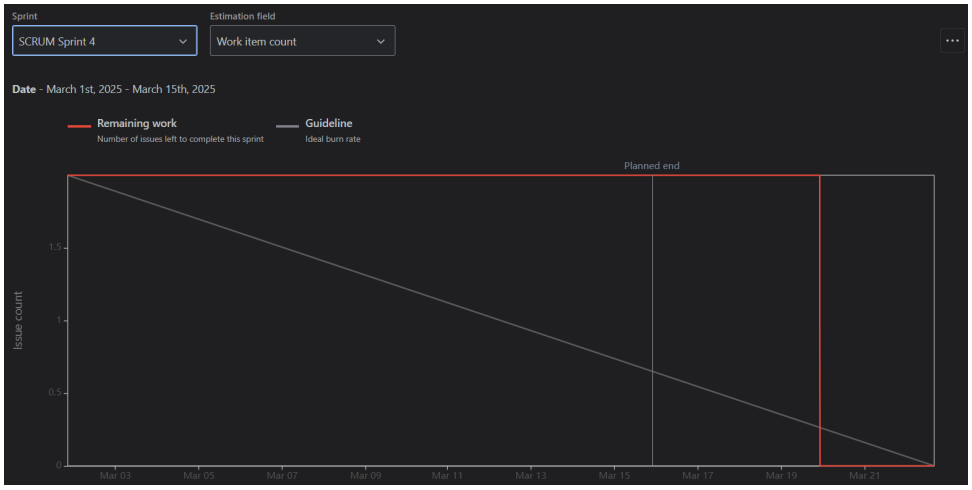


Figure 5.4: Sprint Burndown Chart - Sprint 4

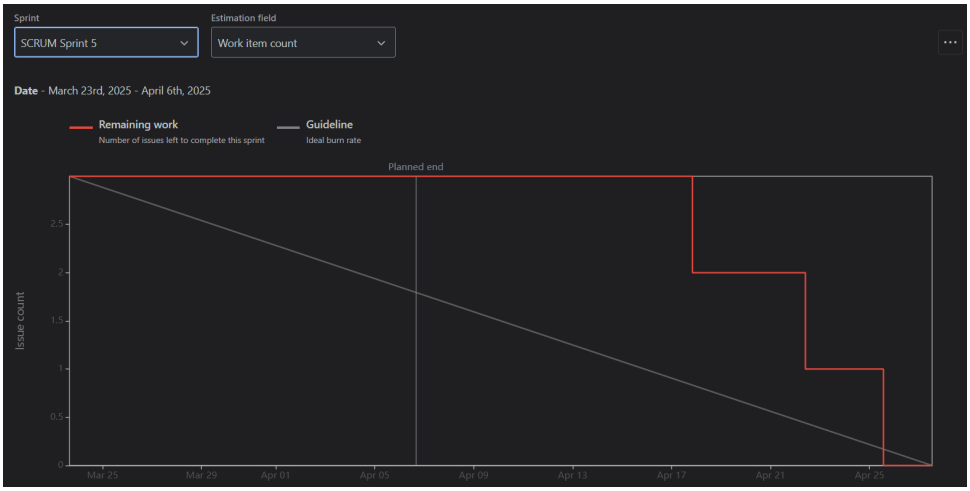


Figure 5.5: Sprint Burndown Chart - Sprint 5

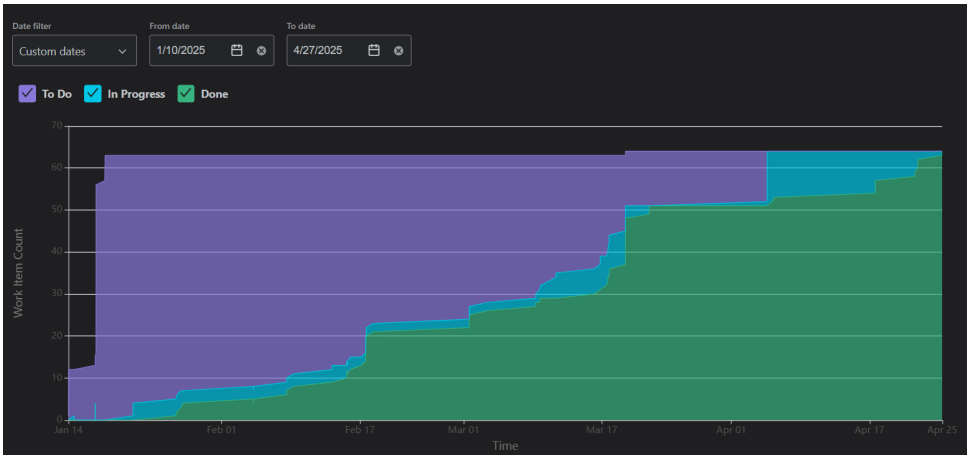


Figure 5.6: Cumulative Flow Diagram

5.3 Github

For project version control we utilize GitHub as mentioned in our previous report. The following GitHub link has all our commits and work. This section highlights key aspects of our work such as Commits Over Time, and Individual Contribution, as shown in the Figures below.

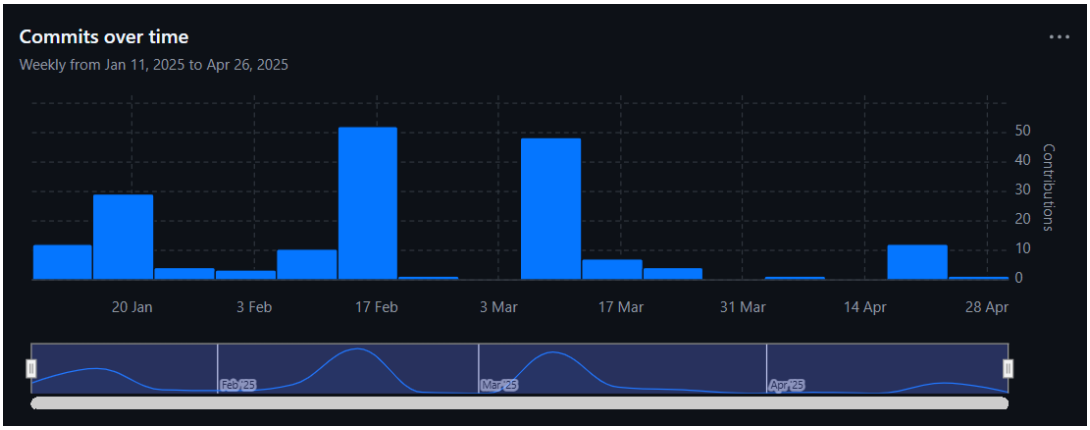


Figure 5.7: Commits Over Time

Figure 5.8: Individual Commit

5.4 Technologies and Tools Utilized

1. Python, Django

Used for backend management and project structure, enabling efficient handling of server-side logic and data processing.

2. HTML, CSS, Javascript, Bootstrap

Utilized for frontend development, ensuring responsive and visually appealing design across devices.

3. Github

Employed for version control, collaboration, and managing merge conflicts during development.

4. JIRA

Used for backlog and task management, organizing sprints, and tracking project progress in an Agile environment.

5. Discord

Leveraged for team communication, providing a platform for real-time discussions, quick issue resolution, and collaboration.

6 Challenges, Limitations & Future Work

Challenges

Throughout the project, we faced a range of challenges that tested our adaptability, technical skills, and teamwork. Early in the development process, setting up the Django environment posed difficulties, especially for team members new to the framework. Configuring views, URLs, templates, and managing dependencies led to initial delays. Additionally, managing collaborative development through GitHub created complications, including merge conflicts and improper file commits due to misconfigured .gitignore files, which required troubleshooting and careful version control practices.

As the project evolved, content sharing across different social media platforms introduced new technical complexities. Each platform had specific requirements for metadata, images, and text formatting, making it difficult to maintain consistency. We also faced challenges with handling local image uploads. Our initial approach using Base64 encoding resulted in bloated HTML and inconsistent rendering, which we later resolved by leveraging Django's File System Storage for cleaner media handling.

Implementing scheduled content posting brought another layer of complexity. Managing time zones, ensuring accurate publishing intervals, and dynamically allowing users to edit or delete scheduled posts required careful backend logic design. Furthermore, adding features like "read out loud" accessibility required multiple iterations to ensure smooth, cross-browser compatibility. Despite these hurdles, our team remained resilient, collaborating effectively to refine our solutions and maintain steady progress.

Limitations

One of the primary limitations of the platform stems from the constraints of free news APIs. Due to usage limits and restricted access, the content fetched from external APIs is often truncated, with only a portion (approximately 500 characters) of each article retrieved. This restricts the depth and richness of the content displayed to users. Additionally, the limited number of API calls per day further impacts the freshness of aggregated content, especially when serving a larger user base. Overcoming these limitations would require moving to premium APIs or developing custom data aggregation pipelines.

Future Work

Looking ahead, there are several clear pathways for enhancing the platform. One immediate focus will be upgrading to premium API services to remove existing content limitations, ensuring the platform can deliver complete articles and real-time updates. In addition, integrating a secure payment system will allow users to subscribe to premium features, such as expanded content access and advanced customization options.

From an architectural perspective, future iterations will prioritize building a more modular and scalable system. This will include supporting multiple content types and distribution channels, enhancing platform flexibility, and making the system infrastructure future-ready. Expanding the ecosystem to include additional services—such as podcast

integration, video content sharing, and broader SNS support—will further enrich the user experience and position the platform for long-term growth.

7 Team Contributions

Team Member	Contributions
Arifuzzaman (Scrum Master)	Developed the homepage, dashboard, and profile pages with an emphasis on a modern, intuitive user interface. Assisted in designing dynamic email templates. Designed a UI for content summarization and implemented summarization using LLMs. Developed a language switch button and supported multi-lingual functionality. Adhered to Agile-Scrum practices, including sprint planning, daily standups, and retrospectives. As Scrum Master, managed team communications, facilitated meetings, and ensured adherence to Agile principles, ensuring a smooth workflow and timely delivery.
Hridoy Rahman	Contributed to setting up the Django project foundation, managing the GitHub repository, resolving conflicts, and assisting with version control. Developed backend functionalities for generating content, updating user preferences, liked button, subscription, search functionality and managing database storage. Ensured seamless integration of all components and conducted system testing. Integrated the LLM summarization functionality, refined sharing functionalities, and redesigned the Dashboard UI. Also, integrated the scheduling functionality, improved code modularity, and enhanced the existing codebase. Worked on API integration and contributed to adding language support to the platform.
Jeffin Sam Joji	Worked on the backend to aggregate news content from multiple free APIs and implemented data scraping. Developed error handling mechanisms, secure API key management, and reusable functions for fetching and processing articles. On the frontend, integrated the share button functionality for various social media platforms and created activity diagrams for process visualization. Contributed to the implementation of language support for the platform. Tested the security and vulnerabilities of the site, including SQL injection and brute force attacks using Hydra.
Yuanhan Huang	Worked on front-end development, designing and styling the input form for user news preferences. Developed dynamic and responsive email templates and assisted with UI/UX design elements. Contributed to database management for storing user preferences.Developed a dynamic preview function for SNS platforms like Instagram and Reddit, enabling live previews of prebuilt templates. Implemented a file upload mechanism for storing and displaying user images correctly. Implemented archiving content functionality and added an option for exporting to PDF.

Team Member	Contributions
Kabir Sethi (Product Owner)	Managed product backlog, worked on front-end development, including user login and registration. Implemented form validation and assisted in setting up database authentication. Wrote test cases for all key functionalities of the project, including unit tests, end-to-end tests, integration tests, manual UI testing, and regression testing. Ensured proper code coverage across all areas.
Sahil Rashid	Developed user authentication with secure login, registration, and account management functionality. Created Figma mockups for design visualization. Implemented email sharing functionality using a Bootstrap modal, allowing users to share articles via Gmail and Outlook. Designed and developed the content scheduling interface and functionality, enabling users to edit, manage, and set scheduled posts.
Jayant Saini	Assisted with the implementation of article summarization and researched the "Read Out Loud" feature, ensuring accurate speech synthesis for improved accessibility. Investigated and defined SNS formatting requirements for seamless content sharing across multiple platforms. Created sequence diagrams and state diagrams to visually represent system processes, data flow, and interactions, contributing to better understanding and communication within the team.