Team B

TypeAI
Design Specification
For Text Correction System

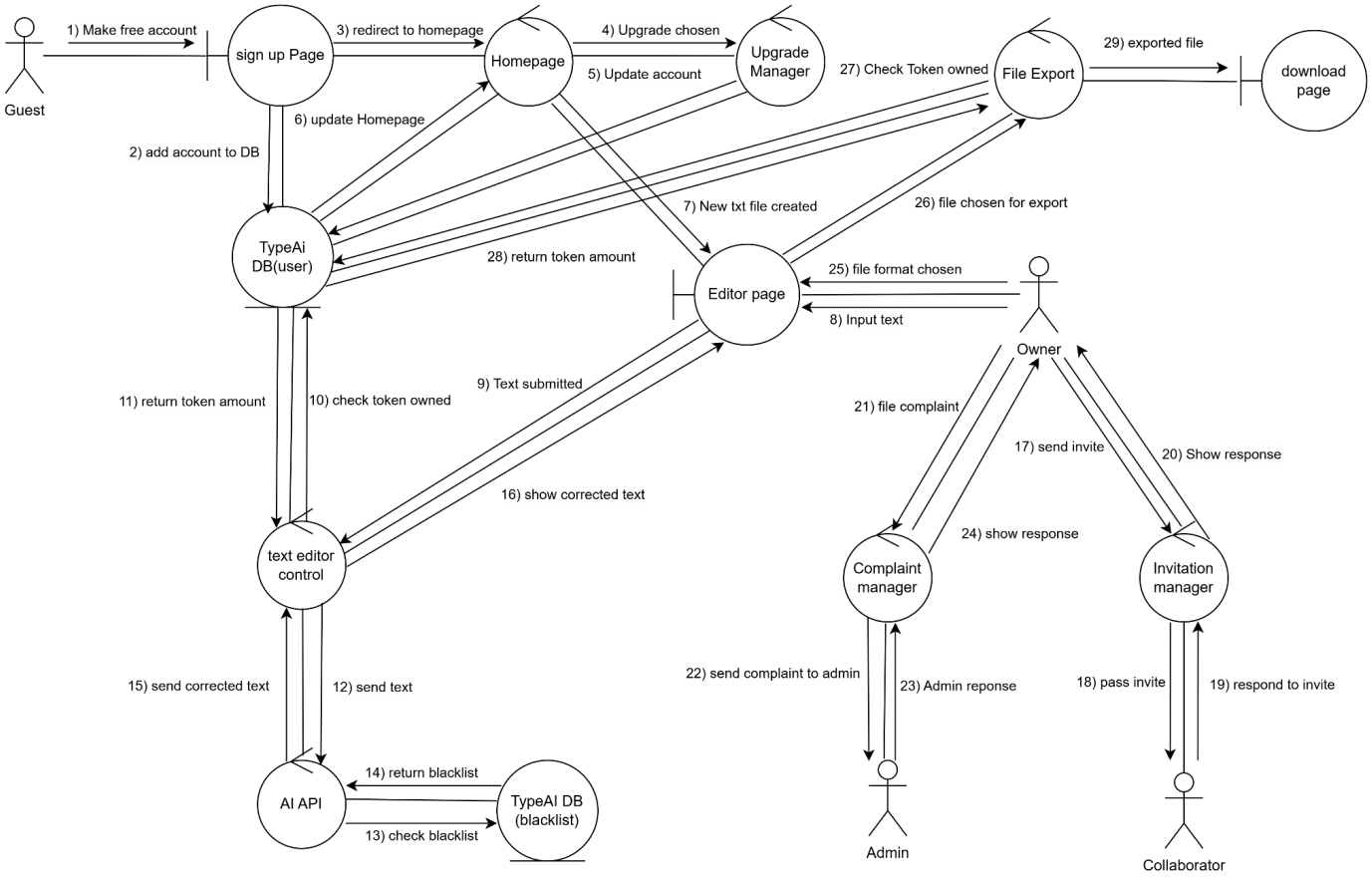
Version <1.0>

Table of Contents

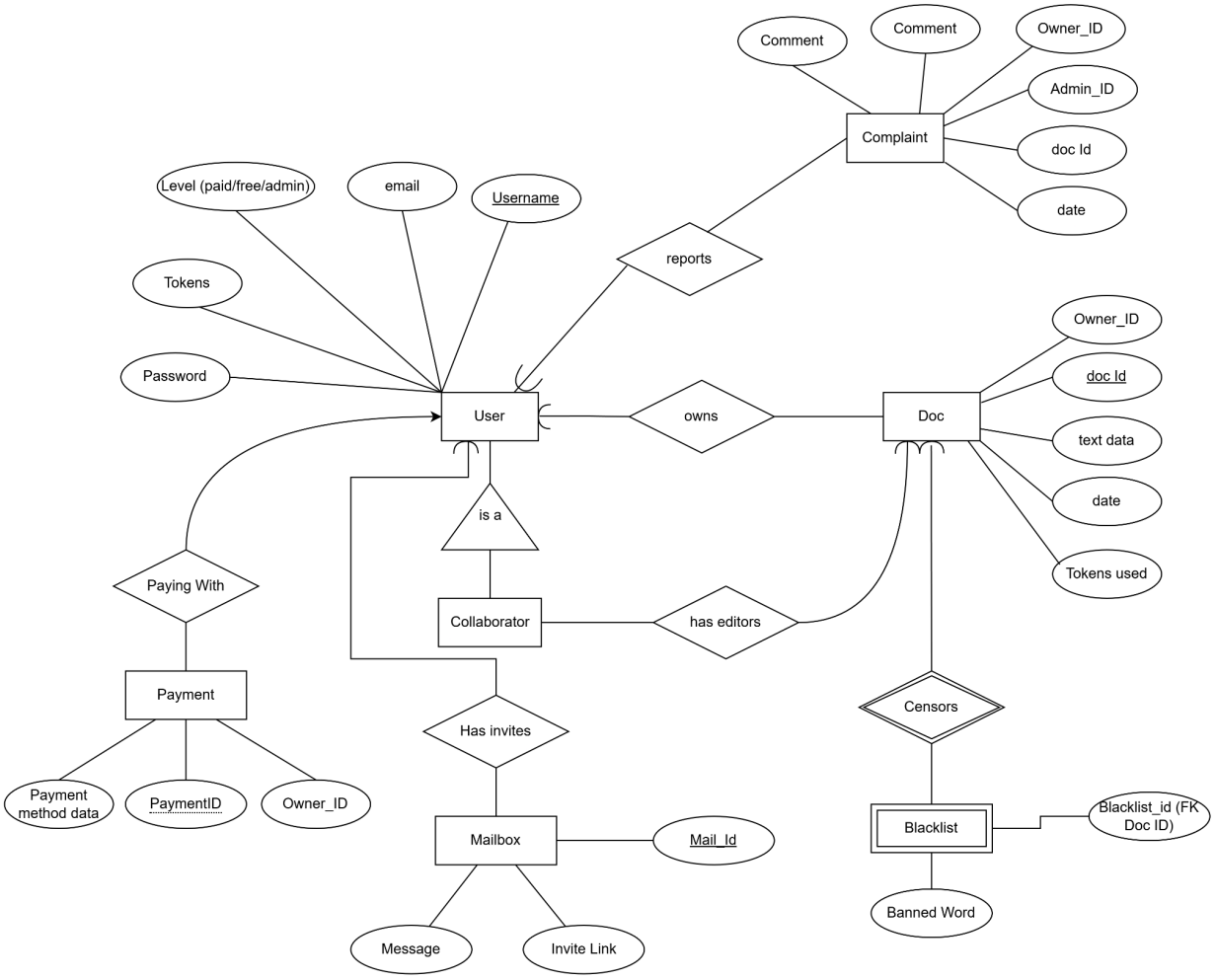
Introduction

This Design Specification outlines the design flow of the Text Correction System, a core component of the TypeAI platform. The system enables users to correct text using AI, filter blacklisted words, and manage token-based charges for paid features. The design flow begins with a guest making an account and upgrading it, followed by file creation and text input, text correction with optional features for blacklist filtering. It supports collaboration for paid users, complaint management, and file exporting. This structured flow ensures efficient text editing, collaboration, and file management while maintaining transparency and accessibility for users.

(*For visual clarity refer to draw.io page in appendix/resources*)



Entity-Relation Diagram



Use-Case Class Diagrams

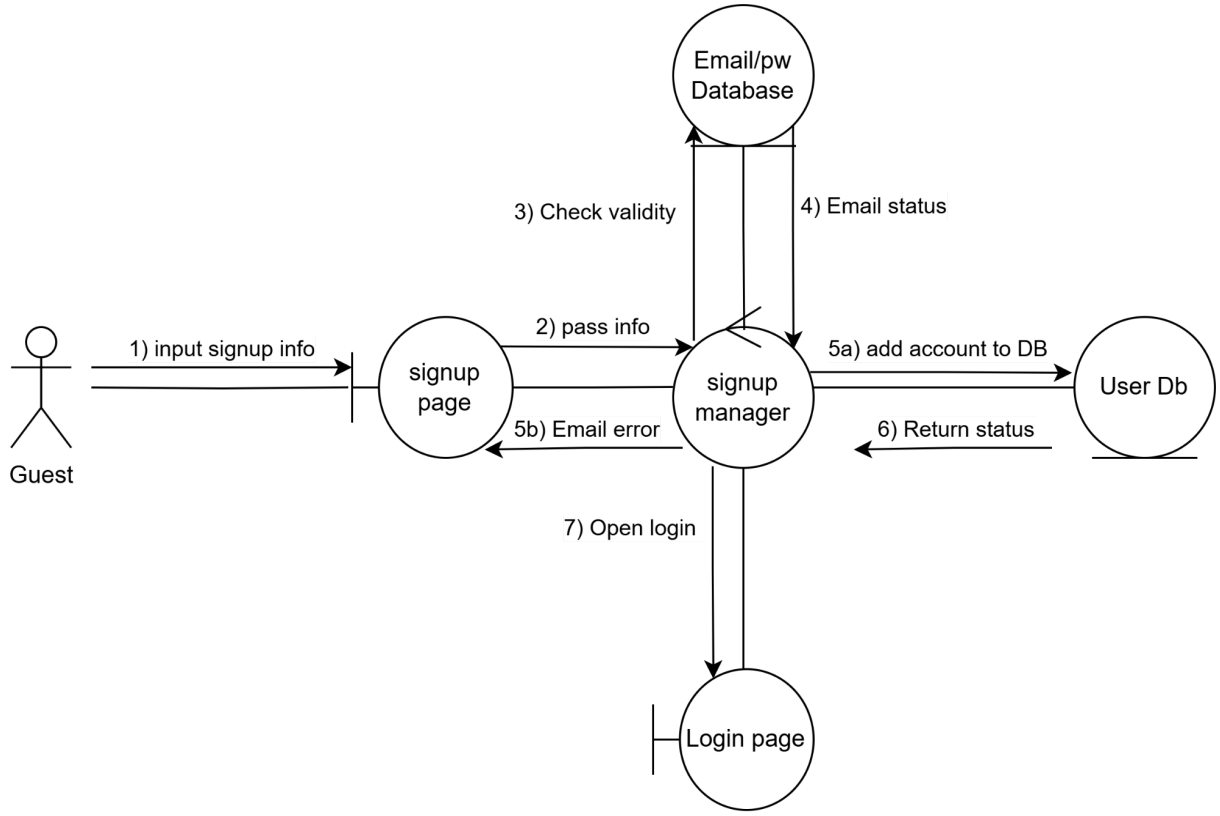
1. Sign up

Normal:

‘Guests’ will input some information that will be used as their verification for using the system as either a free or paid user. Then that info will be handled and after making sure things are valid, pushed into the accounts database. New accounts will be given a starting amount of tokens to use.

Exceptional:

‘Guests’ will not input any/all necessary information and will be warned/prompted to fill in all the necessary parts. So no new accounts will be established.



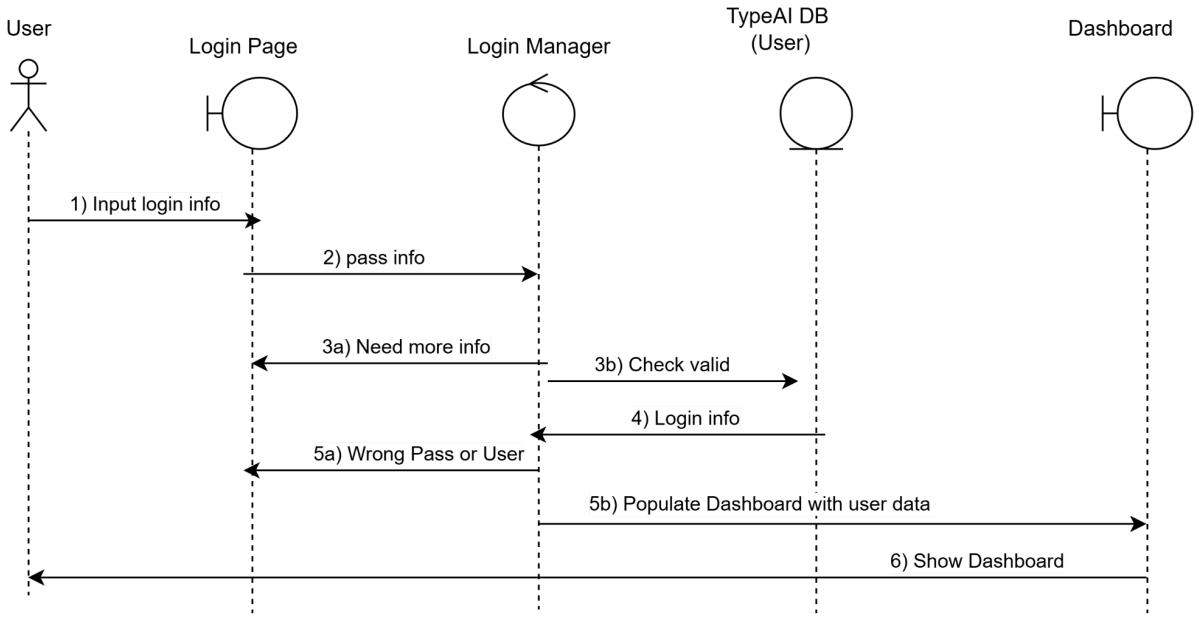
2. Log-in (Paid/Free)

Normal:

Users will input their login information and if it is valid be shown the ui/dashboard for either a free or paid user: depending on their account status (paid user or not).

Exceptional:

- If a user neglects to put in all necessary information an error message will be displayed;
- fails at inputting a correct password while having a correct username, a different related error will be displayed.



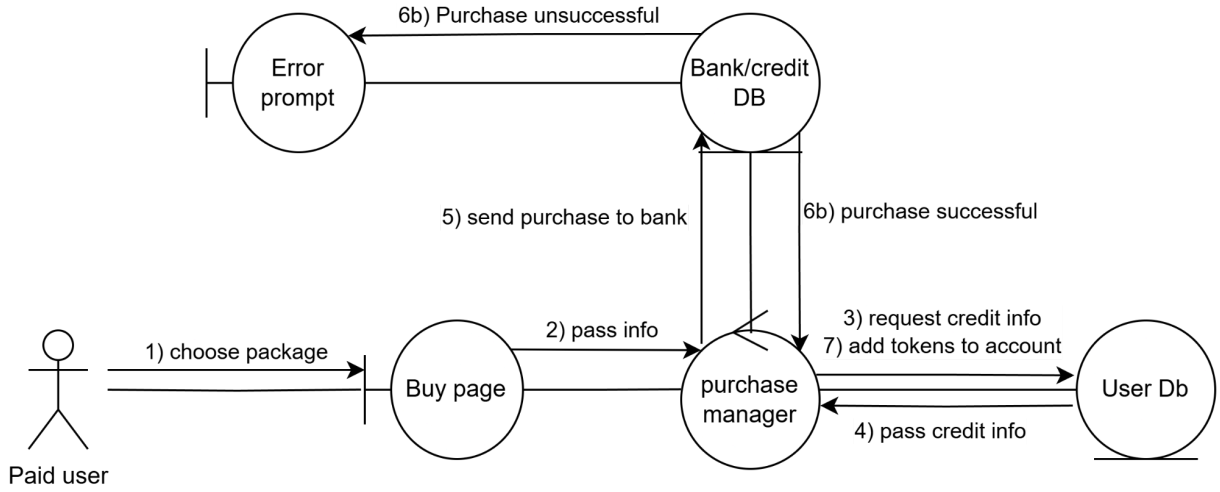
3. Purchase Tokens

Normal:

Paid users will traverse the UI to buy tokens, then select a tokens package and a payment method. Upon successful transaction, tokens are added to the account.

Exceptional:

- On bad transaction (ie invalid bank information) no charge is made and no tokens are distributed
- If tokens aren't distributed or an extra charge comes through, user can report it and customer service will investigate and provide a refund or add the necessary tokens



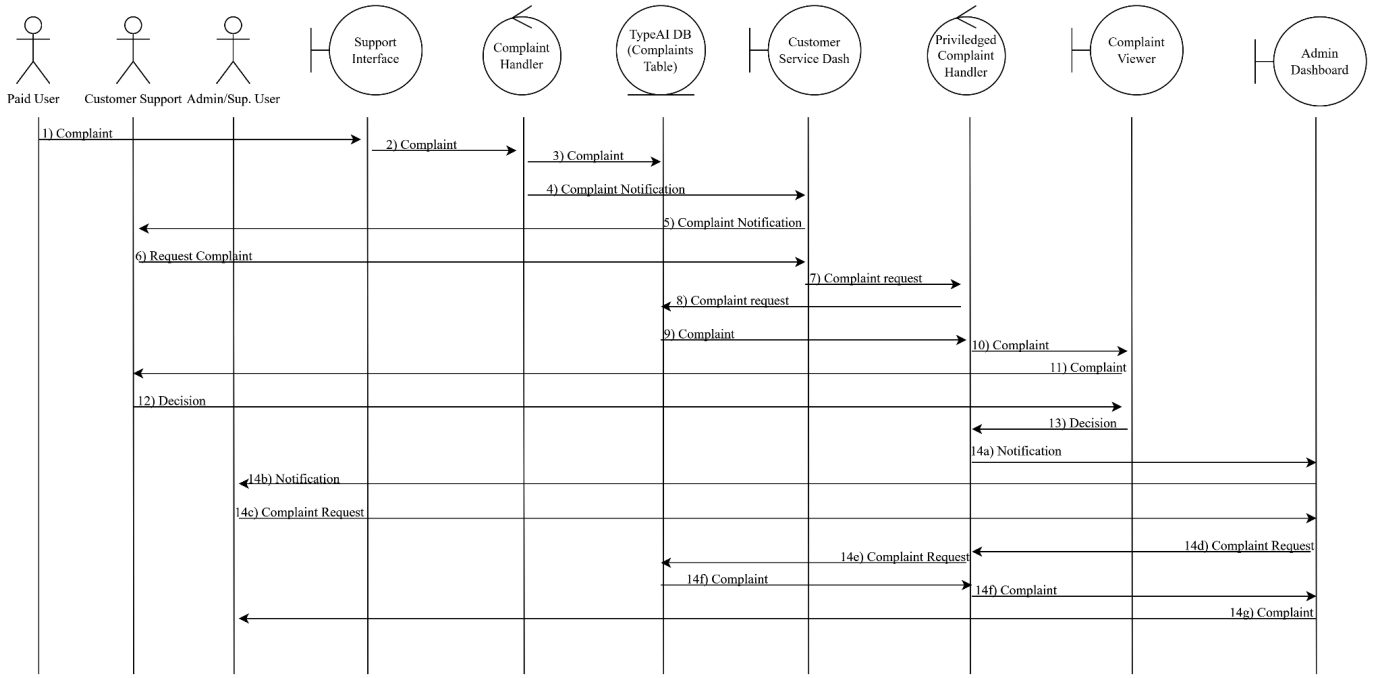
4. Complaining

Normal:

Paid users submit a complaint via the support interface. Customer support investigates and responds appropriately. If needed, the complaint will be sent to the admin.

Exceptional:

Misuse of the complaint system (e.g., spam) may result in an account warning or suspension from the admin.



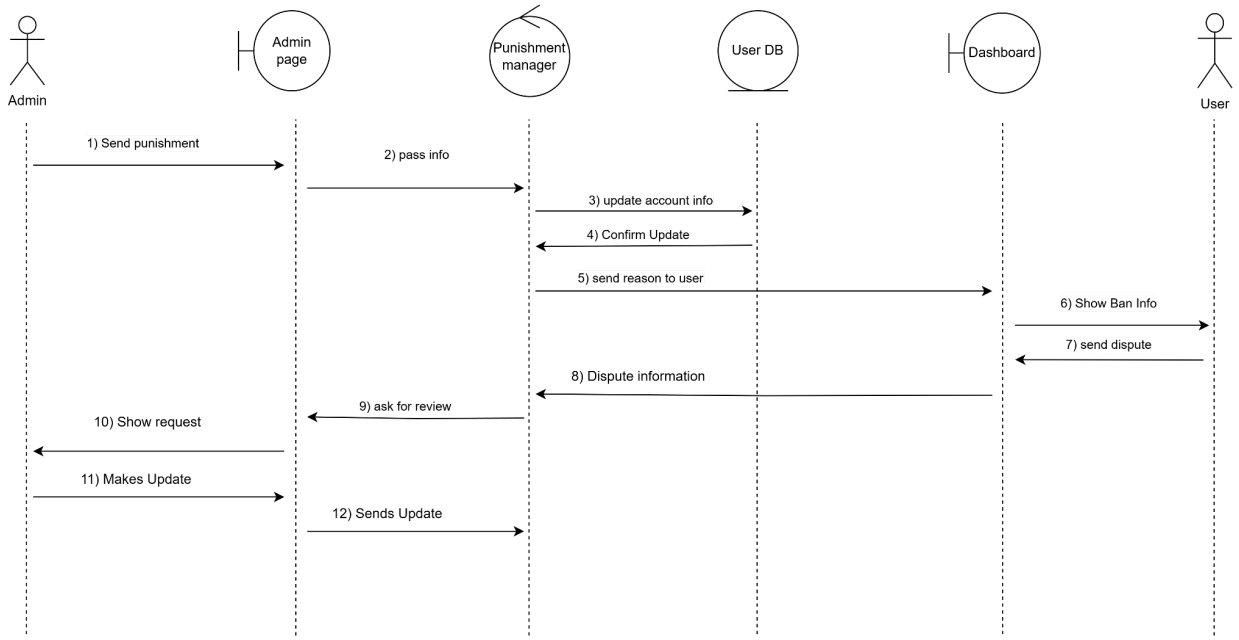
5. Terminate/Suspend

Normal:

Admin reviews the user's activity (e.g., spam, overuse of blacklist words) and terminates/suspend the user if it's justified. User will receive a note about the illegal activity and can dispute it.

Exceptional:

The activity is found to be unjustified, and the admin reverses the termination/suspension.



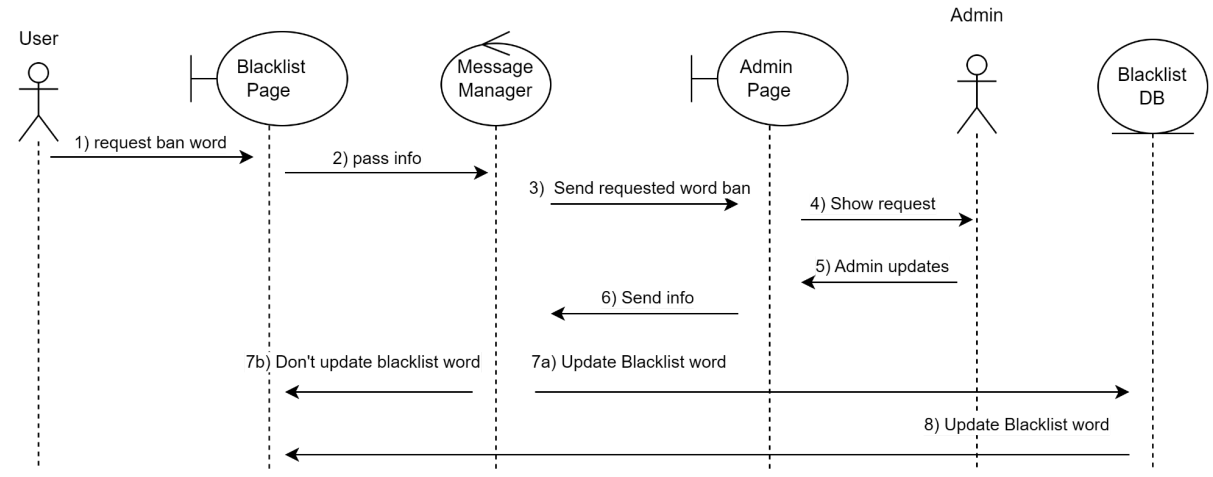
6. Blacklist words

Normal:

The System or the user identifies a word that is offensive/inappropriate and can report it. Admin will review the word and add it to blacklisted words if justified.

Exceptional:

The Admin deemed the word not needed to be inserted into the blacklist.



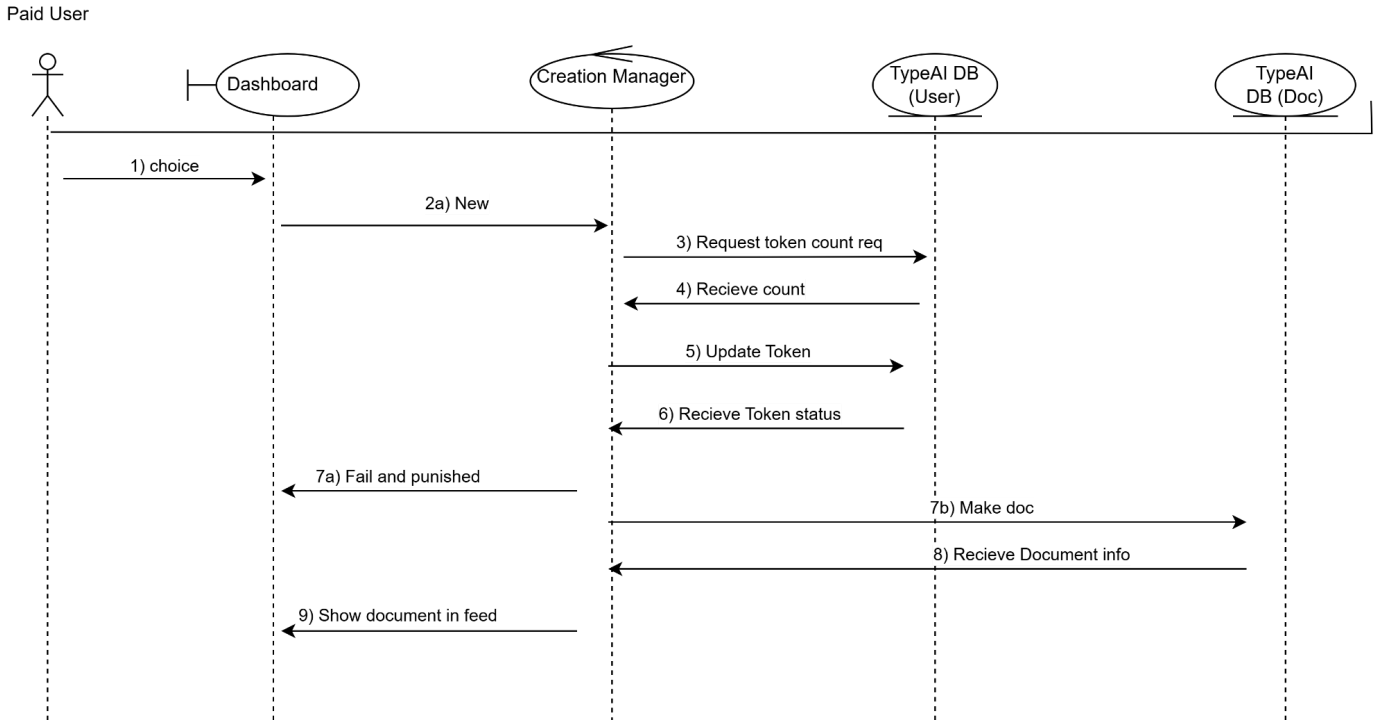
7. Start text (paid)

Normal:

A user inputs text or uploads a text file and the system will charge the number of tokens based on the number of words.

Exceptional:

- If the user doesn't have enough tokens to upload the file, no file will be uploaded and the user will lose half of their remaining tokens.



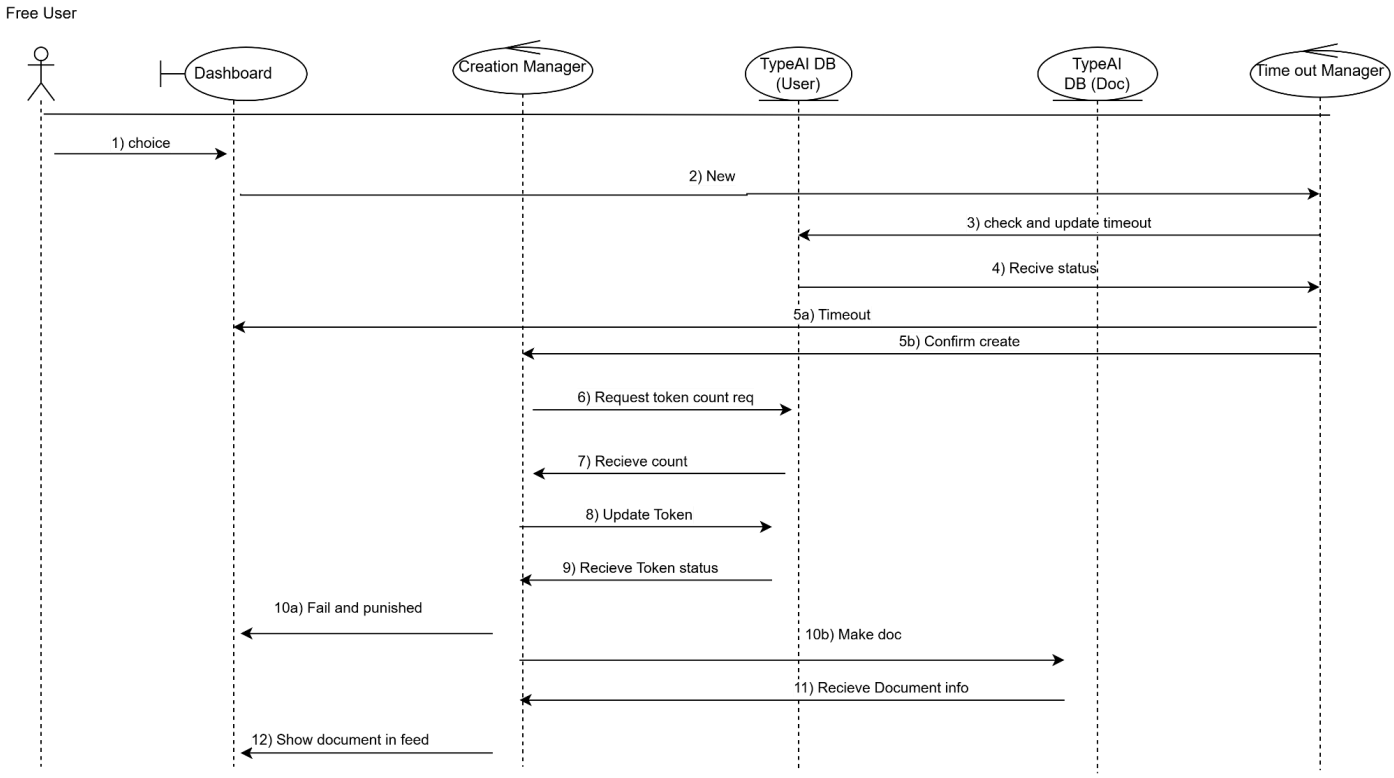
8. Start text (free)

Normal:

A user inputs text or uploads a text file and the system will charge the number of tokens based on the number of words.

Exceptional:

- Before next steps, more than 20 words, throw it out and timeout user
    - If the user doesn't have enough tokens to upload the file, no file will be uploaded and the user will lose half of their remaining tokens.



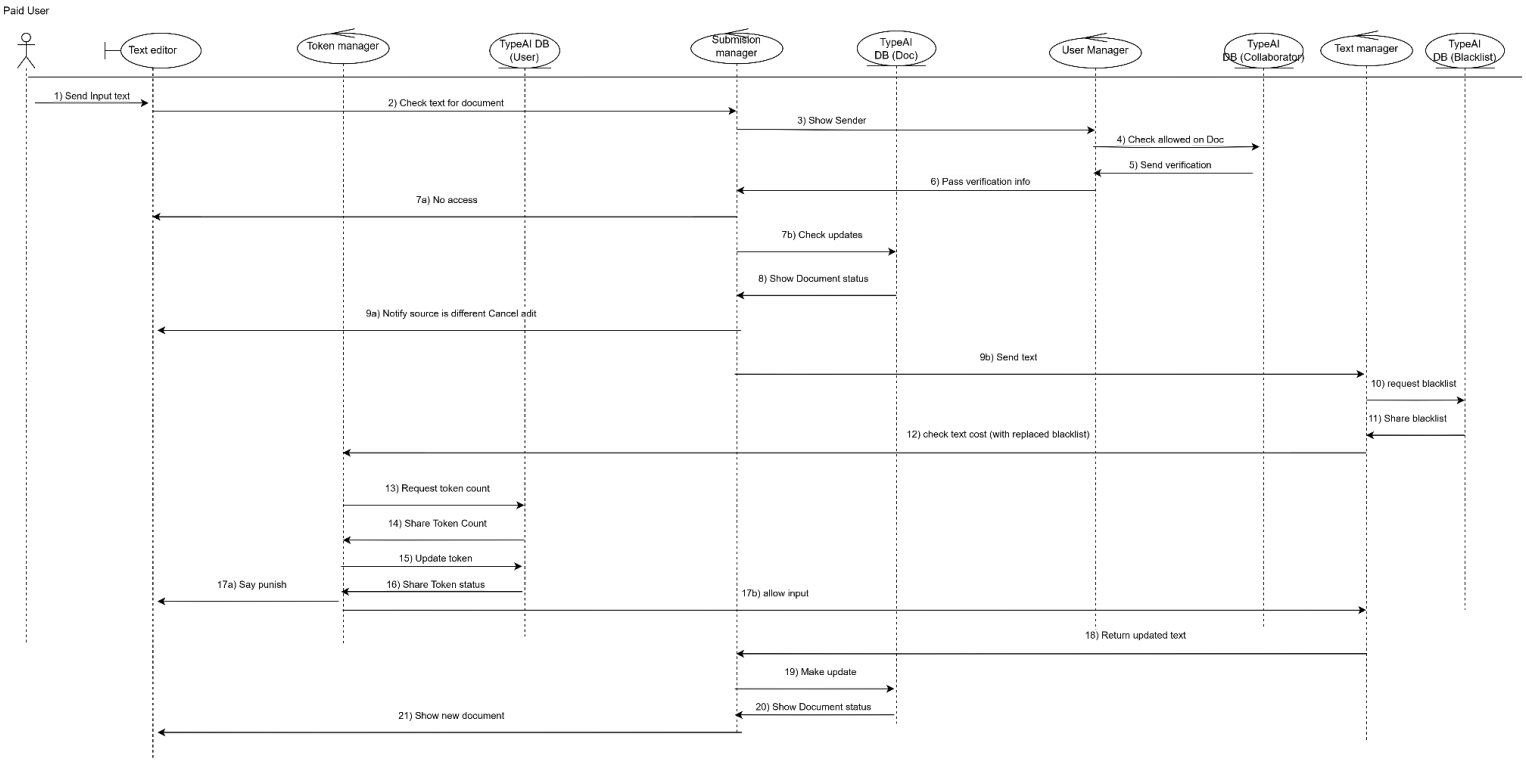
9. Input text (paid)

Normal:

A user inputs more text on an open document and the system will charge the number of tokens based on the number of words.

Exceptional:

- If the user doesn't have enough tokens to input text, the user will lose half of their remaining tokens and the text won't be considered.
- When inputting text, if there are blacklisted words, the text will not upload said word and instead substitute it with asterisks by the word length. The user will lose a token per asterisk



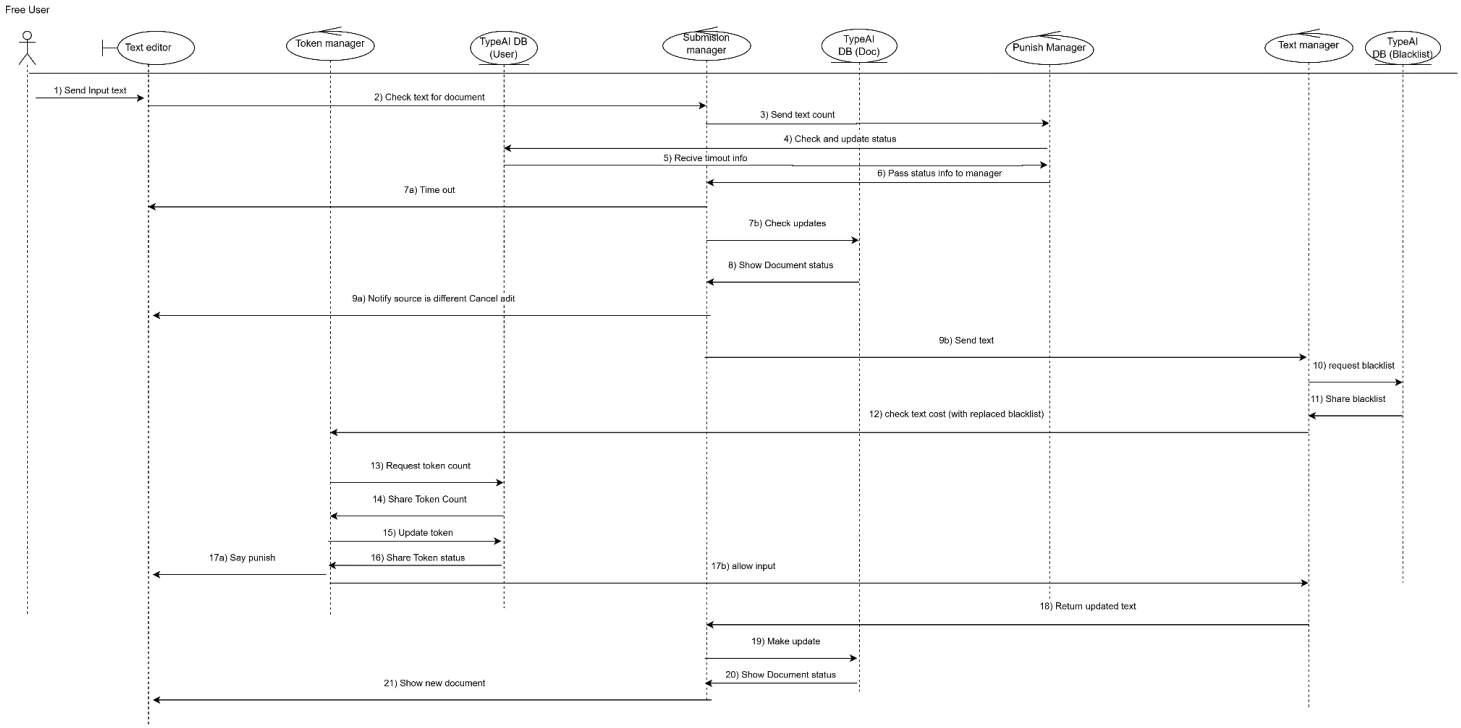
10. Input text (free)

Normal:

A user inputs more text on an open document and the system will charge the number of tokens based on the number of words.

Exceptional:

- Before next steps, more than 20 words, throw it out and timeout user
  - If the user doesn't have enough tokens to input text, the user will lose half of their remaining tokens and the text won't be considered.
  - When inputting text, if there are blacklisted words, the text will not upload said word and instead substitute it with asterisks by the word length. The user will lose a token per asterisk



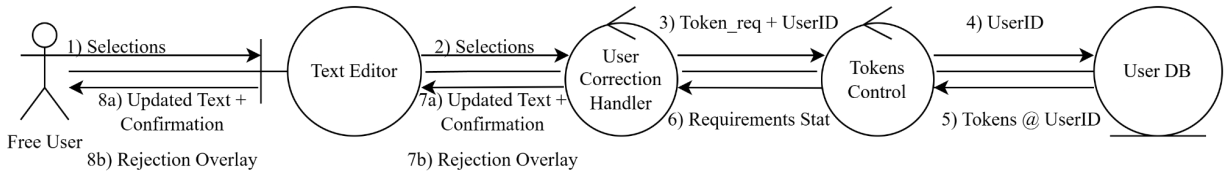
11. Text Correction (free, Self-correct)

Normal:

Free user chooses part of the text to edit or remove etc. Similar to the concept of git staging, they will add any changes they make to the text document, and for the total changes they make, they will press a send button to commit any changes. The changes will update the document, tokens are deducted by sent-words/2 tokens.

Exceptional:

- If the user sends any changes that go above double the amount of tokens they have, the transaction will be rejected. (I think this should just be removed since submit text already covers going over the amount submitted)
- If the update doesn't show, the user can submit a request to customer support for refund of tokens.



I need to go back and fix this for the token deduction I think ****

12. Text Correction (free, AI correction)

Normal:

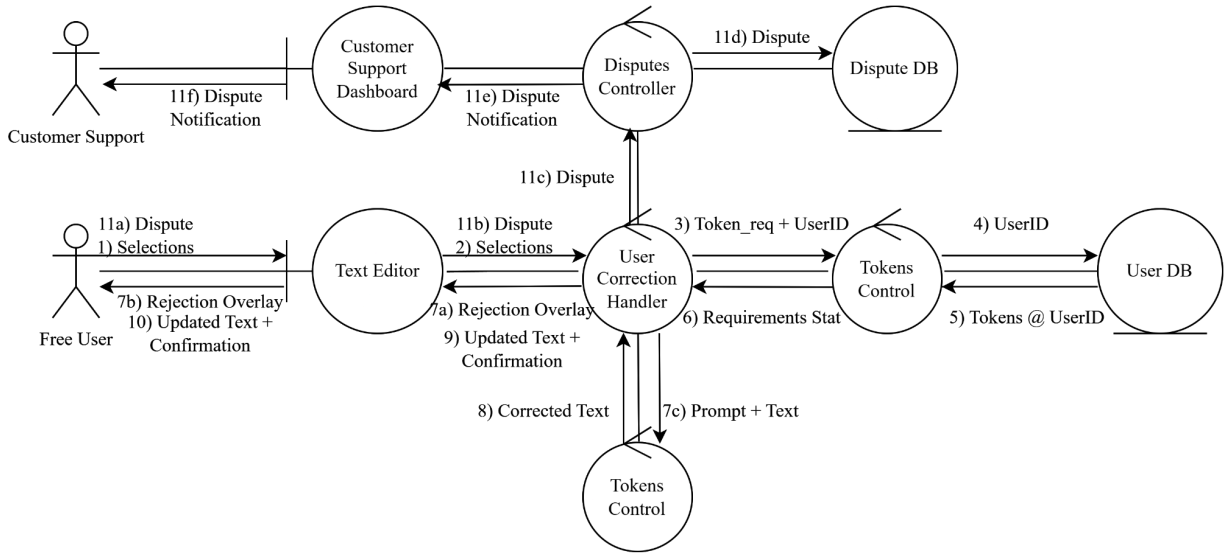
Free user chooses part of the text to edit or remove etc. Although its similar to the idea of git staging, the user will select text that they want to send to the AI. Once they have it, they will press the send to AI button, and the AI will do corrections. The AI will then send the corrections back and will show the user the new updated text. The User can then accept the changes and one token would be deducted.

Exceptional:

If the User rejects the change, 2 things will happen depending on what they do

- users are given the option to save the wrong word labeled by LLM as a correct one, so that later this word will not be highlighted anymore.
- User could outright reject the change and send a reason why the change shouldnt be made. Super user will decide with choose to accept rejection then deduct 1 token, or deny the rejection then deduct 5 tokens

If the update doesn't show, the user can submit a request to customer support for refund of tokens.



I need to go back and fix this for the token deduction I think

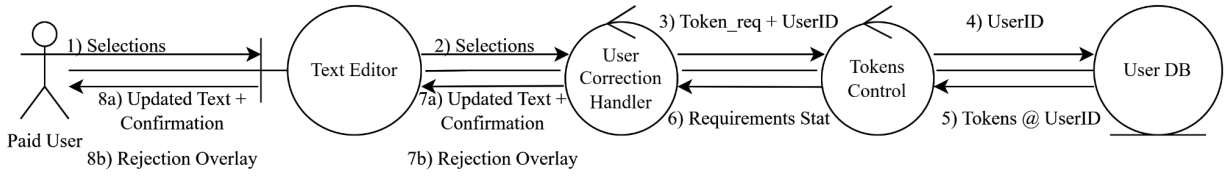
13. Text Correction (paid, Self-correct)

Normal:

Paid user chooses part of the text to edit or remove etc. Similar to the concept of git staging, they will add any changes they make to the text document, and for the total changes they make, they will press a send button to commit any changes. The changes will update the document, tokens are deducted by sent-words/2 tokens.

Exceptional:

- If the user sends any changes that go above double the amount of tokens they have, the transaction will be rejected. (I think this should just be removed since submit text already covers going over the amount submitted)
- If the update doesn't show, the user can submit a request to customer support for refund of tokens.



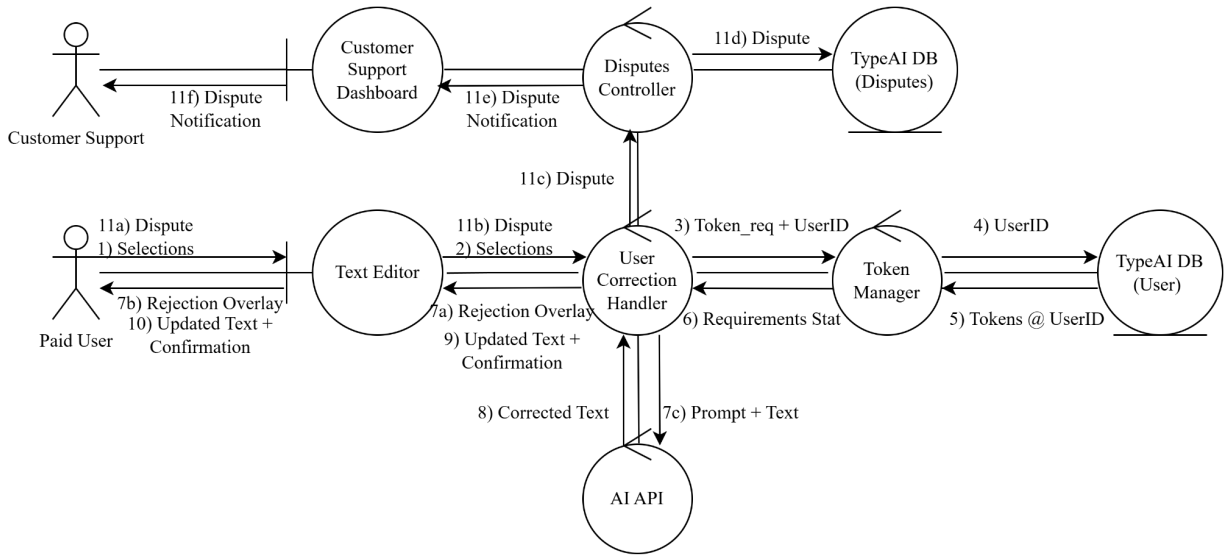
14. Text Correction (paid, AI correction)

Normal:

Paid user chooses part of the text to edit or remove etc. Although its similar to the idea of git staging, the user will select text that they want to send to the AI. Once they have it, they will press the send to AI button, and the AI will do corrections. The AI will then send the corrections back and will show the user the new updated text. The User can then accept the changes and one token would be deducted.

Exceptional:

- If the User rejects the change, 2 things will happen depending on what they do
    - users are given the option to save the wrong word labeled by LLM as a correct one, so that later this word will not be highlighted anymore.
    - User could outright reject the change and send a reason why the change shouldnt be made. Super user will decide with choose to accept rejection then deduct 1 token, or deny the rejection then deduct 5 tokens
- If the update doesn't show, the user can submit a request to customer support for refund of tokens.



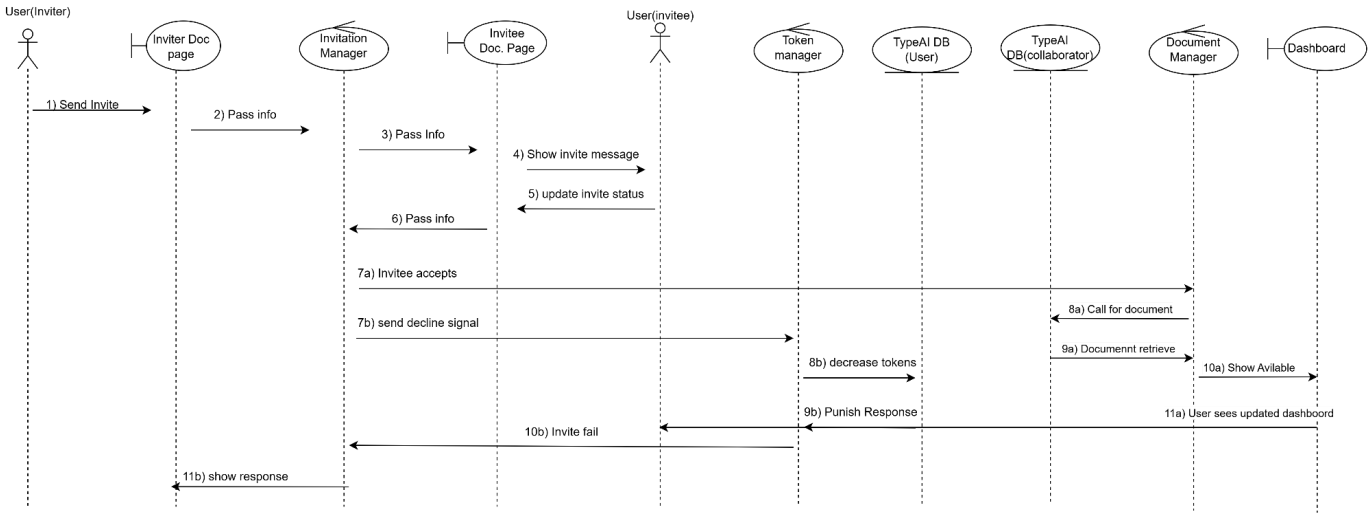
15. Collaboration

Normal:

Paid User sends an invitation to another paid user with a custom message. The invitation will be shown in the recipient's mailbox in which said recipient can accept or reject the invitation. When they click on accept, they have access to the document.

Exceptional:

- Invitee rejects the invitation, the inviter gets a 3 token penalty for reckless inviting. The message sent will be updated showing that the recipient rejected the invitation. On the side of the recipient it will be shown that they have rejected the invite



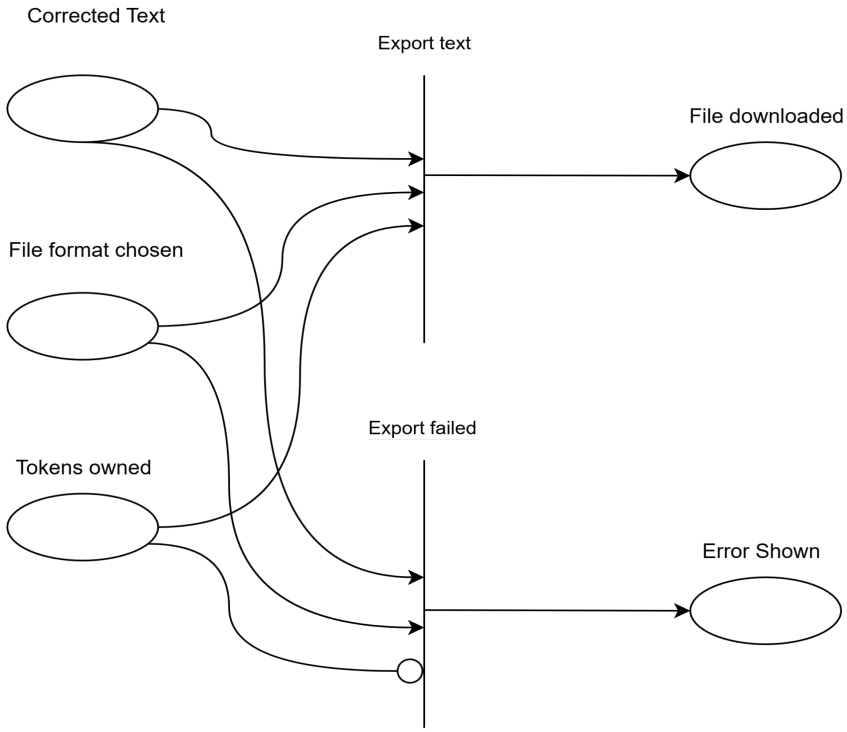
16. Save file text

Normal:

Given the user has some text that has been corrected (self/AI), they will press a button to export the text into a supported file type. The system will automatically start the download onto the user's device.

Exceptional:

If the user doesn't have enough tokens to pay for the export, an error message will be displayed, and no download will be done.



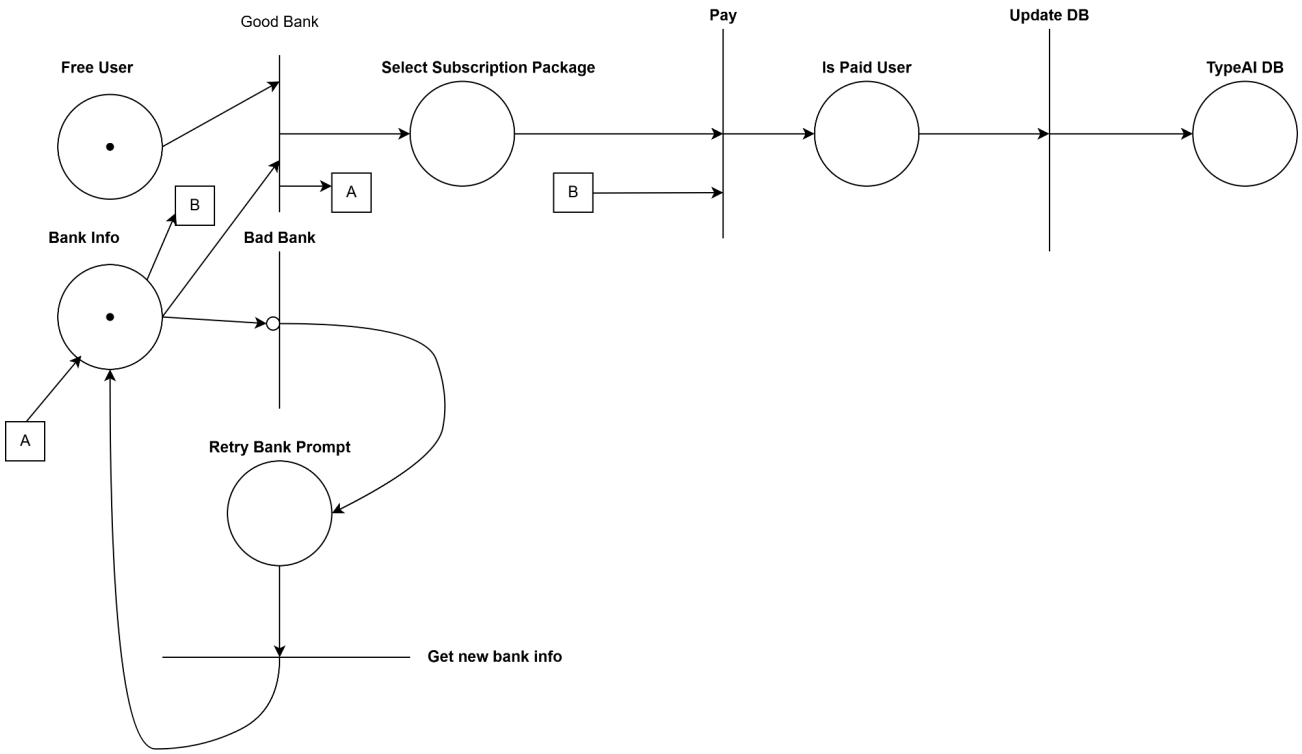
17. Upgrade

Normal:

Free users can sign up to be paid users. The User sends in the required information that is the same as needed for signup for a paid account. The Database updates the free user to a paid user.

Exceptional:

- Despite putting in correct information, they are not a paid user. They can contact customer support to update with transactions made to show that user is a paid user
- Paid User makes a bad password. Tell user to retry password and make it more secure
- Any issues with upgrading would follow any exceptions from signing up to a paid account.



18. Reward

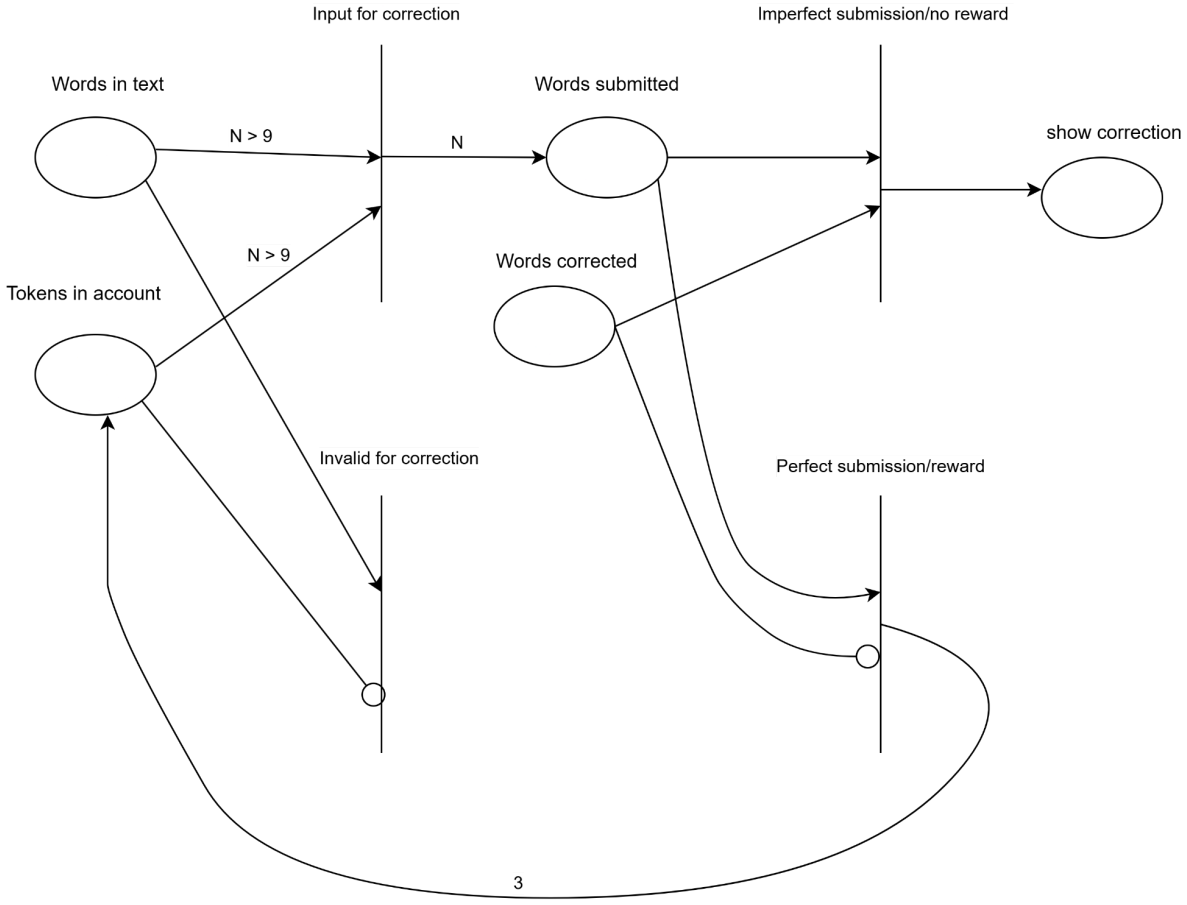
Normal:

When a text is submitted to the AI, the text contains at least 10 words, and the AI determines that there are no errors at all. The AI will continue its process as normal but will additionally add 3 tokens to the submitter's account.

Exceptional:

- There is no error, but no tokens are added. A support ticket can be submitted, and support will add 3 tokens to the account

(Put some diagram here: Petri-net) done on draw.io



Pseudo-Codes

Supporting functions:
```
FUNCTION IsValid(input)
    // Check for valid email and strong password
    RETURN input.email IS NOT NULL AND IsValidEmail(input.email) AND
           input.password IS NOT NULL AND IsStrong(input.password)
END FUNCTION

FUNCTION IsValidLoginInput(input)
    RETURN input.email IS NOT NULL AND input.password IS NOT NULL
END FUNCTION

FUNCTION IsStrong(password)
    RETURN LENGTH(password) ≥ 8 AND ContainsSymbol(password)
END FUNCTION

FUNCTION UserExists(email)
    RETURN QueryUserByEmail(email) IS NOT NULL
END FUNCTION

FUNCTION Hash(password)
    RETURN SecureHash(password)
END FUNCTION

FUNCTION SendVerification(email)
    token ← GenerateToken(email)
    link ← BuildLink(token)
    Email.Send(email, "Verify your account", link)
END FUNCTION

FUNCTION IsPaidUser(user)
    RETURN user.level == "PAID"
END FUNCTION

FUNCTION IsValidPurchaseInput(input)
    RETURN input IS NOT NULL AND input.amount > 0
END FUNCTION

FUNCTION FetchTransactionInfo(user)
    RETURN TypeAI_DB.GetUserTransactionInfo(user.id)
END FUNCTION
```

```
FUNCTION HandleTransaction(transactionInfo, purchaseInput)
  RETURN PaymentGateway.Process(transactionInfo, purchaseInput)
END FUNCTION

FUNCTION AddTokensToUser(user)
  tokensToAdd ← CalculateTokens(purchaseInput.amount)
  TypeAI_DB.AddTokens(user.id, tokensToAdd)
END FUNCTION

FUNCTION FetchUserFromDatabase(email, password)
  storedUser ← TypeAI_DB.FindUserByEmail(email)
  IF storedUser IS NULL THEN RETURN NULL
  IF NOT VerifyPassword(password, storedUser.passwordHash) THEN RETURN NULL
  RETURN storedUser
END FUNCTION

FUNCTION IsValidComplaint(input)
  RETURN input IS NOT NULL AND input.message ≠ ""
END FUNCTION

FUNCTION SaveComplaintToDatabase(user, input)
  complaintRecord ← {
    userId: user.id,
    message: input.message,
    timestamp: GetCurrentTimestamp()
  }
  TypeAI_DB.SaveComplaint(complaintRecord)
END FUNCTION

FUNCTION NotifyCustomerServiceAgent(user, input)
  notification ← {
    type: "New Complaint",
    fromUser: user.email,
    message: input.message,
    time: GetCurrentTimestamp()
  }
  CustomerServiceDashboard.SendNotification(notification)
END FUNCTION

FUNCTION FetchNextComplaint()
  RETURN TypeAI_DB.GetNextUnresolvedComplaint()
END FUNCTION

FUNCTION DisplayComplaintToRep(repId, complaint)
```

```
    CustomerServiceDashboard.ShowComplaint(repId, complaint)
END FUNCTION

FUNCTION WaitForRepDecision()
    DISPLAY "Does this complaint need admin attention? (YES / NO)"
    INPUT decision
    RETURN decision
END FUNCTION

FUNCTION ProcessRepDecision(complaint, decision)
    IF decision == "YES" THEN
        ForwardToAdminTeam(complaint)
    ELSE
        MarkComplaintAsResolved(complaint.id)
END FUNCTION

FUNCTION ForwardToAdminTeam(complaint)
    AdminDashboard.Notify({
        type: "Escalated Complaint",
        complaintId: complaint.id,
        message: complaint.message,
        userId: complaint.userId
    })
END FUNCTION

FUNCTION MarkComplaintAsResolved(complaintId)
    TypeAI_DB.UpdateComplaintStatus(complaintId, "RESOLVED_BY_REP")
END FUNCTION

FUNCTION CheckForAdminNotification()
    RETURN AdminDashboard.GetNextNotification(type = "Escalated Complaint")
END FUNCTION

FUNCTION FetchComplaintFromDatabase(complaintId)
    RETURN TypeAI_DB.GetComplaintById(complaintId)
END FUNCTION

FUNCTION DisplayComplaintToAdmin(adminId, complaint)
    AdminDashboard.ShowComplaint(adminId, complaint)
END FUNCTION

FUNCTION GetUsersOnWarningList()
    RETURN QueryDatabase("SELECT * FROM users WHERE violations >= 3 OR status = 'warning'")
END FUNCTION
```

```
FUNCTION ShouldBeDeleted(user)
   RETURN user.violations >= 5 OR user.status = "banned"
END FUNCTION

FUNCTION SuspendUser(userId)
   UpdateDatabase("UPDATE users SET status = 'suspended' WHERE id = ?", userId)
END FUNCTION

FUNCTION DeleteUser(userId)
   Execute("DELETE FROM users WHERE id = ?", userId)
END FUNCTION

FUNCTION Log(message)
   PrintToLog("[MODERATION] " + message)
END FUNCTION

FUNCTION IsValidWord(word)
   RETURN word IS NOT NULL AND Length(word) > 1 AND NOT ContainsProhibitedChars(word)
END FUNCTION

FUNCTION SavePendingWordRequest(userId, word)
   INSERT INTO pending_blacklist (user_id, word, status, created_at)
   VALUES(userId, word, "pending", CurrentTimestamp())
END FUNCTION

FUNCTION NotifyAdmins(message)
   SendToAdminQueue("blacklist_review", message)
END FUNCTION

FUNCTION CountWords(text)
   RETURN Length(Split(text, " "))
END FUNCTION

FUNCTION CalculateTokens(wordCount)
   RETURN wordCount
END FUNCTION

FUNCTION ChargeUser(userId, tokens)
   user ← GetUserById(userId)
   IF user.tokens < tokens THEN RETURN FALSE
   user.tokens ← user.tokens - tokens
   UpdateUser(user)
   RETURN TRUE
```

```
END FUNCTION

FUNCTION GetText(inputText, uploadedFile)
   IF inputText IS NOT NULL THEN RETURN inputText
   IF uploadedFile IS NOT NULL THEN RETURN FileReader.Read(uploadedFile)
   RETURN NULL
END FUNCTION

FUNCTION HasEnoughTokens(userId, cost)
   user ← GetUser(userId)
   RETURN user.tokens >= cost
END FUNCTION

FUNCTION ApplyTokenPenalty(userId)
   user ← GetUser(userId)
   penalty ← Floor(user.tokens / 2)
   user.tokens ← user.tokens - penalty
   UpdateUser(user)
END FUNCTION

FUNCTION DeductTokens(userId, amount)
   user ← GetUser(userId)
   user.tokens ← user.tokens - amount
   UpdateUser(user)
   RETURN TRUE
END FUNCTION

FUNCTION AppendToDocument(docId, text)
   DocumentService.Append(docId, text)
END FUNCTION

FUNCTION IsBlacklisted(word)
   RETURN word IN GetBlacklistedWords()
END FUNCTION

FUNCTION FilterBlacklistedWords(text)
   words ← Split(text, " ")
   asteriskCount ← 0

   FOR i FROM 0 TO Length(words) - 1 DO
      IF IsBlacklisted(words[i]) THEN
         length ← Length(words[i])
         words[i] ← Repeat("*", length)
         asteriskCount ← asteriskCount + length
```

```
      END IF
    END FOR

    RETURN Join(words, " "), asteriskCount
END FUNCTION

FUNCTION TimeoutUser(userId)
    // Suspend user temporarily or flag for cooldown
    UpdateUserStatus(userId, "timeout")
END FUNCTION

FUNCTION UpdateDocument(docId, newContent)
    RETURN DocumentService.Update(docId, newContent)
END FUNCTION

FUNCTION RecordFailedCommit(userId, docId, tokenCost)
    record ← {
        userId: userId,
        docId: docId,
        tokens: tokenCost,
        timestamp: GetCurrentTimestamp(),
        status: "pending_refund"
    }
    TypeAI_DB.LogFailedCommit(record)
END FUNCTION

FUNCTION SubmitRefundRequest(userId, docId)
    failed ← TypeAI_DB.GetLastFailedCommit(userId, docId)
    IF failed IS NOT NULL AND failed.status == "pending_refund" THEN
        NotifyCustomerSupport(userId, docId, failed.tokens)
        RETURN Success("Refund request submitted.")
    RETURN Error("No eligible failed transaction found.")
END FUNCTION

FUNCTION NotifyCustomerSupport(userId, docId, tokenAmount)
    message ← "User " + userId + " requests token refund of " + tokenAmount +
              " for failed update on document " + docId
    CustomerSupport.Notify(message)
END FUNCTION

FUNCTION DisplayCorrectionToUser(userId, original, corrected)
    UI.ShowCorrection(userId, original, corrected)
END FUNCTION
```

```
FUNCTION SaveToUserDictionary(userId, word)
  INSERT INTO user_dictionary (user_id, word)
END FUNCTION

FUNCTION SubmitRejectionForReview(userId, word, reason)
  rejection ← {
    userId: userId,
    word: word,
    reason: reason,
    status: "pending",
    timestamp: GetCurrentTimestamp()
  }
  TypeAI_DB.SaveRejection(rejection)
END FUNCTION

FUNCTION GetRejection(rejectionId)
  RETURN TypeAI_DB.GetRejectionById(rejectionId)
END FUNCTION

FUNCTION LogReviewOutcome(rejectionId, status, reviewerId)
  TypeAI_DB.UpdateRejectionStatus(rejectionId, status, reviewerId, GetCurrentTimestamp())
END FUNCTION

FUNCTION SaveInviteToMailbox(invite)
  TypeAI_DB.SaveInvite(invite)
END FUNCTION

FUNCTION GetInviteById(inviteId)
  RETURN TypeAI_DB.GetInvite(inviteId)
END FUNCTION

FUNCTION UpdateInviteStatus(inviteId, status)
  TypeAI_DB.UpdateInviteStatus(inviteId, status)
END FUNCTION

FUNCTION GrantDocumentAccess(docId, userId)
  DocumentPermissions.Grant(docId, userId)
END FUNCTION

FUNCTION NotifyInviterOfRejection(senderId, inviteId)
  updatedMessage ← "Your invitation was rejected. You have lost 3 tokens."
  Mailbox.UpdateMessage(senderId, inviteId, updatedMessage)
END FUNCTION
```

```
FUNCTION GetExportCost(fileType)
   // Example: base cost model
   SWITCH fileType
      CASE "txt": RETURN 1
      CASE "pdf": RETURN 2
      CASE "docx": RETURN 3
      DEFAULT: RETURN 2
END FUNCTION

FUNCTION GenerateExportFile(text, fileType)
   RETURN FileBuilder.Create(text, fileType)
END FUNCTION

FUNCTION StartDownload(file)
   FileService.InitiateDownload(file)
END FUNCTION

FUNCTION IsValidUpgradeInput(input)
   RETURN input.email IS NOT NULL AND IsValidEmail(input.email) AND
        input.paymentInfo IS NOT NULL AND input.password IS NOT NULL
END FUNCTION


```

    1. signup:

```
FUNCTION SignUp(input)
   // Validate user input
   IF NOT IsValid(input) THEN
      RETURN Error("Invalid input")

   // Check if user already exists
   IF UserExists(input.email) THEN
      RETURN Error("User already exists")

   // Hash the password for security
   hashedPassword ← Hash(input.password)

   // Save new user to the database
   user ← SaveUser(input.name, input.email, hashedPassword, input.level)

   // Send a verification email
   SendVerification(input.email)
```

```
  RETURN Success("Signup complete. Check your email.")
END FUNCTION

```
    2. Login
```
FUNCTION Login(userInput)
  IF NOT IsValidLoginInput(userInput) THEN
    RETURN Error("Fill out all fields")

  IF NOT IsValidEmail(userInput.email) THEN
    RETURN Error("Bad email.")

  IF NOT IsValidPassword(userInput.password) THEN
    RETURN Error("Bad password.")

  user ← FetchUserFromDatabase(userInput.email, userInput.password)

  IF user IS NULL THEN
    RETURN Error("Bad username or password")

  LoadUserDashboard(user)

  RETURN Success("Login successful")
END FUNCTION

```
    3. Purchase Tokens
```
FUNCTION PurchaseTokens(CurrentUser, purchaseInput)
  IF NOT IsPaidUser(CurrentUser) THEN
    RETURN Error("Only paid users can purchase tokens")

  IF NOT IsValidPurchaseInput(purchaseInput) THEN
    RETURN Error("Missing or invalid purchase information")

  transactionInfo ← FetchTransactionInfo(CurrentUser)

  successful ← HandleTransaction(transactionInfo, purchaseInput)

  IF NOT successful THEN
    RETURN Error("Transaction failed")

  AddTokensToUser(CurrentUser)
```

```
    RETURN Success("Tokens successfully added")
END FUNCTION

```
    4. Complaint:
```
//Paid User
FUNCTION CatchComplaint(complaintInput, CurrentUser)
    IF NOT IsValidComplaint(complaintInput) THEN
        RETURN Error("Complaint cannot be empty")

    SaveComplaintToDatabase(CurrentUser, complaintInput)

    NotifyCustomerServiceAgent(CurrentUser, complaintInput)

    RETURN Success("Your complaint has been submitted")
END FUNCTION

//Customer Service
FUNCTION HandleComplaintRequest(repId)
    complaint ← FetchNextComplaint()

    IF complaint IS NULL THEN
        RETURN Message("No new complaints at this time")

    DisplayComplaintToRep(repId, complaint)

    decision ← WaitForRepDecision()

    ProcessRepDecision(complaint, decision)

    RETURN Success("Complaint handled and logged")
END FUNCTION

//Admin view
FUNCTION HandleEscalatedComplaint(adminId)
    notification ← CheckForAdminNotification()

    IF notification IS NULL THEN
        RETURN Message("No complaints require admin attention")

    complaintId ← notification.complaintId
```

```
    complaint ← FetchComplaintFromDatabase(complaintId)

    DisplayComplaintToAdmin(adminId, complaint)

    RETURN Success("Complaint loaded onto dashboard")
END FUNCTION

```

5. Terminate / Suspend:

```
FUNCTION SuspendFlaggedUsers()
    // Get list of users with multiple violations
    flaggedUsers ← GetUsersOnWarningList()

    FOR EACH user IN flaggedUsers DO
        IF ShouldBeDeleted(user) THEN
            DeleteUser(user.id)
        ELSE
            SuspendUser(user.id)
        END
     END

    Log("Processed " + Count(flaggedUsers) + " flagged users")
END FUNCTION

```

6. Blacklist words:

```
FUNCTION RequestBlacklistWord(userId, word)
    // Validate the word
    IF NOT IsValidWord(word) THEN
        RETURN Error("Invalid or empty word")

    // Save the request in a pending list
    SavePendingWordRequest(userId, word)

    // Notify admins for review
    NotifyAdmins("New word blacklist request: '" + word + "' from user " + userId)

    RETURN Success("Your request has been submitted for review")
END FUNCTION

FUNCTION ReviewBlacklistRequest(requestId, approved)
```

```
    request ← GetPendingRequestById(requestId)

    IF approved THEN
        AddToBlacklist(request.word)
        UpdateRequestStatus(requestId, "approved")
    ELSE
        UpdateRequestStatus(requestId, "rejected")
    END IF
END FUNCTION
```

7. Start text (paid)

```
FUNCTION ProcessUserInput(userId, inputText, uploadedFile)
    // Get text from input or file
    IF inputText IS NOT NULL THEN
        text ← inputText
    ELSE IF uploadedFile IS NOT NULL THEN
        text ← ReadFile(uploadedFile)
    ELSE
        RETURN Error("No input provided")
    END IF

    // Count words
    wordCount ← CountWords(text)

    // Calculate token cost
    tokens ← CalculateTokens(wordCount)

    // Charge user
    success ← ChargeUser(userId, tokens)

    IF NOT success THEN
        RETURN Error("Insufficient tokens")
    END IF

    RETURN Success("Charged " + tokens + " tokens for " + wordCount + " words")
END FUNCTION

```

8. Start text (free)

```
FUNCTION ChargeTokensForText(userId, inputText, uploadedFile)
    // Get text from input or file
    text ← GetText(inputText, uploadedFile)
```

```
    IF text IS NULL THEN
        RETURN Error("No text provided")

    // Count words and calculate token cost
    wordCount ← CountWords(text)
    tokens ← wordCount  // 1 token per word

    // Attempt to charge user
    IF NOT DeductTokens(userId, tokens) THEN
        RETURN Error("Not enough tokens")

    RETURN Success("Charged " + tokens + " tokens for " + wordCount + " words")
END FUNCTION

```

9. Input text (paid)

```
FUNCTION AddTextToDocument(userId, documentId, newText)
    // Filter blacklisted words and count replacements
    filteredText, asteriskCount ← FilterBlacklistedWords(newText)

    // Count valid words after filtering
    wordCount ← CountWords(filteredText)

    // Calculate total token cost
    tokenCost ← wordCount + asteriskCount

    // Try to deduct tokens
    IF NOT HasEnoughTokens(userId, tokenCost) THEN
        // Apply penalty: remove half of remaining tokens
        ApplyTokenPenalty(userId)
        RETURN Error("Not enough tokens. Half your tokens were deducted as penalty.")
    END IF

    DeductTokens(userId, tokenCost)

    // Append filtered text to the document
    AppendToDocument(documentId, filteredText)

    RETURN Success("Added text. Charged " + tokenCost + " tokens.")
END FUNCTION

```

10. Input text (free)

```
FUNCTION AddTextToOpenDocument(userId, docId, newText)
    wordCount ← CountWords(newText)

    // Reject and timeout user if too many words
    IF wordCount > 20 THEN
        TimeoutUser(userId)
        RETURN Error("Input exceeds word limit. User timed out.")

    // Filter blacklisted words, replacing with asterisks
    filteredText, asteriskCount ← FilterBlacklistedWords(newText)

    tokenCost ← wordCount + asteriskCount

    // Check if user can afford token cost
    IF NOT HasEnoughTokens(userId, tokenCost) THEN
        ApplyTokenPenalty(userId)
        RETURN Error("Insufficient tokens. Half your tokens have been deducted.")

    // Deduct total cost and append text
    DeductTokens(userId, tokenCost)
    AppendToDocument(docId, filteredText)

    RETURN Success("Text added. Charged " + tokenCost + " tokens.")
END FUNCTION


```
11. Text Correction (free, Self-correct)
```
FUNCTION CommitStagedChanges(userId, docId, stagedText)
    wordCount ← CountWords(stagedText)
    tokenCost ← Floor(wordCount / 2)

    // Check token balance
    IF NOT HasEnoughTokens(userId, tokenCost) THEN
        RETURN Error("Not enough tokens to commit these changes.")

    // Attempt to update the document
    success ← UpdateDocument(docId, stagedText)

    IF NOT success THEN
        RecordFailedCommit(userId, docId, tokenCost)
        RETURN Error("Update failed. You may request a refund.")
```

```
    // Deduct tokens after successful update
    DeductTokens(userId, tokenCost)

    RETURN Success("Changes committed. " + tokenCost + " tokens deducted.")
END FUNCTION

```
    12. Text Correction (free, AI correction)
```
FUNCTION SendToAICorrection(userId, selectedText)
    suggestedText ← AI.GenerateCorrection(selectedText)

    IF suggestedText IS NULL THEN
        RETURN Error("AI correction failed. Please try again.")

    DisplayCorrectionToUser(userId, selectedText, suggestedText)

    RETURN Success("Correction received. Please review and respond.")
END FUNCTION

FUNCTION AcceptCorrection(userId, docId, correctedText)
    success ← UpdateDocument(docId, correctedText)

    IF NOT success THEN
        RecordFailedCorrection(userId, docId, 1)
        RETURN Error("Update failed. You may request a refund.")

    DeductTokens(userId, 1)
    RETURN Success("Correction accepted. 1 token deducted.")
END FUNCTION

FUNCTION RejectCorrection(userId, originalWord, reason, option)
    IF option == "save_word" THEN
        SaveToUserDictionary(userId, originalWord)
        RETURN Success("Word saved as valid. No token deducted.")

    ELSE IF option == "send_rejection" THEN
        SubmitRejectionForReview(userId, originalWord, reason)
        RETURN Success("Rejection submitted to superuser.")

    RETURN Error("Invalid rejection option.")
END FUNCTION
```

```
FUNCTION ReviewRejection(superUserId, rejectionId, decision)
    rejection ← GetRejection(rejectionId)

    IF decision == "accept" THEN
        DeductTokens(rejection.userId, 1)
        LogReviewOutcome(rejectionId, "accepted", superUserId)
        RETURN Success("Rejection accepted. 1 token deducted.")

    ELSE IF decision == "deny" THEN
        DeductTokens(rejection.userId, 5)
        LogReviewOutcome(rejectionId, "denied", superUserId)
        RETURN Success("Rejection denied. 5 tokens deducted.")

    RETURN Error("Invalid decision.")
END FUNCTION

```

13. Text Correction (paid, Self-correct)

```
FUNCTION CommitPaidUserChanges(userId, docId, stagedText)
    wordCount ← CountWords(stagedText)
    tokenCost ← Floor(wordCount / 2)

    // Check if user can afford the change
    IF NOT HasEnoughTokens(userId, tokenCost) THEN
        RETURN Error("Not enough tokens to commit changes.")

    // Attempt to apply changes
    success ← UpdateDocument(docId, stagedText)

    IF NOT success THEN
        RecordFailedCommit(userId, docId, tokenCost)
        RETURN Error("Update failed. You may request a refund.")

    // Deduct tokens if update succeeded
    DeductTokens(userId, tokenCost)

    RETURN Success("Changes committed. " + tokenCost + " tokens deducted.")
END FUNCTION

```

14. Text Correction (paid, AI correction)

```
FUNCTION SendToAICorrection_PaidUser(userId, selectedText)
```

```
    suggestedText ← AI.GenerateCorrection(selectedText)

    IF suggestedText IS NULL THEN
        RETURN Error("AI failed to generate correction.")

    DisplayCorrectionToUser(userId, selectedText, suggestedText)

    RETURN Success("Correction generated. Awaiting user action.")
END FUNCTION

FUNCTION AcceptCorrection_Paid(userId, docId, correctedText)
    success ← UpdateDocument(docId, correctedText)

    IF NOT success THEN
        RecordFailedCorrection(userId, docId, 1)
        RETURN Error("Update failed. Refund request available.")

    DeductTokens(userId, 1)
    RETURN Success("Correction accepted. 1 token deducted.")
END FUNCTION

FUNCTION RejectCorrection_Paid(userId, originalWord, reason, option)
    IF option == "save_word" THEN
        SaveToUserDictionary(userId, originalWord)
        RETURN Success("Word saved to your dictionary. No token charged.")

    ELSE IF option == "submit_rejection" THEN
        SubmitRejectionForReview(userId, originalWord, reason)
        RETURN Success("Rejection sent to superuser for review.")

    RETURN Error("Invalid rejection option.")
END FUNCTION

FUNCTION ReviewRejection(superUserId, rejectionId, decision)
    rejection ← GetRejection(rejectionId)

    IF decision == "accept" THEN
        DeductTokens(rejection.userId, 1)
        LogReviewOutcome(rejectionId, "accepted", superUserId)
        RETURN Success("Rejection accepted. 1 token deducted.")

    ELSE IF decision == "deny" THEN
        DeductTokens(rejection.userId, 5)
        LogReviewOutcome(rejectionId, "denied", superUserId)
```

```
      RETURN Success("Rejection denied. 5 tokens deducted.")

  RETURN Error("Invalid review decision.")
END FUNCTION
```

15. Collaboration

```
FUNCTION SendDocumentInvite(senderId, recipientId, docId, message)
  IF NOT IsPaidUser(senderId) OR NOT IsPaidUser(recipientId) THEN
    RETURN Error("Both users must be paid members.")

  invite ← {
    senderId: senderId,
    recipientId: recipientId,
    docId: docId,
    message: message,
    status: "pending",
    sentAt: GetCurrentTimestamp()
  }

  SaveInviteToMailbox(invite)
  RETURN Success("Invitation sent to recipient.")
END FUNCTION

FUNCTION AcceptInvite(inviteId, recipientId)
  invite ← GetInviteById(inviteId)

  IF invite.recipientId ≠ recipientId OR invite.status ≠ "pending" THEN
    RETURN Error("Invalid invitation.")

  GrantDocumentAccess(invite.docId, recipientId)
  UpdateInviteStatus(inviteId, "accepted")
  RETURN Success("You now have access to the document.")
END FUNCTION

FUNCTION RejectInvite(inviteId, recipientId)
  invite ← GetInviteById(inviteId)

  IF invite.recipientId ≠ recipientId OR invite.status ≠ "pending" THEN
    RETURN Error("Invalid invitation.")

  UpdateInviteStatus(inviteId, "rejected")
  DeductTokens(invite.senderId, 3)
```

```
  NotifyInviterOfRejection(invite.senderId, inviteId)
  RETURN Success("Invitation rejected. Sender penalized 3 tokens.")
END FUNCTION
```

16. Save file text

```
FUNCTION ExportCorrectedText(userId, text, fileType)
  tokenCost ← GetExportCost(fileType)

  IF NOT HasEnoughTokens(userId, tokenCost) THEN
    RETURN Error("Not enough tokens to export file.")

  file ← GenerateExportFile(text, fileType)
  StartDownload(file)
  DeductTokens(userId, tokenCost)

  RETURN Success("Export successful. " + tokenCost + " tokens deducted.")
END FUNCTION
```

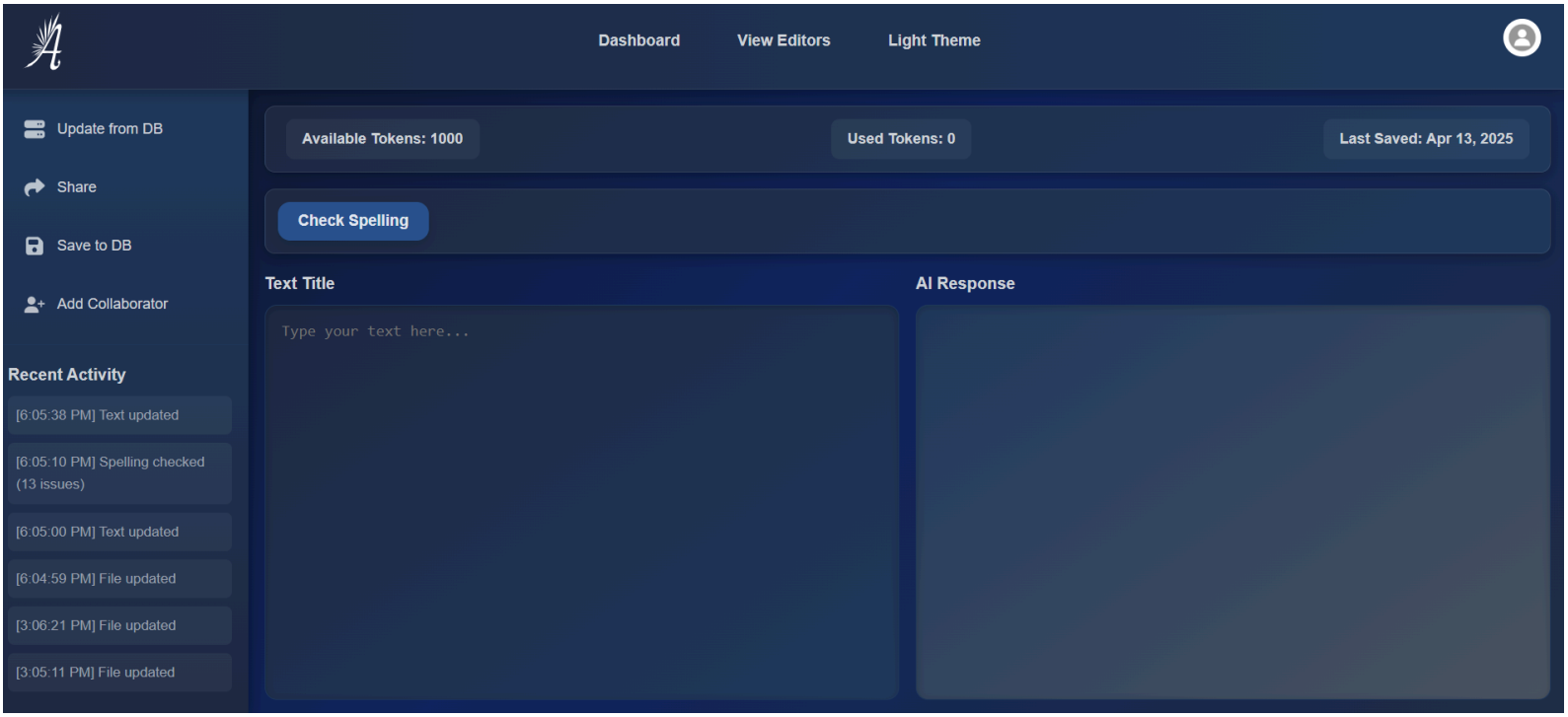
17. Upgrade

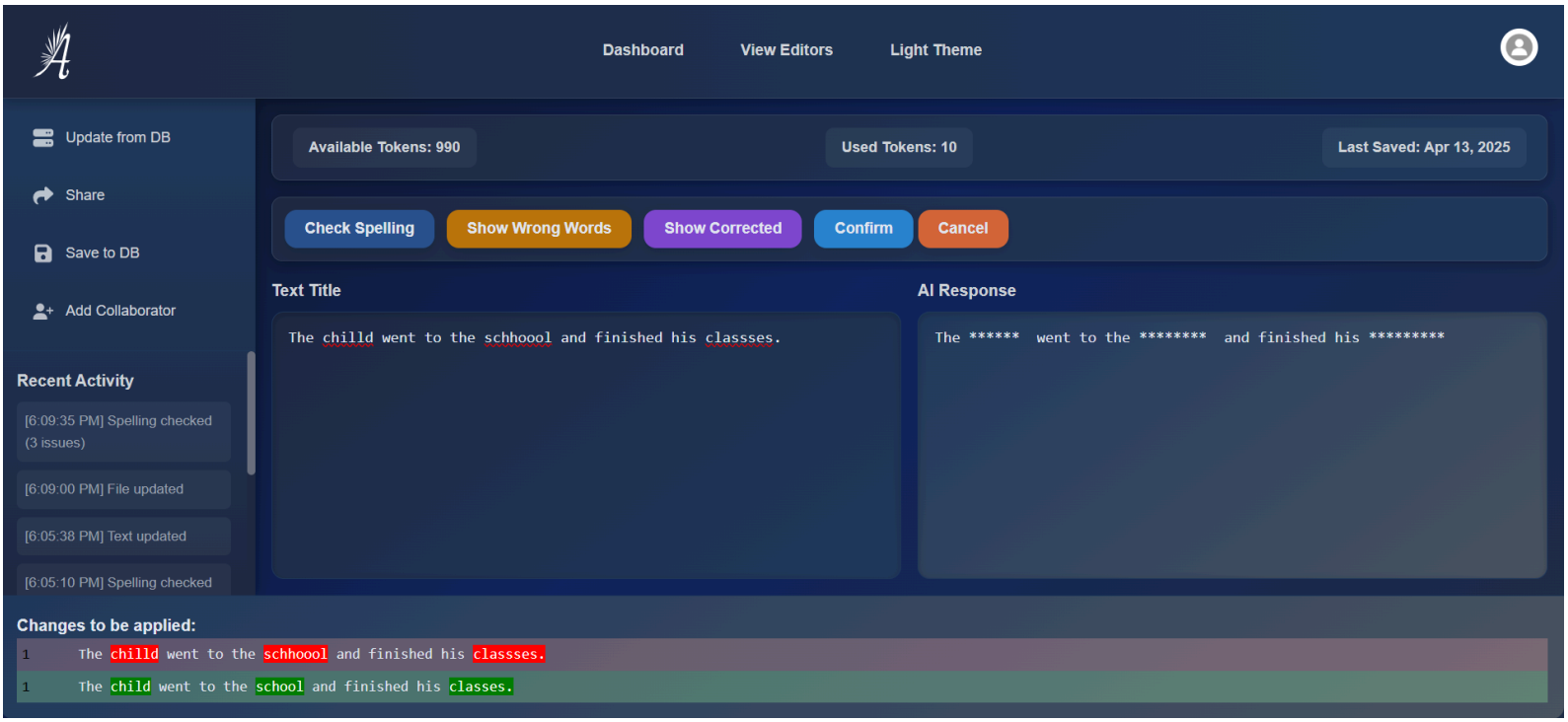
```
FUNCTION UpgradeFreeUserToPaid(userId, input)
  IF NOT IsValidUpgradeInput(input) THEN
    RETURN Error("Invalid input. Please retry.")

  IF NOT IsStrong(input.password) THEN
    RETURN Error("Password too weak. Use a stronger one.")

  success ← UpdateUserToPaid(userId, input)

  IF NOT success THEN
    RETURN Info("Upgrade failed. Contact support with transaction proof.")

  RETURN Success("You are now a paid user!")
END FUNCTION

FUNCTION ContactSupportForUpgrade(userId, transactionDetails)
  message ← {
    userId: userId,
    issue: "Paid upgrade failed",
    details: transactionDetails,
    timestamp: GetCurrentTimestamp()
```

```
    }

    CustomerSupport.Notify(message)
    RETURN Success("Support request sent.")
END FUNCTION

```
    18. Reward
```
FUNCTION SubmitTextToAI(userId, text)
    wordCount ← CountWords(text)

    IF wordCount < 10 THEN
        RETURN Error("Text must have at least 10 words.")

    errors ← AI.CheckForErrors(text)

    IF IsEmpty(errors) THEN
        rewardAdded ← AddTokens(userId, 3)

        IF NOT rewardAdded THEN
            RETURN Info("No errors found. Tokens not added. Contact support if needed.")
        END IF

        RETURN Success("No errors found. You've earned 3 bonus tokens.")
    END IF

    corrected ← AI.ProcessText(text)
    RETURN Success("AI processing complete with corrections.")
END FUNCTION

FUNCTION SubmitNoRewardSupportTicket(userId, text)
    ticket ← {
        userId: userId,
        issue: "No reward for error-free text",
        submittedText: text,
        createdAt: GetCurrentTimestamp()
    }

    SupportDesk.CreateTicket(ticket)
    RETURN Success("Support ticket submitted. 3 tokens will be reviewed.")
END FUNCTION
```

Gui Screenshot

(*Refer to Github Repo for code and implementation of features*)
(features: text typo check and text correction)





Meeting memos

4/8 - Team building and discussing requirements for phase 2
4/10 - Began formulating diagram setup for ER and class diagrams. Also implemented AI integration for text correction
4/11 - Discussed database setup for Django and implemented codebase
4/12 - Finished all class diagrams, Petri-nets, and main GUI for text editing page
4/13 - Finished all pseudo-code for features and polished final document

Appendix/resources

rabbit0227/TypeAI- Git repo link

https://drive.google.com/file/d/14kzTqq3VDunSK4XnlTT4Itv3rqHRuhJq/view?usp=drive_link
(draw.io diagrams)