| **Phase II: Design Report**  **Team D**  **Authors: Diallo, Alhassana**  **Mamun, Md**  **Moussa, Ezzeldin**  **Ojilere, Lesley**  **Wu, Cuiwen** |
| --- |

**Table of Contents**

[**1. Introduction 3**](#_7ptloyo5s3gq)

[1.1 Collaboration Class Diagram 3](#_iofnmklixpuz)

[**2. Use-Case Scenarios and Diagrams 4**](#_htv3ogvphbmv)

[2.1 Scenarios for each use case: normal AND exceptional scenarios 4](#_uzp7lccl28ki)

[2.2 Sequence class diagram for each use case 8](#_bxfd0xm5gsmb)

[2.3 Petri-Net Diagrams for Selected Use Cases 9](#_adj0tm58oypf)

[**3. E/R Diagram for the entire system 10**](#_dhdx2a16jqe3)

[**4. Detailed Design: Pseudo-code for every Method 12**](#_8t3yuiz47gda)

[4.1 Text Submission and Validation 12](#_u1th41po07h4)

[4.2 Correction Processing (Self-Correction and LLM Correction) 12](#_j1cfayl5kq7t)

[4.3 Blacklist Management (For Super Users) 12](#_i9xndh96dlqe)

[4.4 Complaint Handling 12](#_wtosk0n41fg5)

[**5. System Screens and Prototype 13**](#_uzukgso13qal)

[5.1 Major GUI Screens of the System 13](#_lpmfpg3khqbg)

[5.2 Sample Prototype of Functionality: Text Submission 14](#_yppiww8rhpfw)

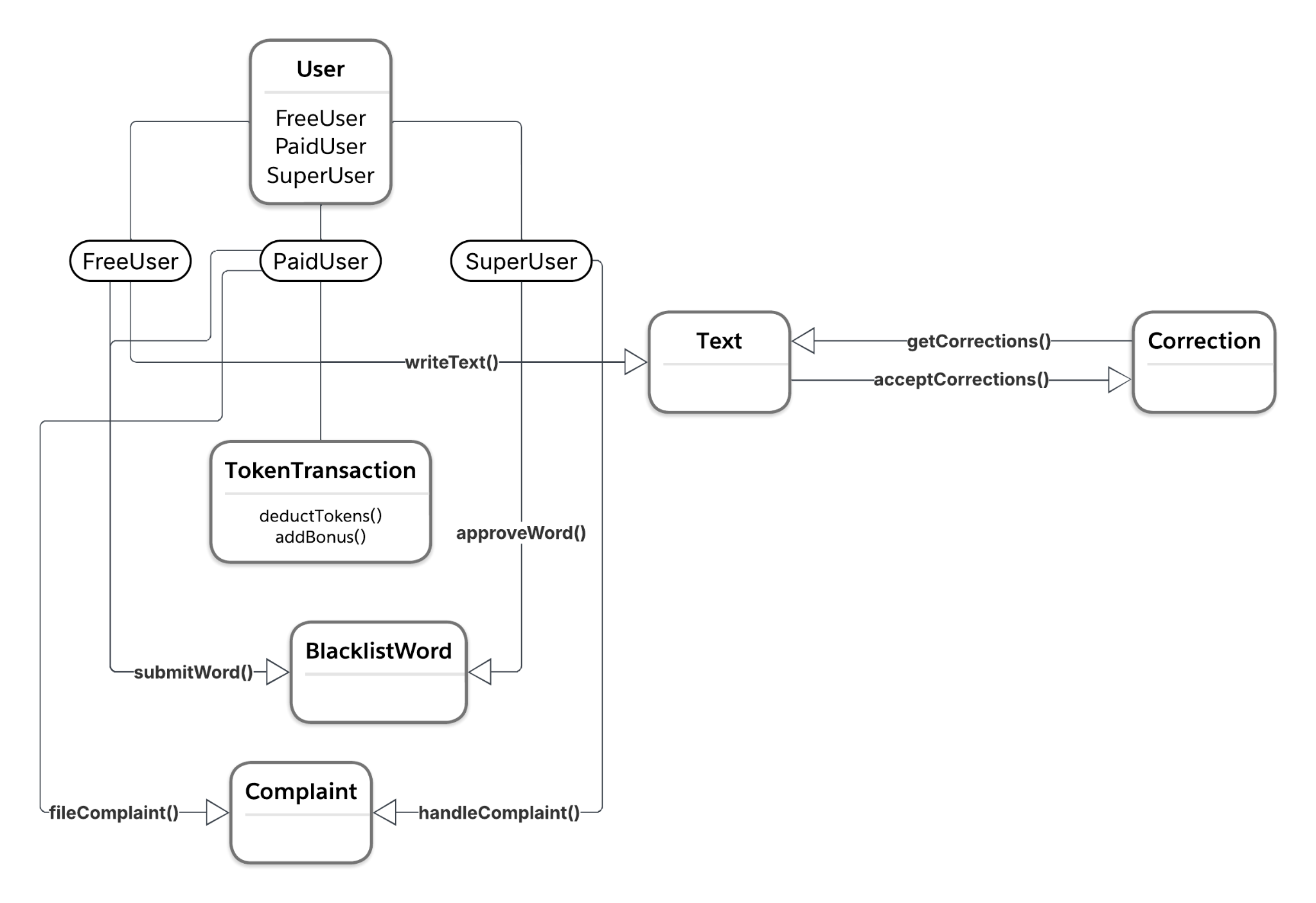
[**6. Group Meeting Memos 15**](#_ptgjmbqvz40o)

[**7. Address of GitHub Repository 15**](#_7ikiobbtl6x5)

# 1. Introduction

This section gives an overview of the system design and introduces the classes. The LLM-Based Cooperative Editor is built to support three user roles (Free, Paid, and Super Users) while providing text correction services using a local free LLM. The core functionalities include text submission, correction (both self- and LLM-assisted), blacklist management, collaboration between paid users, token management, and complaint resolution.

## 1.1 Collaboration Class Diagram



# 2. Use-Case Scenarios and Diagrams

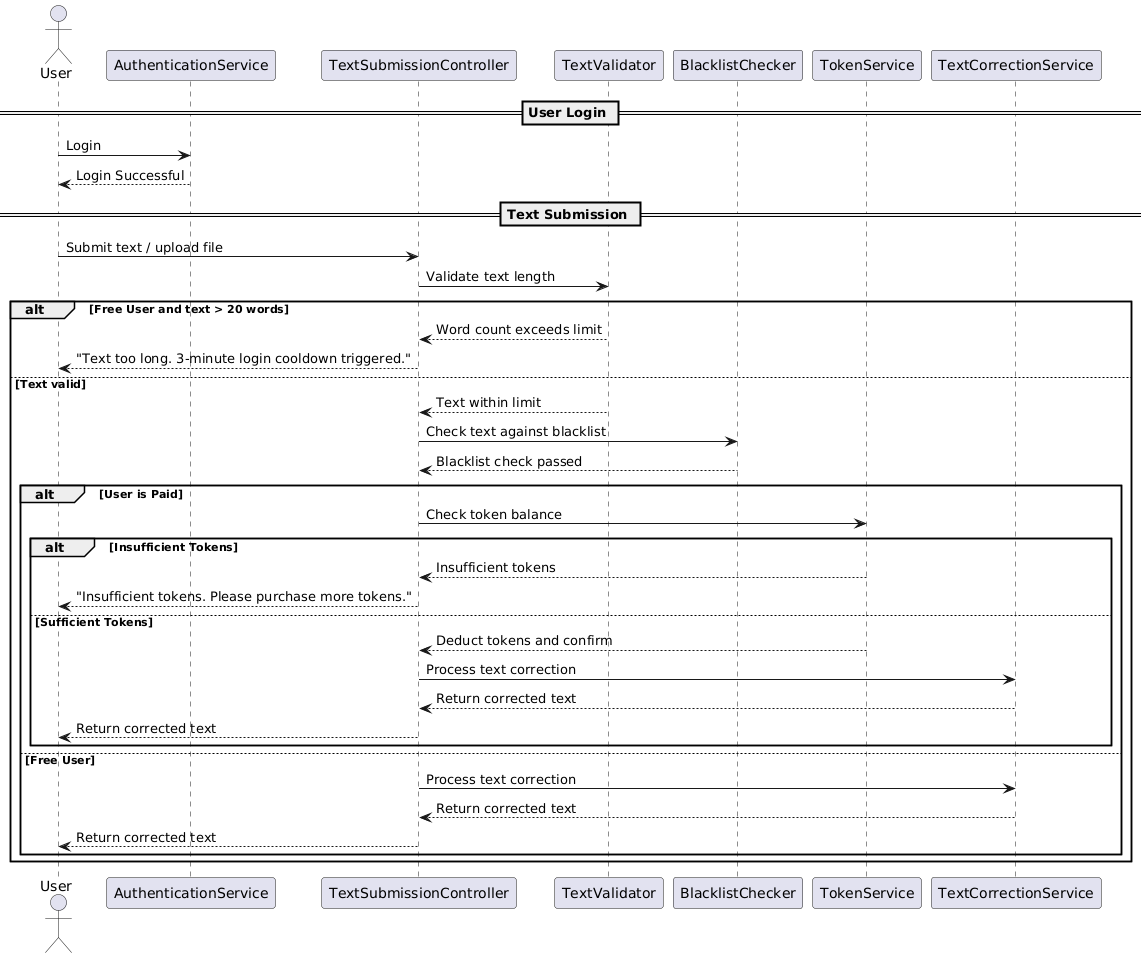
The report includes normal and exceptional flow scenarios for every use case identified in the SRS.

## 2.1 Scenarios for each use case: normal AND exceptional scenarios

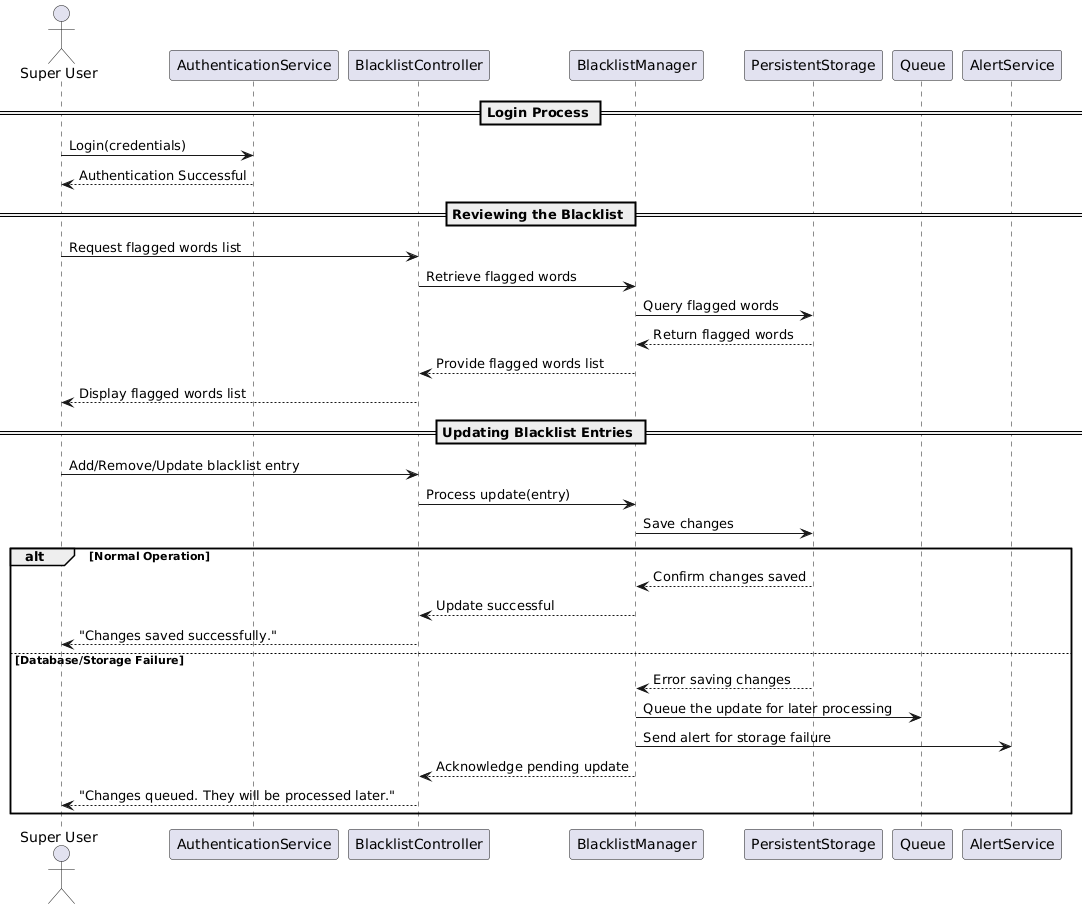
* **Use Case 1: Text Submission**
  + **Normal Scenario (Free User / Paid User):**
    - The user logs in.
    - The user inputs text via the text area or uploads a file.
    - The system validates the text length and checks against the blacklist.
    - For paid users, tokens are deducted based on text length; for free users, a check ensures a maximum of 20 words.
    - The system processes the text (self-correction or LLM correction) and returns the corrected version.
  + **Exceptional Scenario:**
    - **Free User Exceeding Text Limit:**
      * The user enters text longer than 20 words.
      * The system detects the violation, rejects the submission, and triggers a 3-minute login cooldown.
    - **Paid User Lacking Tokens:**
      * The system checks token balance.
      * If there are insufficient tokens, the system interrupts processing and prompts the user to purchase more tokens.
* **Use Case 2: Blacklist Management**
  + **Normal Scenario:**
    - A Super User logs in.
    - The Super User reviews a list of flagged words.
    - The Super User adds, removes, or updates blacklist entries.
    - Changes are saved in the system’s persistent storage.
  + **Exceptional Scenario:**
    - **Database/Storage Failure:**
      * When saving changes, if storage is down, the system queues the changes.
      * An alert is sent to the administrator, and the changes are processed later.
* **Use Case 3: Self-Correction and LLM Correction**
  + **Normal Scenario:**
    - The user submits text.
    - The user selects between self-correction and LLM correction.
    - If LLM correction is selected, the system sends the text to the LLM.
    - The LLM returns highlighted corrections; the user then accepts or modifies the suggestions.
  + **Exceptional Scenario:**
    - **LLM Service Unavailable:**
      * The user selects LLM correction.
      * The system detects that the LLM service is down.
      * The system defaults to enabling self-correction and notifies the user of the service’s unavailability.
* **Use Case 4: Collaboration**
  + **PaidUser invites collaborator**
    - **Normal Scenario:**
      * A paid User logs into their account.
      * Opens an existing document or creates a new one.
      * Click on “Invite Collaborator.”
      * Enter the username or email of another PaidUser.
      * The system verifies the invitee’s status: Must be a valid and active PaidUser.
      * System sends an invitation to the invitee.
      * Confirmation message: “Collaboration invitation sent successfully
    - **Exceptional Scenario:**
      * Invitee is a freeUser or does not exist. → The system displays, Only paid users can be invited.”
      * Invitee already has a pending or active session. → System blocks the invite and shows, User is currently unavailable for collaboration.”
      * Documents are locked or unsaved. → System prompts the user to “ Please save the document before initiating collaboration.”
  + **PaidUser accepts/rejects invitation**
    - **Normal Scenario:**
      * The invitee receives an in-app or email notification.
      * Opens the notification and reviews the collaboration request.
      * Clicks “Accept” to join the editing session.
      * Enters the shared editing interface with real-time document updates.
      * Session is logged, and tokens, if needed, are shared accordingly.
    - **Exceptional Scenario:**
      * Invitee clicks “Reject”: → System records rejection and notifies the sender.
      * Invitees do not respond within 10 minutes: → Invitation expires, and sender is notified: ”Invitation timed out.”
      * Invitee shows: “This collaboration is no longer available.”
* **Use Case 5: Complaint Handling**
  + **PaidUser files complaint**
    - **Normal Scenario:**
      * PaidUser logs into the system.
      * Navigate to the “Report Issue” or “File Complaint” section.
      * Select users involved in the complaint.
      * Provides a written description of the issue.
      * Submit the complaint form.
      * System logs the complaint, sets status to open, and notifies the SuperUser.
    - **Exceptional Scenario:**
      * Complaint field left empty: → System blocks submission with: “Please provide a reason for your complaint.”
      * User attempts to file multiple complaints for the same issue: → System displays. “You’ve already reported this user. Please wait for a resolution.”
      * System error while submitting: → System shows, “Unable to file complaint at this time. Please try again later.”
  + **SuperUser handles complaint**
    - **Normal Scenario:**
      * SuperUser logs into their dashboard.
      * The user views all pending complaints with user details and timestamps.
      * Opens a specific complaint and investigates:
        + Reviews document history, collaboration logs, and chat transcripts
      * Makes a decision:
        + Dismiss, Warn, Suspend, or Terminate the accused user.
      * Updates the complaint status and adds resolution notes.
      * System notifies both parties about the outcome.
    - **Exceptional Scenario:**
      * Complaint: missing context or logs unavailable. → System marks complaint as “ inconclusive”; SuperUser sends follow-up request.
      * Decision fails to save due to backend error: → Error message: “Could not update complaint. Try again later.”
      * Superuser delays action over the 72 hours: → System auto-sends reminders and escalates flagged complaints.
* **Use Case 6: Token Transactions**
  + **PaidUser purchases new tokens**
    - **Normal Scenario:**
      * User logs into their dashboard.
      * Clicks on “Buy Tokens.”
      * Selects a token package, for example (50, 100, 250).
      * Enters payment details and confirms.
      * Payment is processed via secure API.
      * Tokens are added to the user account.
      * Confirmation: “Tokens successfully added.”
    - **Exceptional Scenario:**
      * Invalid payment details or declined card: → System shows, “Transaction failed. Please check your payment info.”
      * Payment API timeout or network issue: → System retries or shows “Connection issue. Try again later.”
      * Tokens not added after successful payment: → System logs the issue and flags it for admin review.
  + **PaidUser receives bonus tokens**
    - **Normal Scenario:**
      * PaidUser submits a corrected text.
      * System analyzes the submission.
        + Check that the exit >= 10 words
        + Verifies there are no errors
      * If both conditions are met, the system adds +3 tokens to the user’s account.
      * Confirmation: “You received 3 bonus tokens for a flawless submission!
    - **Exceptional Scenario:** 
      * Submission <10 words: → No bonus granted. Message: “Bonus applies to 10+ word submissions only.”
      * LLM detects remaining issues: → Bonus not awarded. User is prompted to review corrections.
      * Bonus fails to register due to system error: → System logs the issue and notifies support: “Bonus token transition error.”

## 2.2 Sequence class diagram for each use case

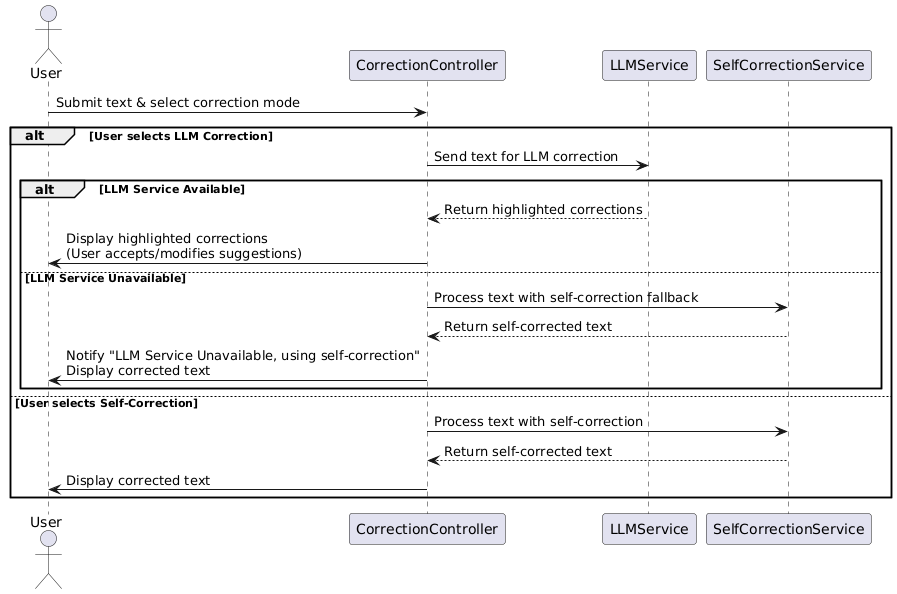
**Use Case 1: Text Submission:**



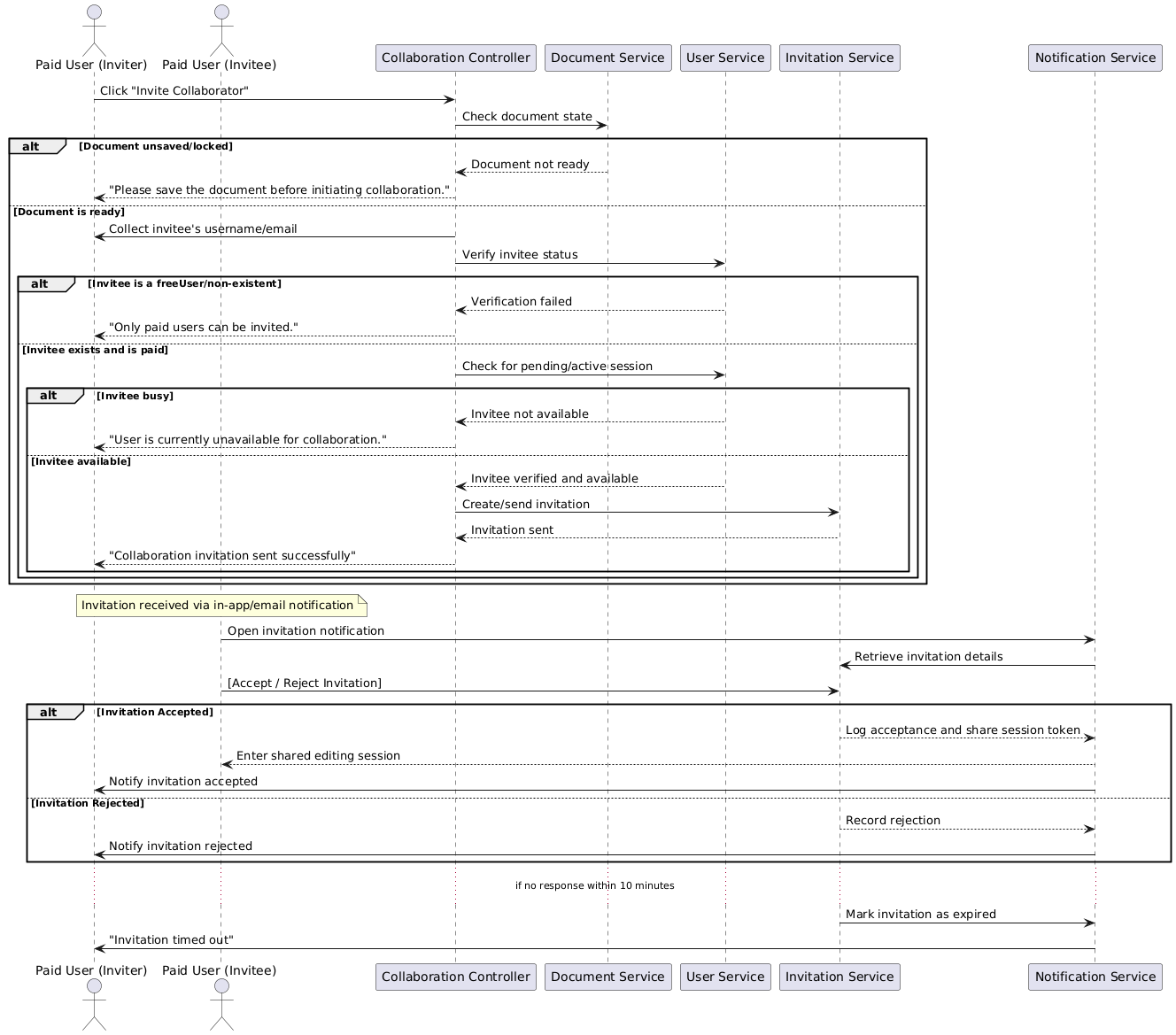
**Use Case 2: Blacklist Management**



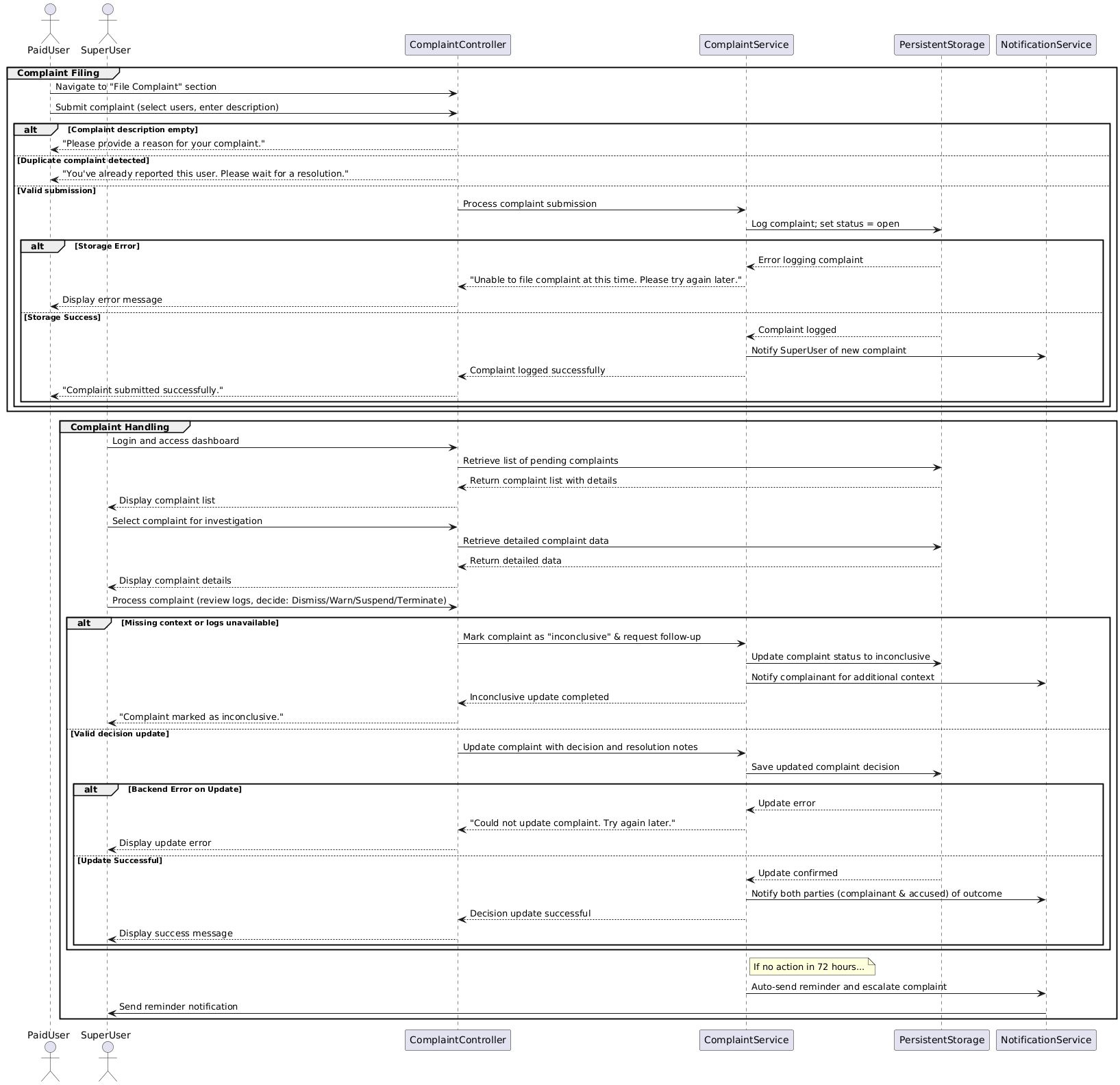
**Use Case 3: Self-Correction and LLM Correction:**



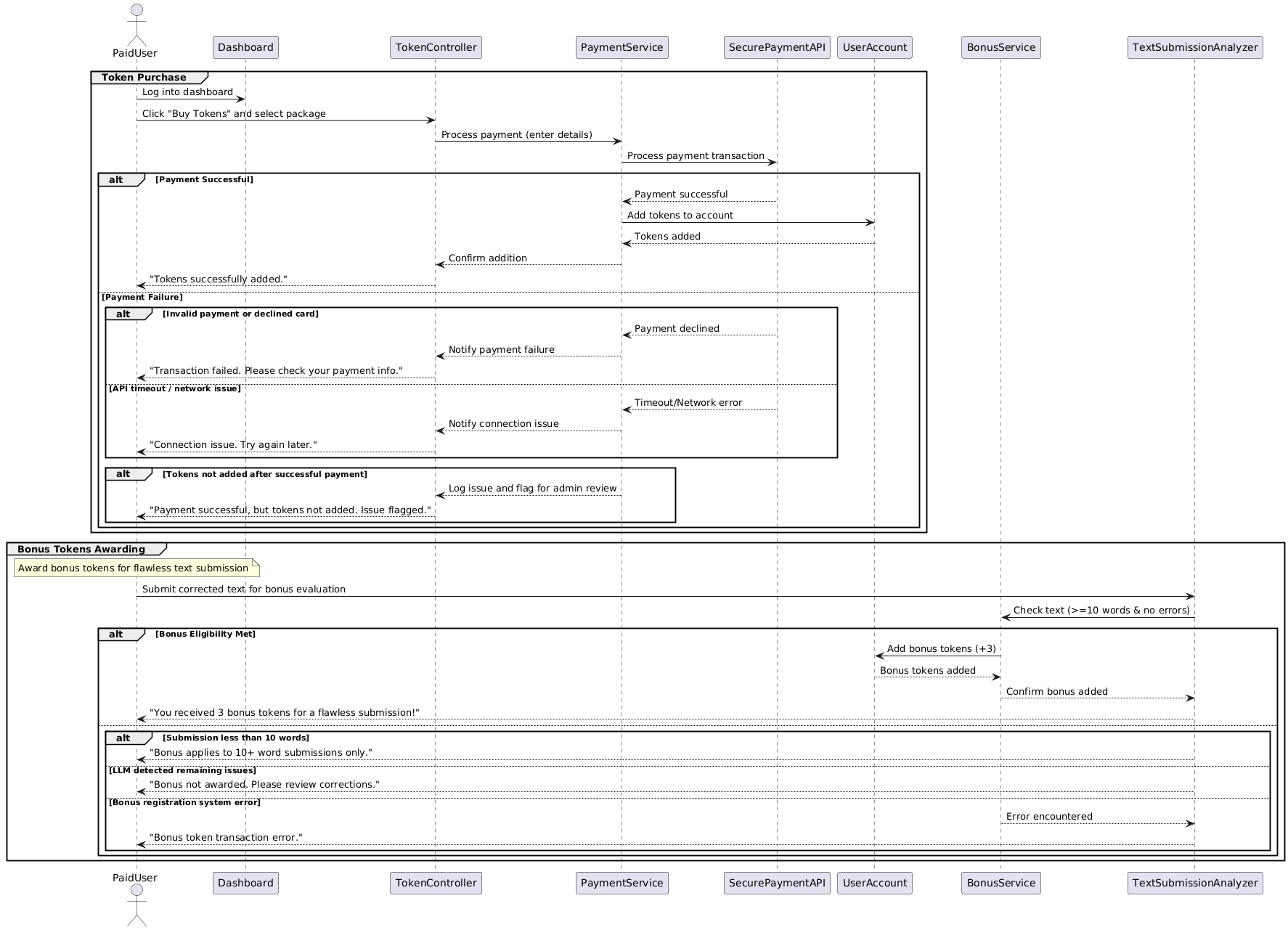
**Use Case 4: Collaboration:**

****

**Use Case 5: Complaint Handling:**

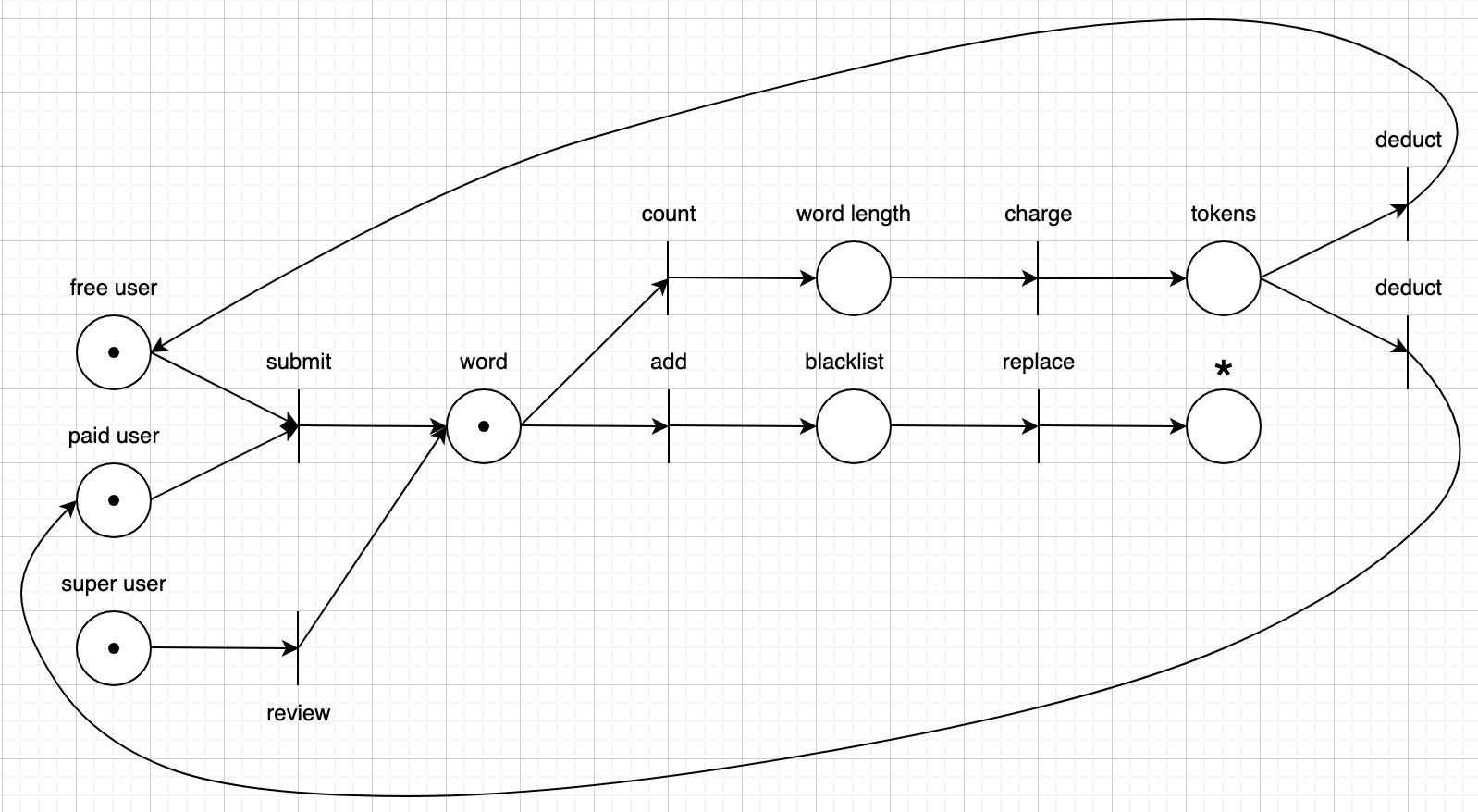


**Use Case 6: Token Transactions:**

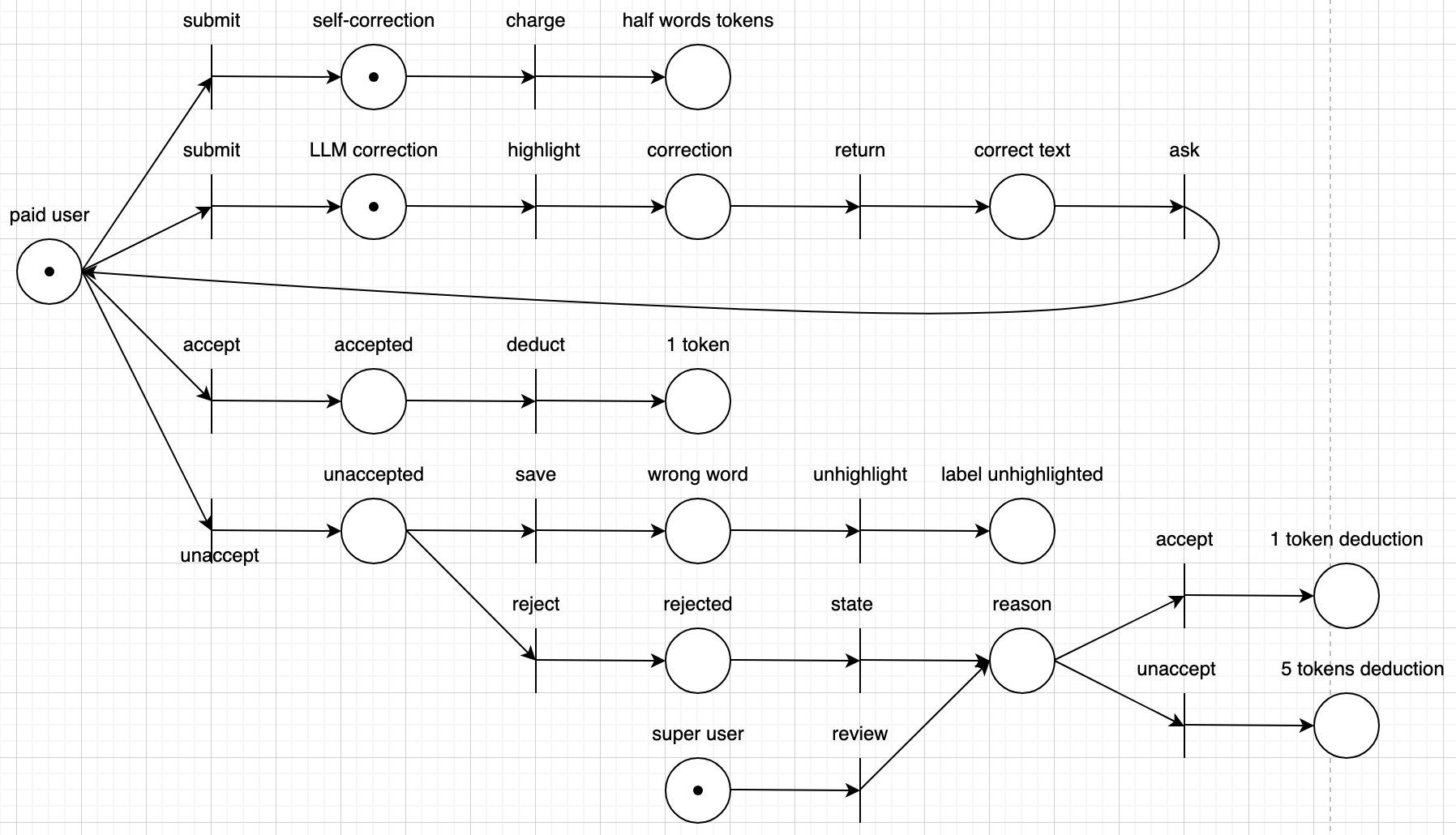


## 2.3 Petri-Net Diagrams for Selected Use Cases

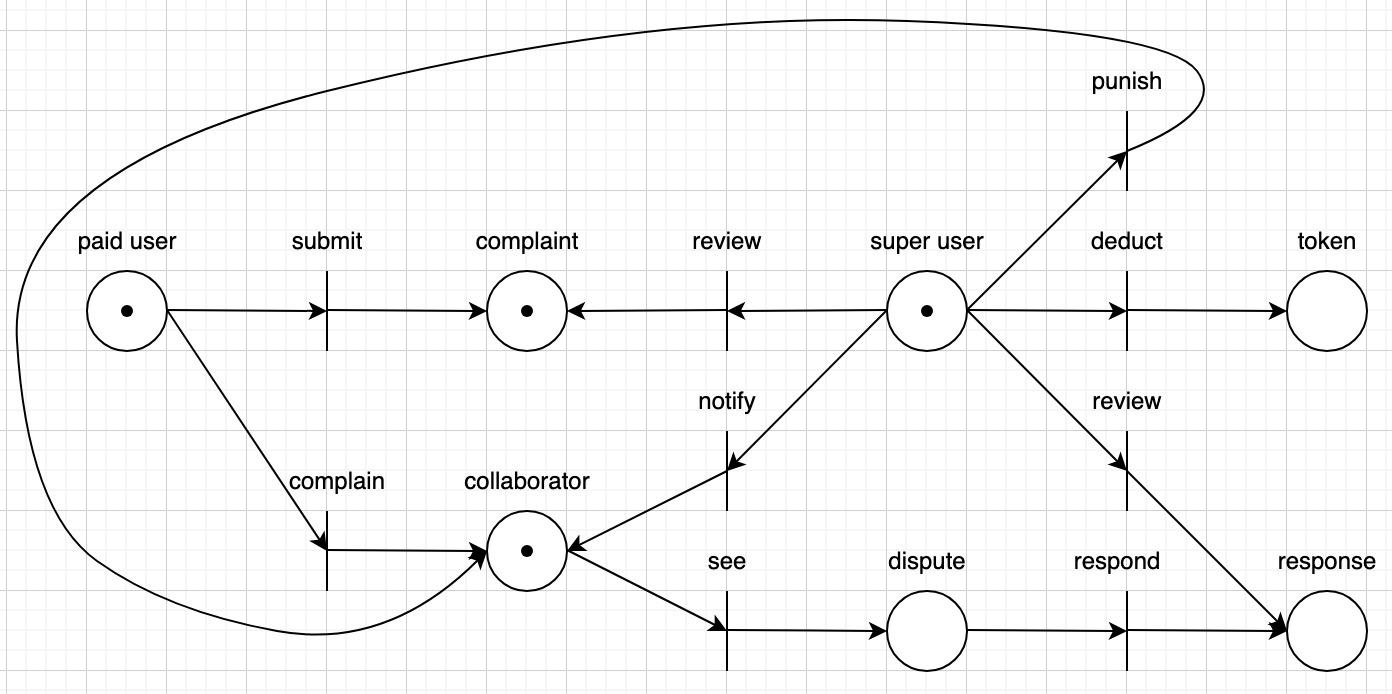
**Use Case 2: Blacklist Management**



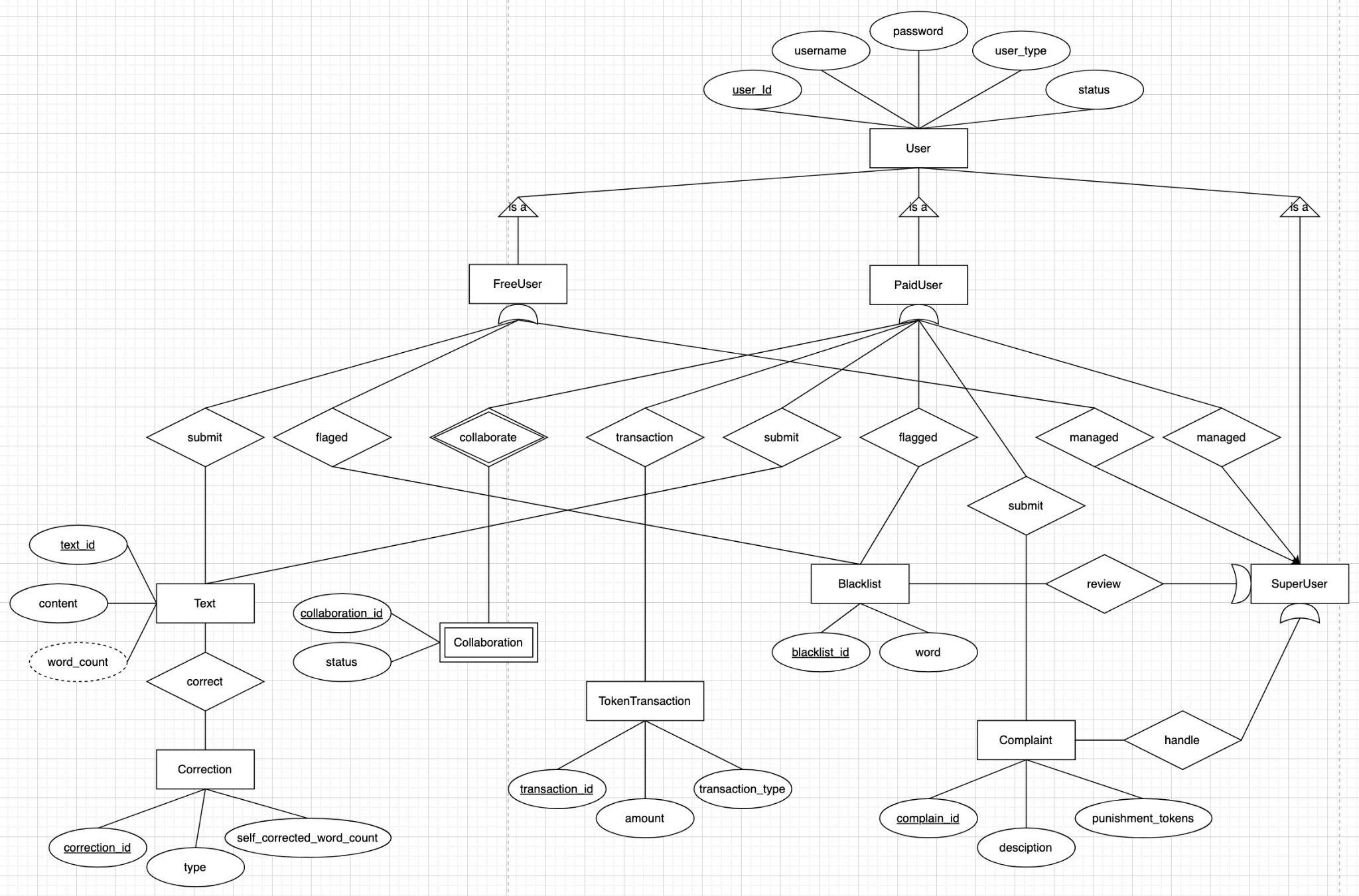
**Use Case 3: Self-Correction and LLM Correction:**

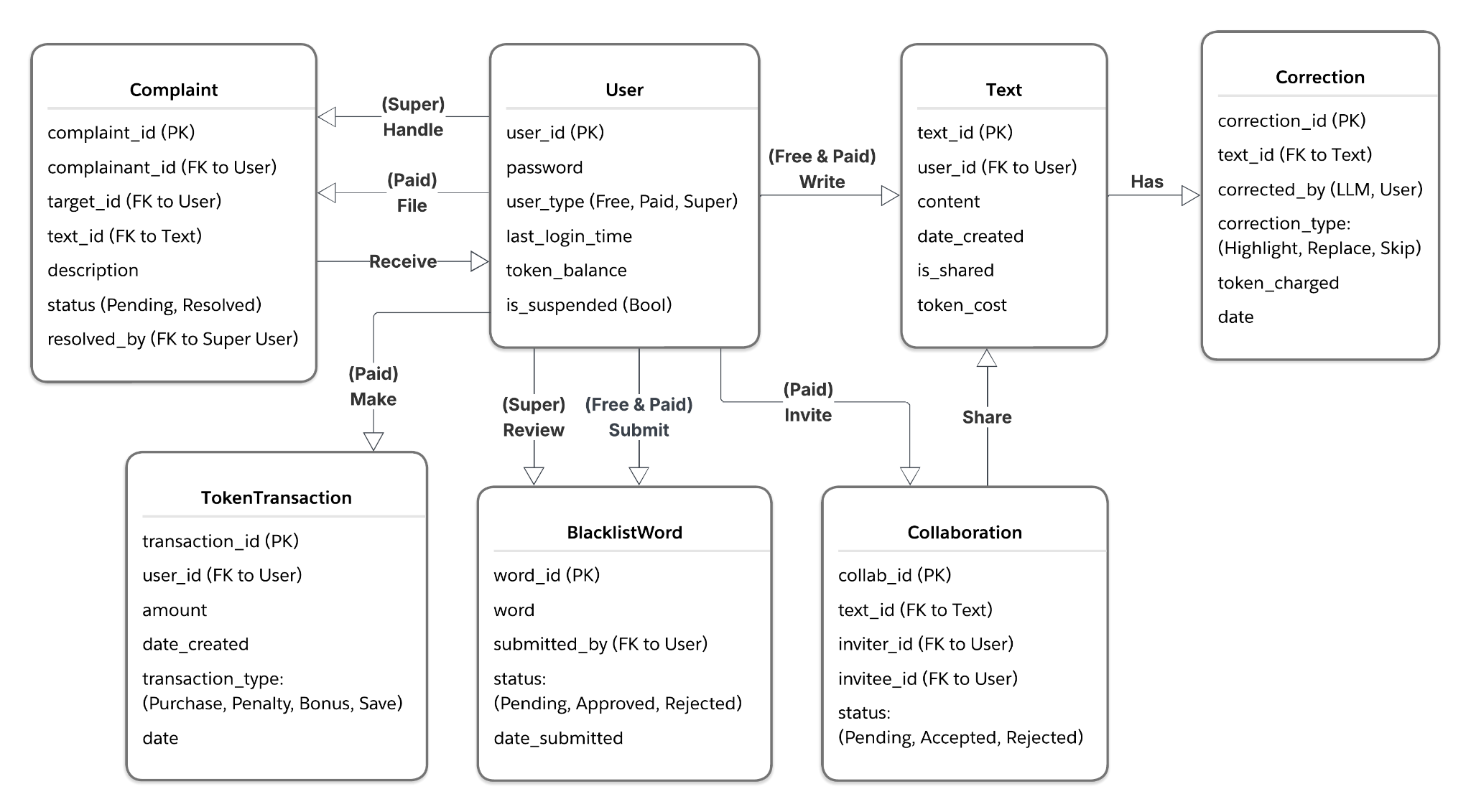


**Use Case 5: Complaint Handling:**



# 3. E/R Diagram for the entire system



****

# 4. Detailed Design: Pseudo-code for every Methods

## 4.1 Text Submission and Validation

function submitText(user, text):

// Preconditions: user is logged in; text is non-empty.

if not user.isLoggedIn():

return "Error: User not logged in"

// Validate text length and blacklist compliance

if user.role == "Free" and wordCount(text) > 20:

triggerCooldown(user, duration=3 minutes)

return "Submission rejected: Exceeds 20-word limit. Cooldown initiated."

if containsBlacklistedWord(text):

return "Submission rejected: Text contains disallowed words."

// For paid users, check token balance

if user.role == "Paid":

tokensRequired = calculateTokenCost(text)

if user.tokenBalance < tokensRequired:

return "Error: Insufficient tokens. Please purchase more tokens."

user.tokenBalance -= tokensRequired

logTransaction(user, tokensRequired, "Deduction")

// Process text correction (self or LLM-based depending on user selection)

result = processCorrection(text, user.selectedCorrectionMode)

return result

## 4.2 Correction Processing (Self-Correction and LLM Correction)

function processCorrection(text, mode):

if mode == "LLM":

if not isLLMServiceAvailable():

notifyUser("LLM service unavailable. Switching to self-correction.")

mode = "Self"

if mode == "LLM":

correctedText = LLMCorrect(text)

else if mode == "Self":

correctedText = selfCorrect(text)

else:

return { status: "Error", correctedText: "Invalid correction mode" }

return { status: "Success", correctedText: correctedText }

## 4.3 Blacklist Management (For Super Users)

function manageBlacklist(action, word, currentList):

if action == "Add":

if word not in currentList:

currentList.add(word)

return "Word added to blacklist."

else if action == "Remove":

if word in currentList:

currentList.remove(word)

return "Word removed from blacklist."

## 4.4 Complaint Handling

function submitComplaint(complainant, againstUser, description):

if description is empty or not valid(description):

return "Error: Complaint description is invalid."

complaint = new Complaint(complainantID = complainant.id, againstUserID = againstUser.id, description = description, status = "Open")

saveComplaint(complaint)

notifySuperUser(complaint)

return "Complaint submitted successfully."

# 5. System Screens and Prototype

## 5.1 Major GUI Screens of the System

The system’s user interface is designed with usability and responsiveness in mind.

**Login Screen:**

* Fields for username and password
* Role indicator (Free/Paid/Super) upon successful login
* Navigation to registration and password recovery

**Text Submission Screen:**

* Text area for manual input and file upload functionality
* Visual indicators for word count
* Correction method selection (Self vs. LLM)
* Submission button and status display area

**Correction Result Screen:**

* Display of original and corrected texts side-by-side
* Option to accept or further edit the highlighted corrections

**Collaboration Interface:**

* Real-time shared editing area with participant indicators
* Chat or comment sidebar for coordination

**Blacklist Management Screen (Super Users):**

* List of flagged words
* Controls to add, remove, or modify entries
* Notification area for pending changes (in case of storage issues)

**Complaint Handling Screen:**

* Form for complaint submission
* Listing of past complaints with status updates

## 5.2 Sample Prototype of Functionality: Text Submission

For a sample prototype, we can develop a text submission and correction screen with the following description:

* Upon login, the user accesses the “Submit Text” tab.
* A real-time word counter notifies the user if the text approaches or exceeds the limit (for Free Users).
* The user selects “LLM Correction” from a toggle.
* Upon submission, the system immediately performs validations (text length, blacklist), and if successful, displays the corrected text in an adjacent panel.
* If the LLM service fails, a message appears and the system reverts to self-correction mode.

# 6. Group Meeting Memos

* **Meeting 1**
  + **Date:** 03/15 - 03/18
  + **Agenda:** Understanding the objectives of completing the software requirement specification (SRS) report, delegating work
  + **Outcome:** Completed SRS report.
* **Meeting 2**
  + **Date:** 04/11 - 04/13
  + **Agenda:** Understanding the objectives of completing the design report, delegating work
  + **Outcome:** Completed design report.

# 7. Address of GitHub Repository

<https://github.com/WenAlgo/An-LLM-based-cooperative-editor>