**Team Q**

**HungryText Software Requirements Second Report**

**Version 2.0**

| HungryText | Version: 2.0 |
| --- | --- |
| Software Requirements Specification | Date: 13/04/2025 |
| https://docs.google.com/document/d/17rqIBVsmEUkUKl66jpTz\_ovucCAesi9mZjLnUkHpLGw/ | |

**Revision History**

| **Date** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
| 17/03/2025 | 1.0 | Initial Specifications | Team Q |
| 13/04/2025 | 2.0 | Second Report | Team Q |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

1. Introduction

| 1.1 | Purpose |  |
| --- | --- | --- |
| 1.2 | Scope |  |
| 1.3 | Definitions, Acronyms, and Abbreviations |  |
| 1.4 | References |  |
| 1.5 | Overview |  |

2. Overall Description

2.1 Use-Case Model Survey

2.2 Assumptions and Dependencies

3. Specific Requirements

3.1 Use-Case Reports

3.2 Supplementary Requirements

4. System Collaboration Diagram

5. Use Case Scenarios

6. System E/R Diagram

7. System Pseudo-Code

8. UI Mockups & Demo

8.1 UI Mockups

8.2 Demo

9. Memos & Teamwork

10. Github Repository

**Software Requirements Specification**

**1. Introduction**

**1.1 Purpose**

The purpose of this document is to define the software requirements for the LLM-Based Cooperative Text Editor. The system, titled HungryText, is designed to enable users to edit texts collaboratively with the assistance of a Large Language Model (LLM), enforcing access control based on account types and integrating a token-based economy for paid users.

**1.2 Scope**

The system functions as an LLM-integrated text editor, allowing users to:

- Submit and edit text using AI assistance.

- Share and collaborate on documents.

- Manage access via different user roles: Free Users, Paid Users, and Super Users.

- Enforce blacklisted words and logging mechanisms.

- Utilize a token-based system to regulate the actions that paid users can take.

- Be hosted as either a local application or an online platform.

**1.3 Definitions, Acronyms, and Abbreviations**

- LLM: Large Language Model, used for text correction.

- Tokens: A virtual currency used by paid users for system features.

**1.4 References**

- [CSC 322 Team Q Project Overview Document](https://docs.google.com/document/d/1BrA2LXJOAdEAfHKduX8m3mRpg3o_v97kZ-IXvs2v-6w/edit?usp=sharing)

- [CSC 322 Team Q Use Case Diagram](https://drive.google.com/file/d/186g3A9s8gIwXW8enjyHQbAzq0wHA2Rn3/view?usp=sharing)

- [Project Requirements Document](https://www.dropbox.com/scl/fi/7o8yt4vgr4l1qm3r08qq7/proj_req.pdf?rlkey=n4t8chm7c69bdlhryxs572jg7&e=2&st=eygrqreb&dl=0)

**1.5 Overview**

This document outlines the functional and non-functional requirements of the system, focusing on user role and permissions, text editing capabilities, collaboration features, and administrative controls.

**2. Overall Description**

**2.1 Use-Case Model Survey**

- Guest Users: Can sign up and sign in.

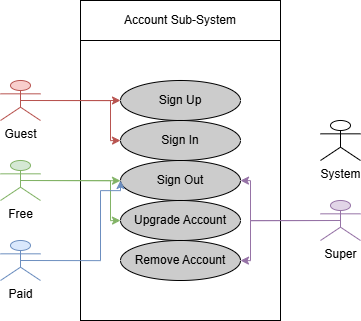
- Free Users: Can submit up to 20 words but have limited privileges.

- Paid Users: Can edit text, purchase tokens, collaborate, and use LLM correction.

- Super Users: Can handle complaints, manage user accounts, and oversee system rules.

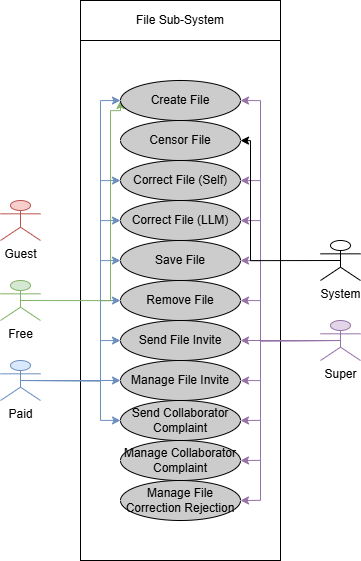
- System: Dummy user type to represent automatic actions.

- Account System: User registration, sign-in, upgrading to paid accounts, account removal.



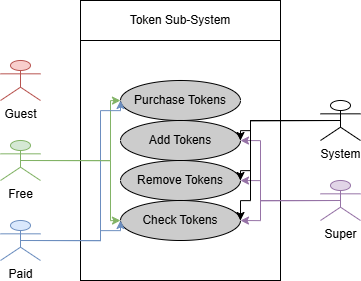
**(Please Proceed To Next Page)**

- File System: Text submission, LLM/self-correction, file management, collaborator complaints.

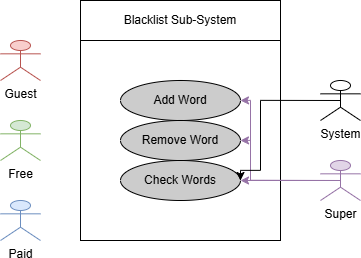


**(Please Proceed To Next Page)**

- Token System: Managing user tokens for accessing features.

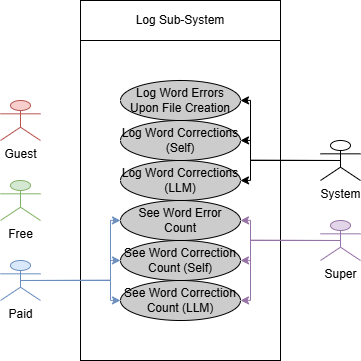


- Blacklist System: Sharing documents, handling invites, and managing user disputes.



**(Please Proceed To Next Page)**

- Log System: Logging actions for statistics and administrative oversight.



**2.2 Assumptions and Dependencies**

- The system assumes users will primarily work with text-based documents.

- LLM features will be powered by an open-source model (e.g., Llama, DeepSeek).

- The token-based system regulates access for paid users.

- Super users will handle moderation and administrative tasks.

**3. Specific Requirements**

**3.1 Use-Case Reports**

1. User Account Management

- Free users can register and submit up to 20 words per text.

- Paid users can upgrade accounts and purchase tokens.

- Super users can suspend, fine, or terminate users for violations.

2. Text Submission & Editing

- Users can input text manually or upload a file.

- Blacklisted words are automatically censored.

- Users can choose between self-correction or LLM correction.

- LLM corrections are highlighted for user review.

3. Token-Based System

- Paid users are charged tokens per submission and correction.

- If a user lacks tokens, they receive a penalty.

- Users receive bonus tokens if LLM finds no errors in long text submissions.

4. Collaboration Features

- Paid users can share documents and invite collaborators.

- Invitees can accept or reject invitations.

- Super users resolve complaints and disputes among collaborators.

5. Logging & Analytics

- User actions are logged for analysis.

- Paid users can view their editing history and correction statistics.

**3.2 Supplementary Requirements**

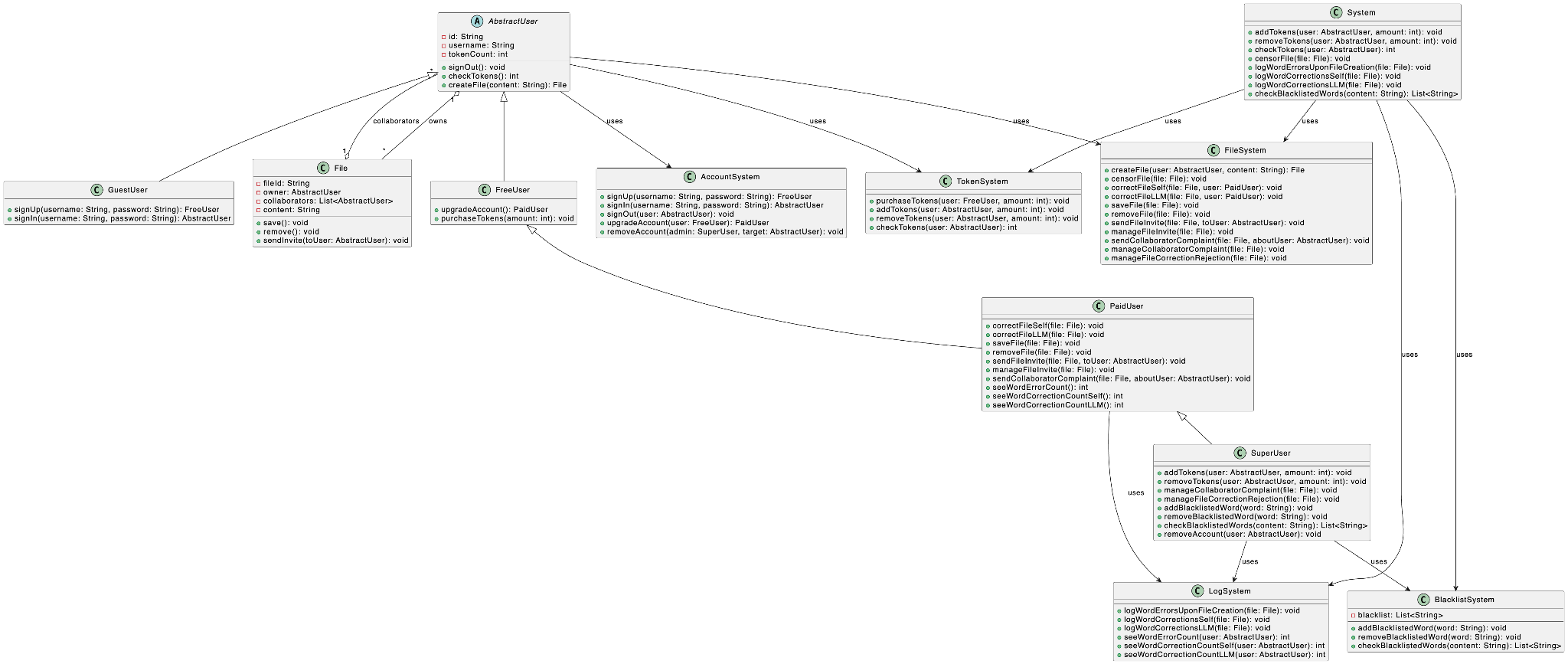
- System should have a graphical user interface (GUI) designed for usability.

- The software should support both online and offline functionality.

- LLM integration must be local or use free APIs to avoid reliance on paid services.

- Users should have access to a statistics panel to monitor token usage and correction history.

**4. System Collaboration Diagram**



[Detailed Image](https://cdn.discordapp.com/attachments/1351726870097301606/1360131025417539584/ClassDiagram.png?ex=67fd4bfa&is=67fbfa7a&hm=c1151f60932f159531fe5f5ee6b5c64d7d3e02c7ccb68507aa99fd5ea01ec0d1&)

**5. Use Case Scenarios**

### Use Case 1: Submit Text

**Actors:** FreeUser, PaidUser, System

**Normal Flow:**

1. User enters or uploads text.
2. System evaluates word count.
3. If the user is a FreeUser and text contains 20 words or fewer, the system accepts submission.
4. If the user is a PaidUser and has sufficient tokens, the system deducts tokens and accepts the text.
5. System scans text for blacklisted words.
6. Any blacklisted words are masked with asterisks; token deduction corresponds to the character count of those words.
7. Final processed text is displayed to the user.

**Exceptional Flow:**

* FreeUser enters over 20 words: submission is blocked; user is logged out and prevented from logging in for 3 minutes.
* PaidUser with insufficient tokens: submission rejected and 50% of available tokens deducted as penalty.
* File unreadable or corrupted: system prompts user to re-submit.

### Use Case 2: LLM Correction

**Actors:** PaidUser, SuperUser, System

**Normal Flow:**

1. PaidUser selects LLM correction.
2. System sends text to LLM and returns suggestions.
3. Suggested corrections are displayed for review.
4. If accepted, each accepted correction deducts 1 token.
5. If rejected, the user may whitelist the word or submit a rejection reason.
6. SuperUser reviews rejection reasons.
7. If reason accepted: 1 token deducted. If rejected: 5 tokens deducted.
8. User completes the correction process.

**Exceptional Flow:**

* LLM fails to respond: system shows error; no tokens deducted.
* User fails to respond: correction session is suspended.
* SuperUser delay: final decision pending; process paused.

### Use Case 3: Send File Invite

**Actors:** PaidUser (Inviter), PaidUser (Invitee), System

**Normal Flow:**

1. Inviter selects file and issues invitation.
2. System prompts for Invitee credentials.
3. System verifies Invitee is a valid PaidUser.
4. Invitation is delivered.
5. If Invitee accepts: file is shared with full edit access.

**Exceptional Flow:**

* Invitee declines: Inviter loses 3 tokens.
* Invitee is not a PaidUser: invitation is cancelled; error shown.
* File altered/deleted before response: system invalidates invite.
* Invitee unresponsive: invitation expires after a fixed time.

### Use Case 4: User Registration / Upgrade to Paid

**Actors:** FreeUser, SuperUser, System

**Normal Flow:**

1. FreeUser submits an upgrade request.
2. System collects and forwards data.
3. SuperUser reviews and approves the request.
4. User becomes a PaidUser.

**Exceptional Flow:**

* Invalid information: system prompts correction.
* Request denied: user remains a FreeUser and is notified.

### Use Case 5: Purchase Tokens

**Actors:** PaidUser, System

**Normal Flow:**

1. PaidUser selects the token package.
2. System processes transactions.
3. Tokens are added to the user's balance.

**Exceptional Flow:**

* Transaction fails: system alerts user and prompts retry.

### Use Case 6: Submit Blacklist Word

**Actors:** FreeUser, PaidUser, SuperUser, System

**Normal Flow:**

1. User submits a proposed blacklisted word.
2. System records and forwards to SuperUser.
3. SuperUser reviews and accepts.
4. Word added to blacklist.

**Exceptional Flow:**

* Word is already blacklisted: system informs user.
* SuperUser rejects the word: user notified; no change made.

### Use Case 7: Save File

**Actors:** PaidUser, System

**Normal Flow:**

1. PaidUser initiates file save.
2. System verifies token balance.
3. If 5 or more tokens: deducts 5 tokens and saves file.

**Exceptional Flow:**

* Insufficient tokens: system blocks save and suggests token purchase.

### Use Case 8: View Usage Statistics

**Actors:** PaidUser, System

**Normal Flow:**

1. PaidUser selects 'View Stats'.
2. System retrieves and displays usage data.

**Exceptional Flow:**

* Stats unavailable: system displays error.

### Use Case 9: Submit Complaint About Collaborator

**Actors:** PaidUser, SuperUser, System

**Normal Flow:**

1. PaidUser submits a complaint.
2. Complaint is forwarded to SuperUser.
3. SuperUser reviews and applies a token penalty to either the complainant or the accused.

**Exceptional Flow:**

* Invalid complaint: no action taken.
* False claims: complainant is penalized.

### Use Case 10: Correction Bonus

**Actors:** PaidUser, System

**Normal Flow:**

1. PaidUser submits a text > 10 words to LLM.
2. If LLM suggests zero corrections:  
   * System rewards users with 3 bonus tokens.

**Exceptional Flow:**

* LLM overlooks error: no reward; no penalty.

### Use Case 11: Log In

**Actors:** FreeUser, PaidUser, SuperUser, System

**Normal Flow:**

1. User launches the system and initiates login.
2. System requests credentials.
3. Credentials are verified.
4. User is redirected to the relevant dashboard.

**Exceptional Flow:**

* Incorrect credentials: system prompts retry.
* FreeUser within lockout window: access denied.
* Expired session: re-authentication required.

### Use Case 12: Log Out

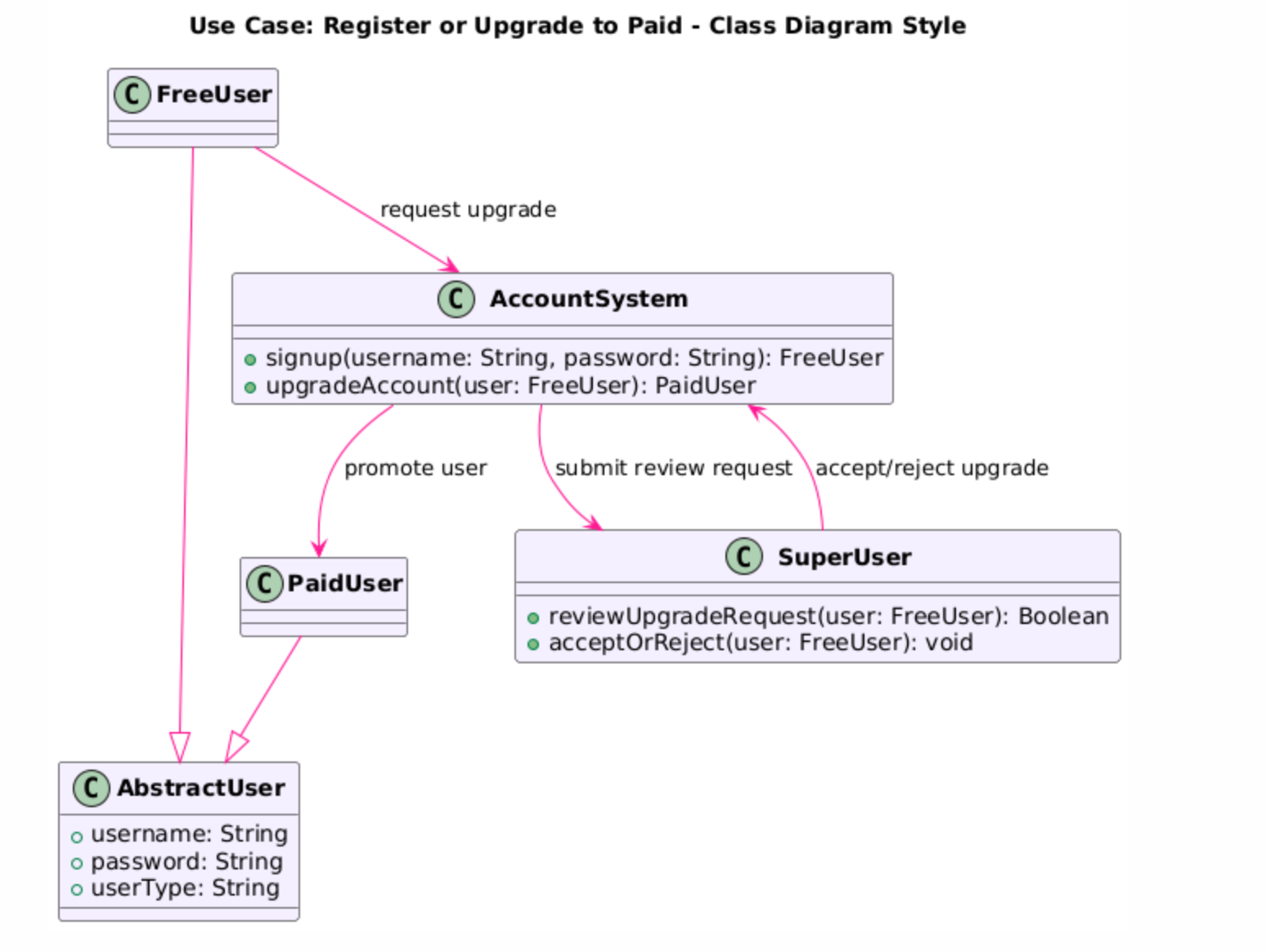
**Actors:** FreeUser, PaidUser, SuperUser, System

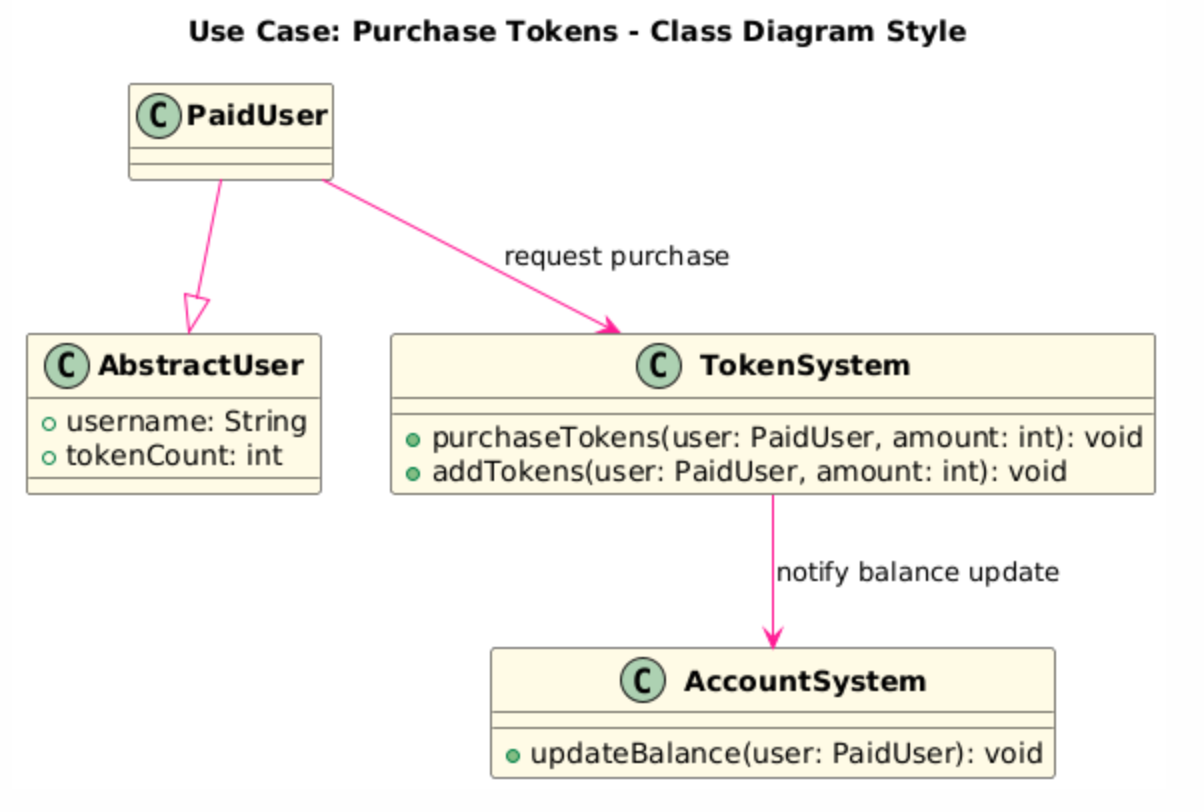
**Normal Flow:**

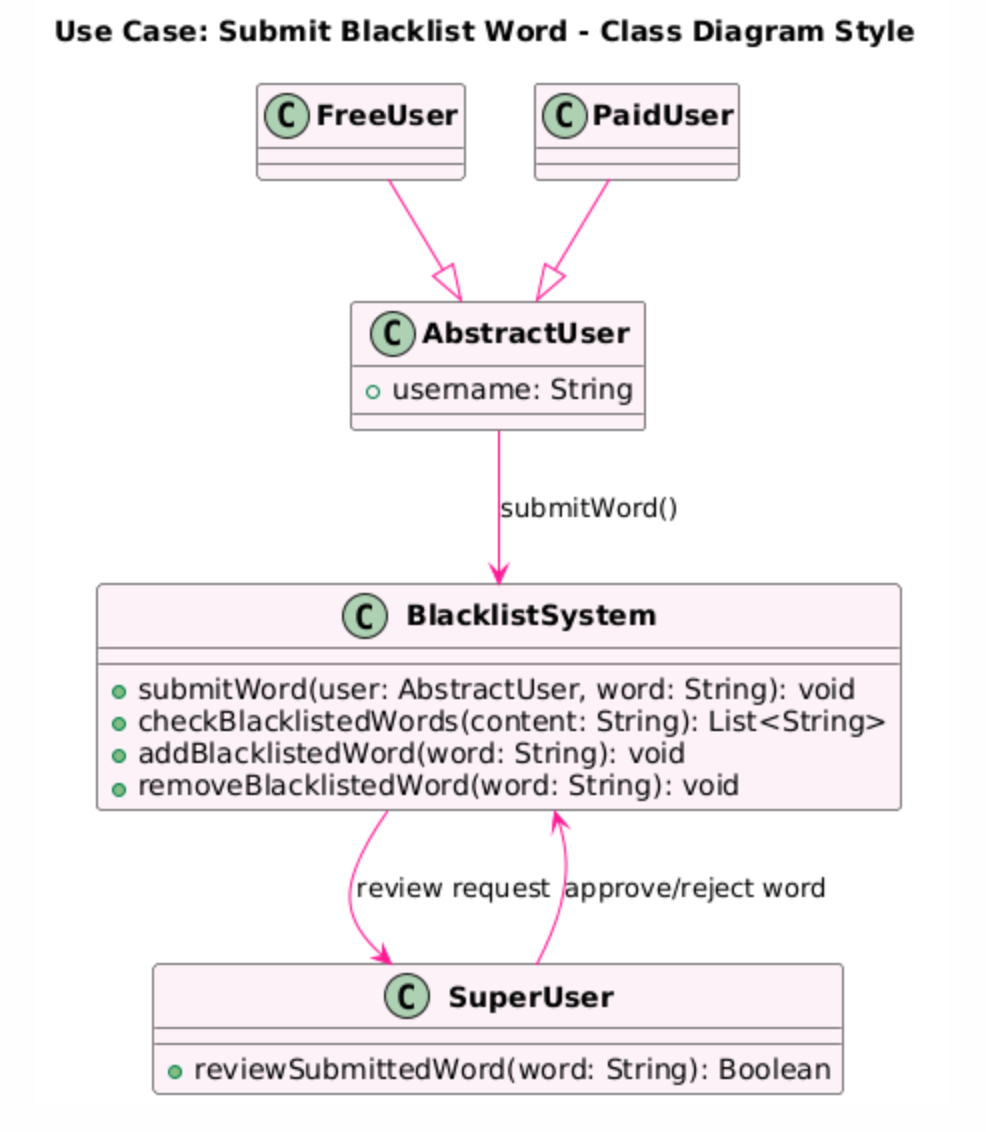
1. User clicks 'Log Out'.
2. System terminates session.
3. FreeUser’s logout time is recorded.
4. User returns to the login screen.

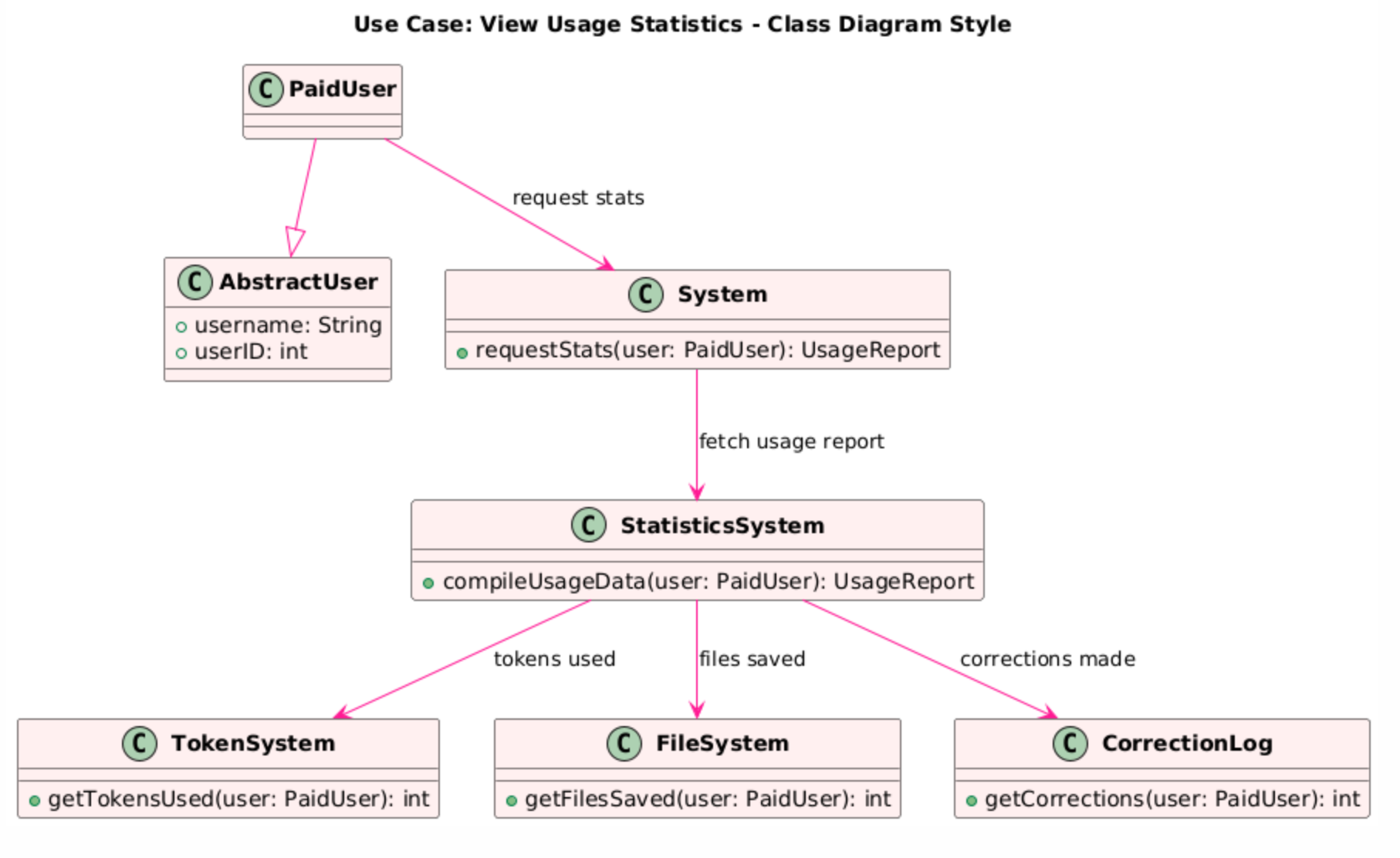
**Exceptional Flow:**

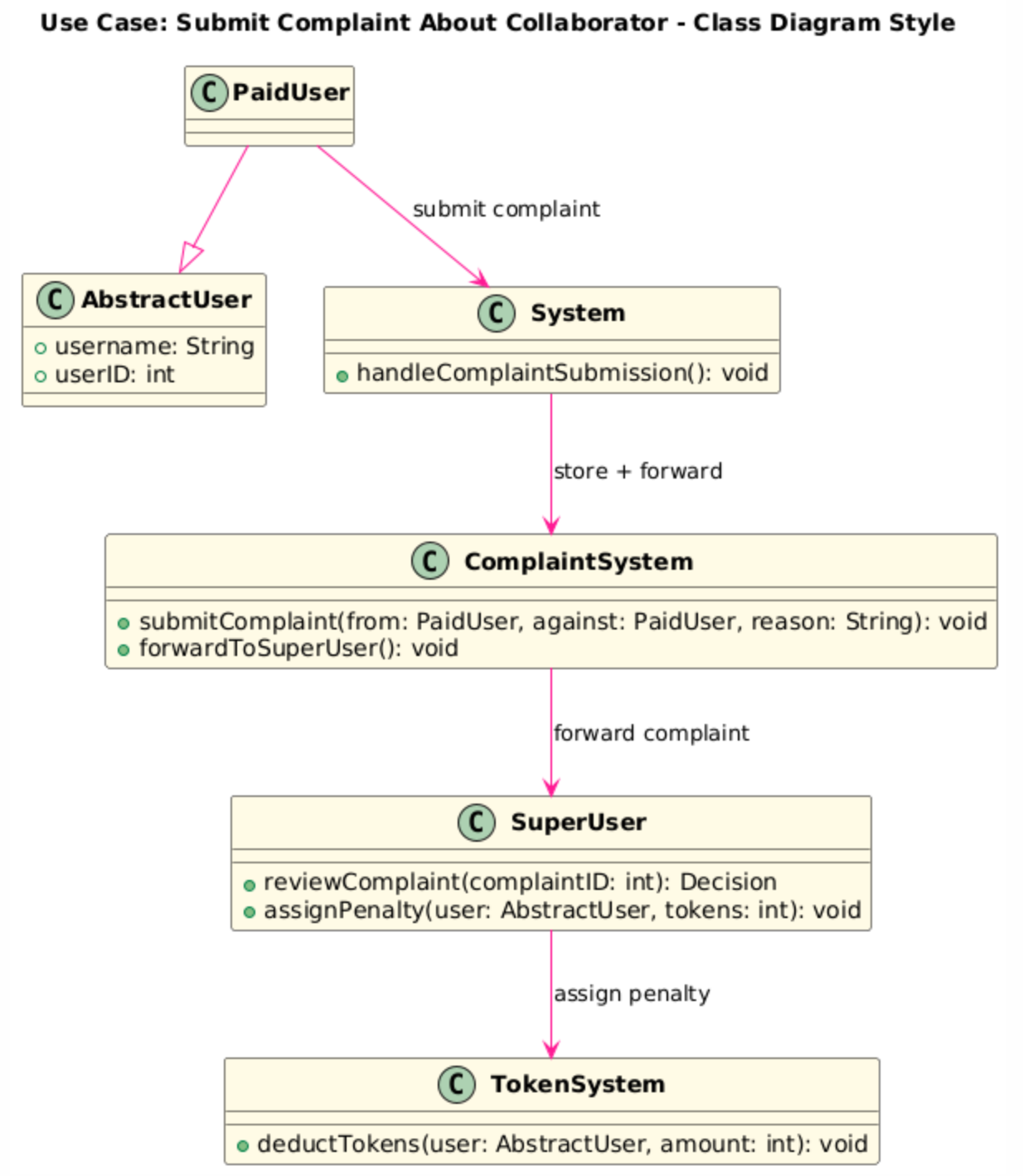
* Log out fails due to system error: session remains active.
* FreeUser closes app without logout: lockout logic may not initiate.

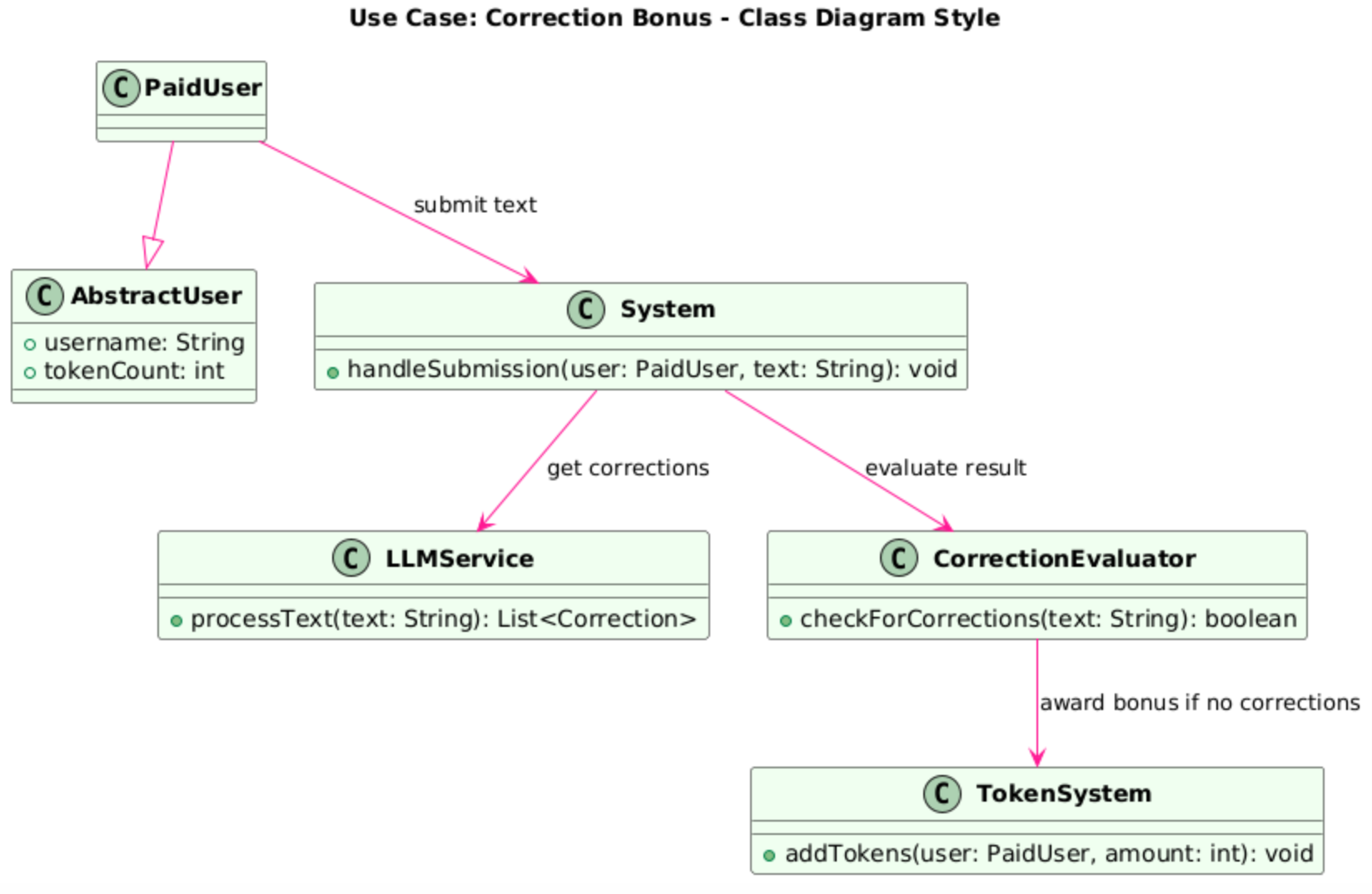


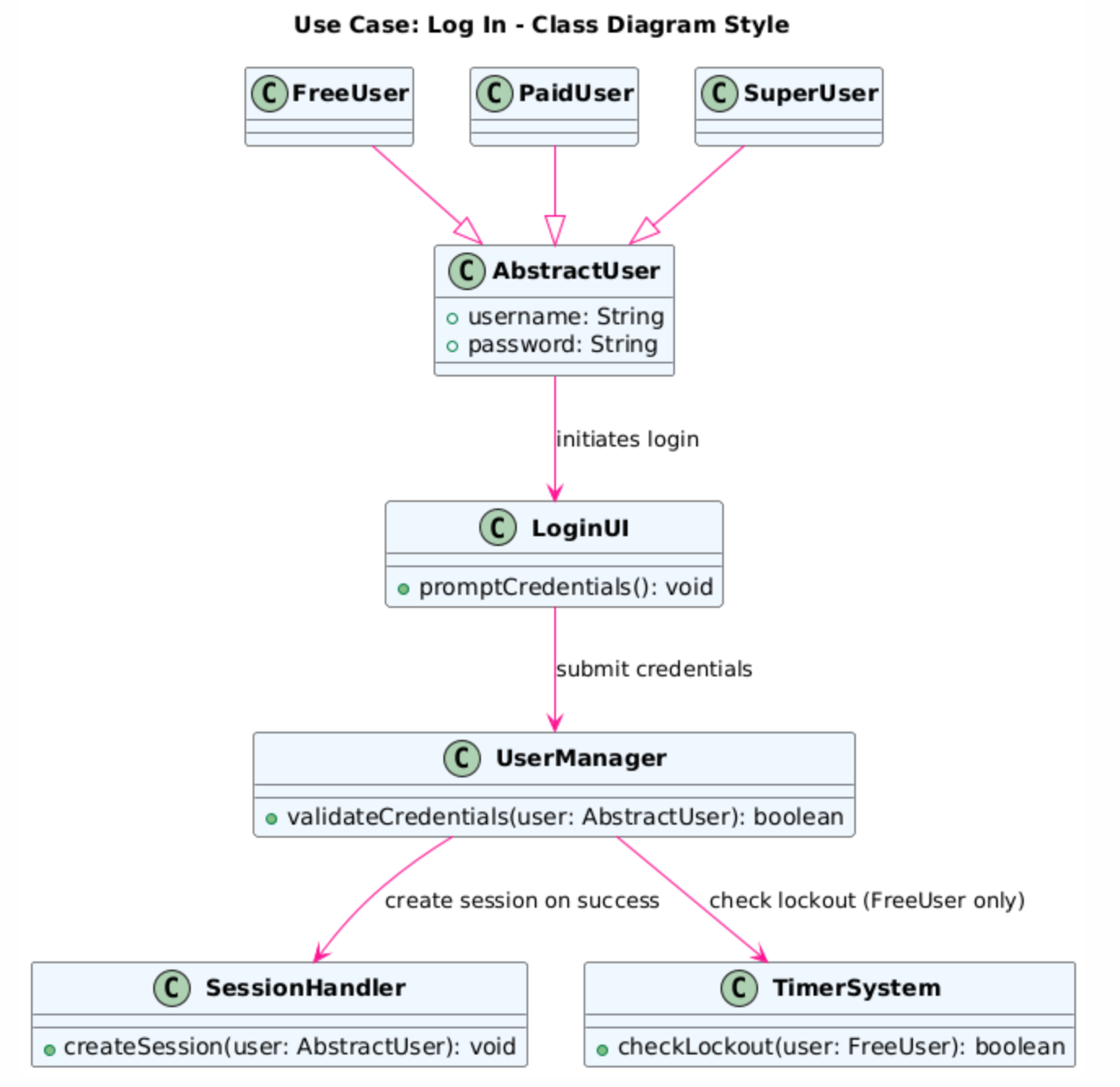


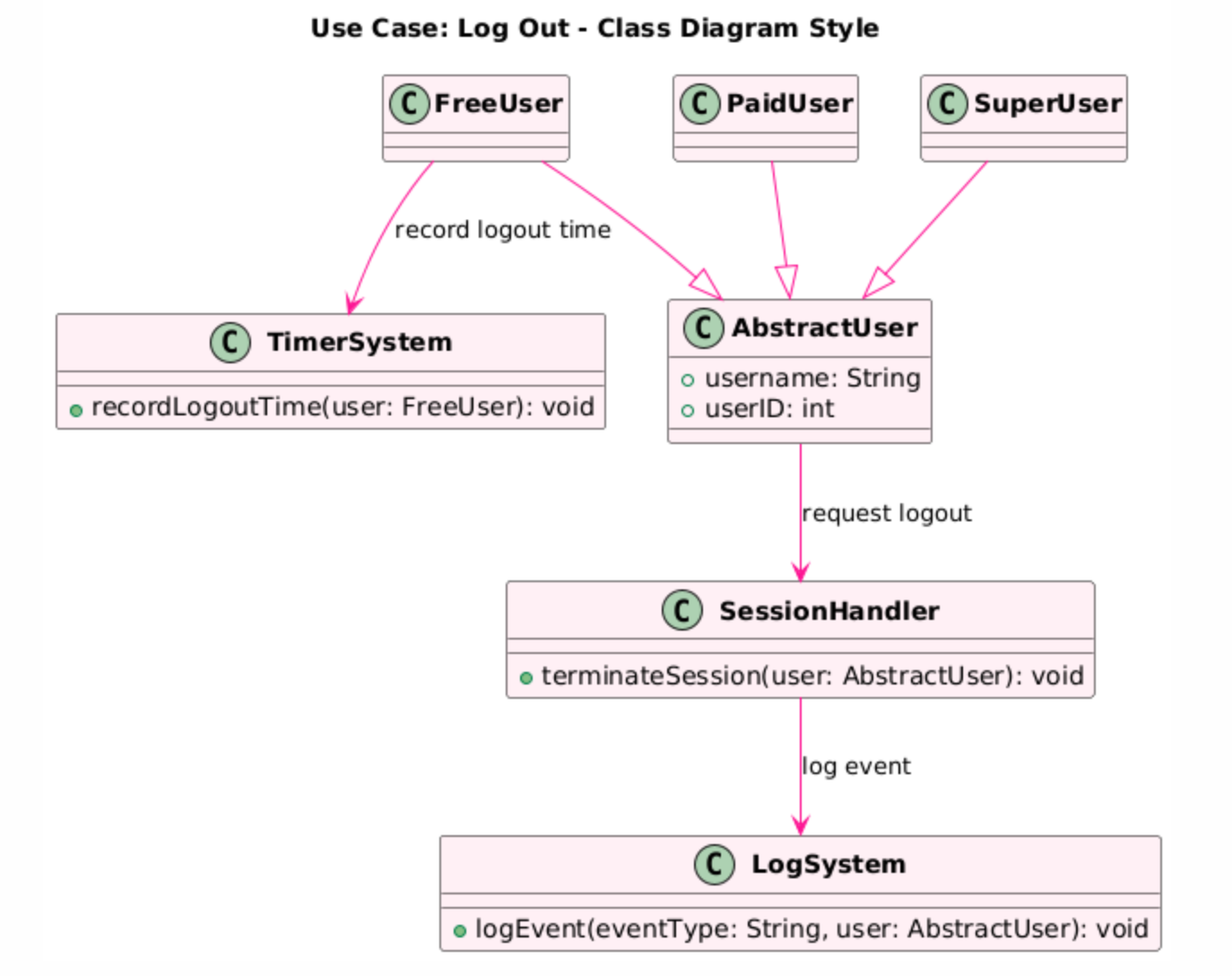


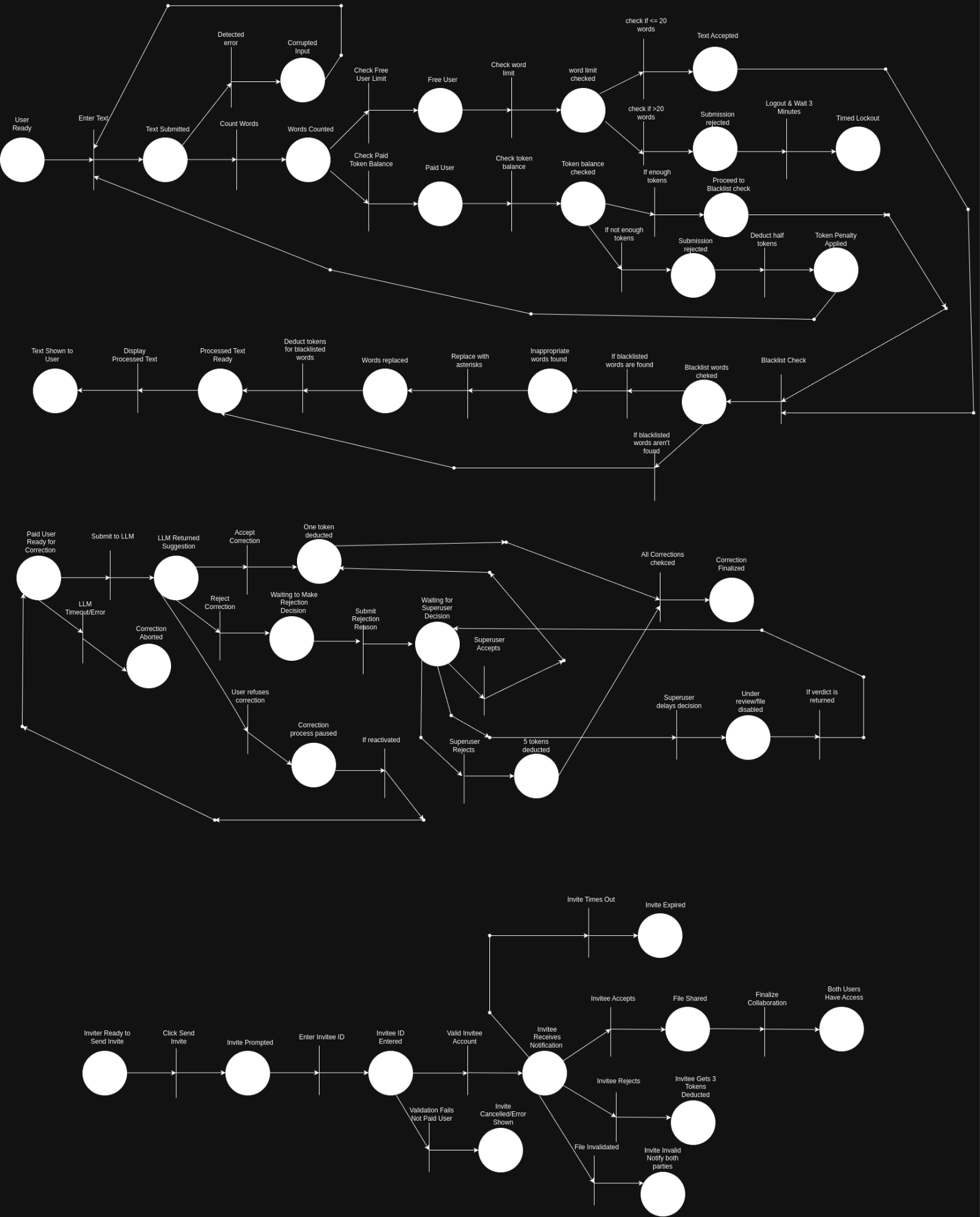


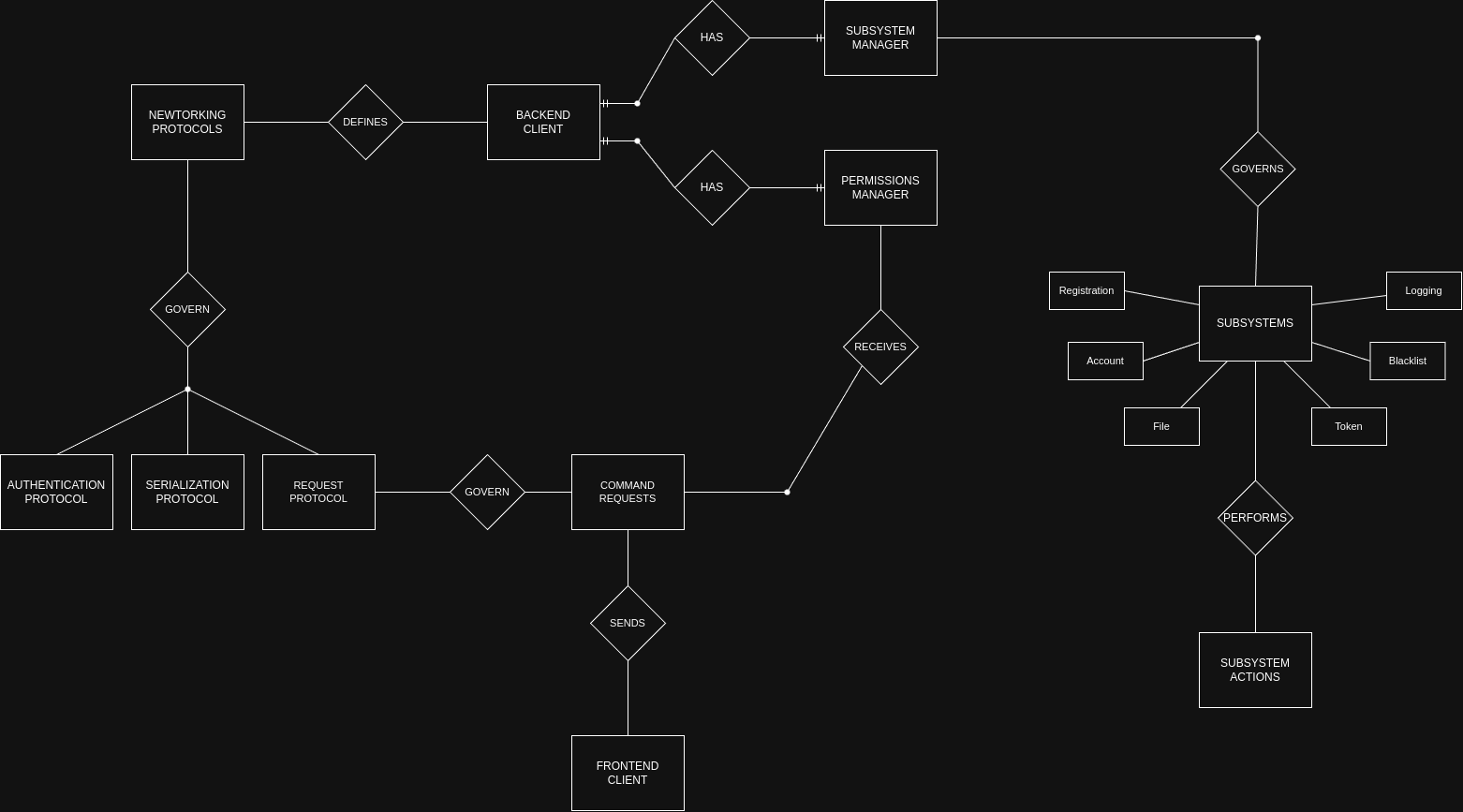


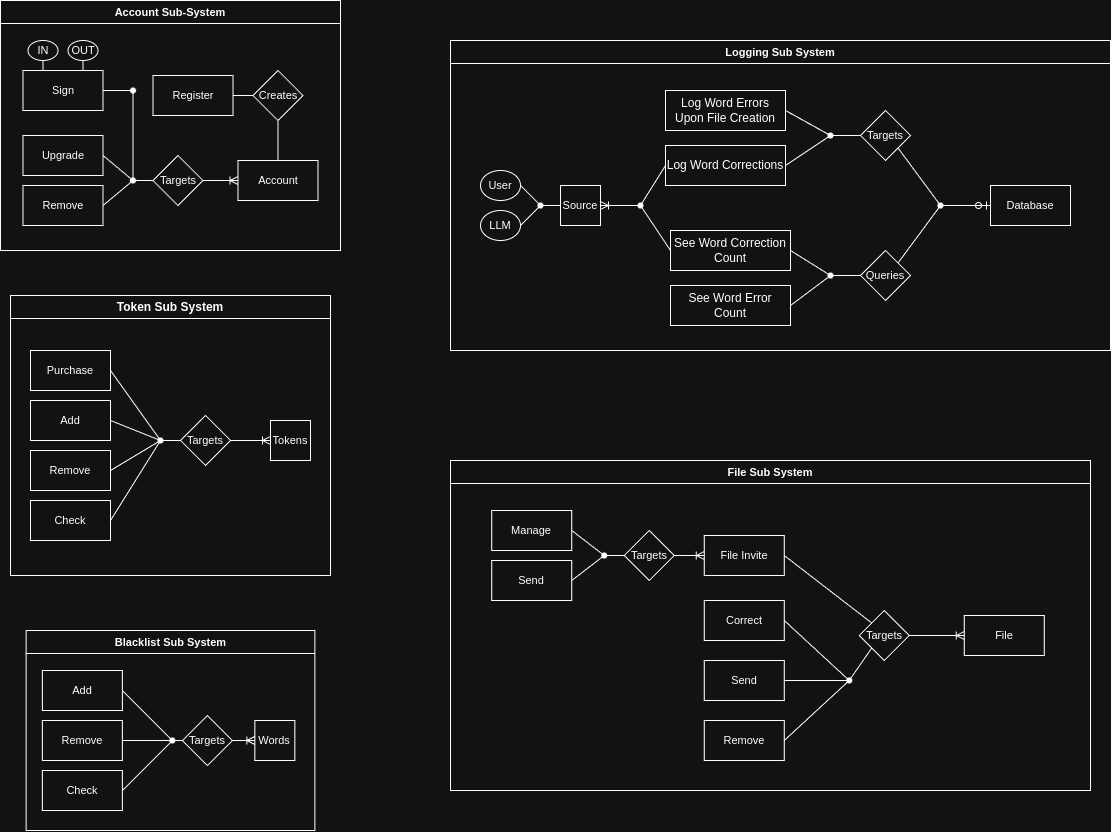


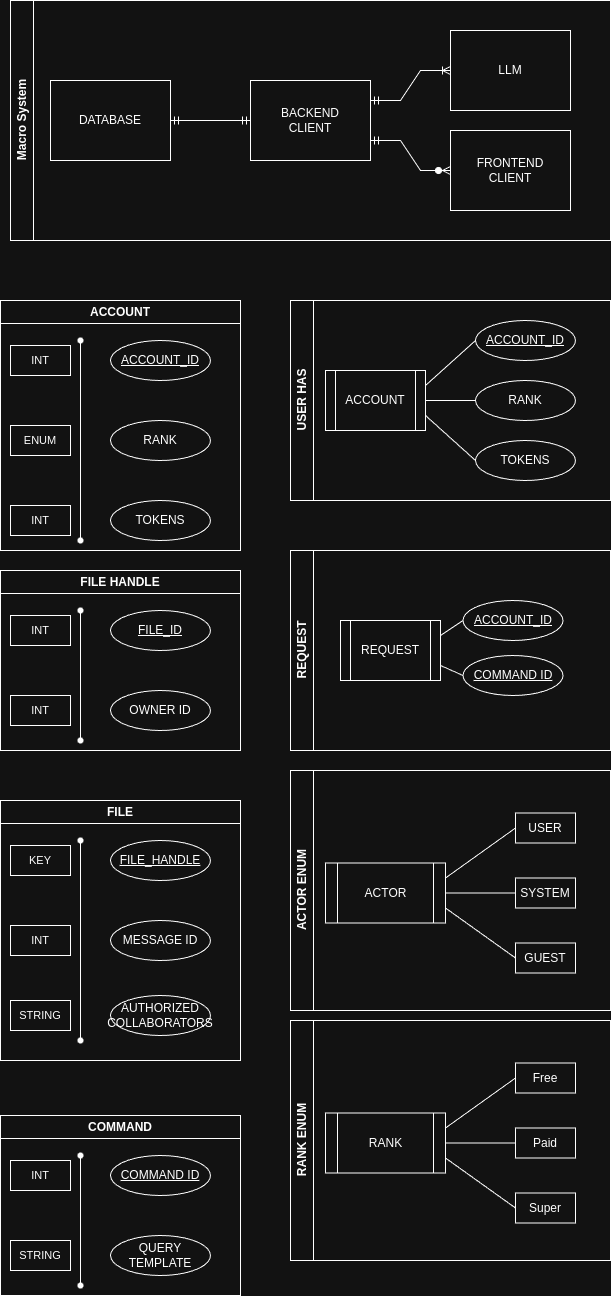




﻿PETRI NETS: Use cases (1), (2), (3)

**6. System E/R Diagram**





**7. System Pseudo-Code**

------------------------------------------------------------------

2. Class and Module Design

------------------------------------------------------------------

---------------------------

2.1 User Management

---------------------------

Data Structures:

- \*\*User Object:\*\*

- Attributes: userID, userType (Free, Paid, Super), availableTokens, loginState, statistics

- Methods: authenticate(), updateTokens(), getStatistics()

Pseudo-code Example for User Authentication:

FUNCTION authenticateUser(username, password)

// Input: username (string), password (string)

// Output: User object if credentials are valid; error message otherwise

userRecord = queryUserDatabase(username)

IF userRecord EXISTS AND userRecord.password == password THEN

user = new User(userRecord)

RETURN user

ELSE

RETURN "Error: Invalid username or password"

ENDIF

END FUNCTION

------------------------------------------------------------------

2.2 Text Submission and Preprocessing

------------------------------------------------------------------

2.2.1 submitText Method

This method processes text submissions from both free and paid users and charges tokens based on word count and blacklist processing.

FUNCTION submitText(text, user)

// Input: text (string), user (User object)

// Output: Processed text object or error message

wordCount = COUNT\_WORDS(text)

// For free users, enforce maximum word count of 20

IF user.userType == "Free" THEN

IF wordCount > 20 THEN

RETURN "Error: Free users can only submit up to 20 words"

ENDIF

// Optionally add a delay (e.g., prevent sign-in for 3 mins) handled elsewhere

ENDIF

// Calculate cost: 1 token per word by default

IF user.availableTokens < wordCount THEN

// Penalize user: deduct half of available tokens

penalty = FLOOR(user.availableTokens / 2)

user.availableTokens = user.availableTokens - penalty

RETURN "Submission failed: Insufficient tokens. Penalty applied."

ENDIF

// Deduct tokens for submission and process blacklist

user.availableTokens = user.availableTokens - wordCount

// Process text for blacklisted words

processedText, blacklistCharge = processBlacklist(text, BLACKLIST)

IF user.availableTokens < blacklistCharge THEN

RETURN "Error: Insufficient tokens to cover blacklist charges"

ENDIF

user.availableTokens = user.availableTokens - blacklistCharge

RETURN processedText // Ready for correction stage

END FUNCTION

2.2.2 processBlacklist Method

Replaces any blacklisted words in the text with asterisks (one per letter) and calculates additional token charges.

FUNCTION processBlacklist(text, blacklist)

// Input: text (string), blacklist (list of banned words)

// Output: processedText (string) with banned words replaced;

// tokenCharge (integer) equal to the length of each banned word replaced

tokenCharge = 0

words = SPLIT(text, " ")

FOR EACH word IN words DO

IF word IN blacklist THEN

length = LENGTH(word)

tokenCharge = tokenCharge + length

// Replace word with asterisks of equal length

word = REPEAT("\*", length)

ENDIF

END FOR

processedText = JOIN(words, " ")

RETURN (processedText, tokenCharge)

END FUNCTION

------------------------------------------------------------------

2.3 Correction Engine Module

------------------------------------------------------------------

The editor supports two types of corrections: self-correction and LLM-based correction.

2.3.1 Self-Correction Method

Users manually edit the text. Tokens are charged at half the number of words that were changed.

FUNCTION selfCorrection(originalText, user, corrections)

// Input: originalText (string), user (User object), corrections (map from index to corrected word)

// Output: correctedText (string) with user changes applied and token deduction applied

words = SPLIT(originalText, " ")

numCorrections = 0

FOR EACH index, newWord IN corrections DO

IF words[index] ≠ newWord THEN

words[index] = newWord

numCorrections = numCorrections + 1

ENDIF

END FOR

// Calculate token cost: half the number of corrected words (rounded up)

tokenCost = CEIL(numCorrections / 2)

IF user.availableTokens < tokenCost THEN

RETURN "Error: Insufficient tokens for self-correction"

ENDIF

user.availableTokens = user.availableTokens - tokenCost

correctedText = JOIN(words, " ")

RETURN correctedText

END FUNCTION

2.3.2 LLM Correction Method

Uses the LLM for suggestions, allows interactive acceptance or rejection of each correction, and adjusts tokens accordingly.

FUNCTION llmCorrection(originalText, user)

// Input: originalText (string), user (User object)

// Output: finalCorrectedText (string) after interactive user decisions, token adjustments applied

// Step 1: Get LLM suggestions (simulate as a list of corrections with index positions)

suggestions = LLM\_CALL(originalText)

// suggestions = list of { index: i, original: word, suggestion: correctedWord }

// Step 2: Highlight suggested corrections

highlightedText = originalText

FOR EACH suggestion IN suggestions DO

highlightedText = HIGHLIGHT\_CORRECTION(highlightedText, suggestion.index, suggestion.suggestion)

END FOR

// Step 3: Interactively process each suggestion with the user

FOR EACH suggestion IN suggestions DO

decision = GET\_USER\_DECISION("Accept correction for word '" + suggestion.original + "'? (yes/no/rejectAll)")

IF decision == "yes" THEN

// Accept correction and charge 1 token

IF user.availableTokens < 1 THEN

RETURN "Error: Insufficient tokens for accepting correction"

ENDIF

user.availableTokens = user.availableTokens - 1

originalText = APPLY\_CORRECTION(originalText, suggestion.index, suggestion.suggestion)

ELSE IF decision == "no" THEN

// Ask if the word should be saved as correct for future submissions

saveWord = GET\_USER\_DECISION("Save '" + suggestion.original + "' as correct? (yes/no)")

IF saveWord == "yes" THEN

UPDATE\_USER\_CORRECT\_WORDS(user, suggestion.original)

ENDIF

ELSE IF decision == "rejectAll" THEN

// Entirely reject the LLM suggestion with a reason

reason = GET\_USER\_INPUT("Provide a reason for rejecting all suggestions:")

// Forward to Super User for evaluation

penalty = superUserEvaluateLLMRejection(reason)

IF user.availableTokens < penalty THEN

RETURN "Error: Insufficient tokens for penalty"

ENDIF

user.availableTokens = user.availableTokens - penalty

// No corrections applied

RETURN originalText

ENDIF

END FOR

finalCorrectedText = originalText

RETURN finalCorrectedText

END FUNCTION

Note:

- LLM\_CALL(), HIGHLIGHT\_CORRECTION(), GET\_USER\_DECISION(), APPLY\_CORRECTION(),

UPDATE\_USER\_CORRECT\_WORDS(), and superUserEvaluateLLMRejection() are helper functions

that abstract the LLM interaction, UI updates, and super user review processes.

------------------------------------------------------------------

2.4 File Handling & Collaboration

------------------------------------------------------------------

2.4.1 Save Text to File

Allows paid users to save their corrected text as a file for a fixed token cost.

FUNCTION saveTextToFile(text, filename, user)

// Input: text (string), filename (string), user (User object)

// Output: Success message or error message

cost = 5

IF user.availableTokens < cost THEN

RETURN "Error: Insufficient tokens to save file"

ENDIF

// Deduct tokens

user.availableTokens = user.availableTokens - cost

// Save file to disk (simulate file saving)

status = FILE\_SYSTEM\_SAVE(filename, text)

IF status == SUCCESS THEN

RETURN "File saved successfully"

ELSE

RETURN "Error: Could not save the file"

ENDIF

END FUNCTION

2.4.2 Invite Collaborator

Shares text files with other paid users. Acceptance adds the collaborator; rejection penalizes the inviter.

FUNCTION inviteCollaborator(inviter, invitee, file)

// Input: inviter (User object), invitee (User object), file (File object)

// Output: Collaboration status message

IF invitee.userType ≠ "Paid" THEN

RETURN "Error: Invitee must be a paid user"

ENDIF

invitationStatus = SEND\_INVITATION(invitee, file)

// Simulate response from invitee

decision = WAIT\_FOR\_RESPONSE(invitee)

IF decision == "accept" THEN

ADD\_COLLABORATOR(file, invitee)

RETURN "Invitation accepted. " + invitee.userID + " is now a collaborator on " + file.filename

ELSE IF decision == "reject" THEN

penalty = 3

IF inviter.availableTokens >= penalty THEN

inviter.availableTokens = inviter.availableTokens - penalty

ELSE

RETURN "Error: Insufficient tokens for penalty deduction"

ENDIF

RETURN "Invitation rejected. Penalty of 3 tokens applied to inviter."

ENDIF

END FUNCTION

------------------------------------------------------------------

2.5 Bonus Token Awarding

------------------------------------------------------------------

If a paid user submits a text (of more than 10 words) with no errors via LLM correction, they receive a bonus.

FUNCTION awardBonusIfNoErrors(correctedText, errorCount, user)

// Input: correctedText (string), errorCount (integer), user (User object)

// Output: Updated user token balance and bonus message

wordCount = COUNT\_WORDS(correctedText)

IF wordCount > 10 AND errorCount == 0 THEN

user.availableTokens = user.availableTokens + 3

RETURN "Bonus awarded: 3 tokens added."

ELSE

RETURN "No bonus awarded."

ENDIF

END FUNCTION

------------------------------------------------------------------

2.6 Statistics and Reporting

------------------------------------------------------------------

Paid users can view all errors, corrections, and system usage statistics.

FUNCTION showStatistics(user)

// Input: user (User object)

// Output: statistics report (string or structured object)

stats = {

"Total Tokens": user.statistics.totalTokensDeducted,

"Used Tokens": user.statistics.usedTokens,

"Available Tokens": user.availableTokens,

"Total Submissions": user.statistics.totalSubmissions,

"Total Corrections": user.statistics.totalCorrections,

"Error Details": user.statistics.errorLog // This can be a list or dictionary of errors and corrections

}

RETURN stats

END FUNCTION

------------------------------------------------------------------

2.7 Complaint and Dispute Management

------------------------------------------------------------------

2.7.1 Submit Complaint

Allows paid users to file complaints regarding collaborators. The complaint is forwarded to a super user for review.

FUNCTION submitComplaint(complainant, accused, disputeDetails, file)

// Input: complainant (User object), accused (User object), disputeDetails (string), file (File object)

// Output: Complaint ID and status message

complaintID = GENERATE\_UNIQUE\_ID()

complaintRecord = {

"ID": complaintID,

"complainant": complainant.userID,

"accused": accused.userID,

"disputeDetails": disputeDetails,

"file": file.filename,

"status": "Pending",

"response": NULL

}

SAVE\_COMPLAINT\_RECORD(complaintRecord)

NOTIFY\_USER(accused, "A complaint has been filed against you. Please log in to respond.")

RETURN "Complaint submitted with ID: " + complaintID

END FUNCTION

2.7.2 Super User Complaint Evaluation

The super user reviews a complaint and applies the appropriate penalty.

FUNCTION evaluateComplaint(complaintRecord, superUser)

// Input: complaintRecord (object), superUser (User object)

// Output: Updated complaintRecord and token adjustments

decision = GET\_SUPERUSER\_DECISION("Is the complaint valid? (yes/no)")

IF decision == "yes" THEN

penaltyDecision = GET\_SUPERUSER\_DECISION("Penalty severity? (mild/strict)")

IF penaltyDecision == "mild" THEN

penalty = 1

ELSE

penalty = 5

ENDIF

DEDUCT\_TOKENS\_FROM\_USER(complaintRecord.accused, penalty)

complaintRecord.status = "Resolved: Accused penalized " + penalty + " tokens"

ELSE

complaintRecord.status = "Resolved: Complaint dismissed"

ENDIF

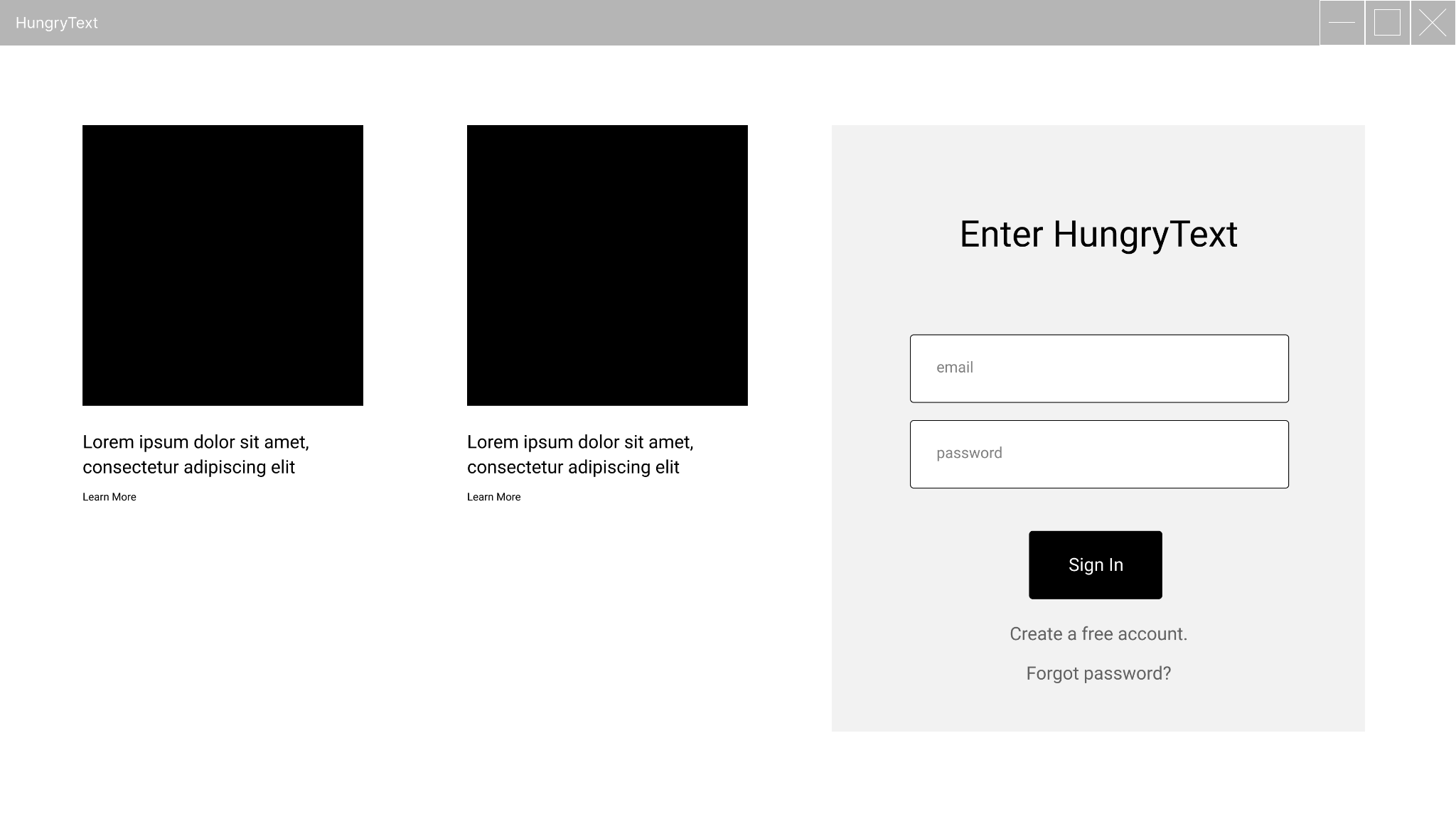
UPDATE\_COMPLAINT\_RECORD(complaintRecord)

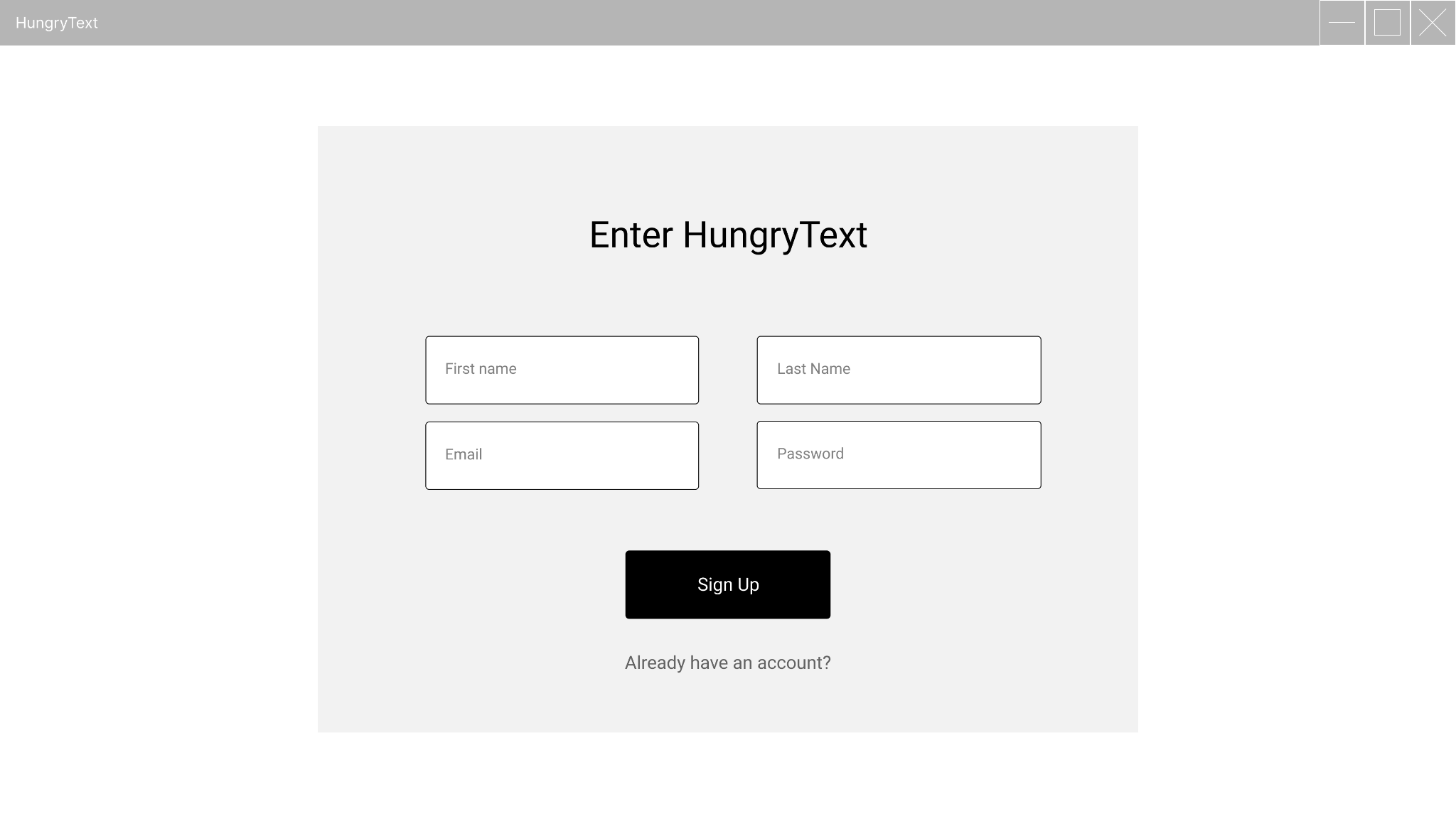
RETURN complaintRecord.status

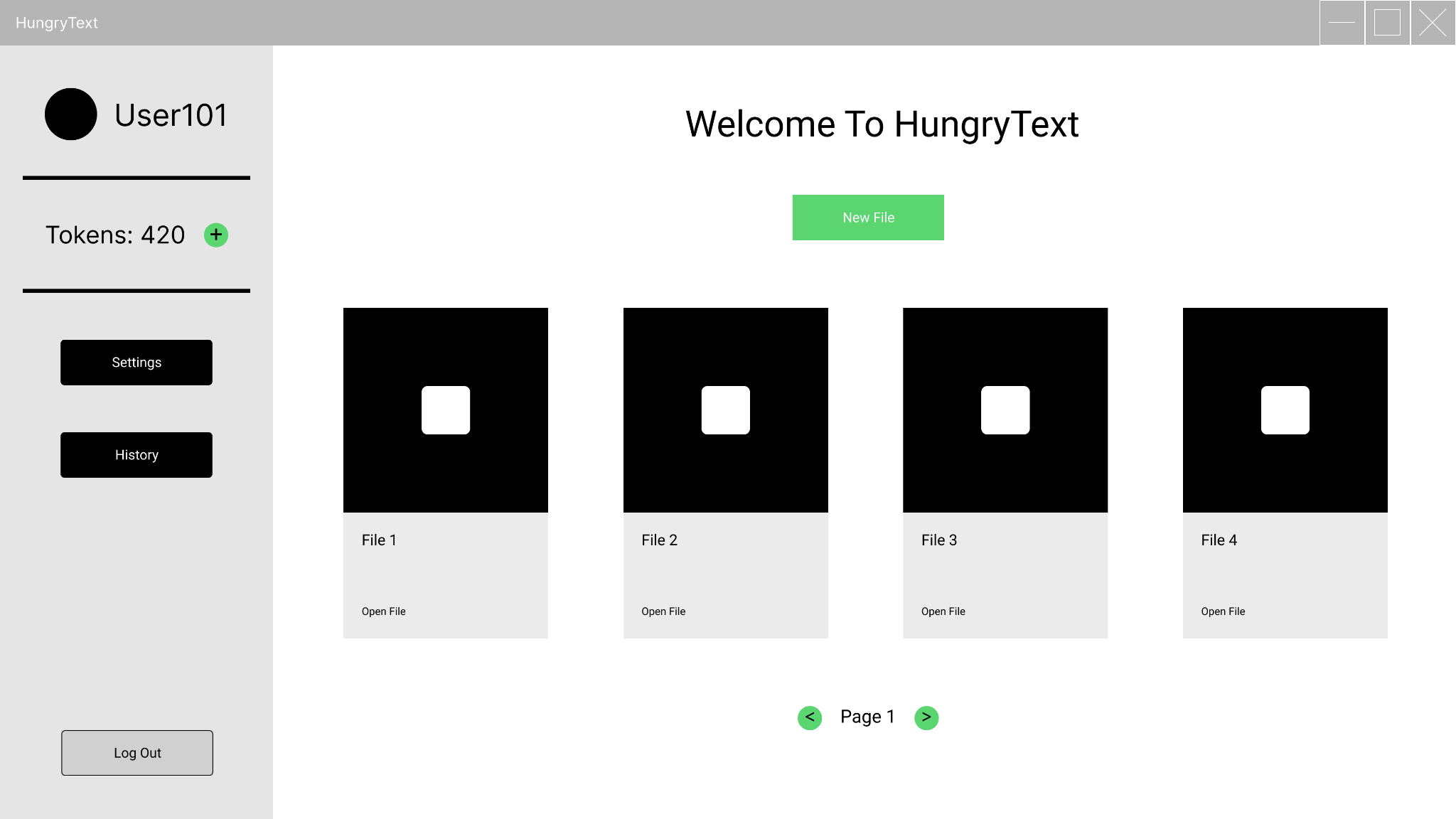
END FUNCTION

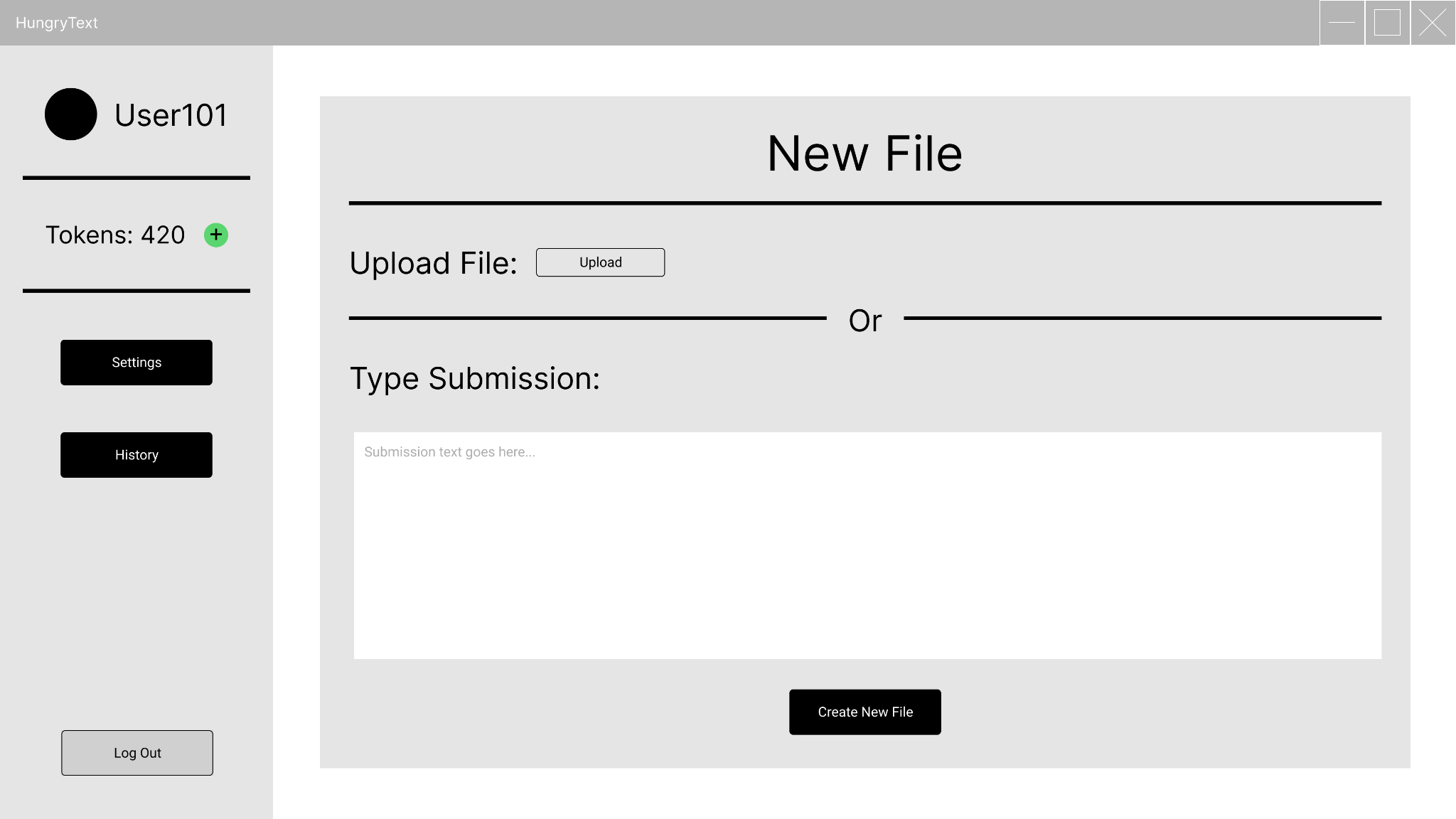
**8. UI Mockups & Demo**

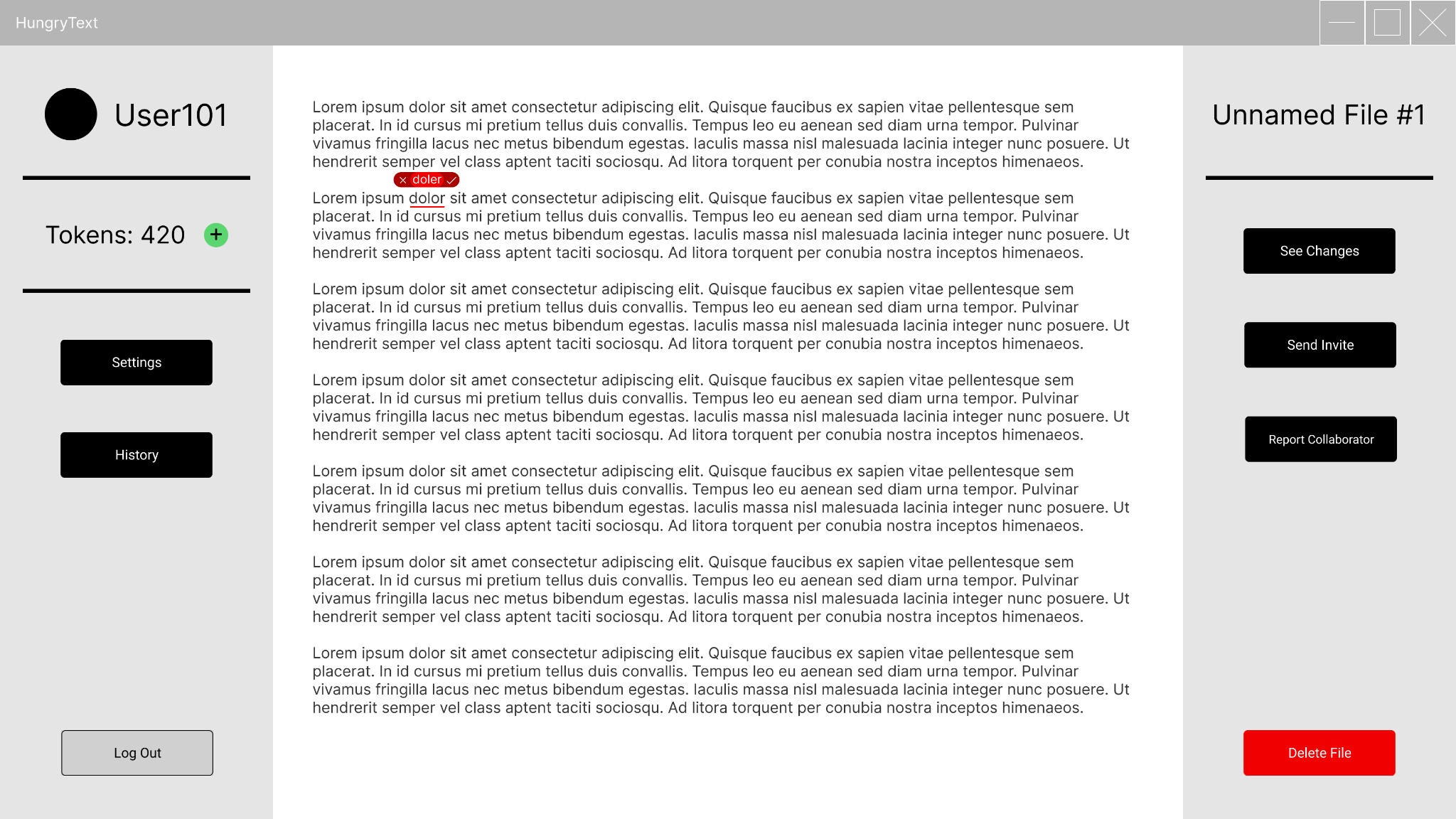
**8.1 UI Mockups**





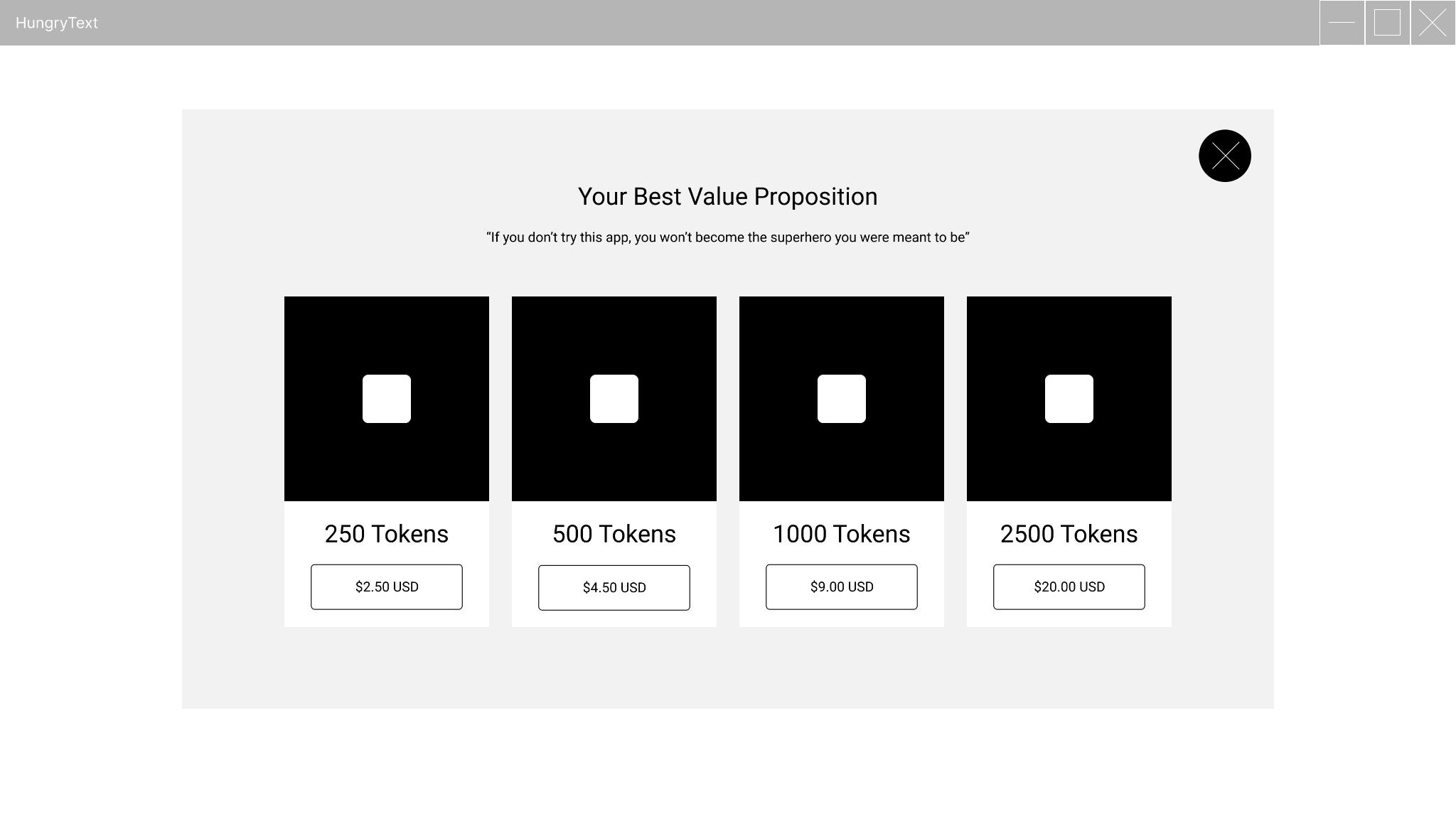












**8.2 Demo**

[Demo Video (YouTube)](https://youtu.be/K1HJn7UPdUo)

**9. Memos & Teamwork**

As a group we discussed how we will plan our project. We spoke about which tech stack to use, making diagrams before implementing code, and deciding whether we should make a web application or a desktop/native application. We also had an in-depth discussion about how we will construct the backend, more specifically how we will handle the implementation of an LLM, whether each user will install the LLM on their machine, or if the LLM will be cloud hosted, or if a group member will host it.

We also had a meeting about revising our tech stack and finally decided to go with Python to do a native application. We talked about which libraries and frameworks to use and chose to go with PyQt6, Flask, PostgreSQL, and Ollama. During one of our meetings we spoke about which member will occupy which role of the project. For example, who’s working on the backend, frontend, LLM, Database.

In one of the meetings, we mentioned that we need to educate ourselves more on the topics we’ll be working on, we were all aware that we lacked experience in a few things. This helped us decide the roles. We also discussed concerns regarding teamwork, like our lack of communication and organization. We mentioned a few times that communication is key and that we must do our best at it.

Organization has come into our discussion when talking about planning and implementing for the project. Another concern we had was our time and availability, with both being limited, we had to adjust accordingly, this makes communication even more important. Another concern is that we all had doubts about our abilities and gave each other confidence to get started. One of our first concerns was being able to understand each other, and we’ve improved greatly at that.

**10. Github Repository**

<https://github.com/TylersHub/AI-TEXT-EDITOR>