IT314
SOFTWARE ENGINEERING

Software Requirements Specification

Group ID 6
Topic: Online Blogging Platform

SR	Student ID	Name
1.	202201008	Smruti Parmar
2.	202201014	Deep Patel
3.	202201030	Jainil Patel
4.	202201034	Harshit Kumar
5.	202201037	Rishi Patel
6.	202201041	Dhriti Goenka
7.	202201080	Kanishk Kunj
8.	202201083	Sahil Vaghasiya
9.	202201090	Denil Antala

Under guidance of:
Prof Saurabh Tiwari

INDEX:

1. INTRODUCTION:

The Software Requirements Specification (SRS) provides an overview of the entire document, covering its purpose, scope, definitions, abbreviations, references, and a general summary. This document aims to thoroughly analyze and offer a comprehensive understanding of the Blogging Platform by clearly defining the problem statement. Additionally, it highlights the functionalities required by stakeholders and their needs while outlining the high-level features of the platform. More detailed requirements for the blogging platform will be outlined in subsequent sections of the document.

1.1 Purpose of the document:

This document describes the requirements for the blogging platform, with an emphasis on user needs and functionality. It clarifies how the platform will be utilised while also offering insight into the project's aims and future development. The SRS addresses critical issues such as target audience, user interface, and software needs from the viewpoints of customers, the development team, and users. It also helps designers and developers through the whole software development life cycle (SDLC).

1.2 Problem Description:

In the current digital landscape, there is a need for a platform that empowers users to create and share engaging blog content. Existing solutions often lack features like rich text editing, media uploads, tagging, and seamless social sharing, which are essential for modern blogging. Additionally, users require the ability to follow their favorite bloggers and receive notifications for new posts, enhancing content discovery and engagement. Effective management of users and content is also a challenge, highlighting the need for an intuitive admin panel. This platform aims to address these gaps by providing a comprehensive and user-friendly solution for bloggers and readers alike.

1.3 Scope:

The Blogging Platform will serve as a comprehensive solution for content creators and readers. It will provide advanced tools like rich text editing, media uploads, tagging, and social sharing to enhance blogging capabilities. Readers will benefit from features such as commenting, upvoting, following bloggers, and receiving notifications about new posts. The platform will also include a secure authentication system and a scalable backend to support optimal performance. Furthermore, an admin panel will be implemented for streamlined user and content management. With a focus on usability, scalability, and interactivity, this platform will foster an engaging and dynamic blogging community.

1.4 Technologies Used:

Frontend: Tailwind, Custom CSS
Backend: Next JS
Authorization: Firebase
Database: Firebase
Unit Testing: Mocha, Jest
GUI Testing: Selenium Web Driver
Load Testing: Apache Load Tester

2. OVERALL DESCRIPTION:

This section gives you the gist of our whole project, including details of the users and stakeholders involved in this and also the constraints under which we have made this project. We have also made some assumptions, which are defined below.

2.1 Users and Stakeholders

Users:

- Bloggers/Writers: Create and manage blog posts.
- Readers: View, interact, and comment on blogs.
- Premium Subscribers: People who have access to premium blogs and some special features for an enhanced blogging experience.

Stakeholders:

- Developers: responsible for building and maintaining the platform.
- Admins: Manage user accounts and oversee content quality (content management team).

2.2 Constraints

- Secure authentication must be implemented.
- The system must be scalable to handle increased traffic.
- Data storage and processing must adhere to GDPR compliance.

2.3 Assumptions and Dependencies

- Users have internet access and modern web browsers.
- Third-party services (e.g., email notifications, analytics) function as expected.

2.4 Proposed Model:

1. **Requirements gathering:**
   Goal: Identify essential features for the blogging platform, such as user profiles, blog posts, comments, likes and sharing options.
   Actions: Conduct user interviews to gather insights on what bloggers and readers expect from the platform. Analyze other blogging platforms for inspiration and understand user preferences. Create user personas to guide feature development based on potential user needs.
   Output: A comprehensive list of prioritized features, along with initial wireframes or prototypes showcasing the core functionalities.

2. **Sprint Planning:**
   Goal: Break down the required features into smaller tasks and prioritize them.
   Actions: Identify the primary components (e.g., post creation, commenting system, user profiles) and divide them into manageable tasks. Prioritize tasks based on their importance and establish the order of dependencies.
   Output: A well-organized sprint backlog containing tasks for the next sprint (typically 2-3 weeks), focusing on high-priority features.

3. **Development and Testing:**
   Goal: Develop features within the sprint timeline, ensuring each feature is functional and integrates seamlessly.
   Actions: Build the platform’s UI, backend, and API endpoints. Test individual features (unit tests) and ensure overall integration (integration testing). Conduct code reviews to ensure quality and consistency.

Output: Fully developed and tested features that are ready for stakeholder review and feedback.

4. **Sprint Review:**
   Goal: Present completed features to stakeholders, collect feedback, and make adjustments as needed.
   Actions: Showcase the developed features (or demo version) to stakeholders, including potential users and team members, to receive feedback on the design, functionality, and usability.
   Output: Gather insights and feedback that will inform future sprints, ensuring the platform aligns with user expectations.

5. **Sprint Retrospective:**
   Goal: Reflect on the sprint's successes and identify areas for improvement.
   Actions: Hold a retrospective meeting with the team to discuss what went well and what could be improved. Identify adjustments to processes, tools, or team collaboration methods to enhance future sprints.
   Output: Actionable improvements to apply in the next sprint, promoting better efficiency and quality.

6. **Repeat Sprint Cycles Until Completion**
   - Continue iterating through sprints, gathering feedback and using retrospectives to refine the platform. This iterative process ensures the blogging platform meets all essential requirements.
   - Once all core features are implemented, enter a stabilization phase to thoroughly test and fine-tune the platform before the full launch.

7. Final Review and Launch Preparation

- Conduct a comprehensive review once all key features are complete.
- Perform final rounds of testing, including security checks, performance optimization, and usability testing.
- Plan for a soft launch or beta testing phase to gather final feedback and ensure the platform is ready for its full release.

Benefits of Agile for Social Media Website Project

Adaptability: Agile allows for continuous adjustments based on user feedback, ensuring the platform evolves in line with real user needs.
Incremental Releases: Features are released incrementally, allowing users to test them as they become available, providing early insights and reducing risks.
Collaboration: Agile promotes regular collaboration with users and stakeholders, ensuring the final platform meets user expectations and enhances engagement.

3. REQUIREMENT ANALYSIS:

3.1 Functional Requirements:

1. User Registration and Authentication:

Allows users to sign up with an email or social login and authenticate using their credentials. The system verifies login details and grants access based on valid authentication, ensuring secure access to the platform.

2. Authentication and role-based access control:

After logging in, the system assigns different roles (e.g., reader, writer, admin) and grants access to specific features based on the user's role, ensuring role-based access control and security.

3. Blog Post Creation, Editing, and Deletion:

Writers can create, edit, or delete their blog posts through an editor, allowing them to manage content. Editing includes updating title, content, and tags, while deletion removes the blog from the platform.

4. Support Media Upload in Blog Posts:

Writers can upload media files (e.g., images, videos) to enhance their blog posts, allowing for multimedia content in blog creation, which is displayed within the post.

5. Categorize Blogs to Sort Them Based on Interest:

Blogs are categorized by topics or tags, allowing readers to filter and view posts that match their interests, making content discovery easier.

6. Save Blog Drafts to Enable Working Over Time:

Writers can save their blog posts as drafts before publishing, allowing them to work on the content over time and publish when ready.

7. Like, share, and comment on blog posts by readers:

Readers can engage with blog posts by liking, sharing, or commenting, fostering interaction with the content and author while contributing to the blog’s visibility.

8. Advanced Search and Filters:

Readers can search for specific blogs using keywords, tags, or categories and apply filters to narrow results, making it easier to find relevant content.

9. Bookmark a Blog Post for Later Reference:

Readers can bookmark blog posts to save them for future reference, creating a personal collection of preferred posts they can revisit at any time.

10. Follow Bloggers:

Readers can follow their favorite bloggers, receiving updates on new posts or content shared by those bloggers, enhancing user engagement and content discovery.

11. Recommend blogs to readers according to their preferences and interests:

The system recommends blogs to users based on their browsing history, interests, and preferences, providing a personalized reading experience.

12. Premium Subscription to View Locked Blogs and Better Blogging Experience:

Users can subscribe to premium plans to access exclusive content, unlock special features (e.g., better blogging tools), and enjoy an enhanced user experience.

13. Supervise blog content to prevent Rumours:

Admins review flagged content for inappropriate or false information, ensuring content adheres to platform guidelines and preventing the spread of rumours.

14. Block and Report Users:

Admins have the authority to block users who violate platform rules and respond to reports filed against users, ensuring a safe and respectful community.

3.2 Non-functional requirements:

- Security: data encryption and secure storage practices.
- Usability: interactive interface with responsive design.
- Scalability: support for increased user base and content growth.
- Backup: data must be backed up daily, stored securely in a separate location, and allowed recovery within 1 hour to minimize downtime during failures.
- Reliability: ensure 99.9% uptime and handle errors gracefully with user-friendly notifications and robust error recovery mechanisms.

4. USE CASES:

4.1 General Use Cases:

- Sign up and login securely.
- View recent and trending blogs.
- Search blogs based on tags and categories.

4.2 Logged In User:

- Write, edit, and delete blogs.
- Update profile.
- Bookmark blogs and follow people.
- Save draft.
- Like, share, and comment on blogs.
- Take a subscription for premium users.

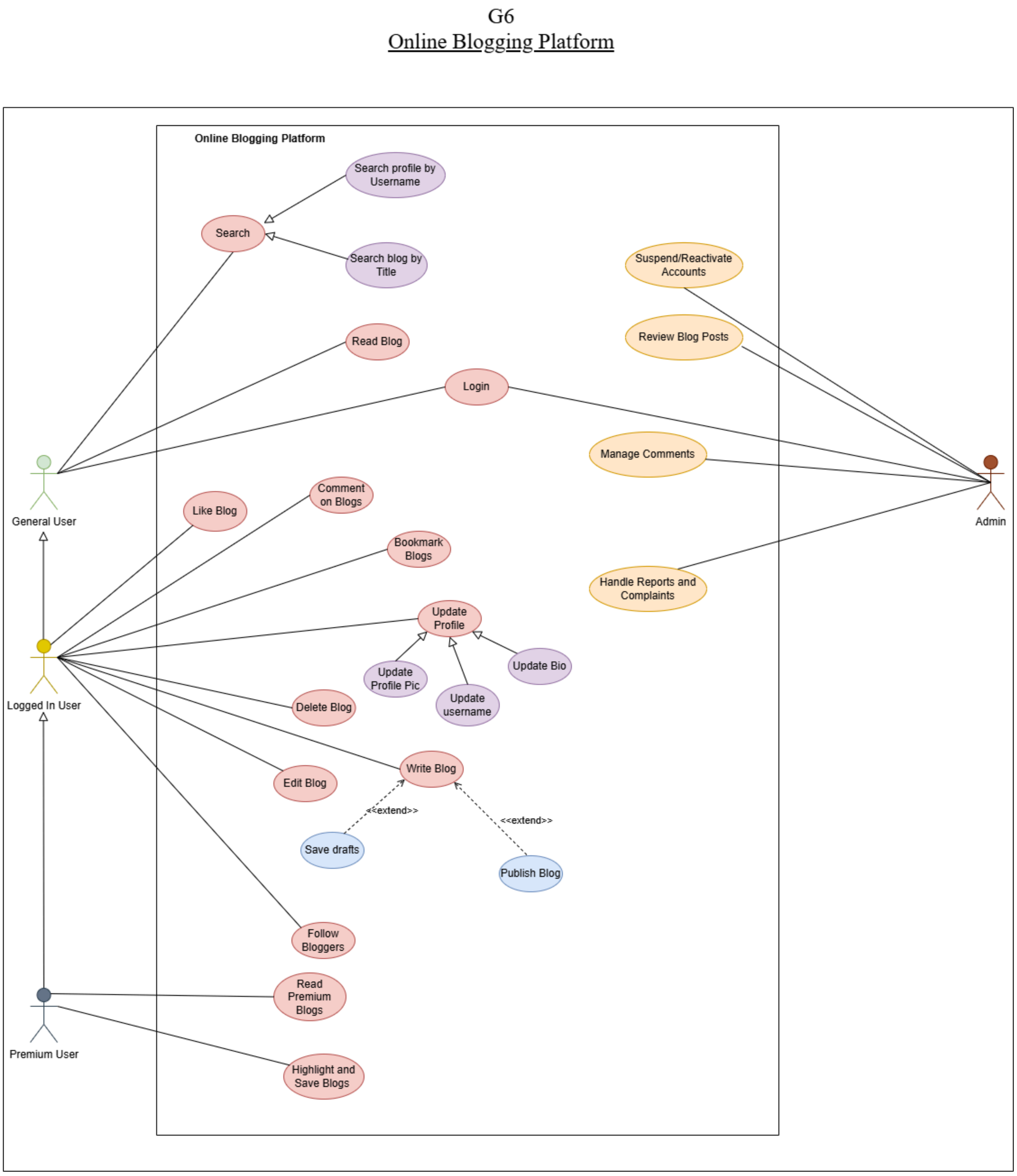
4.3 Premium User:

- Access premium content.
- Highlight while reading a blog and save it.

4.4 Admin:

- Review the blog post and comments, and check for suspicious content.
- Handle reports and block users.
- Suspend/reactivate blog posts.

4.5 Use Case Diagram:



5. USE CASE DESCRIPTION:

1. Use Case: Sign Up Securely

Description:
Allows users to create an account on the platform securely using email/password or social login.

Actors:
General User

Preconditions:

- The user has access to the platform.
- The user does not already have an account.

Main Flow:

1. The user clicks on the "Sign Up" button/link.
2. The system displays the sign-up form.
3. The user enters the required details (e.g., email, password, name, and optional fields like phone number).
4. Optionally, the user clicks on "Sign Up with Google" to use social login.
5. The system validates the provided details:
   5.1. For email/password sign-up, the system checks for the following:
   A valid email format.
   A password meeting security requirements (e.g., minimum length, special characters).
   5.2. For social login, the system verifies the Google account details.
6. The system checks if the email already exists in the database.
7. For new accounts, the system creates the user account and sends an email verification link (if applicable).

8. The user is redirected to the home/dashboard page or shown a success message with further instructions.

Alternative Flow:
4.1. If any required fields are missing, the system shows an error and prompts the user to complete the form.
5.1. If validation fails (e.g., invalid email, weak password), the system displays an error message and requests corrections.
5.2. If Google login fails (e.g., invalid token, canceled permission), the system shows a relevant error message and prompts retry.
6.1. If the email is already registered, the system prompts the user to login instead.

Postconditions:

- A new user account is created.
- The user is either logged in or prompted to verify their email.
- Account creation attempts are logged for auditing and security purposes.

2. Use Case: Login Securely

Description:
Allows users to log in to the platform securely using email/password or social login.

Actors: General User

Preconditions:

- The user has an existing account.
- The platform is operational.

Main Flow:

1. The user clicks on the "Login" button/link.
2. The system displays the login form.
3. The user enters their credentials (email and password) or clicks "Login with Google."
4. The system validates the credentials:
   4.1. For email/password login, the system checks the following:
   Whether the email is registered.
   Whether the password matches the stored hash for the email.
   4.2. For social login, the system verifies the Google account details and retrieves user information.
5. Upon successful authentication, the user is redirected to the home/dashboard page or the last visited page.

Alternative Flow:

4.1. If the credentials are invalid:

4.1.1 The system shows an error message and prompts the user to retry.

4.1.2 After multiple failed attempts, the system triggers additional security measures (e.g., CAPTCHA or account lock).

4.2. If the account does not exist:

4.2.1 The system shows an error and provides a link to the sign-up page.

4.3. If Google login fails (e.g., revoked access, invalid token):

4.3.1 The system shows a relevant error and prompts a retry.

Postconditions:

- The user is logged in successfully.
- The system logs failed login attempts for monitoring and security.

- If authentication fails, appropriate error messages and options are provided to the user.

3. Use Case: View Recent and Trending Blogs

Description:
Allows users to browse recent and popular blogs displayed prominently on the homepage for easy access.

Actors: General User

Preconditions:

1. The platform has blogs stored in the system, each with relevant metadata, such as:
    - Creation date.
    - Popularity metrics (e.g., number of views, likes, shares, or comments).
2. The homepage is accessible to the user.

Main Flow:

1. The user navigates to the homepage via a direct URL.
2. The system fetches recent blogs based on their creation date (sorted in descending order).
3. The system identifies trending blogs using predefined popularity metrics, such as:
        - Blogs with the highest views or likes in a specified time frame.
        - Blogs with significant user engagement (comments/shares).

4. The system displays two distinct sections for "Recent Blogs" and "Trending Blogs" on the homepage.

5. The user scrolls through the blog listings to browse available titles, excerpts, and featured images.

6. The user selects a blog of interest by clicking on "Read More" link.

7. The system loads the blog’s content in a detailed view/page for the user to read.

Alternative Flows:

2.1. If no blogs exist in the system:

2.1.1 The system displays a placeholder message, such as “No blogs available,” or suggests actions like creating a new blog.

2.2. If the user is offline or the blog data fails to load:

2.2.1 The system displays an error message, such as “Unable to load blogs. Please check your internet connection.”

2.2.2 The user is prompted to retry or view cached blog data if available.

3.1. If the blog fails to load due to an error:

3.1.1 The system displays an error message and provides options to navigate back to the homepage or retry loading the blog.

Postconditions:

1. The selected blog opens in a detailed view for the user to read.
2. The blog’s metadata is updated:
   - The view count for the blog is incremented.
   - Any additional engagement metrics (e.g., read time or clicks) are recorded.
3. The user may engage further by liking, commenting on, or sharing the blog, if these features are available.

4. Use Case: Search Blogs Based on Tags and Categories

Description:
Allows users to search for specific blogs using keywords, tags, or categories for a more focused browsing experience.

Actors: General User

Preconditions:

1. Blogs in the system are assigned relevant metadata, including:
    - Tags (e.g., "Technology," "Health").
    - Keywords from blog content or titles for search indexing.
2. The system has a functional and accessible search feature on the platform.

Main Flow:

1. The user accesses the search bar on the homepage or a designated search page.
2. The user types a keyword or selects predefined tags/categories from a dropdown or tag cloud.
3. The user clicks the search icon/button or presses "Enter."
4. The system processes the query by:
        - Searching blog titles, content, tags, and categories for matches.
        - Ranking results based on relevance (e.g., keyword density or recency).
5. The system shows a list of blogs matching the query, including:
        - Blog title.
        - Short excerpt.
        - Tags and category information.
        - Featured image or thumbnail.
6. The user clicks on a blog from the results to view its content in detail.

Alternative Flows:
3.1. If no results match the query:

3.1.1 The system displays a message: "No blogs found for your search."

3.1.2 The user is prompted to refine the search or explore suggested tags or popular categories.

3.2. If the search input is invalid (e.g., empty, special characters only):

3.2.1 The system displays a message: "Please enter a valid keyword or select a tag/category."

4.1. If the search system fails due to a backend issue:

4.1.1 The system shows a generic error message: "Search is currently unavailable. Please try again later."

Postconditions:

1. The user accesses relevant blogs through the search results.
2. Search metrics are recorded for analytics, such as:
   - Frequently searched keywords.
   - Most selected tags or categories.
3. If a blog is opened, its view count and engagement metrics are

- **Personalization:** Results can be personalized based on the user's preferences, past searches, or browsing history.

5. Use Case: Write a Blog

Description:
Allows writers to create and publish new blog posts or save drafts for later editing.

Actor: Writer

Preconditions:

1. The user is logged into the platform.
2. The system provides access to a functional blog editor.

Main Flow:

1. The writer clicks the "Write Blog" button/link on the platform.
2. The system opens the blog editor.
3. The writer inputs the following required fields:
        - Blog title.
        - Blog content/body.
    - The writer may also add optional details such as:
        - Tags or categories for organization.
        - Featured image for display.
4. The writer selects one of the following options:
        - **Publish Blog:** The system performs validation and publishes the blog, making it visible to readers.
        - **Save as Draft:** The system saves the blog as a draft, accessible only to the writer for future editing.
5. Upon successful save or publish, the system displays a confirmation message (e.g., "Blog published successfully" or "Draft saved successfully").

Alternative Flow:

2.1. If required fields (title or content) are missing:

2.1.1 The system shows an error message (e.g., "Title and content are required") and prevents publishing or saving.

3.1. If publishing fails due to a system error:

3.1.1 The system displays a message (e.g., "An error occurred while publishing. Please try again later.") and retains unsaved content.

Postconditions:

1. A new blog is created, either as a published post or draft.
2. The database is updated to reflect the new blog's status and details.

6. Use Case: Edit a Blog

Description:
Allows writers to update content, tags, or other details of their existing blogs.

Actor: Writer

Preconditions:

1. The writer is logged in.
2. The blog to be edited exists in the system and is accessible to the writer.

Main Flow:

1. The writer navigates to their blog list or dashboard.
2. The writer selects the blog they wish to edit and clicks the "Edit" button.
3. The system opens the blog in the editor with the current title, content, tags, and other details pre-filled.
4. The writer updates the blog's details (e.g., correcting typos, adding content, or changing tags).
5. The writer clicks "Save Changes."
6. The system validates the updated fields and saves the changes.
7. The system displays a success message (e.g., "Blog updated successfully").

Alternative Flow:
2.1. If the blog cannot be accessed (e.g., it has been deleted):

2.1.1 The system displays an error message (e.g., "The selected blog no longer exists").

3.1. If required fields are removed during editing:

3.1.1 The system prompts the writer to re-enter the missing information before saving.

Postconditions:

1. The blog is updated with the writer’s changes.
2. The database reflects the updated blog details.

7. Use Case: Delete a Blog

Description:
Allows writers to permanently delete their own blogs.

Actor: Writer

Preconditions:

1. The writer is logged in.
2. The blog to be deleted exists in the system and is accessible to the writer.

Main Flow:

1. The writer navigates to their blog list or dashboard.
2. The writer selects the blog they wish to delete and clicks the "Delete" button.
3. The system prompts the writer to confirm the action with a warning (e.g., "Are you sure you want to delete this blog? This action cannot be undone.")
4. The writer confirms by clicking "Yes" or "Confirm Delete."
5. The system removes the blog from the database.

6. The system displays a success message (e.g., "Blog deleted successfully").

Alternative Flow:
2.1. If the blog to be deleted does not exist:

2.1.1 The system displays an error message (e.g., "The selected blog no longer exists").

2.2. If the writer cancels the deletion process:

2.2.1 The system aborts the deletion and returns the writer to the previous screen.

Postconditions:

1. The blog is permanently deleted from the system.
2. The database is updated to reflect the deletion.
3. The writer’s dashboard no longer lists the deleted blog.

8. Use Case: Update Profile

Actors: Reader, Writer

Preconditions:

1. The user is logged into the platform.
2. The system provides a functional profile page with editable fields for user information, such as:
    - Name.
    - Bio.
    - Profile picture.
3. The system supports input validation for the profile update fields.

Main Flow:

1. The user clicks on their profile icon or selects "Profile" from the navigation menu.
2. The system loads the profile page, displaying the user's current details.
3. The user updates one or more of the available fields:
    - Name: Updates the display name.
    - Bio: Provides a brief introduction or description.
    - Profile Picture: Uploads a new image file for the profile.
4. The user clicks the "Save" or "Update Profile" button.
5. The system performs validation checks, such as:
    - Ensuring the name is not empty.
    - Validating the bio's character limit (e.g., 250 characters max).
    - Checking that the profile picture is in an acceptable format (e.g., PNG, JPEG).
6. Upon successful validation, the system saves the updated details in the database.
7. The profile page reloads to display the updated information.
8. The system displays a success message: "Profile updated successfully."

Alternative Flow:

3.1. If any input is invalid:

3.1.1 The system displays a specific error message, such as:

- "Name cannot be empty."
- "Bio exceeds the character limit of 250 characters."
- "Invalid image format. Please upload a PNG or JPEG file."

3.1.2 The system highlights the problematic fields and allows the user to correct the errors.

3.2. If the system encounters a backend error while saving:

> 3.2.1 The system shows an error message: "An error occurred while updating your profile. Please try again later."

Postconditions:

1. The user's profile details are updated in the database.
2. The updated details are immediately reflected on the user's profile page.
3. The user receives a confirmation notification of the successful update.

8. Use Case: Bookmark Blogs

Description:
Allows users to save blogs for easy access later by adding them to their bookmarks.

Actor: Reader, Writer

Preconditions:

1. The user is logged into the platform.
2. The system supports bookmarking functionality and maintains a list of bookmarked blogs in the user's profile.

Main Flow:

1. The user browses through the list of blogs or views an individual blog.
2. The user clicks the "Bookmark" button/icon displayed alongside the blog.
3. The system checks if the blog is already bookmarked by the user.
4. If not already bookmarked:

        - The system saves the blog to the user’s list of bookmarks in the database.
5. The system displays a success message: "Blog bookmarked successfully."

Alternative Flow:
3.1. If the user attempts to bookmark a blog that is already bookmarked:

3.1.1 The system displays a message: "This blog is already bookmarked."

3.1.2 The bookmark action is not duplicated.

Postconditions:

1. The blog is added to the user’s list of bookmarks.
2. The bookmarks are accessible from the user’s profile or a designated "Bookmarks" section.

9. Use Case: Follow People

Description:
Enables users to follow authors to stay updated on their latest blogs or activities.

Actor: Reader, Writer

Preconditions:

1. The user is logged into the platform.
2. The system supports following functionality and maintains a list of followed authors in the user’s profile.

Main Flow:

1. The user navigates to an author’s profile through a blog post or the "Authors" section.
2. The user clicks the "Follow" button displayed on the author’s profile.
3. The system checks if the author is already followed by the user.
4. If not already followed, the system adds the author to the user’s list of followed authors in the database.
5. The system displays a success message: "You are now following [Author Name].”

Alternative Flow:
3.1. If the user attempts to follow an author they are already following:

3.1.1 The system displays a message: "You are already following this author."

3.1.2 The follow action is not duplicated.

Postconditions:

1. The author is added to the user’s list of followed authors.
2. The user is notified of the author’s future blog posts or activities, depending on the platform’s notification settings.

10. Use Case: Save Draft

Description:
Allows writers to save incomplete blogs as drafts for further editing and publishing later.

Actor: Writer

Preconditions:

1. The writer is logged into the platform.

2. The system provides a blog editor with a "Save as Draft" option.
3. The system is connected to a database capable of storing drafts.

Main Flow:

1. The writer opens the blog editor by clicking "Write Blog" or a similar option in their dashboard.
2. The writer enters the blog content, including:
    - Title.
    - Body text.
    - Optional metadata like tags or categories.
3. The writer clicks the "Save as Draft" button.
4. The system performs the following:
    - Validates the input for saving (e.g., ensuring the title is not empty).
    - Marks the blog as a draft (unpublished).
    - Saves the draft in the database.
5. The system displays a success message: "Draft saved successfully."
6. The draft becomes accessible from the writer's dashboard under a "Drafts" section.

Alternative Flow:
3.1. If the draft cannot be saved due to a system error:

3.1.1 The system displays an error message, such as "An error occurred while saving the draft. Please try again later."

3.1.2 The writer's unsaved content is temporarily preserved in the editor for retrying.

Postconditions:

1. The draft is saved in the database with the following status:
    - Unpublished.
    - Associated with the writer's account.

2. The draft is accessible from the writer's dashboard for future editing or publishing.

11. Use Case: Like a Blog

Description:
Allows users to express appreciation for a blog by liking it.

Actor: Reader

Preconditions:

1. The user is logged into the platform.
2. The blog exists in the system and is visible to the user.
3. The system supports a liking feature and tracks likes per blog.

Main Flow:

1. The user navigates to the blog list or opens a specific blog to read.
2. The user clicks the "Like" button/icon associated with the blog.
3. The system checks if the user has already liked the blog.
4. If not already liked, the system records the like in the database, associating it with the user and the blog.
5. **T**he system updates the like count on the blog in real time.
6. The "Like" button/icon changes state (e.g., highlighted) to indicate the user's action.

Alternative Flow:
3.1. If the user has already liked the blog:

3.1.1 The system prevents duplicate likes and displays a message (e.g., "You have already liked this blog.").

3.2. If a system error occurs while recording the like:

3.2.1 The system displays an error message, e.g., "An error occurred. Please try again later."

Postconditions:

1. The like is recorded in the database and reflected in the blog’s statistics.
2. The user’s action is visually represented on the blog page (e.g., an updated like count).

12. Use Case: Share a Blog

Description:
Allows users to share a blog via a direct link or social media platforms.

Actor: Reader

Preconditions:

1. The user is logged into the platform.
2. The blog exists in the system and is visible to the user.
3. The system provides a "Share" button with options for generating a link or sharing to social media.

Main Flow:

1. The user clicks the "Share" button/icon.
2. The system displays sharing options, such as:
    - ■ Copying a direct link to the blog.
    - ■ Sharing via social media platforms (e.g., Facebook, Twitter).
3. If the user completes a share action, the system logs the share action in the database for analytics purposes.
4. For direct link sharing, the system displays a message (e.g., "Link copied to clipboard.").

5. For social media sharing, the system confirms the successful share, e.g., "Blog shared successfully on [Platform].".

Alternative Flow:
3.1. If the sharing process fails (e.g., network issue):

3.1.1 The system displays an error message (e.g., "Failed to share. Please try again later.")

Postconditions:

1. The share is recorded in the database for analytics purposes.
2. The blog's share count is updated and displayed.

13. Use Case: Comment on a Blog

Description:
Allows users to leave comments under blogs to engage in discussions.

Actor: Reader

Preconditions:

1. The user is logged into the platform.
2. The blog exists in the system and is visible to the user.
3. The system provides a comment box and a mechanism to display comments.

Main Flow:

1. The user navigates to the blog list or opens a specific blog to read.
2. The user types their comment into the comment box.
3. The user clicks the "Submit" button.
4. The system validates the comment to ensure it:
    - Is not empty.
    - Does not contain invalid content (e.g., profanity, spam).

5. The system saves the comment in the database, associating it with the blog and the user.
6. The system updates the comment section to display the new comment.
7. The system displays a message: "Your comment has been posted successfully."

Alternative Flow:
3.1. If the comment is invalid (e.g., empty, or contains profanity):

3.1.1 The system displays an error message (e.g., "Comment cannot be empty" or "Inappropriate content detected.").

3.1.2 The user is prompted to revise the comment.

3.2. If a system error occurs while saving the comment:

3.2.1 The system displays an error message (e.g., "Failed to post a comment." Please try again later."

Postconditions:

1. The comment is saved in the database and associated with the blog and user.
2. The new comment is displayed under the blog, visible to other users.

14. Use Case: Take a Subscription for Premium Users

Description:
Allows users to purchase a subscription plan to access premium features.

Actors: Reader, Writer

Preconditions:

1. The user is logged into the platform.
2. Subscription plans are available for selection.
3. The system integrates a functional and secure payment gateway.
4. The user's account supports subscription status updates.

Main Flow:

1. The user navigates to the "Subscriptions" page.
2. The system displays available plans (e.g., monthly, yearly) with details like pricing and benefits.
3. The user selects a desired subscription plan.
4. The system redirects the user to the payment page.
5. The user enters required payment details (e.g., card number, expiration date, CVV).
6. The user reviews the plan details and payment amount.
7. The user clicks "Confirm Payment."
8. The system processes the payment via the integrated payment gateway.
9. If successful, the system updates the user’s subscription status in the database.
10. The system grants access to premium features (e.g., exclusive content, analytics tools).
11. The system displays a success message: "Your subscription is now active."

Alternative Flow:

2.1. Payment Failure:

2.1.1 If the payment fails (e.g., insufficient funds, invalid card):

- The system displays an error message: "Payment failed. Please check your details and try again."
- The user is prompted to re-enter payment details or select a different payment method.

2.2. User Cancels Transaction:

2.2.1 If the user cancels the payment process, the system aborts the transaction.

2.2.2 The system redirects the user to the subscription selection page.

Postconditions:

1. The user's subscription is activated and recorded in the database with the following details:
    - Plan type (e.g., monthly/yearly).
    - Activation date.
    - Expiry/renewal date.
2. The user can immediately access premium features based on their plan.
3. The user can view their subscription details in the "Account" or "Subscriptions" section.

15. Use Case 1: Access Premium Content

Actors: Premium User

Preconditions:

1. The user is logged in.
2. The user has an active subscription to the premium plan.
3. The system has premium content available (e.g., articles, videos, tools).

Main Flow:

1. The premium user logs in and navigates to the section containing exclusive or premium content, such as a "Premium" tab or a section on the homepage.
2. The system checks if the user has an active subscription to the premium plan. This is done by verifying the user’s account information in the database.
3. If the user has an active premium subscription, the system retrieves the premium content from the database and displays it to the user.
4. The content may include blogs, videos, courses, or special features available only to premium users.
5. The user is able to view, interact with, or download the premium content, depending on the type of content and platform features.

Alternative Flow:
3.1 Inactive Subscription:

3.1.1 If the user does not have an active subscription, the system displays a message: "You need an active premium subscription to access this content."

3.1.2 The user is redirected to the subscription page to renew or purchase a new subscription.

Postconditions:

- If the user has an active subscription, they successfully access the premium content.
- If the user does not have an active subscription, they are prompted to purchase or renew their subscription.

16. Use Case 2: Highlight While Reading a Blog and Save It

Actors: Premium User

Preconditions:

1. The user is logged in as a premium user.
2. The system supports highlighting text in blog posts.

Main Flow:

1. The premium user navigates to a blog post and begins reading.
2. The user highlights a portion of text within the blog post by clicking and dragging over the text they want to save.
3. The system applies a visual highlight (e.g., changing the background color of the selected text).
4. The user clicks the "Save Highlight" button to save the highlighted text for later reference.
5. The system stores the highlighted section in the user's profile for future reference, making it accessible from their dashboard or profile page.

Alternative Flow:
4.1 If the user decides not to save the highlight, they can cancel the action by clicking "Cancel" or by simply unhighlighting the text.

Postconditions:

- The highlighted text is saved under the user's account.
- The user can view and manage saved highlights from their profile at any time.

17. Use Case: Review Blog Post and Comments, and Check for Suspicious Content

Actors: Admin

Preconditions:

1. The admin is logged into the system.
2. The system has flagged content (posts or comments) that is potentially violating platform guidelines.

Main Flow:

1. The admin navigates to the "Flagged Content" section in the admin dashboard, which lists blog posts and comments that have been reported or flagged.
2. The system displays a list of flagged content, including blog posts or comments that have been reported by users or detected by the system.
3. The admin reviews the content for compliance with platform guidelines (e.g., offensive language, inappropriate behavior, spam).
4. The admin evaluates the content and determines the necessary action:
    - **Approve:** The content does not violate any guidelines and can be restored to normal visibility.
    - **Suspend:** The content violates platform guidelines and should be hidden from public view.
    - **Escalate:** The content requires further investigation or legal review, so it is escalated to higher authorities.

5. If approved, the content is restored to public view.
6. If suspended, the content is hidden from public view, and users can no longer interact with it.
7. If escalated, the content is forwarded to the relevant team for further action.

Alternative Flow:
1.1 If no content is flagged, the system notifies the admin with a message: "No suspicious content flagged."

Postconditions:

- The flagged content is either approved, suspended, or escalated.
- The content's status is updated in the system database.
- The author of the content is notified of any action taken.

18. Use Case: Handle Reports and Block Users

Actors: Admin

Preconditions:

1. The admin is logged into the system.
2. The system has user reports available in the admin dashboard, filed against users for violating platform policies.

Main Flow:

1. The admin navigates to the "Reports" section of the admin dashboard, where user reports are displayed.
2. The system lists reports filed against users, which include details about the reported behavior (e.g., harassment, spam, inappropriate content).
3. The admin reviews the report and assesses whether it is valid.

4. The admin evaluates the severity of the reported behavior and decides on one of the following actions:
    - **Block:** The reported user is blocked from the platform, and their account is suspended.
    - **Warn:** The user receives a warning, which is logged in their profile.
    - **No Action:** If the report is found to be false or non-serious, no action is taken.
5. If the user is blocked, the system suspends their account, and they are prevented from accessing the platform.
6. If the user is warned, the system logs the warning in the user's profile for future reference.
7. If no action is taken, the system notifies the admin that the report is dismissed.

Alternative Flow:
3.1 If the report is found to be unsubstantiated or non-serious, the admin may choose to take no action.

Postconditions:

- The reported user is either blocked or warned, depending on the admin's decision.
- The system updates the user's status accordingly.
- The user is notified of any action taken against them (e.g., warning, block).

19. Use Case: Suspend/Reactivate Blog Posts

Actors: Admin

Preconditions:

1. The admin is logged into the system.
2. The blog post exists and is visible to the public.

Main Flow:

1. The admin selects a blog post from the admin dashboard for review.
2. The system displays the blog post's details (e.g., title, author, content) for the admin's review.
3. The admin chooses one of the following actions:
    - **Suspend:** If the blog post violates platform policies or needs temporary removal, the admin suspends the blog post.
    - **Reactivate:** If the blog post was previously suspended or hidden, the admin reactivates it.
4. If the post is suspended, the system hides it from public view.
5. If the post is reactivated, the system restores the blog post to its original state, making it visible again.

Alternative Flow:
5.1 If there is an error in processing (e.g., database issue), the system displays an error message: "Error processing request."

5.2 The admin is prompted to try again later.

Postconditions:

- The blog post is either suspended or reactivated as per the admin's decision.
- The status of the blog post is updated accordingly in the database.
- The author of the blog post is notified of any action taken.

6. USER STORIES:

TITLE	USER STORY	PRIORITY	ACCEPTANCE CRITERIA
User Registration	As a “new user,” I want to “register my account” so that I can “become a verified user of the platform.”	Must Have	- The registration page should prompt the user to enter their name, email address, and password. - After submitting, the user should receive a confirmation email indicating successful registration. - Upon successful registration, the user’s account should be created and securely stored.
Create and publish blogs	As a “blog writer,” I want to “create and publish blog posts” so that I can “share my content with readers.”	Must Have	- The blog writer should have access to a rich text editor to format their content. - Writers should be able to add images to their blogs. - Posts should have options for tags and categories for better organization.
Blog Post Management	As an "administrator," I want to “monitor and manage blog posts” so that the platform can “maintain high-quality content standards.”	Must Have	- The system should flag certain posts for review based on automated filters or user reports. - The blog writer should be notified that his post is under review. - Administrators should be able to approve, reject, or request edits to flagged posts. - If a post is not approved, it should not be visible to other users. - Once reviewed, it should be visible to everyone.
Comment Management	As an “administrator,” I want to “monitor and manage comments” to “ensure the platform promotes a positive and respectful community.”	Must Have	- The system should automatically flag comments for review based on predefined filters or user reports. - Administrators should have the ability to approve, reject, or modify flagged comments. - Comments that are not approved should be removed or hidden from public view. - A record of all actions taken on comments should be maintained. - Administrators should have the option to restore or unflag comments when necessary.
User Report Management	As an “administrator,” I want to “review and address user reports” to	Must Have	- The system should alert administrators whenever a user reports a post, comment, or another

	“ensure that inappropriate content or behavior on the platform is managed quickly and effectively.”		user. - Administrators must have access to view the reported content and understand the reason for the report. - The system should offer options to dismiss the report, take corrective actions, or escalate the issue if necessary. - A record of all actions taken should be maintained. - Users who file reports should receive updates on the outcome.
Bookmarking	As a “logged-in user,” I want to “bookmark blog posts” so that I can “save them for later reading.”	Should Have	- The blog post page should have a "bookmark" button. - The reader's profile should contain a section with all bookmarked posts. - The reader should be able to remove blog posts from their bookmarks. - After bookmarking a post, it should be available in the bookmarked section.
Following Bloggers	As a “logged-in user,” I want to “follow bloggers” so that I can “receive notification when new content is published.”	Should Have	- Users should have a follow button on a blogger’s profile. - Once followed, users should receive notifications of new blog posts from that blogger. - Users should be able to view and manage a list of followed bloggers in their profile.
Subscription Management	As a “logged-in user,” I want to “subscribe to exclusive content” so that “I can access premium blog posts and features that enhance my reading experience.”	Should Have	- The creator profile page should have a premium subscriber button that redirects to payment. - The subscription page should present various plans, clearly outlining the benefits. - Once subscribed, the user should have access to exclusive content. - The user should be able to manage their subscription directly from their profile, including options to upgrade or cancel their plan.
Analytics and	As a “blog writer,” I want	Should Have	- Bloggers should have access to

Performance	to “track the performance of my blog posts” so that “I can understand what resonates with my audience and improve future content.”		an analytics dashboard showing data on views, likes, comments, and shares. - The dashboard should allow filtering by date range and post type.
Search and Filtering	As a “logged-in user,” I want to “be able to apply features and search by categories, keywords, or tags” so that “I can quickly find relevant content that matches my interests.”	Should Have	- Users can enter keywords in a search bar to find blogs; results should display blogs containing the keyword. - A "Clear Filters" button should be available to reset all applied filters and search terms.
Commenting	As a “logged-in user,” I want to “leave comments on blog posts” so that I can “engage with the content writer and other readers.”	Could Have	- Readers should be able to comment on posts directly. - The comment section should allow for threaded replies. - Users should be able to edit or delete their own comments.
Giving tags to blog posts	As a “blog writer,” I want to “give tags to my blog posts” so that readers can “easily discover content based on specific topics or themes.”	Could Have	- The blogging interface should allow assigning one or more categories or tags. - The platform should provide predefined categories and allow admins to create new ones. - Readers should be able to browse and filter posts by category. - Categories should be clearly displayed on the blog post and blogger’s profile.
Text Highlighting	As a “premium user,” I want to “highlight text within blog posts” so that I can “easily refer back to important sections later.”	Could Have	- Premium users should have the ability to highlight text in different colors. - The highlighted sections should be saved and easily accessible. - The system should allow users to edit or remove highlights. - Highlights should be visibly marked when revisiting a post.
Shareable Links	As a “logged-in user,” I want to “generate a shareable link for blog posts” so that I can “share the content in other platforms or messaging apps.”	Could Have	- Each blog post should have an option to generate a unique shareable link. - Users should be able to copy the link easily and share it on any platform.
Drafts Management	As a “blog writer,” I want to “save drafts of my blog posts” so that I can “work on them over time before	Could Have	- The blogging interface should allow saving posts as drafts. - Drafts should be easily accessible from the writer's profile or

	publishing.”		dashboard. - The system should automatically save drafts at regular intervals. - Writers should be able to edit, update, or delete drafts. - The platform should clearly differentiate between drafts and published posts.
Content Discovery	As a “logged-in user,” I want to “find content of my interest” so that I can “read content of my choice directly.”	Could Have	- Users on Signup should have a screen to enter their interests. - Creators uploading content should select the interests their content is based on.
User Profile Management	As a “logged-in user,” I want to “update my profile picture and personal information” so that I can “keep my profile accurate and up-to-date.”	Could Have	-The platform should allow users to upload a new profile picture, crop it, and replace the existing one with instant updates across the platform. -Users should be able to edit personal information, including their name, email, and bio, with changes saved upon submission. -The profile settings page should validate the information and provide confirmation messages once the updates are successfully applied. -The system should ensure that the updated profile picture and personal information are visible on the user’s public profile immediately.

7. SPRINTS:

Sprint 1: User Registration & Basic User Interactions

- User Registration: As a “new user,” I want to “register my account” so that I can “become a verified user of the platform.”
- User Profile Management: As a “logged-in user,” I want to “update my profile picture and personal information” so that I can “keep my profile accurate and up-to-date.”

Sprint 2: Blog Post Creation

- Create and publish blogs: As a “blog writer,” I want to “create and publish blog posts” so that I can “share my content with readers.”
- Drafts Management: As a “blog writer,” I want to “save drafts of my blog posts” so that I can “work on them over time before publishing.”

Sprint 3: Commenting and Adding Tags

- Commenting: As a “logged-in user,” I want to “leave comments on blog posts” so that I can “engage with the content writer and other readers.”
- Giving tags to blog posts: As a “blog writer,” I want to “give tags to my blog posts” so that readers can “easily discover content based on specific topics or themes.”

Sprint 4: Search, Filtering & Content Discovery

- Search and Filtering: As a “logged-in user,” I want to “be able to apply features and search by categories, keywords, or tags” so that “I can quickly find relevant content that matches my interests.”
- Content Discovery**:** As a “logged-in user,” I want to “find content of my interest” so that I can “read content of my choice directly.”

Sprint 5: User Interaction Features

- Following Bloggers: As a “logged-in user,” I want to “follow bloggers” so that I can “receive notification when new content is published.

- Bookmarking: As a “logged-in user,” I want to “bookmark blog posts” so that I can “save them for later reading.
- Shareable Links: As a “logged-in user,” I want to “generate a shareable link for blog posts” so that I can “share the content in other platforms or messaging apps.”

Sprint 6: Subscription & Analytics

- Subscription Management: As a “logged-in user,” I want to “subscribe to exclusive content” so that “I can access premium blog posts and features that enhance my reading experience.”
- Analytics and Performance: As a “blog writer,” I want to “track the performance of my blog posts” so that “I can understand what resonates with my audience and improve future content.”

Sprint 7: Enhancements & Admin Controls

- Text Highlighting: As a “premium user,” I want to “highlight text within blog posts” so that I can “easily refer back to important sections later.”
- User Report Management: As an “administrator,” I want to “review and address user reports” to “ensure that inappropriate content or behavior on the platform is managed quickly and effectively.”
- Blog Post Management: As an "administrator," I want to “monitor and manage blog posts” so that the platform can “maintain high-quality content standards.”
- Comment Management: As an “administrator,” I want to “monitor and manage comments” to “ensure the platform promotes a positive and respectful community.

8. FUTURE WORK:

The remaining work for the project includes several important features aimed at improving both user experience and platform management. Subscription Management will allow logged-in users to subscribe to exclusive content, granting access to premium blog posts and features. Additionally, Analytics and Performance features will enable blog writers to track the performance of their posts, providing insights into audience engagement to refine future content.

Another key feature to be added is Text Highlighting, which will allow premium users to highlight important sections within blog posts for easy reference later. For platform administrators, there will be enhancements in User Report Management to address inappropriate content or behavior effectively. Administrators will also gain the ability to manage blog posts and monitor comments, ensuring high-quality content standards and fostering a positive, respectful community environment. These updates will significantly enhance the platform's functionality and user experience.