Team H

LLM Editor
Design Report
For Application

Version <1.0>

Revision History

Date	Version	Description	Author
<13/04/25>	<1.0>	Second Phase Report	Khandaker, Fardin Ismam Saraf, Fardin Ferdous Uddin, Raida Z Yeung, Tak Kit

Table of Contents

1. Introduction 4

2. Use cases 4
   - 2.1 Scenarios for each use case 5
   - 2.2 Class diagram for each use case 9
   - 2.3 Petri-net for selected use cases 15

3. E/R diagram for system 17

4. Detailed design 17

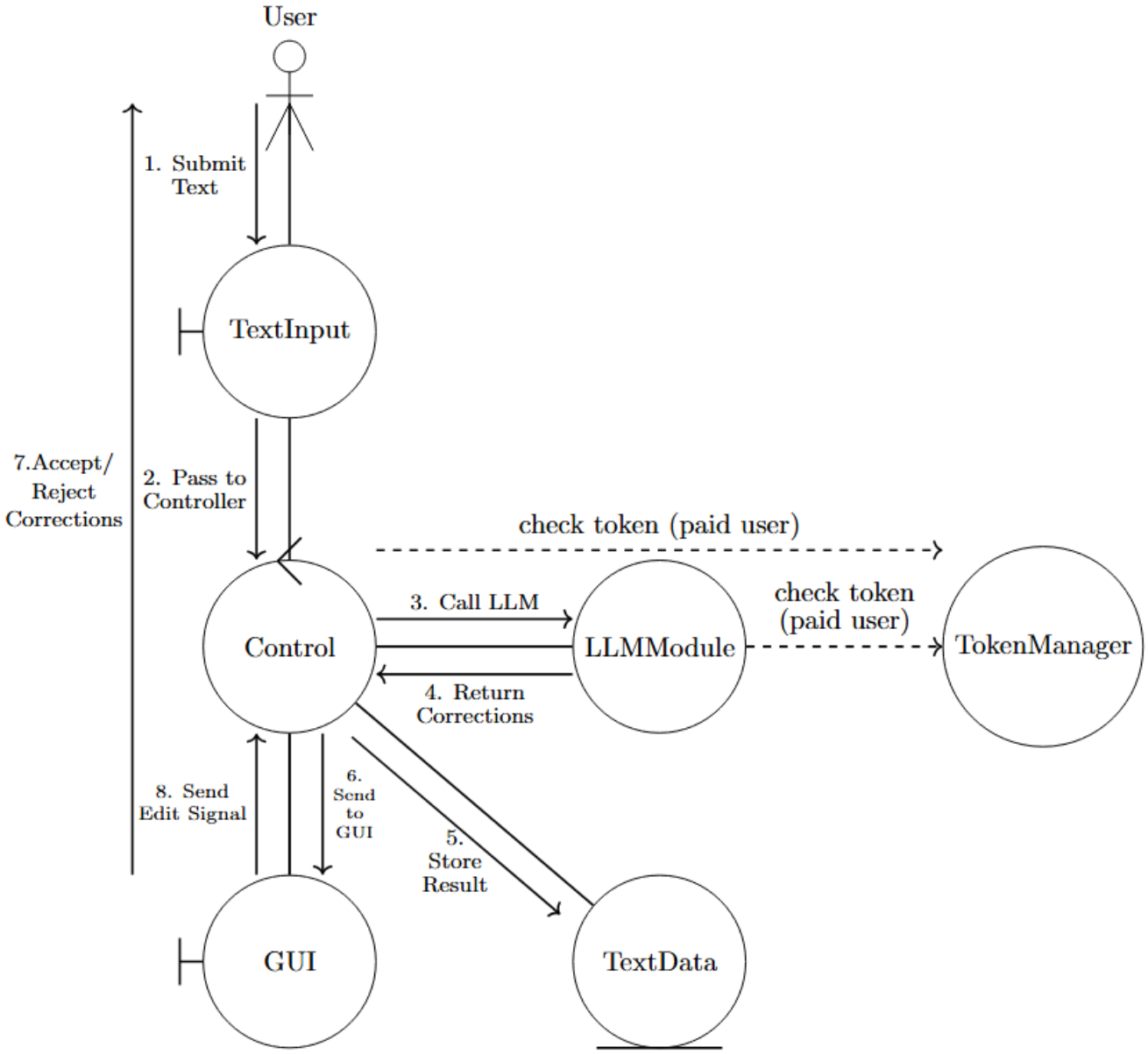
5. System screens 18

6. Memos of group meetings 19

7. Address of git repository 20

Design Report

1. Introduction



2. Use cases

2.1 Scenarios for each use case

2.1.1: Submit text (free user)

1. Free user → Text input: “try to submit text”

   The user submits text through the input interface

2. Text input → Control: “check if word count exceed limit”

   The system receives the input and checks if the number of words exceeds the limit for free users

3. a. Control → Text data: “accept text submission” (normal scenario)

   If the word count is 20 or less, the system accepts the submission and stores the text in the system memory.
   b. Control → Free user: “throw user out of system” (exceptional scenario)
   If the word limit exceeds 20, the system rejects the submission and logs the user out or restricts their access.

2.1.2: Submit text (paid users)

1. Paid user → Text input: “try to submit text”

   User initiates the process by submitting text

2. Text input→ Token data: “check tokens available”

   The system checks if the user has enough tokens

3. Token data → Control: “check if word count exceed limit”

   System checks the word count

4. a. Control → text data: “accept text submission” (normal scenario)

   If the user has enough tokens, the system accepts the submission
   b. Control → paid user: “penalize user” (exceptional scenario)
   If the user doesn't have enough tokens, the submission fails, and user is penalized

5. Token data: “update tokens available”

   Number of tokens are deducted from the user’s balance

2.1.3: Sign up

1. Free user → Signup form: “sign up as paid user”

   The user submits a form requesting paid access

2. Signup form → control: “review signup request”

   The request is passed, the super user may review the application

3. Control → Account data: “update user type”

   If the application is approved the system updates the user’s role from free to paid

4. Control → Free user: “accept/decline application”

   The system sends a notification to the user indicating the signup request was accepted/denied

2.1.4: Complain and fine/suspend/terminate paid users

1. User → Complaint form: “file complaint”

   User submits a complaint

2. Complaint form → Control: “handle complaint”

   Complaint is forwarded to the super user

3. a. Control → complaint data: “add complaint”

   Complaint is stored in the system
   b. Control → user: “notify complaint”
   User is notified
   c. control → user: “fine/suspend/terminate user” (exceptional scenario)
   If the user violated rules they may be fined, suspended or terminated

4. Control: “resolve complaint”

   The super user reviews responses and makes decision

5. Control → complaint data: → “delete complaint”

   Once resolved, the complaint is removed from the database

2.1.5: Purchase tokens

1. Paid user → transaction: “enter purchase amount”

   User specifies how many they want to buy

2. a. Transaction → control: “complete transaction”

   Payment is processed and verified
   b. Transaction → paid user: “transaction fail” (exceptional scenario)
   Occurs if the payment process fails

3. Control → token data: “update tokens available”

   User’s token balance is updated to reflect the purchase

2.1.6: Submit blacklist of words, and add words to the blacklist

1. User → blacklist input: “submit words to blacklist”

   User enters one or more words they think should be restricted

2. Blacklist input → control: “review submission”

   The submission is forwarded to a super user

3. a. Control → blacklist data: “add words to blacklist”

If approved, the submitted words are added to the system's blacklist
b. Control → user: "not add to blacklist" (exceptional scenario)
The control rejects the submission if the word is considered not valid for blacklisting

4. Blacklist data → text data: "replace words in blacklist"

   The new blacklist is used to censor words in the future user submissions

2.1.7: Submit to self-correction

1. Paid user → self-correction: "submit to correction"

   The user chooses to correct their own text and begins editing

2. Self-correction → control: "process correction"

   System analyzes the manual corrections

3. Control →text data: "correct words"

   Corrected version is saved in the system

4. Text data → token data: "charge tokens"

   Tokens are deducted based on the number of words corrected

2.1.8: Submit to LLM correction, and accept/reject correction

1. Paid user →LLM correction: "submit to correction"

   User sends their text to be reviewed

2. LLM correction → control: "process correction"

   System forwards input to LLM

3. a. Control → text data: "correct and highlight text"

   LLM returns corrections
   b. Control → paid user: "ask user to accept correction"
   User is prompted to either accept or reject the suggestions

4. a. Paid user → control: "accept correction"

   For each accepted correction, 1 token is deducted
   b. Paid user → control: "reject correction" (exceptional scenario)
   User may reject the LLM's suggestion

5. Control → token data: "deduct tokens"

   Tokens are deducted based on the number of accepted corrections

2.1.9: Save text in text file

1. Paid user → text data: "save text"

   User requests to save the current text to a file

2. Text data → control: “process”

   System prepares the text and verifies token availability

3. a. Control → file output: “output text to file”

   System generates a local file
   b. Control → token data: “charge tokens”
   System charges tokens from the user’s balance for saving

2.1.10: Accept/reject invitation to share text files

1. Paid user → control: “share text files”

   User initiates the sharing process

2. Control → paid user: “invite other paid users”

   System allows the other user to send an invitation to another paid user

3. a. Invited users → control: “accept invitation”

   The invited user accepts the collaboration invite
   b. Invited users → control: “decline invitation” (exceptional scenario)
   Invited user chooses not to collaborate

4. a. Control →text file: “invited users become collaborators”

   System grants access to the text file for both users
   b. Control →token data: “charge inviter for tokens”
   Inviter is charged tokens for inviting

2.1.11: Receive bonus of tokens

1. Paid user → LLM correction: “submit text to LLM”

   User submits a text with more than 10 words for correction

2. LLM correction → control: “find text with >10 words that has no error”

   System checks the text and detects no errors

3. a. Control → token data: “user receives bonus of tokens”

   User is awarded bonus tokens for submitting text with no errors

2.1.12: See errors/corrections/statistics of usage

1. Paid user → GUI: “log in”

   User begins log in process

2. GUI → Control: “process information”

   System processes log in credentials

3. Control → account data: “retrieve user account type”

Checks if the user is paid and determines what to display

4. a. Account data → token data: “retrieve used and available tokens”

   System pulls the current token balance and usage history
   b. Account data → text data: “retrieve user account”
   Brings information about corrections or file history

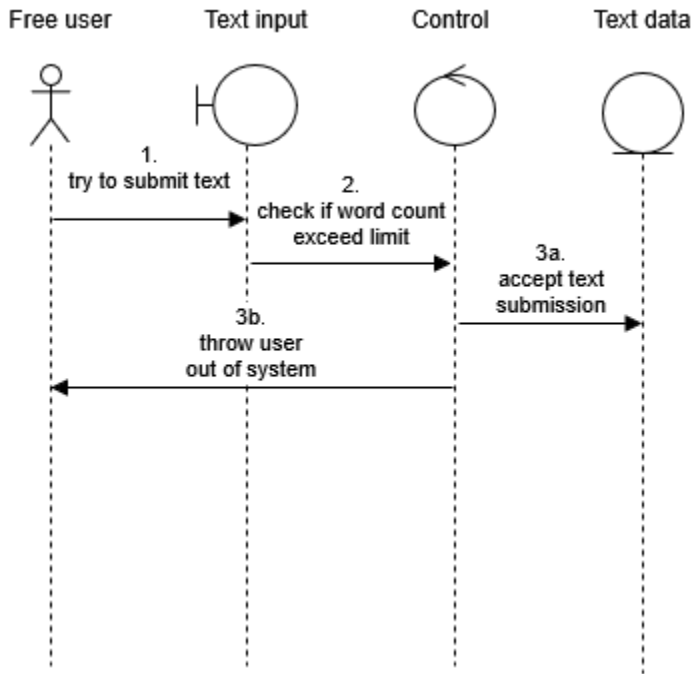
5. a-b. Control → GUI: “display data to GUI”

   GUI is updated with account information

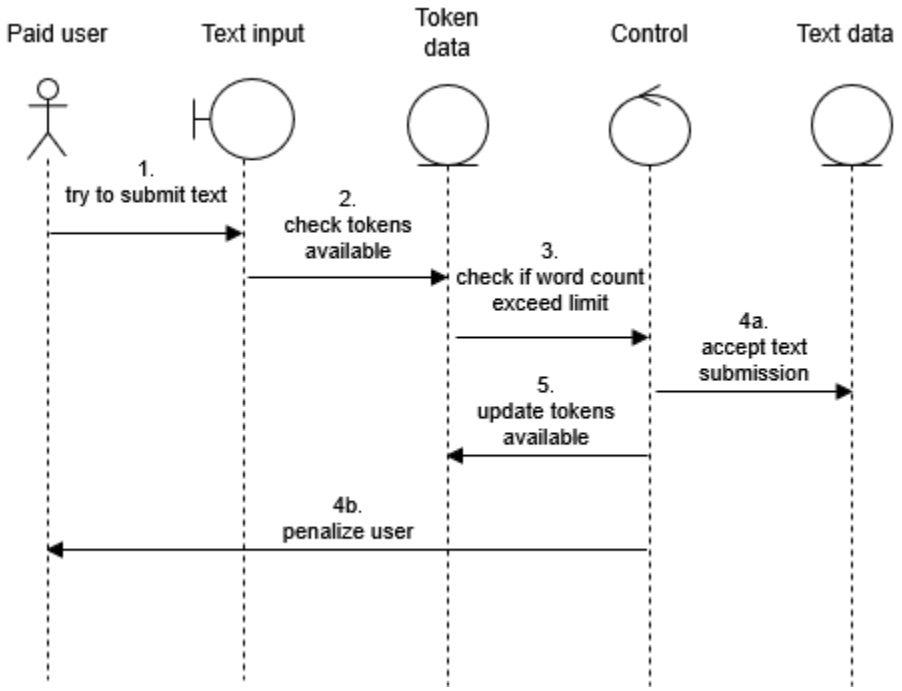
2.2 Class diagram for each use case

For each use case, the sequence class diagrams are provided to describe how objects work together in order.

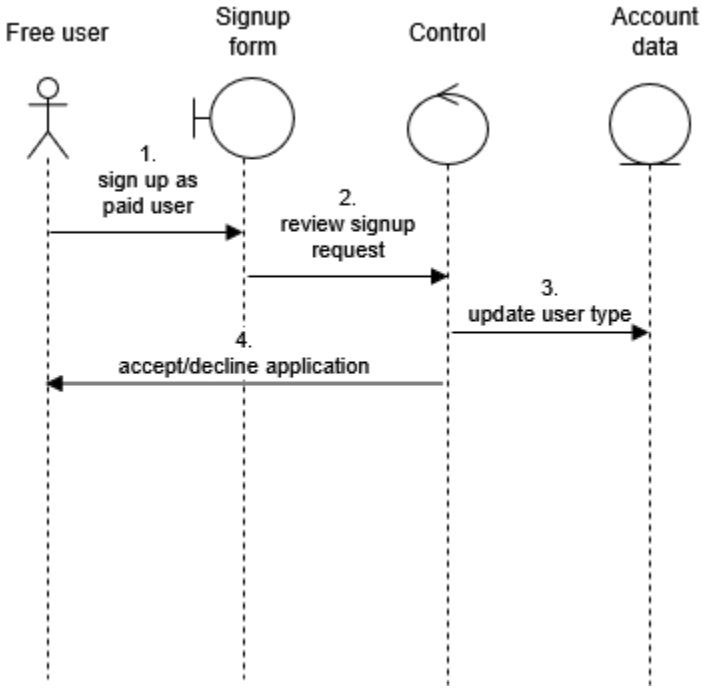
2.2.1 Submit text (free users)



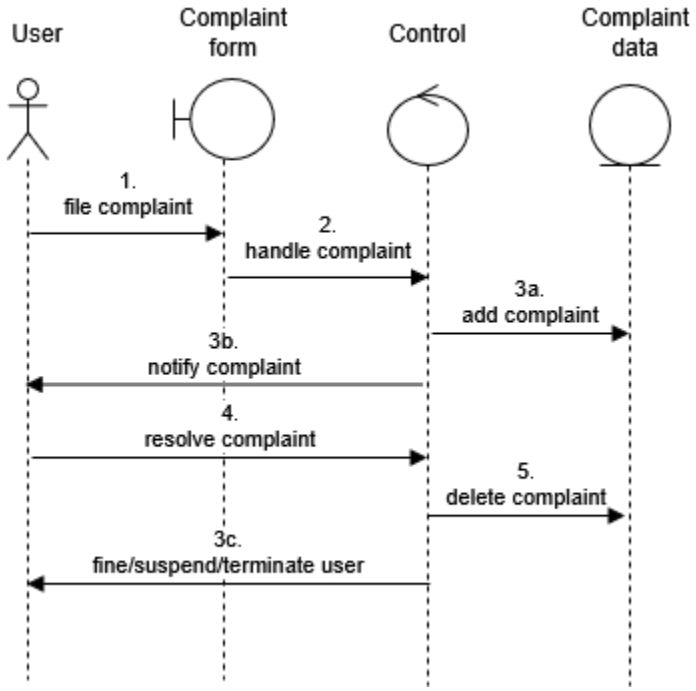
2.2.2 Submit text (paid users)



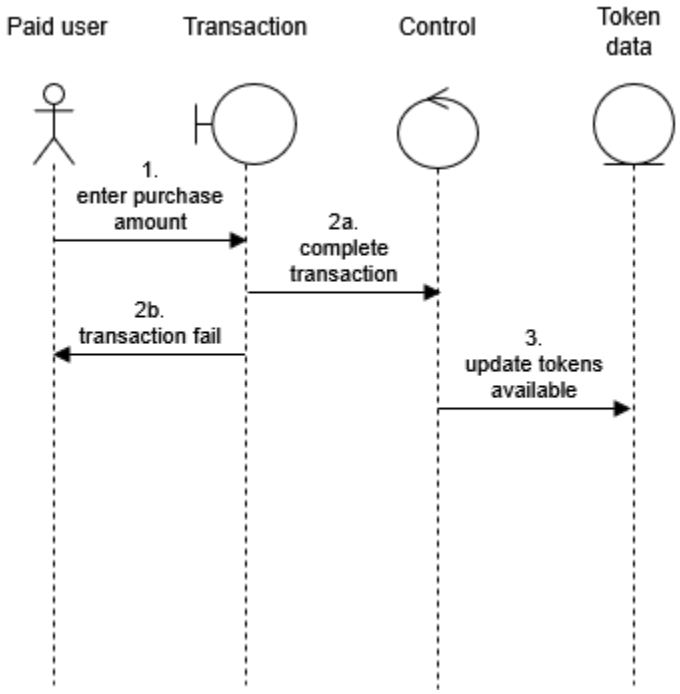
2.2.3 Sign up



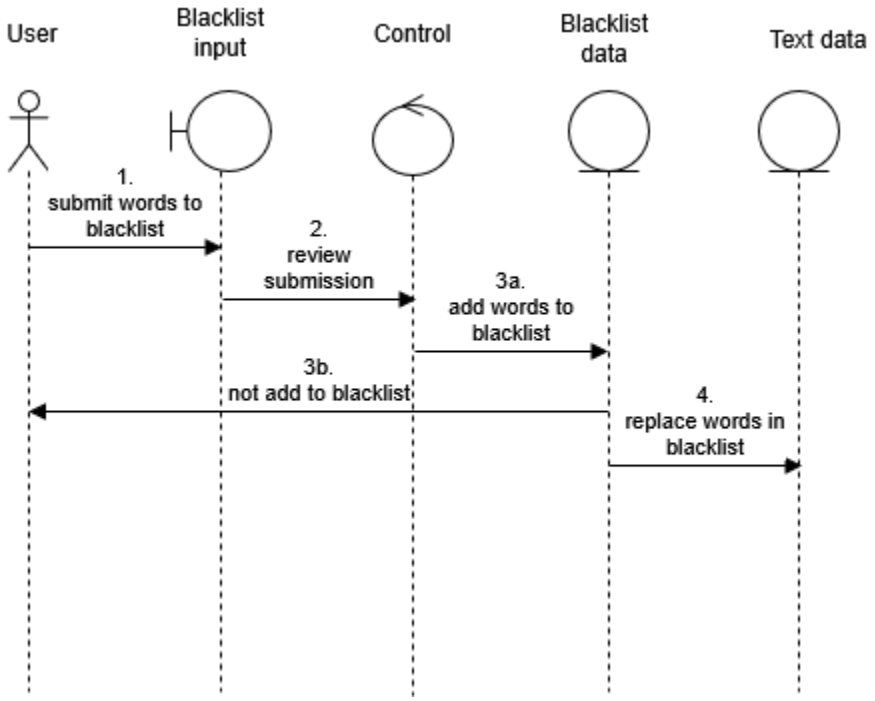
2.2.4 Complain, and fine/suspend/terminate paid users



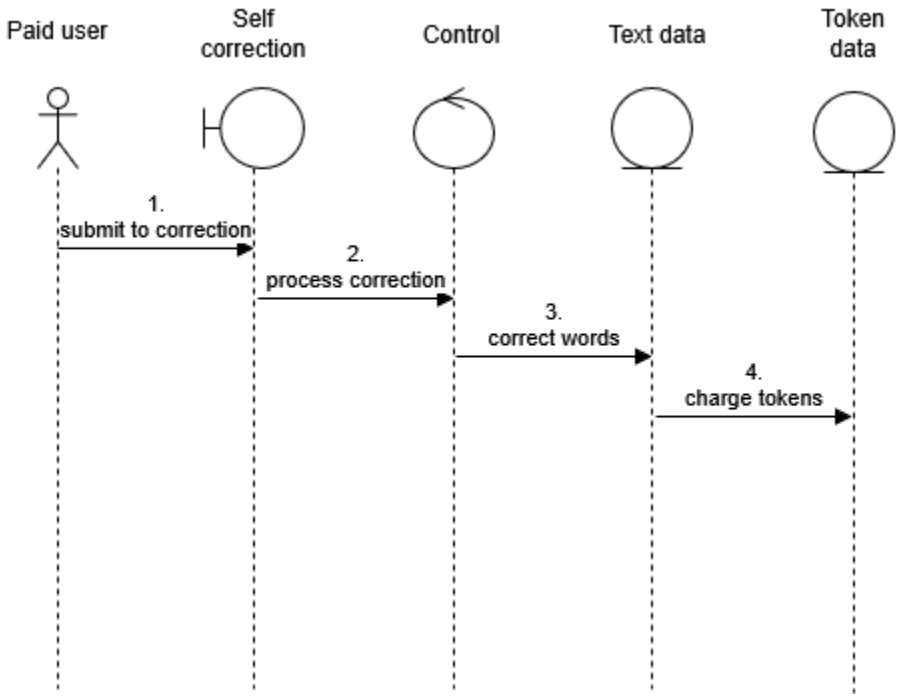
2.2.5 Purchase tokens



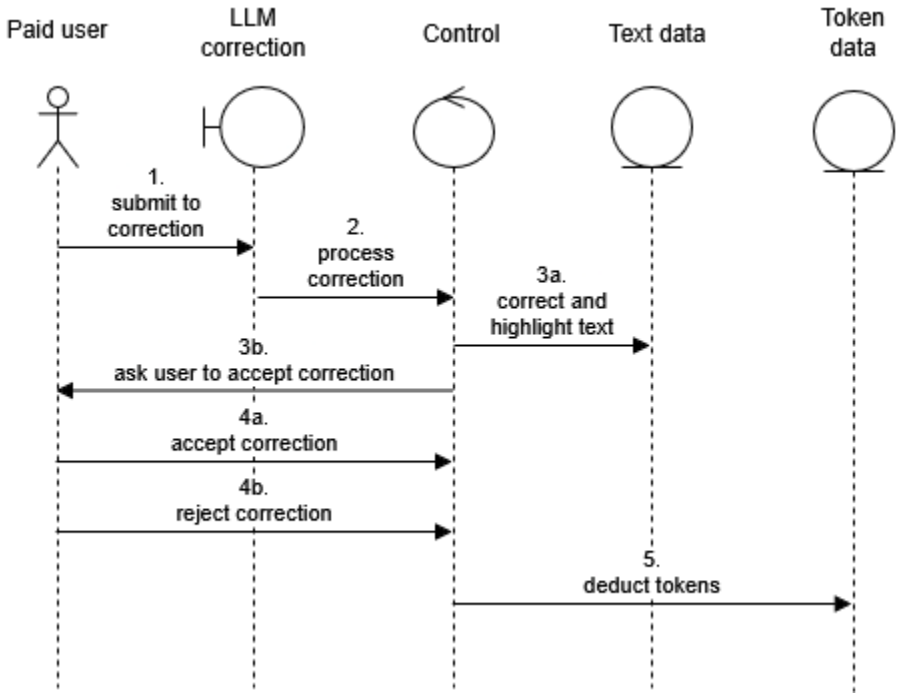
2.2.6 Submit blacklist of words, and add words to blacklist



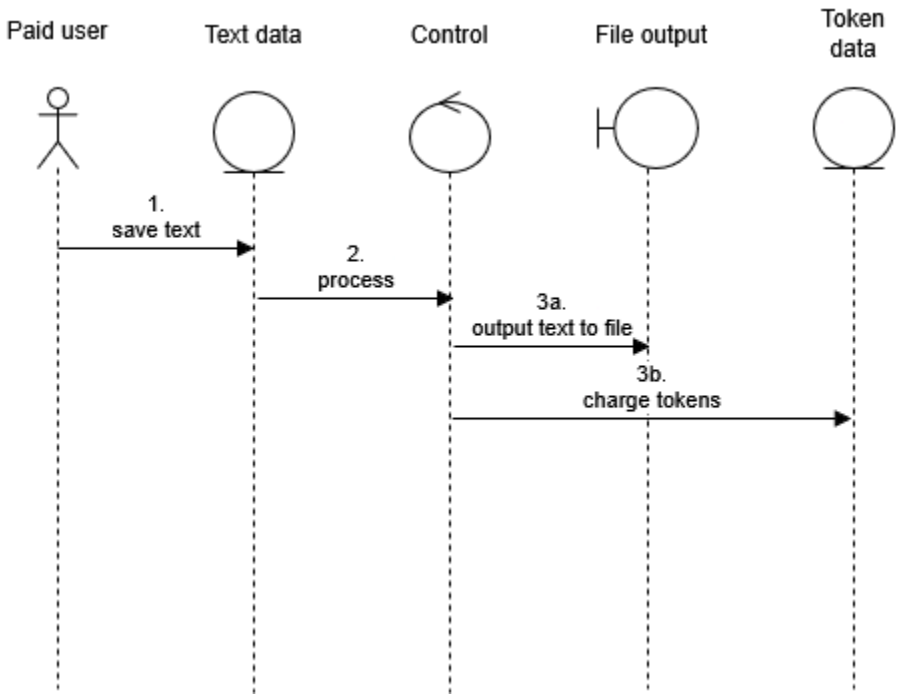
2.2.7 Submit to self-correction



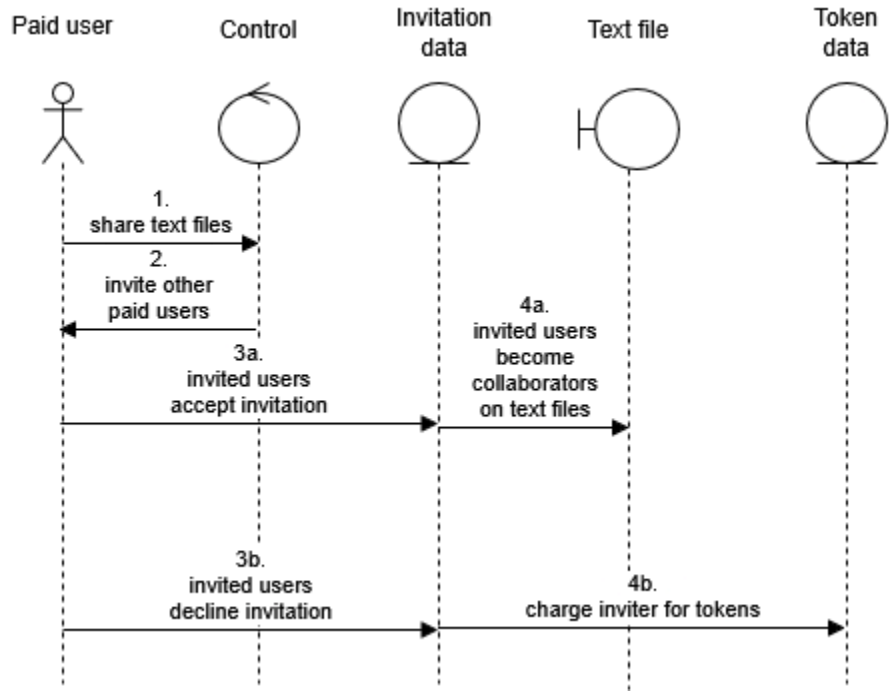
2.2.8 Submit to LLM correction, and accept/reject correction



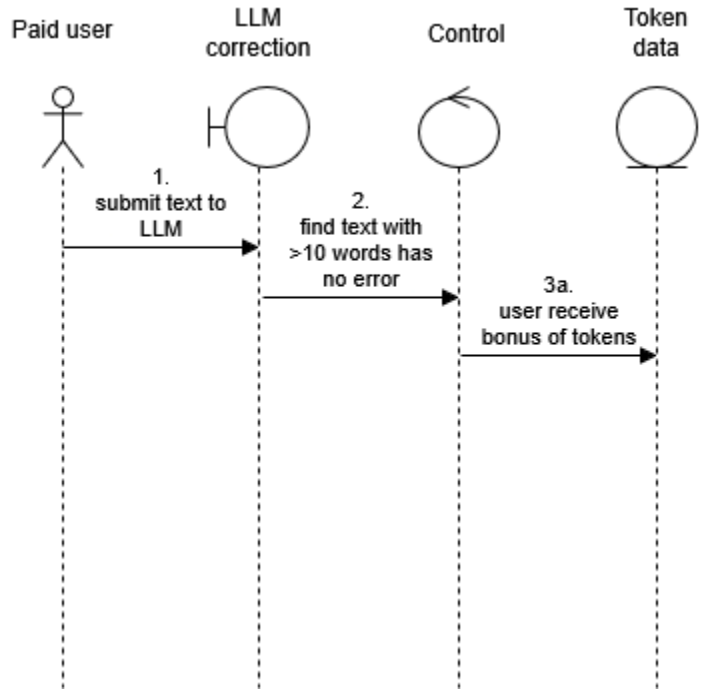
2.2.9 Save text in text file



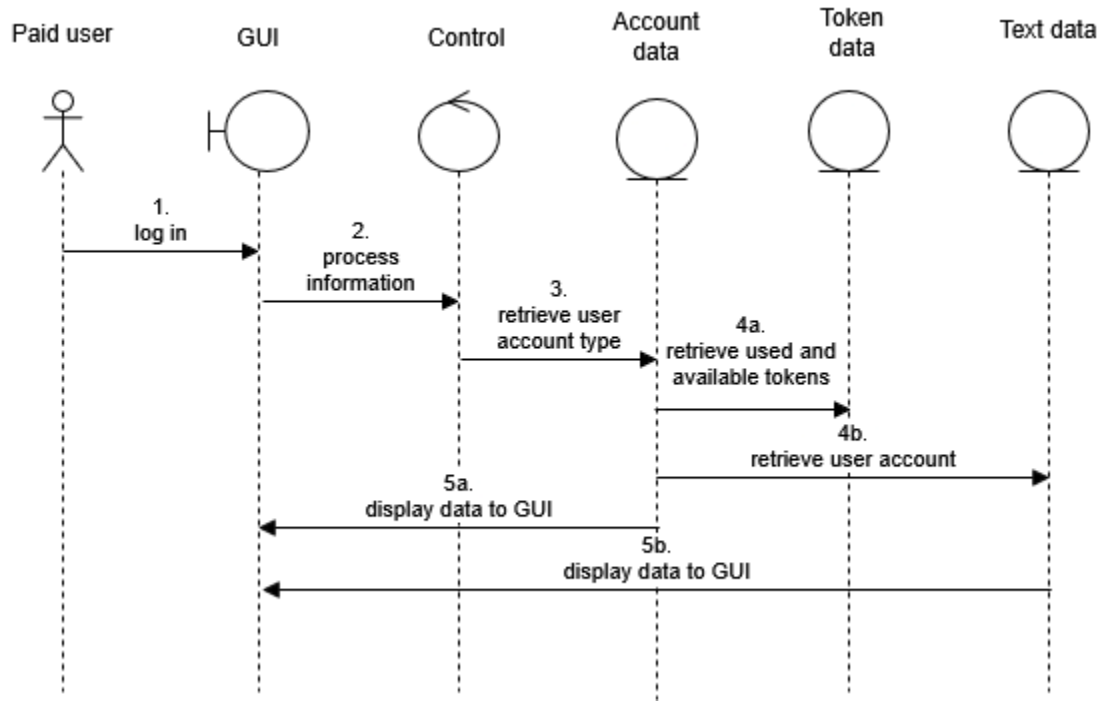
2.2.10 Accept/reject invitation to share text files



2.2.11 Receive bonus of tokens

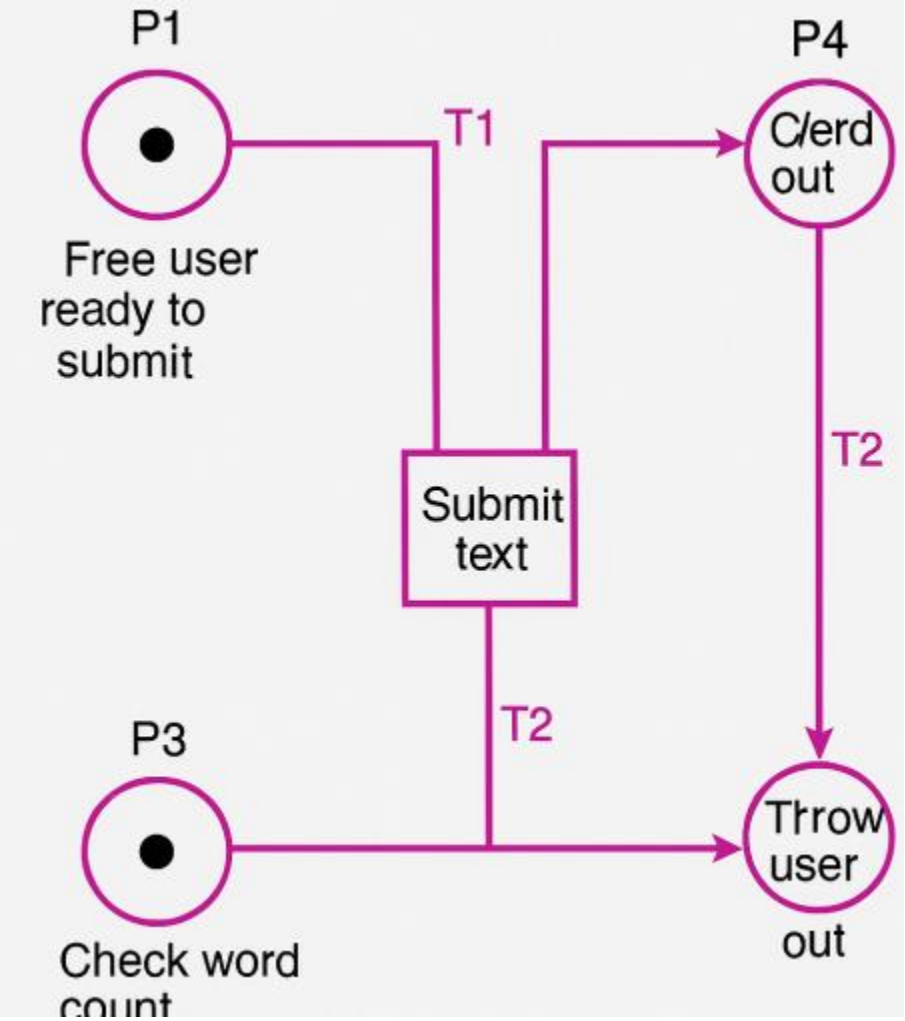


2.2.12 See errors/corrections/statistics of usage



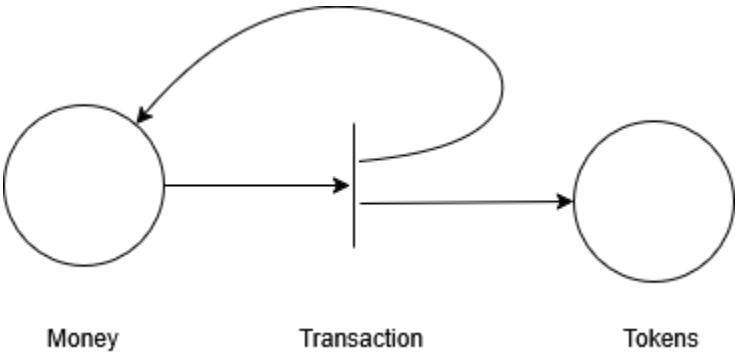
2.3 Petri-net for selected use cases

2.3.1 Submit text (free users)



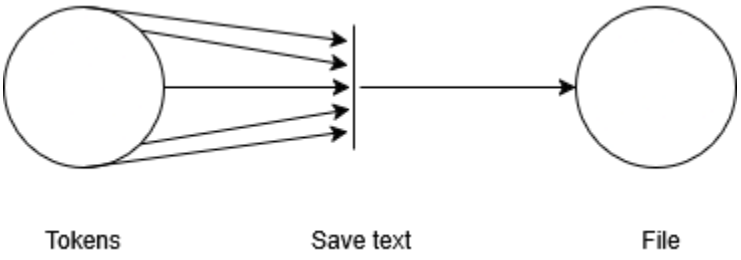
2.3.2 Purchase tokens

Paid users can purchase a certain number of tokens. If the transaction succeeds, the system receives money from the users and the users receive an equivalent number of tokens. If the transaction fails, the system sends back the money to the users.



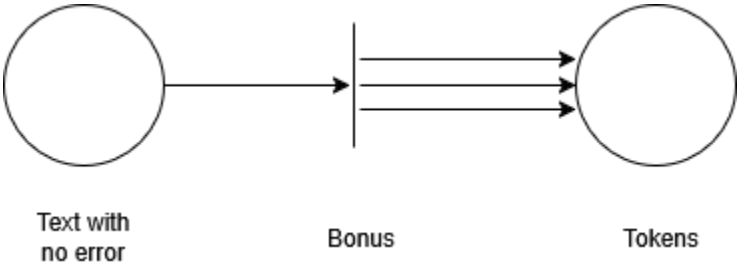
2.3.3 Save text in text file

After any correction, the paid users can choose to save the text in a text file with a charge of 5 tokens.



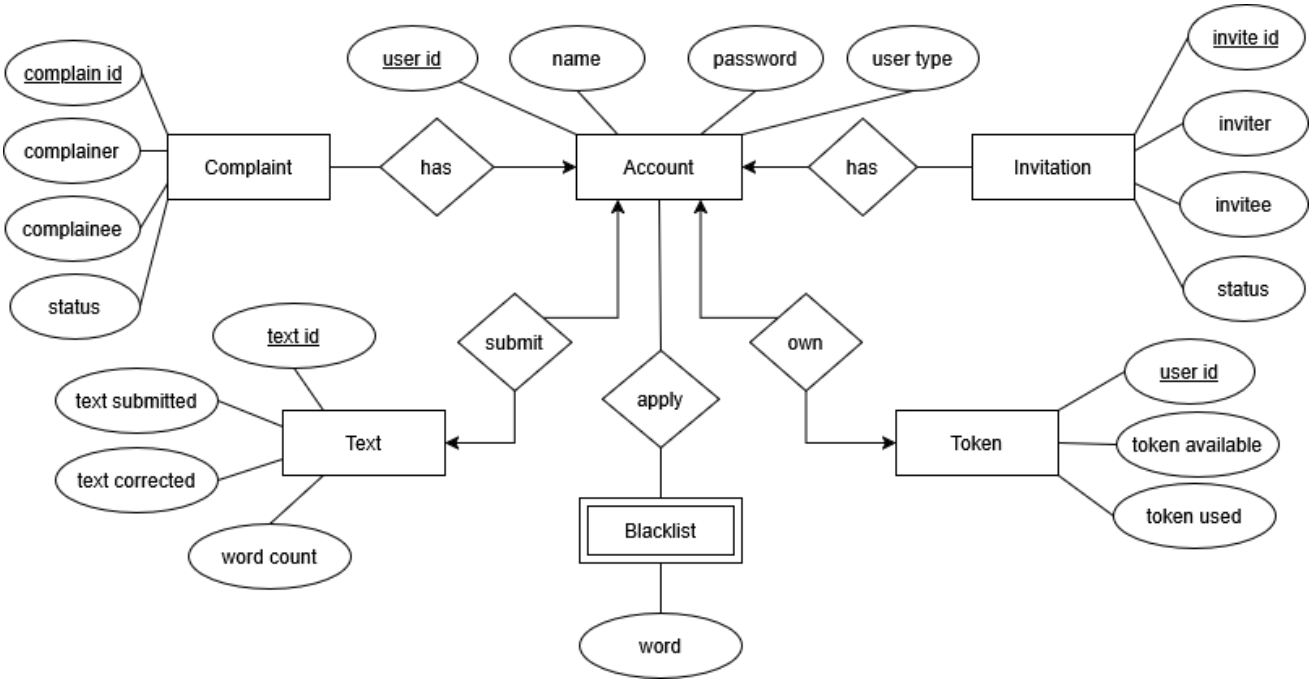
2.3.4 Receive bonus of tokens

If paid users submit a text with more than 10 words to LLM with no error, the user will receive a bonus of 3 tokens deposited to the account.



3. E/R diagram for system

The entity relationship diagram is used for the high-level database design that illustrates the relationship between the tables needed for the system.



4. Detailed design

4.1 Main Streamlit Application

Function Name: *Main UI Logic*

Inputs:
- user_input – user-entered text
- LLM_instruction – instruction string for the LLM

Outputs:
- Rendered, corrected text with highlighted changes

Pseudo-code:

```
Set page config and header
Create text input box
If submit is clicked and input is not empty:
    Send input and instruction to LLM
    Get corrected output
    Split both input and output into word lists
    Compare word lists using difflib

    For each comparison result:
        If words match, keep them as-is
        If words differ, wrap corrected words in styled HTML button
    Display the joined result using st.markdown
Else:
    Show warning if input is empty
```

4.2 Reject Edit Feature (Toggle Correction View)

Function Name: *Toggle Word View / render_toggleable_text*

Inputs:
- user_input: original text from user
- corrected_text: LLM output
- index: word index to toggle on click

Outputs:
- Dynamically rendered sentence
- Highlighted corrected or original words based on toggle state

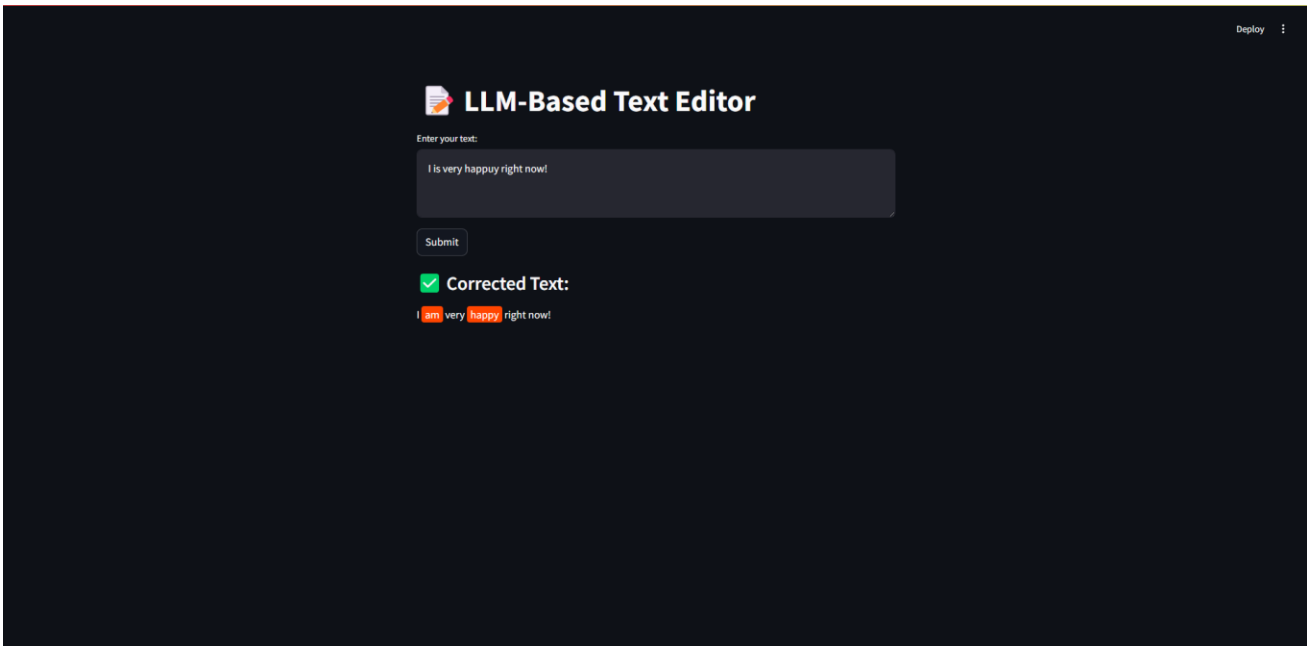
Pseudo-code:

```
On submit:
    Send user input to LLM
    Get corrected text response
    Tokenize original and corrected text
    Use difflib to compare them

    For each comparison result:
        If equal → store (word, word)
        If changed → store (corrected_word, original_word)

    Save this list in session_state.display_words
```

```
    Initialize session_state.word_status = { i: False }

Render:
    For each word in display_words:
        If word_status[i] is False → show corrected word (orange)
        If True → show original word (gray)

        Wrap each word in HTML button or span with onclick
        All words rendered inline using st.markdown

When word is clicked:
    Toggle word_status[i]
    Rerun the app to reflect change
```

5. System screens

The prototype GUI for text submission with LLM correction is shown below:



6. Memos of group meetings

3/14:
- Set up team communication channel
- Discussed on team roles and tech stack
- Worked on first phase report

3/18:
- Decided on Streamlit for UI and Mistral in Ollama for LLM
- Set up GitHub repository

4/11:
- Worked on second phase report

7. Address of git repository

https://github.com/Dece1st/LLM-Based-Text-Editor