
Team V

**Mini Twitter
Software Requirements Specification**

Version: 2.0

**Ramim Tarafdar, Oluwanifesimi Kukoyi,
Mahmud Hasan, Taha Nayyar**

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

Revision History

Date	Version	Description	Author
10/27/2023	1.0	Initial specs doc created	Mahmud Hasan, Ramim Tarafdar, Oluwanifesimi Kukoyi, Taha Nayyar
11/7/2023	2.0	Fixed issues from previous report. Included use cases and use case diagram	Mahmud Hasan, Ramim Tarafdar, Oluwanifesimi Kukoyi, Taha Nayyar
11/14/2023	2.0	Created sections for new report requirements and began updating information	Mahmud Hasan, Ramim Tarafdar, Oluwanifesimi Kukoyi, Taha Nayyar
11/19/2023	2.0	Updated use case reports, er diagram	Mahmud Hasan, Ramim Tarafdar, Oluwanifesimi Kukoyi, Taha Nayyar
11/20/2023	2.0	Created method pseudo codes, collaboration diagram, system screens	Mahmud Hasan, Ramim Tarafdar, Oluwanifesimi Kukoyi, Taha Nayyar
11/21/2023	2.0	Created sequence diagrams and petri nets	Mahmud Hasan, Ramim Tarafdar, Oluwanifesimi Kukoyi, Taha Nayyar
11/22/2023	2.0	Finished all remaining edits	Mahmud Hasan, Ramim Tarafdar, Oluwanifesimi Kukoyi, Taha Nayyar

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

Table of Contents

1.	<u>Introduction</u>	5
1.1.	Purpose	5
1.2.	Scope	5
1.3.	Definitions, Acronyms, & Abbreviations	6
1.4.	References	7
1.5.	Overview	8
2.	<u>Overall Description</u>	8
2.1.	Use-Case Model	9
2.1.1.	User Types	9
2.1.2.	Use Cases	9
2.2.	Assumptions and Dependencies	11
3.	<u>Specific Requirements</u>	12
3.1.	Use-Case Reports	12
3.2.	Supplementary Information	20
3.2.1.	Functional Requirements	20
3.2.2.	Non-functional Requirements	21
4.	<u>Design Diagrams</u>	22
4.1.	Collaboration Class Diagram	22
4.2.	Use-Case Diagram	23
4.3.	ER Diagram	24
4.4.	Sequence Class Diagrams	25
4.4.1.	Login Sequence	25
4.4.2.	Create Profile Sequence	26
4.4.3.	Apply to be OU/CU Sequence	27
4.5.	Petri-Nets	28
4.5.1.	Post Message.	28
4.5.2.	SU Accept/Deny Application	28
4.5.3.	SU Manage Taboo List	29
5.	<u>Detailed Design</u>	30
5.1.	User Login	30
5.2.	User Sign Up	30
5.3.	User Log Out	31
5.4.	Get User Followers	31

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

5.5.	Add new Followers/Followings	32
5.6.	Get User Followings	32
5.7.	Get All Posts.	33
5.8.	Get User Posts.	33
5.9.	Create New Post.	34
5.10.	Get Post Details.	34
5.11.	Comment on Post.	35
5.12.	Like a Post.	35
5.13.	Dislike a Post.	37
5.14.	Delete a Post.	37
5.15.	Find a User.	38
5.16.	Handling Taboo.	38
5.17.	Creating Ad.	38
5.18.	Banning User.	38
5.19.	Warn User.	39
5.20.	Get Ads.	39
5.21.	Delete Ads.	39
5.22.	Add Trendy	39
5.23.	Remove Trendy.	39
5.24.	Get Top 3	40
5.25.	Get User Details.	40
5.26.	Create Profile	41
6.	<u>Supporting Information</u>	42
6.1.	System Screens/GUI Screens	42
6.2.	Group Meetings Memos	48
6.3.	Concerns	48
6.4.	Git Repo	48

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

Software Requirements Specification

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) is to outline the requirements for the development of an online social networking application, similar to Twitter, referred to as “Mini Twitter.” This document will serve as a guide for the development team of Team V, the Professor, and fellow students.

The SRS will outline the core features and capabilities of Mini Twitter, describing all external behaviors of the application, including system responses to user interactions, non-functional requirements, design constraints, and other factors required for a successful application.

1.2 Scope

The scope of this project applies to the development of a user-friendly web application called “Mini Twitter” that will replicate the functionalities of the existing Twitter/X application on a smaller scaler. Users will be able to create, share, and interact with content and other users in various formats through posts (aka tweets). These tweets can contain text, videos, photos, or links. Users will be able to authenticate themselves and interact with posts by commenting, posting, reacting, liking and following other users. The backend of the application will utilize a relational database to handle all the users, posts, comments that are created, and followers. The frontend will be developed using React.

This project is associated with the microblogging use-case model. Microblogging enables users to weigh in on trending topics or communicate with friends or family, and organizations to reach their prospective audiences. Our target audience includes anyone

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

interested in growing their network and communicating quickly in various formats such as audio, video, images, texts.

The scope of the Mini Twitter project encompasses the following aspects:

- Development of a user-friendly web application that facilitates user authentication and interaction with posts through actions such as posting, commenting, reacting, and following other users.
- Implementation of a robust relational database to manage user profiles, posts, and comments efficiently.
- Construction of the application's frontend using the React framework.
- Inclusion of essential features, such as the ability to post and view various content types, including text, videos, photos, and links.

1.3 Definitions, Acronyms, & Abbreviations

- **User** - An individual who creates an account to participate in the platform's features and interact with other users.
- **Tweets**: Posts created by users that are able to be viewed, commented, and liked by other users.
- **DB**: Database
- **HTML**: HyperText Markup Language
- **Javascript**: JavaScript is a scripting language that allows you to create dynamically updated information, manage multimedia, animate visuals, and everything else web related.
- **React**: React is an open source frontend library used to create component based interfaces
- **SQL**: Stands for Structured Query Language. SQL is a programming language used to manage databases and perform various data manipulations.
- **PSQL**: Stands for PostgreSQL. PSQL is a command-line interface (CLI) is an open-source, dependable relational database management system (RDBMS). PostgreSQL is well-known for its complex functionality, versatility, and standard adherence.

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

- **ER:** Entity relationship model
- **API:** Stands for Application Programming Interface. An API is a collection of guidelines and procedures that permits communication between different software programs. It outlines the procedures and file types that apps may utilize to communicate and request data. Through the use of APIs, disparate software systems may be integrated and made to cooperate and exchange data.
- **Super User (SU):** An individual with the authority to issue warnings, add, or delete any users and/or messages within the system.
- **Corporate User (CU):** An entity granted the capability to post advertisements and job openings on the platform.
- **Trendy User (TU):** A subset of ordinary users is defined by meeting the following criteria: being subscribed by more than 10 users, having received tips or likes minus dislikes exceeding \$100, and having authored at least 2 messages classified as trendy.
- **Surfer User (S):** A user restricted to viewing and searching messages. They can report or complain to the super-user regarding any misinformation encountered on the platform.
- **Ordinary User (OU):** In addition to the features available to a surfer, an ordinary user can post, delete, comment, tip, like/dislike messages, file complaints, follow messages, and subscribe to other users.

1.4 References

Reference documents and sources used in the development of Mini Twitter include:

- Twitter Website
- Prisma Documentation
- PSQL Documentation
- React Documentation

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

- Node Documentation
- Express Documentation

1.5 Overview

This document is organized into four main sections. The first section, “Introduction,” provides an overview of the purpose, scope, definitions, acronyms, abbreviations, and references related to Mini Twitter. The second section, “Overall Description,” offers a detailed description of Mini Twitter, including its use-case model and assumptions and dependencies. The third section, “Specific Requirements,” outlines the use-case reports and supplementary requirements for Mini Twitter. The final section, “Supporting Information,” provides additional information to support the understanding of the system.

2. Overall Description

Mini Twitter is an online social networking application designed to replicate the functionalities of prominent platforms like Twitter. Users can create and share content in multiple formats, including text, videos, photos, and links. The application emphasizes user-friendliness and responsiveness to offer a seamless user experience. Mini Twitter will provide users with a platform where they can empower themselves to connect, self-express, stay informed, and have a sense of belonging.

The product is an online social networking application that allows users to post (aka tweet), view, comment on, react to, and repost other users’ tweets. These tweets can contain text, videos, photos, or links. The application will be user-friendly and will utilize a relational database for handling all the users, posts, and comments that are created. The frontend of the application will be built using React.

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

2.1 Use-Case Model Survey

2.1.1 User Types

- **SU (Super user)** : Someone who can warn/add/delete any users and/or messages
- **CU (Corporate User)** : Someone who can post ads and job openings
- **TU (Trendy User)** : the subset of ordinary users who were (all must be true) :
 - a) subscribed by >10 users
 - b) received >\$100 tips or #likes - #dislikes>10
 - c) author of at least 2 trendy messages
- **S (Surfer User)**: who can only view/search the messages and report/complain to the super - user about the misinformation
- **OU (Ordinary User)**: besides having all features for a surfer, who can post/delete, comment, tip, like/dislike , complain, follow messages, and subscribe other users

2.1.2 Use Cases

Functions	Description
Login/Register/Reset Password	All SU/TU/OU/CU can log and register to the website as well as reset password if they forgot it.
Apply to be OU/CU	Surfers can apply to create an account and choose an account type.
Give Warning/Demote User	SU receives reports and issues warnings or demotions to users.

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

Report Message/Profile of Users	S and OU can file a complaint against other users' messages or profiles.
Taboo List	SU sets up and manages all the taboo words.
Deny Users/Accept Users	SU can deny or accept users trying to make an account.
Illegal Posting	OU/TU will be fined for posting ads/job posts that violate rules.
Comment on Other User Profile	OU/TU/CU can all comment on other people's profiles.
Search for Message	OU/TU/CU/S can all search for messages based on various criteria.
Deposit Money	SU adds a certain amount of money to a user's account from bank
Posting Ads/Job Link	CU has to pay SU and the website for posting ads/job links.
Providing Temporary Password	SU provides a temporary password to all users they accept.
Construct Own Profile	OU/TU/CU can all customize their own profiles.
Exceed Word Count	OU/TU have to pay if they exceed 20 words in their posts.
Payment for clicks on Ad/Job Postings	Interaction between CU and SU, ensuring fair and transparent transactions based on accurate click tracking.
Trendy Post	TU posts are considered as trending posts. OU can become a TU and have trending posts as well.
Activity History Recommendations	The system suggests users to follow trendy users

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

2.2 Assumptions and Dependencies

The successful deployment of Mini Twitter relies on several assumptions and dependencies:

- **User Interface and User Experience:** The success of Mini Twitter is tied to a user-friendly and intuitive interface. This necessitates well-designed UI/UX resources to enhance the overall user experience. Users are expected to have a basic understanding of web applications and social media platforms to navigate and interact effectively.
- **Content Promotion:** The identification and highlighting of "trendy posts" hinge on Mini Twitter's algorithms, analyzing message reads, likes, and dislikes. The effectiveness of these algorithms is crucial for promoting relevant and engaging content.
- **Monetary Transactions:** Mini Twitter relies on a secure payment gateway for financial transactions, covering tasks like fund deposits, user billing, and payments from Corporate Users (CU). Any glitches with this payment gateway may impact the platform's financial operations.
- **Advertisements and Job Postings:** Mini Twitter's revenue model depends on the active participation of advertisers and employers for posting ads and job opportunities. Corporate Users (CU) play a vital role in this aspect.
- **Privacy and Data Security:** Mini Twitter places a high priority on ensuring the privacy and security of user data. Adherence to privacy regulations and robust data protection measures is non-negotiable. Users should also behave respectfully and treat everyone fairly.
- **Tech Proficiency:** Users are assumed to have basic computer skills, familiarity with internet browsing, and an understanding of social media functions, similar to popular platforms like Youtube, Twitter, Facebook/Meta, Discord, etc. Mini Twitter is designed to work seamlessly across various browsers, ensuring accessibility for users on Windows, Linux, and macOS. The application will be

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

hosted on a reliable web server capable of handling expected traffic and providing uptime.

3. Specific Requirements

These requirements are detailed enough to enable designers to design a system that satisfies these requirements and testers to test that the system satisfies these requirements.

3.1 Use-Case Reports

3.1.1 Use Case: Login

Actor: OU/TU/SU/CU

Description: This use case outlines the process when an ordinary (OU), trendy (TU), super (SU), or corporate user (CU) logs into the website.

Preconditions: The user has a valid account and credentials.

Postconditions: The user is successfully logged into the system.

Normal Flow: 1. The user navigates to the login page. 2. The user enters their username and password. 3. The user clicks on the "Login" button. 4. The system verifies the credentials and logs the user in.

3.1.2 Use Case: Sign Up

Actor: All users mentioned in the login (OU/TU/SU/CU) and Surfer

Description: This use case describes the process when users sign up for an account.

Surfers can also sign up and choose an account type.

Preconditions: The user does not have an existing account.

Postconditions: The user's account is successfully created, and the user is logged in.

Normal Flow: 1. The user accesses the sign-up page. 2. The user provides the necessary information, including choosing an account type. 3. The user clicks on the "Sign Up" button. 4. The system validates the information and creates the user account.

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

3.1.3 Use Case: Post/Delete Message

Actor: TU/OU/CU

Description: This use case outlines the process when TU/OU/CU posts or deletes a message.

Preconditions: The user is authenticated and logged into the system.

Postconditions: For posting: The message is successfully posted. For deleting: The message is successfully removed from the user's profile and followers' timelines.

Normal Flow - Post: 1. The user navigates to the message composition area. 2. The user enters the message content (≤ 20 words for free). 3. The user clicks on the "Post" button. 4. The system validates the message content. 5. If valid, the system posts the message and displays a confirmation message to the user.

Normal Flow - Delete: 1. The user navigates to the message they want to delete. 2. The user selects the delete option. 3. The system prompts the user for confirmation. 4. If confirmed, the system removes the message from the user's profile and followers' timelines.

3.1.4 Use Case: Like/Dislike Message

Actor: TU/OU/CU

Description: This use case describes the process when TU/OU/CU reacts to a message by liking or disliking it.

Preconditions: The user is authenticated, and the message exists.

Postconditions: The user's reaction is registered, updating the like count on the message.

Normal Flow: 1. The user clicks on the "Like" or "Dislike" button on a message. 2. The system updates the like count on the message.

3.1.5 Use Case: Reply to Message

Actor: TU/OU/CU

Description: This use case details the process when TU/OU/CU replies to a message.

Preconditions: The user is authenticated, and the message exists.

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

Postconditions: The user's reply is successfully added to the conversation thread.

Normal Flow: 1. The user navigates to the message they want to reply to. 2. The user enters their response in the designated area. 3. The user clicks on the "Reply" button. 4. The system validates the reply content. 5. If valid, the system adds the reply to the conversation thread.

3.1.6 Use Case: Give Warning/Demote User

Actor: SU

Description: This use case outlines the process when the super-user receives reports and issues warnings or demotions to users.

Preconditions: User reports have been received.

Postconditions: The user is either warned or demoted based on the super-user's decision.

Normal Flow: 1. The super-user reviews user reports. 2. For each report: - If warranted, the super-user issues a warning or demotion.

3.1.7 Use Case: Report Message/Profile of Users

Actor: S and OU

Description: This use case describes the process when S and OU file a complaint against other users' messages or profiles.

Preconditions: 1. The user is logged into the account. 2. The user encounters objectionable content. 3. The reporting functionality is available and operating.

Postconditions: The reported message or profile is reviewed for further action.

Normal Flow: 1. The user accesses the reporting feature. 2. The user provides details about the objectionable message or profile. 3. The report is submitted for review.

3.1.8 Use Case: Taboo List

Actor: SU

Description: This use case outlines the process when the super-user sets up all the taboo words.

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

Preconditions: None

Postconditions: The taboo list is successfully updated.

Normal Flow: 1. The super-user accesses the system settings. 2. The super-user adds or modifies words in the taboo list. 3. The system updates the taboo list.

3.1.9 Use Case: Deny Users/Accept Users

Actor: SU

Description: This use case outlines the process when the super-user denies or accepts users trying to make an account.

Preconditions: User applications are pending.

Postconditions: The user's application is either accepted or denied.

Normal Flow: 1. The super-user reviews pending user applications. 2. For each application: - If accepted, a temporary password is sent to the user. - The user must change the password upon first login. - An amount of money is deposited to the system. - If denied, a justification is provided.

3.1.10 Use Case: Comment on Other User Profile

Actor: OU/TU/CU

Description: This use case outlines the process when OU/TU/CU can all comment on other people's profiles.

Preconditions: The user is authenticated and views another user's profile.

Postconditions: The comment is successfully added to the user's profile.

Normal Flow: 1. The user navigates to another user's profile. 2. The user enters the comment in the designated area. 3. The user clicks on the "Comment" button. 4. The system validates the comment content. 5. If valid, the system adds the comment to the user's profile.

3.1.11 Use Case: Search for Message

Actor: OU/TU/CU/S

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

Description: This use case outlines the process when OU/TU/CU/S can all search for messages based on various criteria.

Preconditions: The user wants to search for specific messages.

Postconditions: The search results are displayed based on the user's criteria.

Normal Flow: 1. The user accesses the search feature. 2. The user provides search criteria. 3. The system displays the search results.

3.1.12 Use Case: Deposit Money

Actor: SU

Description: This use case outlines the process when SU adds a certain amount of money to a user's account when created.

Preconditions: The user's account is created.

Postconditions: The specified amount of money is successfully added to the user's account.

Normal Flow: 1. The super-user accesses the user management section. 2. The super-user selects a user to deposit money to. 3. The super-user specifies the amount and confirms the deposit.

3.1.13 Use Case: Posting Ads/Job Link

Actor: CU

Description: This use case outlines the process when CU has to pay SU and the website for posting ads/job links.

Preconditions: The CU wants to post ads or job links.

Postconditions: The CU is successfully billed for posting ads or job links.

Normal Flow: 1. The CU accesses the posting feature. 2. The CU provides details about the ad or job link. 3. The system calculates the posting fee and bills the CU.

3.1.14 Use Case: Providing Temporary Password

Actor: SU

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

Description: This use case outlines the process when SU provides a temporary password to all users they accept.

Preconditions: The user's application is accepted.

Postconditions: The user receives a temporary password for their initial login.

Normal Flow: 1. The super-user reviews accepted user applications. 2. For each accepted application, a temporary password is generated and sent to the user.

3.1.15 Use Case: Construct Own Profile

Actor: TU/OU/CU

Description: This use case describes the process when TU/OU/CU can all customize their own profiles.

Preconditions: The user is logged into the system.

Postconditions: The user profile is successfully created or modified.

Normal Flow: 1. The user navigates to the profile editing section. 2. The user adds or modifies profile information. 3. The system displays the updated user profile.

3.1.16 Use Case: Trendy Post

Actor: TU/OU

Description: This use case outlines the process when TU posts are considered as trending posts. OU can become a TU and have trending posts as well.

Preconditions: TU posts meet specific criteria for being considered trendy.

Postconditions: TU posts are featured as trending, and OU can achieve trendy status.

Normal Flow: 1. The system analyzes TU posts for specific criteria. 2. If met, the TU post is marked as trending. 3. OU can achieve trendy status if they have trendy post.

3.1.17 Use Case: Visit System (Surfer)

Actor: Surfer

Description: This use case describes the process when a surfer visits the system.

Preconditions: None

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

Postconditions: The top 3 most liked messages and the top 3 trendy users are featured on the top page.

Normal Flow: 1. The surfer accesses the system. 2. The system displays the top 3 most liked messages. 3. The system displays the top 3 trendy users.

3.1.18 Use Case: Apply for User Type

Actor: Surfer

Description: This use case describes the process when a surfer applies to be an ordinary (OU) or corporate user (CU) with their chosen ID.

Preconditions: None

Postconditions: The user's application is submitted for processing.

Normal Flow: 1. The surfer indicates the desire to apply for OU or CU status. 2. The surfer provides a chosen ID. 3. The application is submitted for processing.

3.1.19 Use Case: Process User Application (Super-User)

Actor: Super-User

Description: This use case describes the process when the super-user processes user applications.

Preconditions: User applications are pending.

Postconditions: The user's application is either accepted or denied.

Normal Flow: 1. The super-user reviews pending user applications. 2. For each application: - If accepted, a temporary password is sent to the user. - The user must change the password upon first login. - An amount of money is deposited to the system. - If denied, a justification is provided.

3.1.20 Use Case: Construct User Profile

Actor: TU/OU/CU

Description: This use case describes the process when TU/OU/CU constructs their own profiles.

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

Preconditions: The user is logged into the system.

Postconditions: The user profile is successfully created.

Normal Flow: 1. The user navigates to the profile editing section. 2. The user adds or modifies profile information. 3. The system displays the updated user profile.

3.1.21 Use Case: Activity History Recommendations

Actor: Surfer

Description: This use case describes how the system suggests users to follow based on a surfer's activity history. It ensures that surfers are exposed to content that aligns with their interests.

Preconditions: 1. The surfer is logged into the platform. 2. The recommendation system is operational and functioning properly. 3. The surfer has a sufficient history of reading, liking, and following other users.

Postconditions: 1. The surfer has successfully discovered and followed recommended users based on their activity history. 2. The surfer's feed is now filled with relevant content that aligns with their interests.

Normal Flow: 1. The surfer logs into the platform, granting the system access to their profile and activity history. 2. The recommendation system analyzes top three trending users and posts and displays them.

3.1.22 Use Case: Payment for clicks on Ad/Job Postings

Actors: CU, SU

Description: The process of paying for clicks on ads or job openings post by CU. The SU is responsible for tracking and generating invoices for the CU.

Preconditions: 1. The CU has successfully posted an ad or job opening on the platform. 2. The SU has access to the system to monitor and track clicks on ads and job openings. 3. The payment system is functioning correctly and can process invoices and payments.

Postconditions: 1. The CU has successfully paid the SU for clicks on their ad or job posting. 2. Their payment System has accurately recorded and invoiced for clicks on ads

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

or job openings. 3. The SU has received payment from the CU for clicks on their ads or job openings.

Normal Flow: 1. CU posts and ad/job opening. 2. SU tracks clicks on the ad/job opening. 3. Payment system calculates clicks and generates an invoice. 4. CU pays SU. 5. SU confirms payment.

3.2 Supplementary Requirements

The Mini Twitter project requires a stable internet connection, cloud server space, and a hosting domain. Users must access the website from a personal computer using a React-supported browser with up-to-date JavaScript. Browser compatibility extends to Chrome, Firefox, Safari, and Edge, ensuring a consistent experience. Mobile responsiveness is essential for users on various devices. The application should adhere to accessibility standards, provide user documentation, and implement data backup for recovery. Scalability considerations involve accommodating potential growth in users and interactions. Continuous monitoring, user authentication, and compliance with legal standards are crucial. Additionally, the platform needs community guidelines, moderation features, and adherence to industry-specific regulations for a secure and user-friendly experience.

3.2.1 Functional Requirements

Functional requirements describe the actions and capabilities of Mini Twitter. The system should allow users to:

1. Authenticate themselves securely using established authentication mechanisms.
2. Create and post tweets with a wide range of content types, including text, videos, photos, and links.
3. View tweets created by other users and navigate through content seamlessly.
4. Comment on tweets to engage in discussions and provide feedback.
5. React to tweets with predefined reactions, expressing their sentiments.

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

6. Repost tweets to share content from other users with proper attribution.
7. Follow other users to stay updated on their posts and activities.

3.2.2 Non-functional Requirements

Non-functional requirements focus on the quality attributes and constraints of Mini Twitter, encompassing the following aspects:

- User-Friendliness: The application should feature an intuitive and user-friendly interface, ensuring that users can navigate and use the platform with ease.
- Responsiveness: Mini Twitter should provide quick response times, ensuring that user interactions are smooth and efficient.
- Privacy and Security: The system must prioritize the privacy and security of user data and interactions, implementing measures to protect user information and ensure secure transactions.

Performance Requirements:

- The system should be able to support multiple concurrent users without significant degradation in response time.
- The system should load user interfaces within 2 seconds.
- The system should be able to handle a large number of tweets and interactions efficiently.

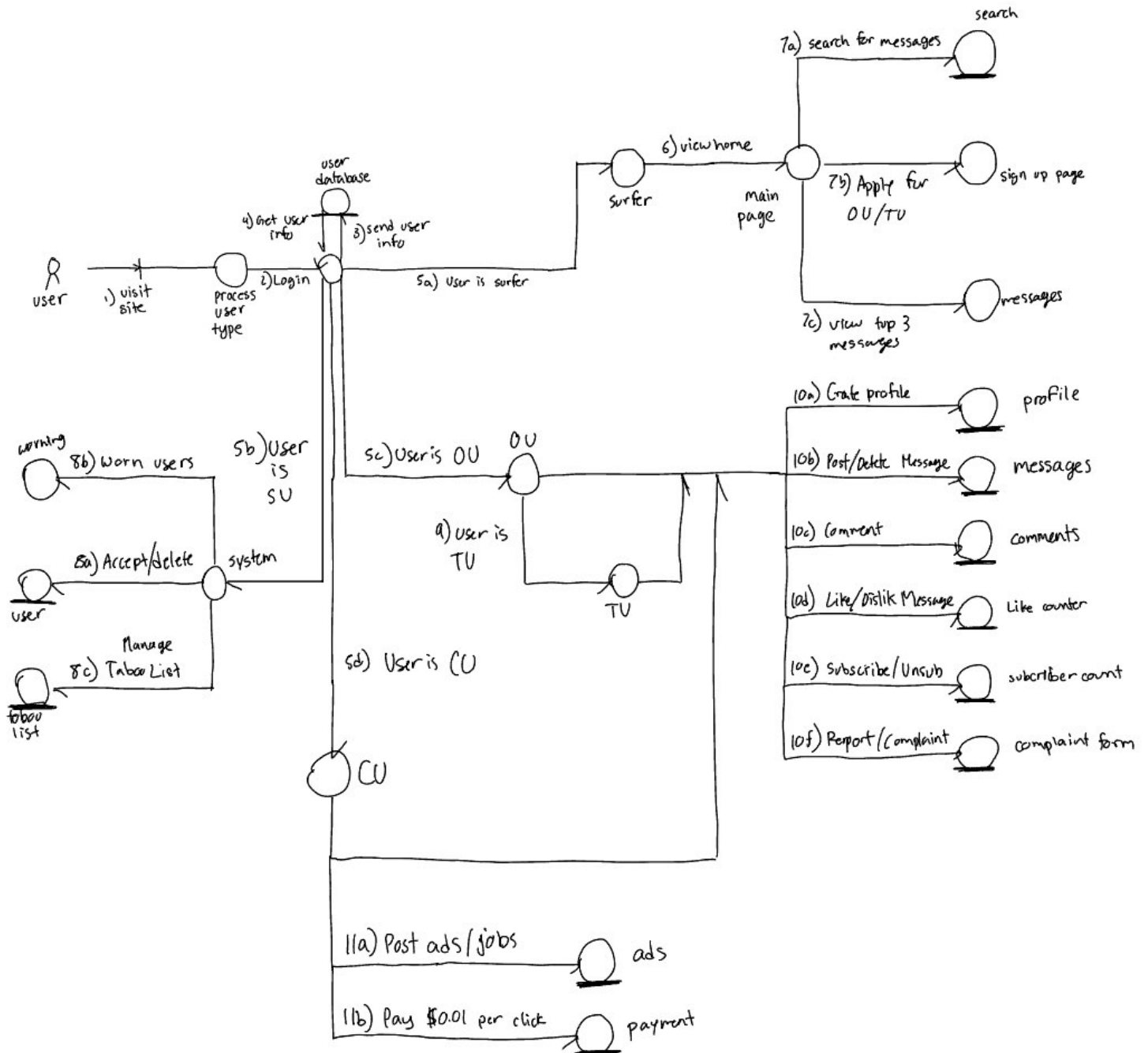
Security Requirements:

- User data should be stored securely, with sensitive data such as passwords being hashed.
- The system should provide role-based access control, ensuring that users can only access data and functions that they are authorized for.

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

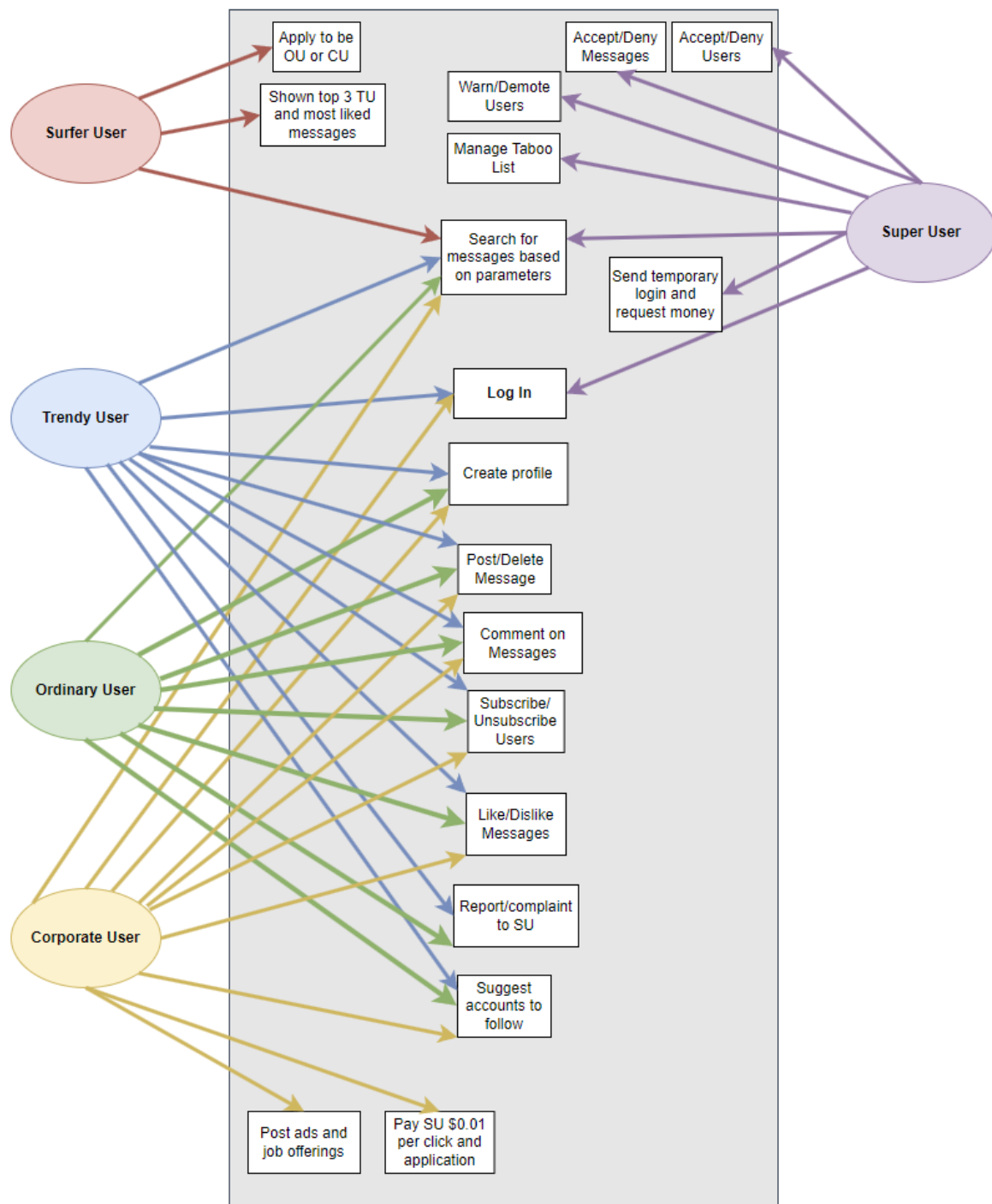
4. Design Diagrams

4.1 Collaboration Class Diagram



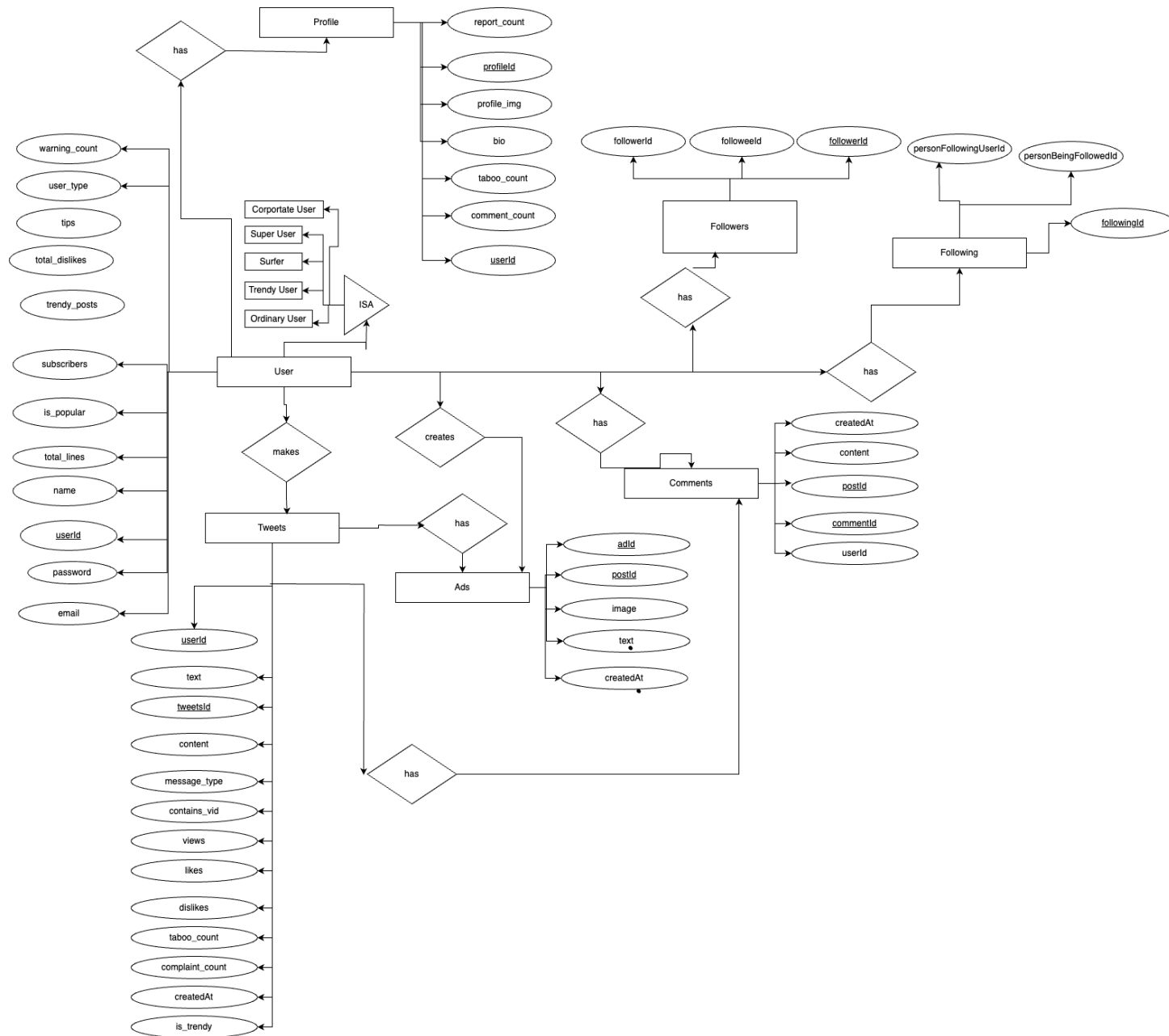
Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

4.2 Use-Case Diagram



Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

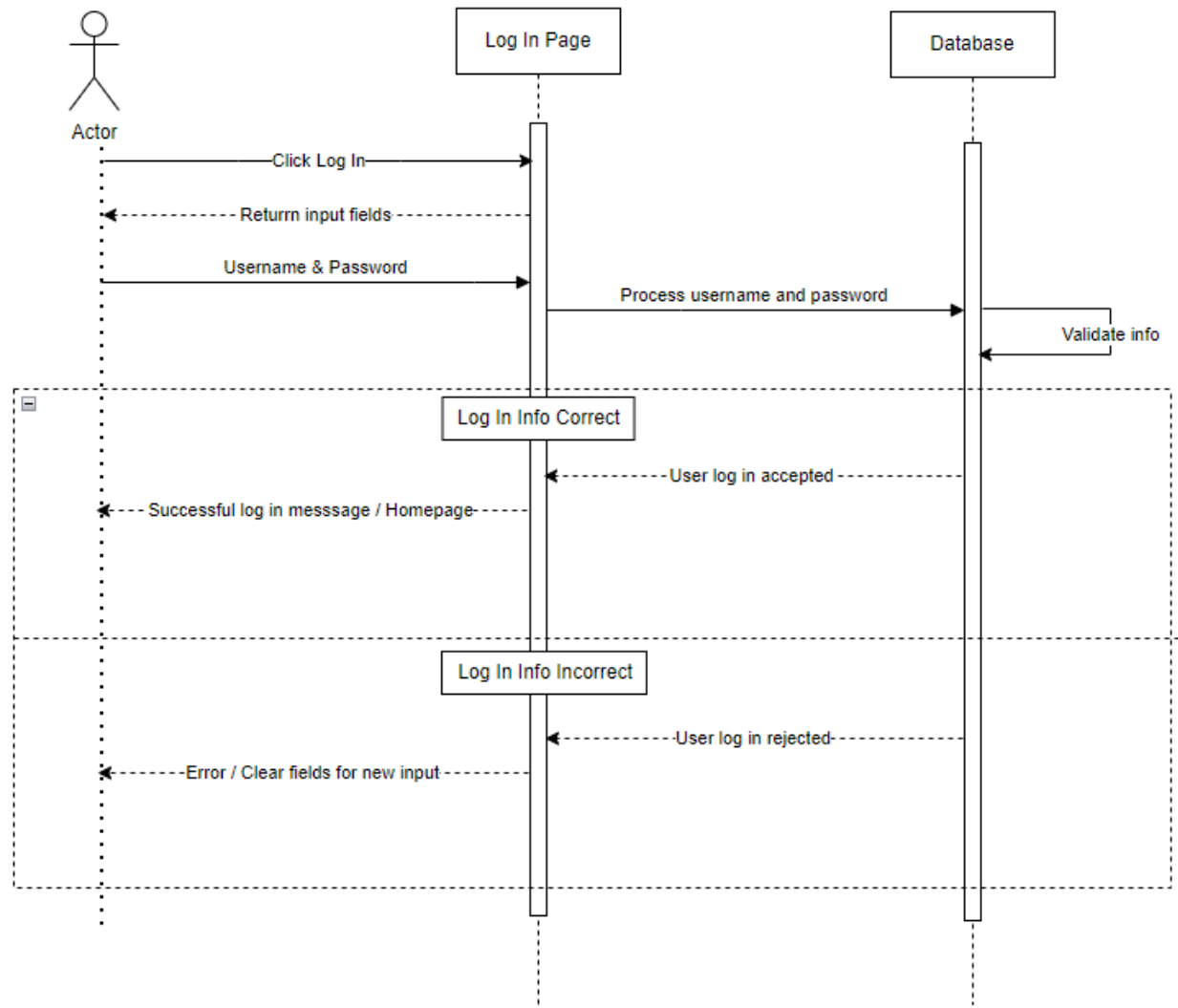
4.3 ER Diagram



Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

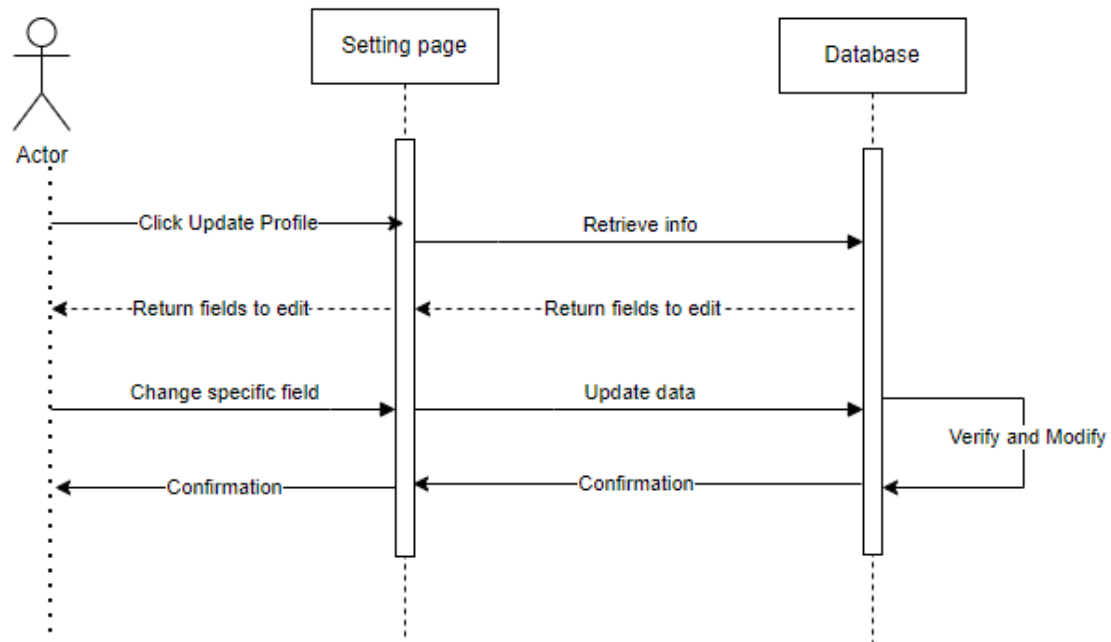
4.4 Sequence Class Diagrams

4.4.1 Login



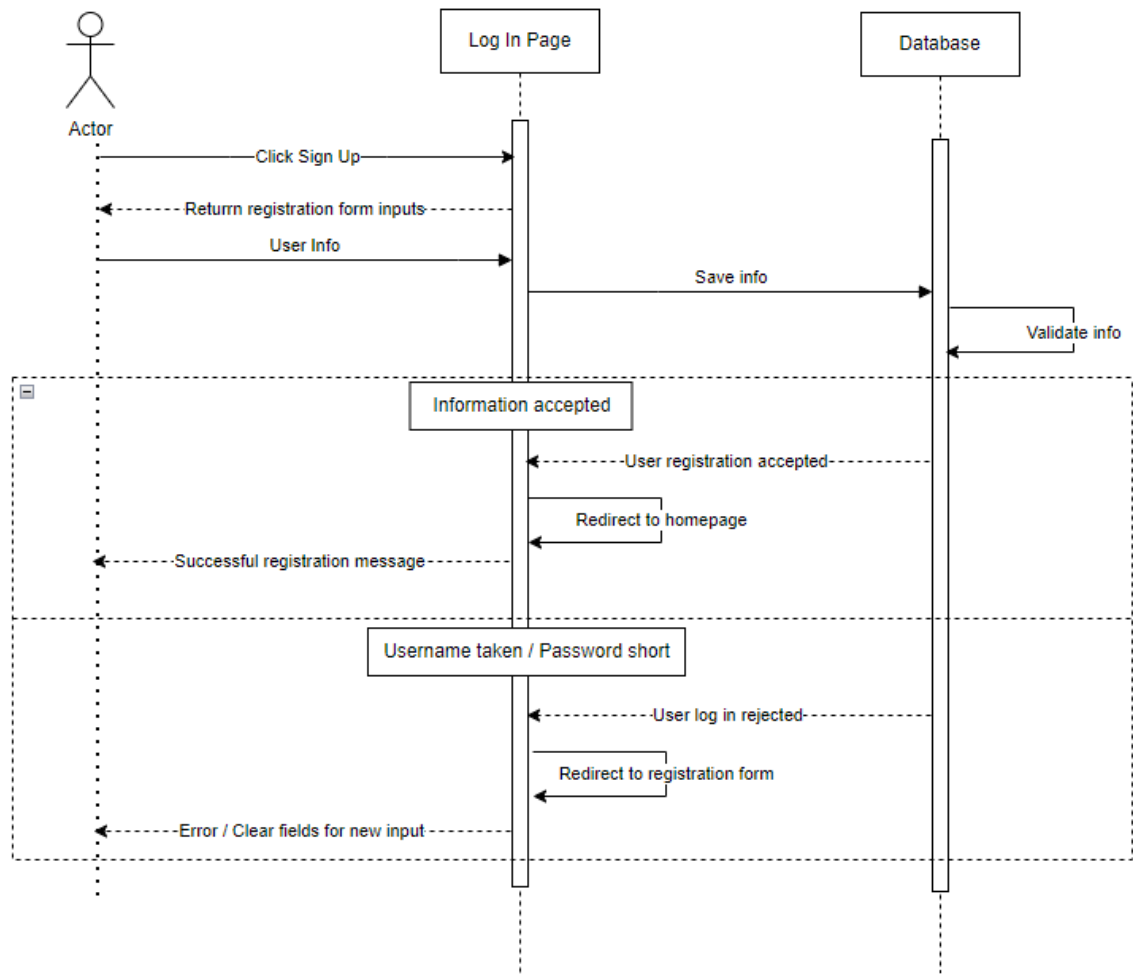
Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

4.4.2 Create Profile



Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

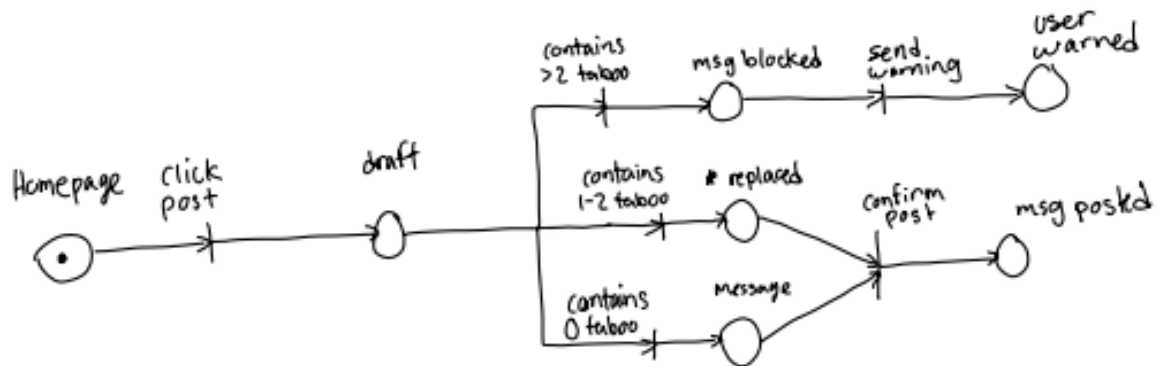
4.4.3 Apply to be OU/CU



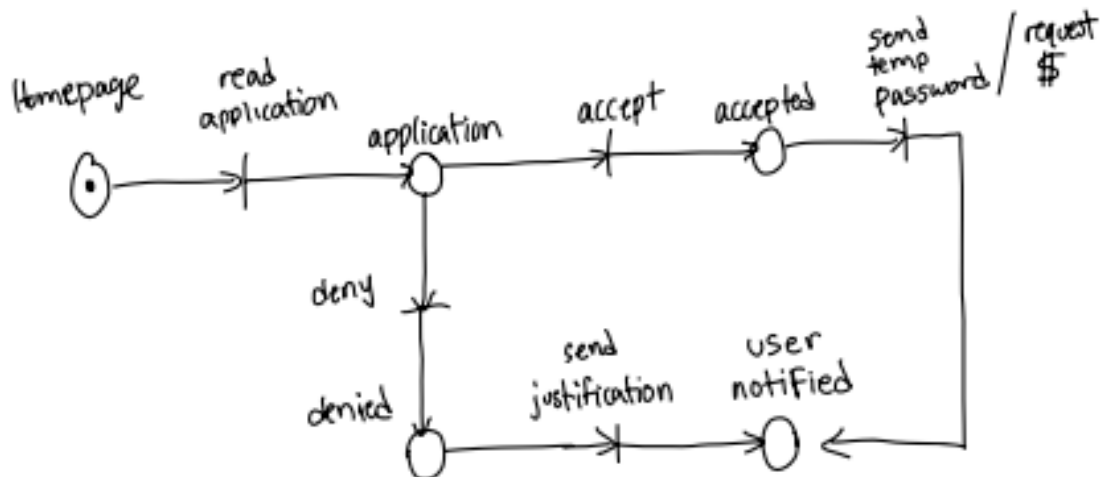
Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

4.5 Petri Nets

4.5.1 Post Message

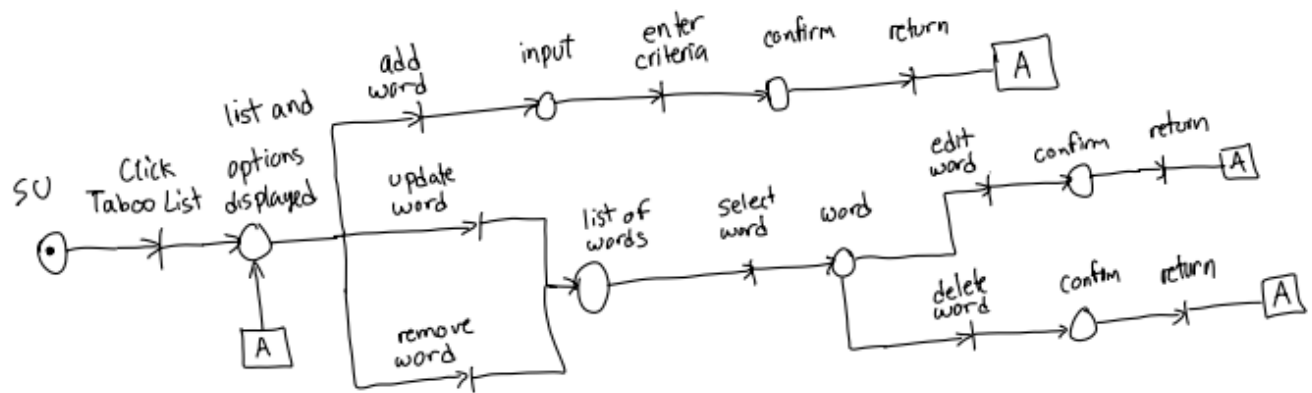


4.5.2 SU Accept/Deny Application



Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

4.5.3 SU Manage Taboo List



Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

5. Detailed Design

5.1 User Login:

```
function userLogin(req, res):
  try:
    { email, password } = req.body

    existingUser = await prisma.user.findUnique({ where: { email } })

    if existingUser === null:
      res.json({ message: "email does not exist" })
      return

    { id, name } = existingUser

    if existingUser.password === password:
      token = jwt.sign({ id, name }, process.env.ACCESS_TOKEN_SECRET, { expiresIn: "10m" })
      refreshToken = jwt.sign({ id, name }, process.env.REFRESH_TOKEN_SECRET)

      // store tokens in cookies
      res.cookie("access-token", token, { maxAge: 600000 })
      res.cookie("refresh-token", refreshToken)

      res.json({
        user: existingUser,
        token: token,
        refreshToken: refreshToken
      })
    else:
      res.json({ message: "incorrect password" })

  catch err:
    console.log("error")
    res.json({ error: err })
```

5.2 User Sign-up:

```
function userSignUp(req, res):
  try:
    { email, name, password } = req.body

    existingUser = await prisma.user.findUnique({ where: { email } })

    if existingUser !== null:
      res.json({ message: "email already exists" })
      return

    newUser = await prisma.user.create({
      data: {
        email,
        name,
        password
      }
    })

    res.json({ user: newUser })

  catch err:
    console.log(err)
```

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

```
res.json({ error: err })
```

5.3 User Log-out:

```
function userLogOut(req, res):
  // Clear access-token cookie
  res.cookie("access-token", "", {
    maxAge: 1
  })

  // Clear refresh-token cookie
  res.cookie("refresh-token", "", {
    maxAge: 1
  })

  // Redirect to the home page
  res.redirect("/")
```

5.4 Get User Followers:

```
function getUserFollowers(req, res):
  // Get the user ID from the request parameters
  id = req.params.id

  const currentUser = await prisma.user.findUnique({
    where : { id : id }
  })

  if(currentUser === null) {
    // handle error accordingly
    res.status(404) // send error to client
  }

  // Retrieve followers from the database
  followers = await prisma.followers.findMany({
    where: {
      followeeId: id
    },
    select: {
      personFollowingUser: {
        select: {
          id: true,
          name: true,
          email: true
        }
      }
    }
  })
```

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

```
// Log the first follower (for debugging purposes)
console.log(followers[0])
```

```
// Return the followers as JSON response
res.json({ followers: followers })
```

5.5 Add new Followers/Followings:

```
function addFollower(req, res):
    // Extract personBeingFollowed and personFollowingUser from request body
    { personBeingFollowed, personFollowingUser } = req.body

    // check if personBeingFollowed and personFollowingUser are existing users
    const userOne = await prisma.user.findUnique({
        where : {
            id : personBeingFollower
        }
    })
    const userTwo = await prisma.user.findUnique({
        where : {
            id : personFollowingUser
        }
    })
    if(userOne === null || userTwo === null) {
        handle errors accordingly
        res.status(404)
    }
```

```
// Create a new follower in the database
newFollower = await prisma.followers.create({
    data: {
        followeeId: personBeingFollowed,
        follower: personFollowingUser
    }
})
```

```
// Return the newly created follower as JSON response
res.json({ newFollower: newFollower })
```

5.6 Get User Followings:

```
function getUserFollowing(req, res):
    // Get the user ID from the request parameters
    id = req.params.id
```


Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

```
// Retrieve users being followed from the database
following = await prisma.following.findMany({
  where: {
    personBeingFollowed: id
  },
  select: {
    User: {
      select: {
        name: true
      }
    }
  }
})
```

5.7 Get All Posts:

function getAllPost(req, res):

```
// Retrieve all posts from the database
posts = await prisma.post.findMany()

// Return the posts as a JSON response
res.json({ message: "post route", posts: posts })
```

```
// Return the users being followed as JSON response
res.json({ following: following })
```

5.8 Get User Post:

function getUserPost(req, res):

```
// Get the user ID from the request parameters
id = req.params.id

const currentUser = await prisma.user.findUnique({
  where : {
    id : id
  }
})

if(currentUser === null) {
  // handle error accordingly
  res.status(404) // send error to client
}
```

```
// Retrieve posts of a specific user from the database
```

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

```

userPost = await prisma.post.findMany( {
  where: {
    userId: id
  }
})

res.json({ userPost: userPost })

```

5.9 Create New Post:

function createPost(req, res):

```

// Extract body, title, and userId from request body
{ body, title, userId } = req.body

// Create a new post in the database
newPost = await prisma.post.create( {
  data: {
    title,
    body,
    userId
  }
})

// Return the newly created post as JSON response
res.json({ newPost: newPost })

```

5.10 Get Post Details:

function getPostDetails(req, res):

```

// Get the post ID from the request body
id = req.body.id

// Retrieve post details from the database
postDetails = await prisma.post.findUnique( {
  where: {
    id: id
  }
})

// Return the post details as JSON response
res.json({ postDetails: postDetails })

```

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

5.11 Comment on Post:

function commentOnPost(req, res):

```

    // Extract post ID, commenter ID, and comment text from request body
    { postId, userId, commentText } = req.body

    const currentUser = await prisma.user.findUnique( {
      where : {
        id : userId
      }
    })

    if(currentUser === null) {
      // handle error accordingly
      res.status(404) // send error to client
    }

    const post = await prisma.post.findUnique( {
      where: { id : postId }
    })
    if(post === null) res.status(404)

    // Create a new comment in the database
    newComment = await prisma.comment.create( {
      data: {
        postId,
        userId,
        text: commentText
      }
    })

    // Return the newly created comment as JSON response
    res.json( { newComment: newComment } )

```

5.12 Like a Post:

function likePost(req, res):

```

    // Extract post ID and user ID from request body
    { postId, userId } = req.body

    const currentUser = await prisma.user.findUnique( {
      where : {
        id : userId
      }
    })

```

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

```

    })

    if(currentUser === null) {
        // handle error accordingly
        res.status(404) // send error to client
    }

    const currentPost = await prisma.post.findUnique({
        where : {
            id : postId
        }
    })

    if(currentUser === null) {
        // handle error accordingly
        res.status(404) // send error to client
    }

    // Check if the user has already liked the post
    existingLike = await prisma.postLike.findFirst({
        where: {
            postId,
            userId
        }
    })

    if existingLike:
        // User has already liked the post, handle accordingly (e.g., remove the like)
        // Code to handle removing the like goes here
        res.json({ message: "Post already liked by the user" })
    else:
        // User has not liked the post, create a new like in the database
        newLike = await prisma.postLike.create({
            data: {
                postId,
                userId
            }
        })

        // Return the newly created like as JSON response
        res.json({ newLike: newLike })

```

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

5.13 Dislike a Post:

```
function dislikePost(req, res):

    // Extract post ID and user ID from request body
    { postId, userId } = req.body

    // Check if the user has already disliked the post
    existingDislike = await prisma.postDislike.findFirst({
      where: {
        postId,
        userId
      }
    })

    if existingDislike:
      // User has already disliked the post, handle accordingly (e.g., remove the dislike)
      // Code to handle removing the dislike goes here
      res.json({ message: "Post already disliked by the user" })
    else:
      // User has not disliked the post, create a new dislike in the database
      newDislike = await prisma.postDislike.create({
        data: {
          postId,
          userId
        }
      })

      // Return the newly created dislike as JSON response
      res.json({ newDislike: newDislike })
```

5.14 Delete a Post:

```
function deletePost(req, res):

    // Extract post ID and user ID from request body
    const id = req.body.id

    const deletedPost = await prisma.posts.delete({
      where : {
        id:id
      }
    })

    res.json({ deletedPost:deletedPost})
```

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

5.15 Find a User:

```
function findUser(req, res):

  const username = req.params.username
  const user = await prisma.user.findUnique( {
    where : { name: username}
  }
  if(user === null) res.json( {error: "user does not exist"})
  return res.json({user: username})
```

5.16 Handling Taboo:

```
function findUser(req, res):

  const tabooList =["word1","word2",.....]
  let words = messageText.split(" ")
  let tabooCount = 0
  for (let word of words) {
    if (tabooWordsList.includes(word)) {
      count tabooWordCount++
      messageText = messageText.replace(word, "*".repeat(word.length)) }
    if (tabooWordCount > 2) {
      // block message
      // warn author
    } return messageText
  };
```

5.17 Creating Ad:

```
function createAd(req, res):

  const ad = await prisma.ad.create( {
    data : { // data for new ad }
  }
  res.json({newAd: ad})
```

5.18 Banning User:

```
function badUser(req,res):

  const id = req.body.id
  // check if user exist in db
  const user = await prisma.user.findUnique(...)

  if(user === null) res.status(404)

  // update user obj to denote that user is banned

  const updatedUser = await prisma.updateUser( // mark user as banned )
  res.json({updatedUser : updatedUser})
```

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

5.19 Warn User:

```
function warnUser(req,res):
  const id = req.body.id

  // check if user exist in db

  const user = await prisma.user.findUnique(..)

  // create new warning and link to user

  const newWarning = await prisma.warnings.create({...}) // pass in userId, reason , etc

  res.json({newWarning: newWarning})
```

5.20 Get Ads:

```
function getAds(req,res):
  const ads = await prisma.ads.findMany()
  res.json({ads:ads})
```

5.21 Delete Ads:

```
function deleteAds(req,res):
  const id = req.body.id

  const deletedAd = await prisma.ads.delete(....) // pass in id for ad to delete
  res.json({ads:ads})
```

5.22 Add Trendy:

```
function makeTrendyUser(req,res):
  const id = req.body.id

  const updatedUser = await prisma.user.update({ trendy :true})
```

5.23 Remove Trendy:

```
function removeTrendyUser(req,res):
  const id = req.body.id

  const updatedUser = await prisma.user.update({ trendy :false})
```

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

5.24 Get Top 3:

```
function getTopTrendy(req,res):
const id = req.body.id
const trendyUsers = await prisma.messages.findMany( { where :{ likes > }, selected :{userId}})
let i = 0 ;
const topThree = []
while i <= 3:
    if(trendyUsers[i] !==null):
        topThree.push[tredyUser[i])

res.json({topThree:topThree})
```

5.25 Get User Details:

```
function getUserDetails(req, res):

    // Get the user ID from the request parameters
    id = req.params.id

const currentUser = await prisma.user.findUnique( {
    where : {
        id : id
    }
})

if(currentUser === null) {
    // handle error accordingly
    res.status(404) // send error to client
}

// Retrieve user details from the database, excluding the password
{ password, ...userObject } = await prisma.user.findUnique( {
    where: {
        id: id
    },
    include: {
        Followers: {
            select: {
                personFollowingUser: {
                    select: {
                        id: true,
                        name: true,
                        email: true
                    }
                }
            }
        }
    }
})
```


Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

```

    }
  }
},
Following: {
  select: {
    User: {
      select: {
        id: true,
        name: true,
        email: true
      }
    }
  }
}
})

```

```

// Return the user details (excluding password) as JSON response
res.json({ user: userObject })

```

5.26 Create Profile Details:

```

function createProfile(req, res):

  // Get the user ID from the request parameters
  id = req.params.id

  const user = await prisma.user.find(where: {id:id})

  // check if user exist

  const newProfile = await prisma.profile.create(data: { // user , and other info for profile }

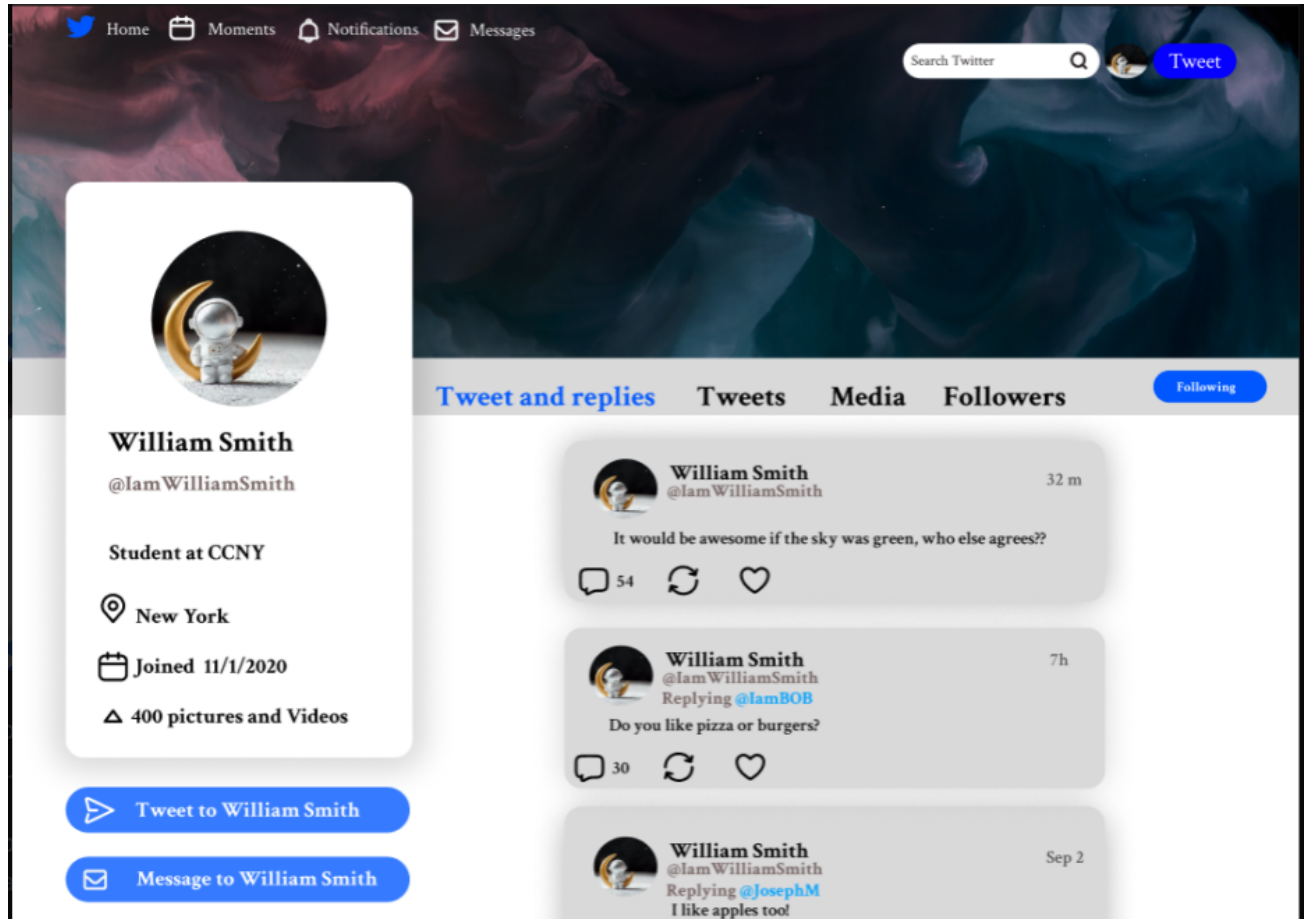
  res.json({newProfile:newProfile})

```

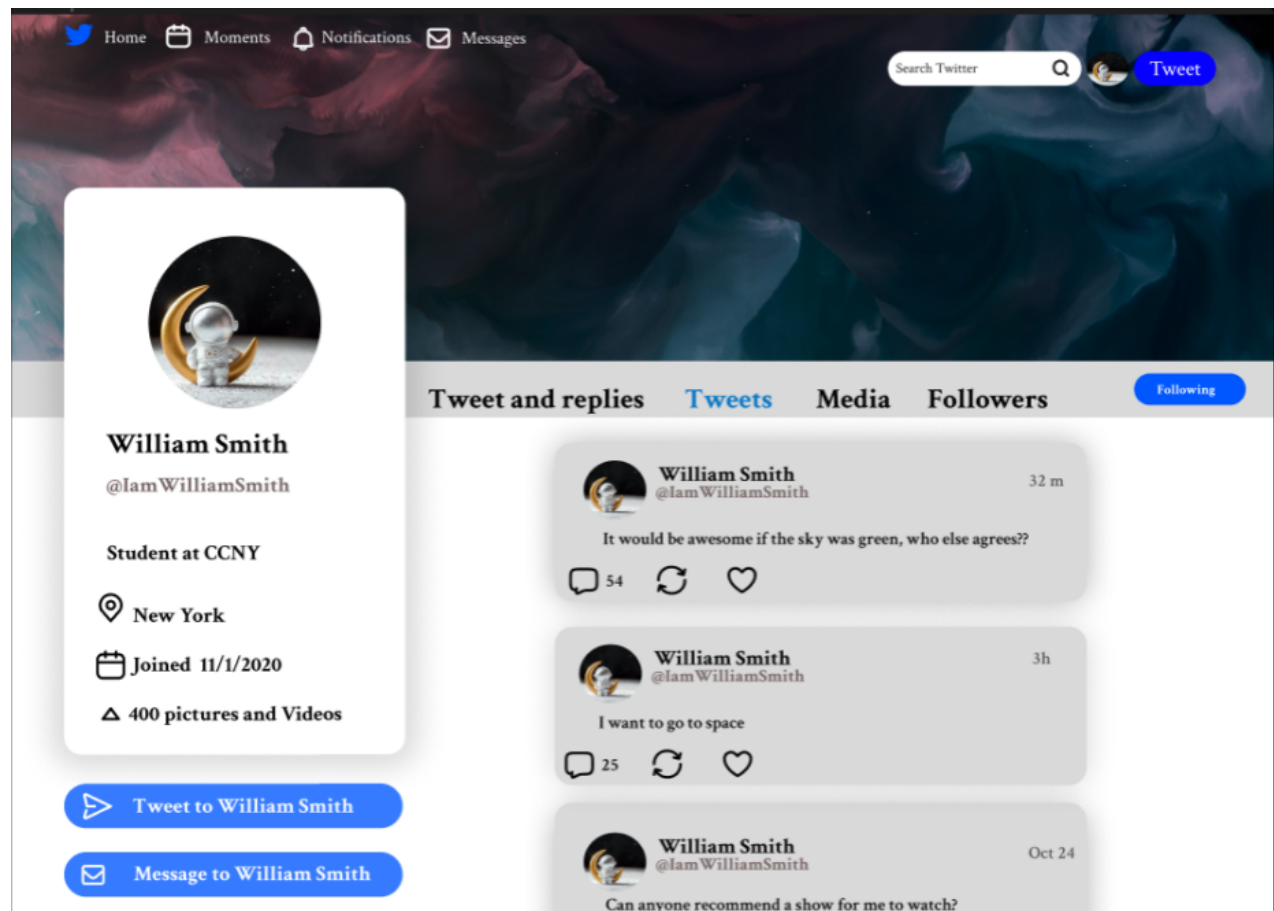
Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

6. Supporting Information

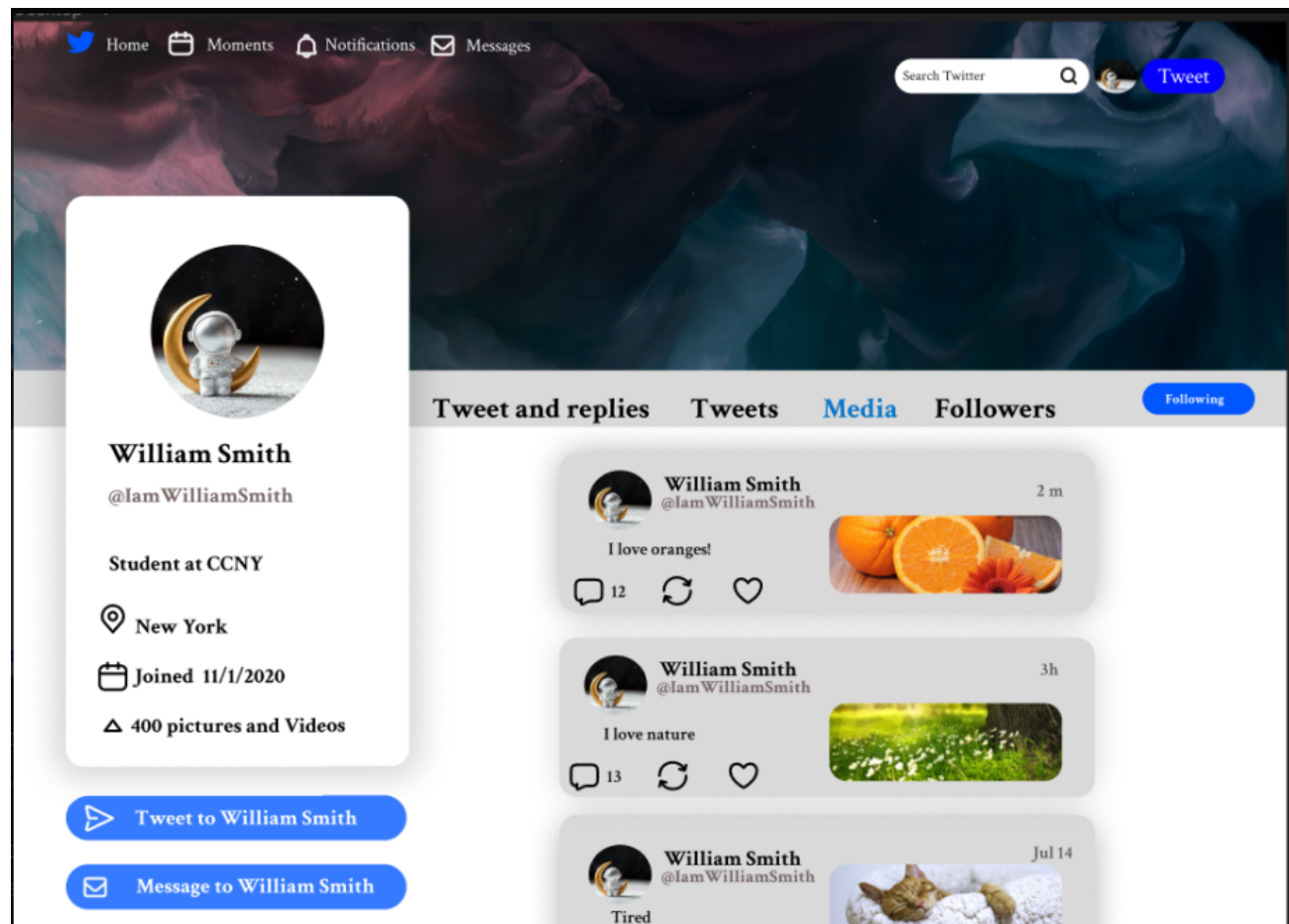
6.1 System Screens/GUI Screens



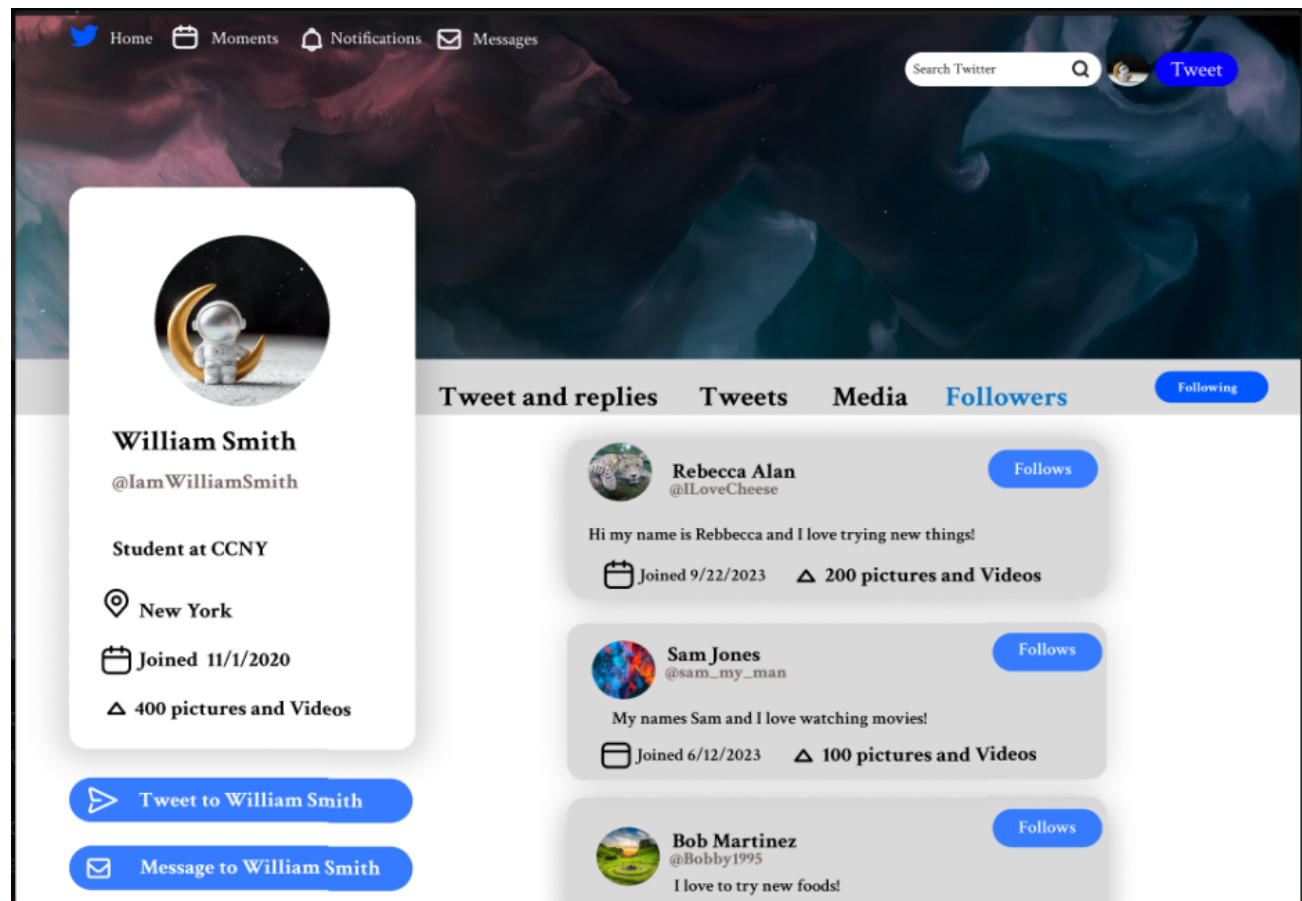
Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	



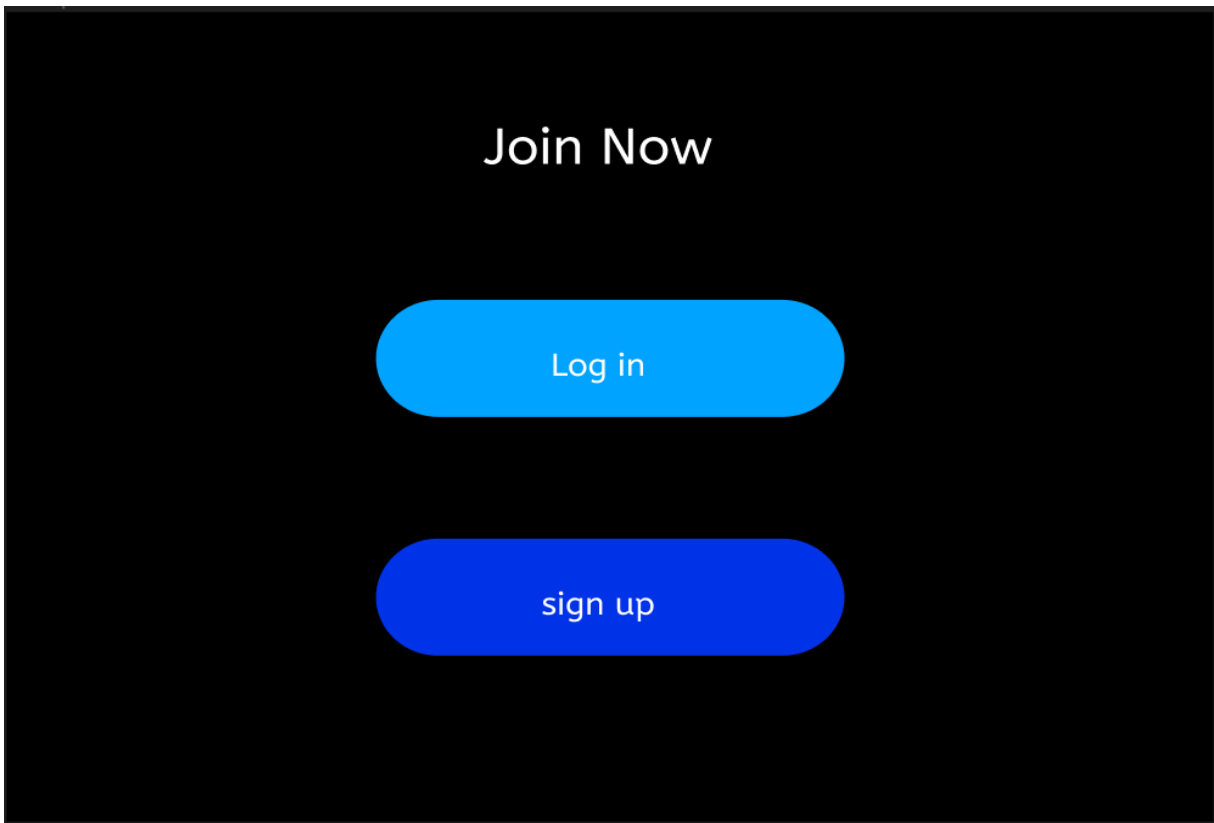
Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	



Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	



Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	



Sign in to your account

Email address

Password

[Forgot my password](#)

Dont have a account? [Register](#)

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	


Sign up

Email address

Password

Confirm Password

Already have a account? [Log in](#)



trouble logging in?

Enter the email address you used when creating your account.

Send login link

OR

Create New Account

Back to login

Mini Twitter	Version: 2.0
Software Requirements Specification	Date:11/22/2023
<Document Identifier>	

6.2 Group Meeting Memos

Meeting Date	Details	Members
10/17/2023	Initial meeting discussing project ideas, languages, experience level	Mahmud Hasan, Ramim Tarafdar, Oluwanifesimi Kukoyi, Taha Nayyar
10/24/2023	Planning report 1	Mahmud Hasan, Ramim Tarafdar, Oluwanifesimi Kukoyi, Taha Nayyar
10/27/2023	Finalizing report 1 and submitting	Mahmud Hasan, Ramim Tarafdar, Oluwanifesimi Kukoyi, Taha Nayyar
11/19/2023	Heavy working on report 2 and began diagrams	Mahmud Hasan, Ramim Tarafdar, Oluwanifesimi Kukoyi, Taha Nayyar
11/20/2023	More progress on report 2, had to scrap some sections bc of misunderstanding and restart.	Mahmud Hasan, Ramim Tarafdar, Oluwanifesimi Kukoyi, Taha Nayyar
11/22/2023	Finalizing all parts of report 2 and submitting	Mahmud Hasan, Ramim Tarafdar, Oluwanifesimi Kukoyi, Taha Nayyar

6.3 Concerns

Our group has been consistently communicating through the semester and have dedicated assignments to each member in regards to both the report questions and project functionalities. We have essentially given backend, frontend, UI designer, and product manager roles to ensure steady progress is achieved. The only concern at the moment is that all members have other responsibilities outside of classes, which makes it difficult to meet and work together as well as meet personal deadlines. Otherwise, no major issues.

6.4 Git Repo

<https://github.com/olukukoyi/twitter-clone-322-ccny>