Team S

LLMscribe
Software Design Report
For an LLM-based cooperative editor

Version <2.0>

Revision History

Date	Version	Description	Author
03/18/2025	1.0	1st Phase Software Requirements Specification Report of LLM-based cooperative editor project	Sean Jenkins, Adama Faye, Rajiv Seeram, Nischal Thapa, Junaet Mahbub
04/13/2025	2.0	2nd Phase Design Report of LLM-based cooperative editor project	Sean Jenkins, Adama Faye, Rajiv Seeram, Nischal Thapa, Junaet Mahbub

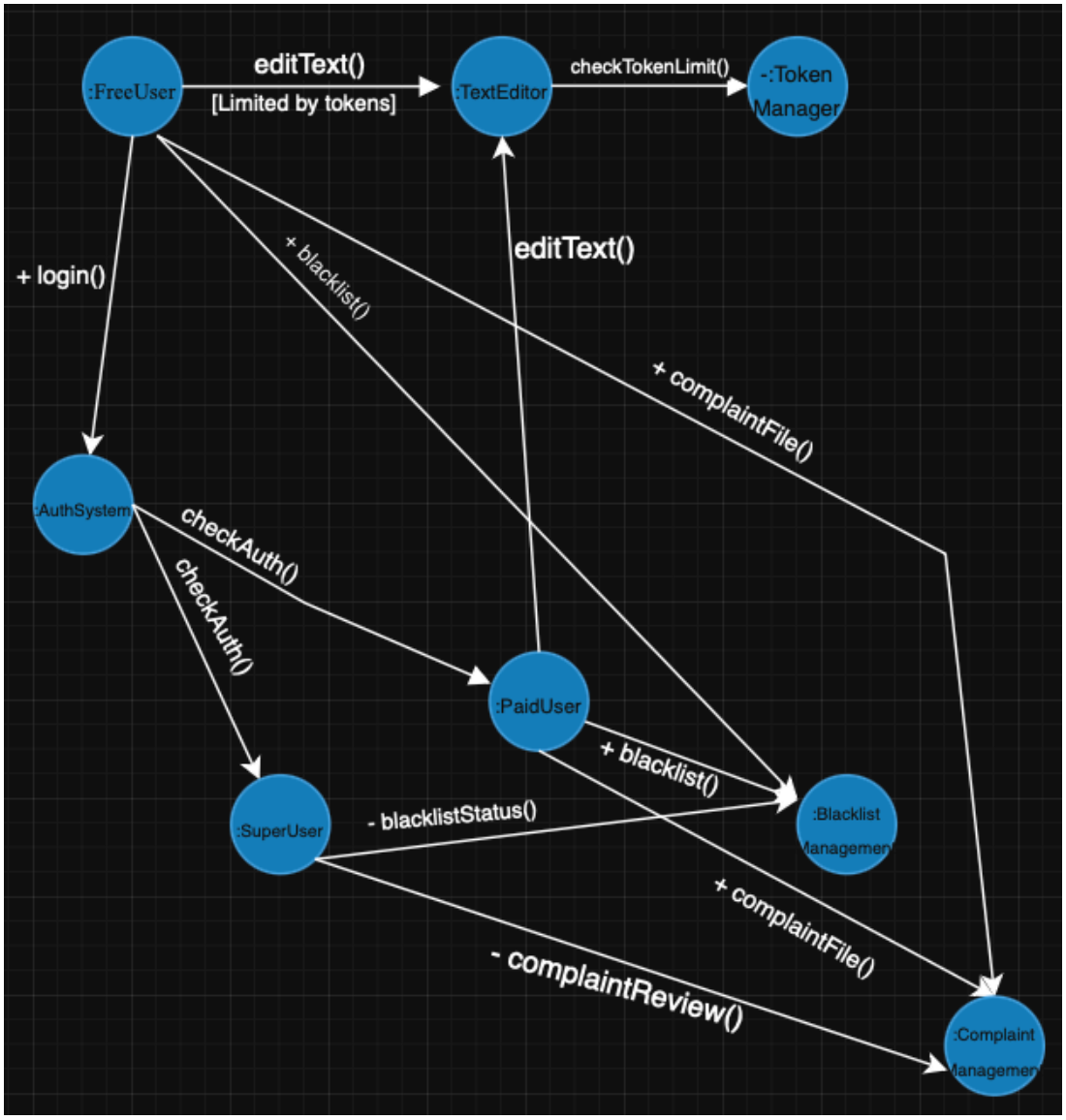
Table of Contents

Table of Contents

1. Introduction

This section provides a high-level overview of the system, including its structure, main components, and how they interact. It introduces the overall architecture through a collaboration class diagram, which outlines the relationships and message flow between system objects. The purpose of this section is to give readers a foundational understanding of the system's design philosophy and operational framework before delving into specific functionalities, data structures, and implementation details.

1.1 Overview of the System Using a Collaboration Class Diagram



2. Use Cases

This section outlines the system's core functionalities through a series of detailed use cases. Each use case includes normal (expected) and exceptional (edge/error) scenarios to demonstrate how the system behaves under varying conditions. To enhance understanding, each use case is accompanied by either a collaboration or sequence diagram that visually illustrates the interaction between system components during the process. Additionally, for at least three selected use cases, Petri-net models are presented to depict the dynamic behavior and state transitions within the system in a formal and structured way.

2.1 Use-Case Scenarios

Use Case: User Authentication

Actors: Free Users, Paid Users, Super Users
Description: Users authenticate using a login system that differentiates roles.
Preconditions: Users must have valid credentials (except for Free Users who do not require login).
Flow of Events:
1. User enters username and password (if applicable).
2. System verifies credentials.
3. Users are granted access based on their role.

Postconditions: User gains appropriate access rights.

Use Case: Text Submission

Actors: Free Users, Paid Users
Description: Users submit texts for LLM-based correction.
Preconditions: Users must be logged in (except for Free Users), and Paid Users must have enough tokens.
Flow of Events:
1. User inputs text in the editor.
2. System checks word count (limit for Free Users, token calculation for Paid Users).
3. If within the limit, the text is processed; otherwise, the submission fails with penalties.

Postconditions: Text is submitted for correction.

Use Case: Text Correction

Actors: Paid Users, LLM
Description: Users choose self-correction or LLM-assisted correction.
Preconditions: Submitted text must be available.
Flow of Events:

1.

User selects the correction method.

2. If self-correction, the system deducts half the required tokens.
3. If LLM correction, AI highlights errors for user approval.
4. User accepts/rejects corrections.

Postconditions: The corrected text is updated.

Use Case: Token Management

Actors: Paid Users
Description: Paid users manage their tokens.
Preconditions: User must be logged in as a Paid User.
Flow of Events:

1. User checks token balance.
2. User purchases additional tokens if needed.
3. System updates token balance.

Postconditions: Token balance reflects transactions.

Use Case: Blacklist Management

Actors: Free Users, Paid Users, Super Users
Description: Users suggest blacklist words; Super Users approve/reject them.
Preconditions: User must be logged in.
Flow of Events:

1. Free/Paid Users submit words for blacklisting.
2. Super Users review and decide.
3. System updates blacklist.

Postconditions: Blacklist is updated accordingly.

Use Case: Complaint Handling

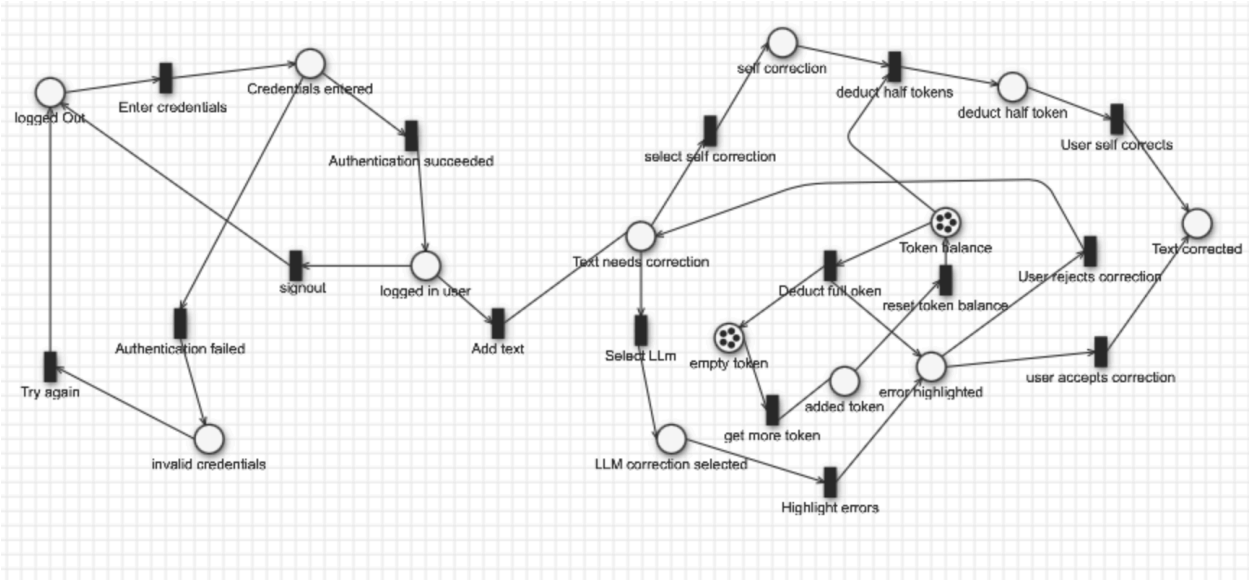
Actors: Paid Users, Super Users
Description: Users submit complaints about collaboration or system issues.
Preconditions: User must have a dispute.
Flow of Events:

1. Paid User files a complaint.
2. Super User reviews the case.
3. Super User issues a resolution (penalty, warning, dismissal).

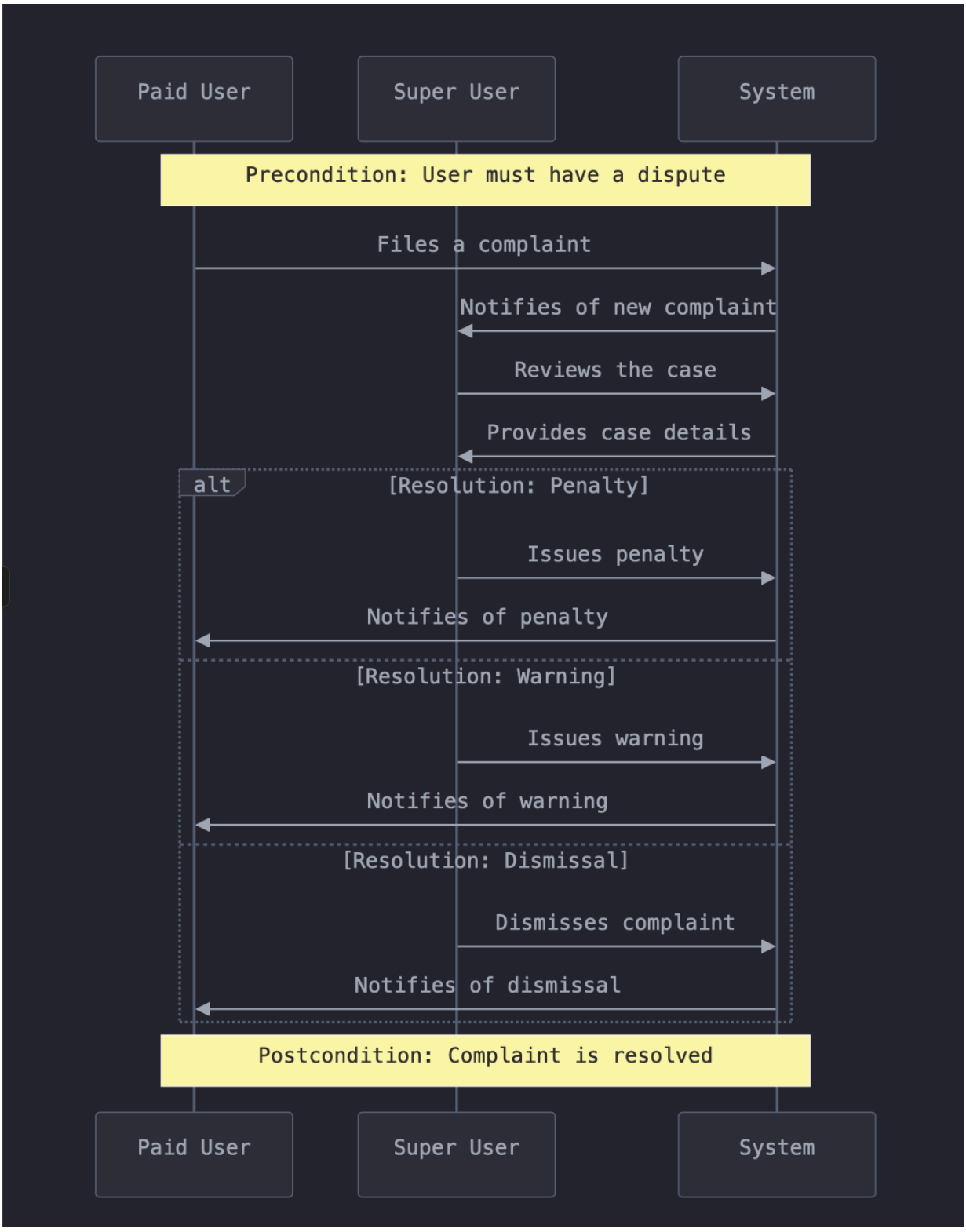
Postconditions: Complaint is resolved.

2.2 Diagram Visualization for Use Cases

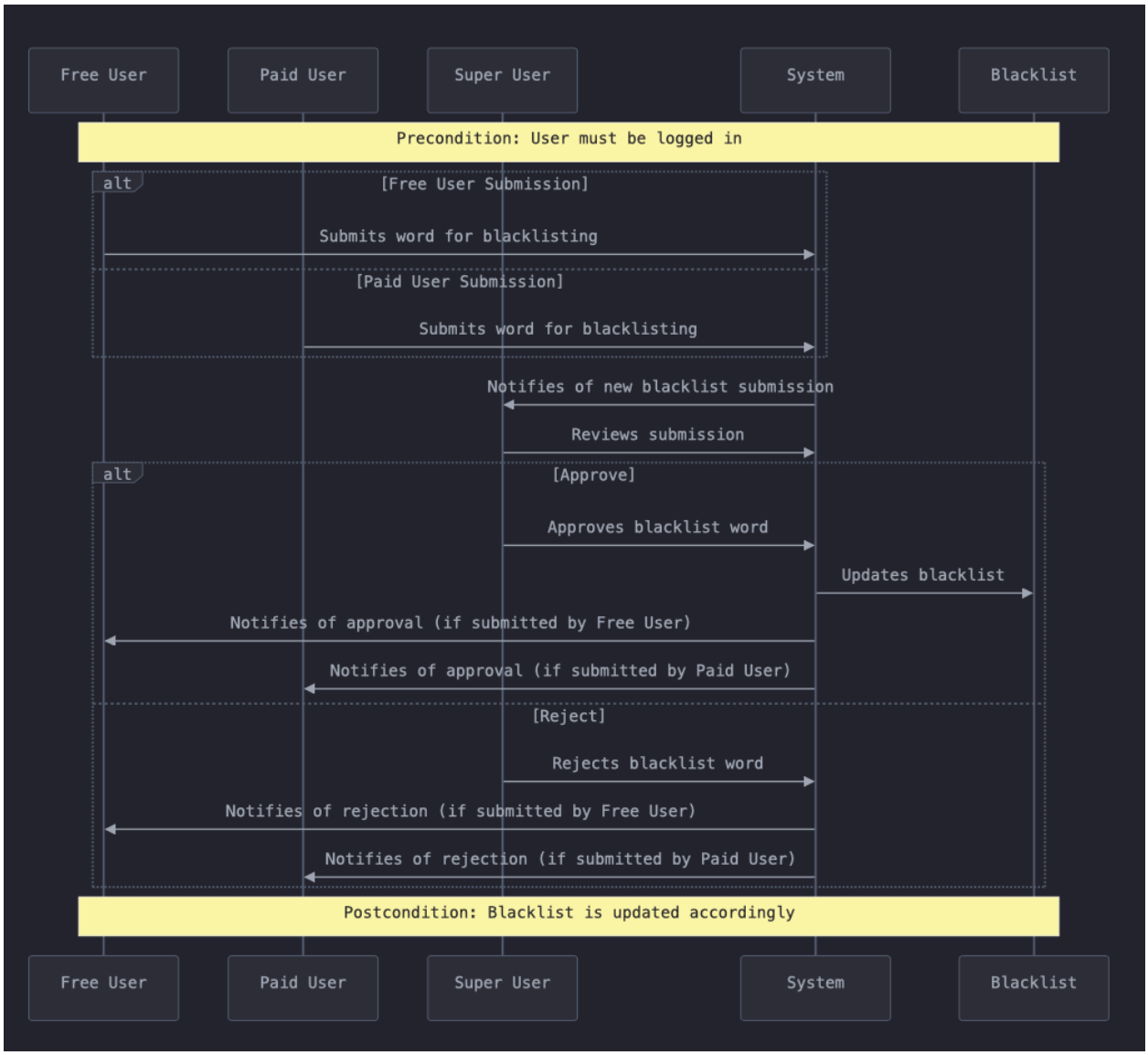
Petri net for use cases: User Authentication, Text Correction, Text Submission, and Token Management



Sequential Diagram for Complaint Handling



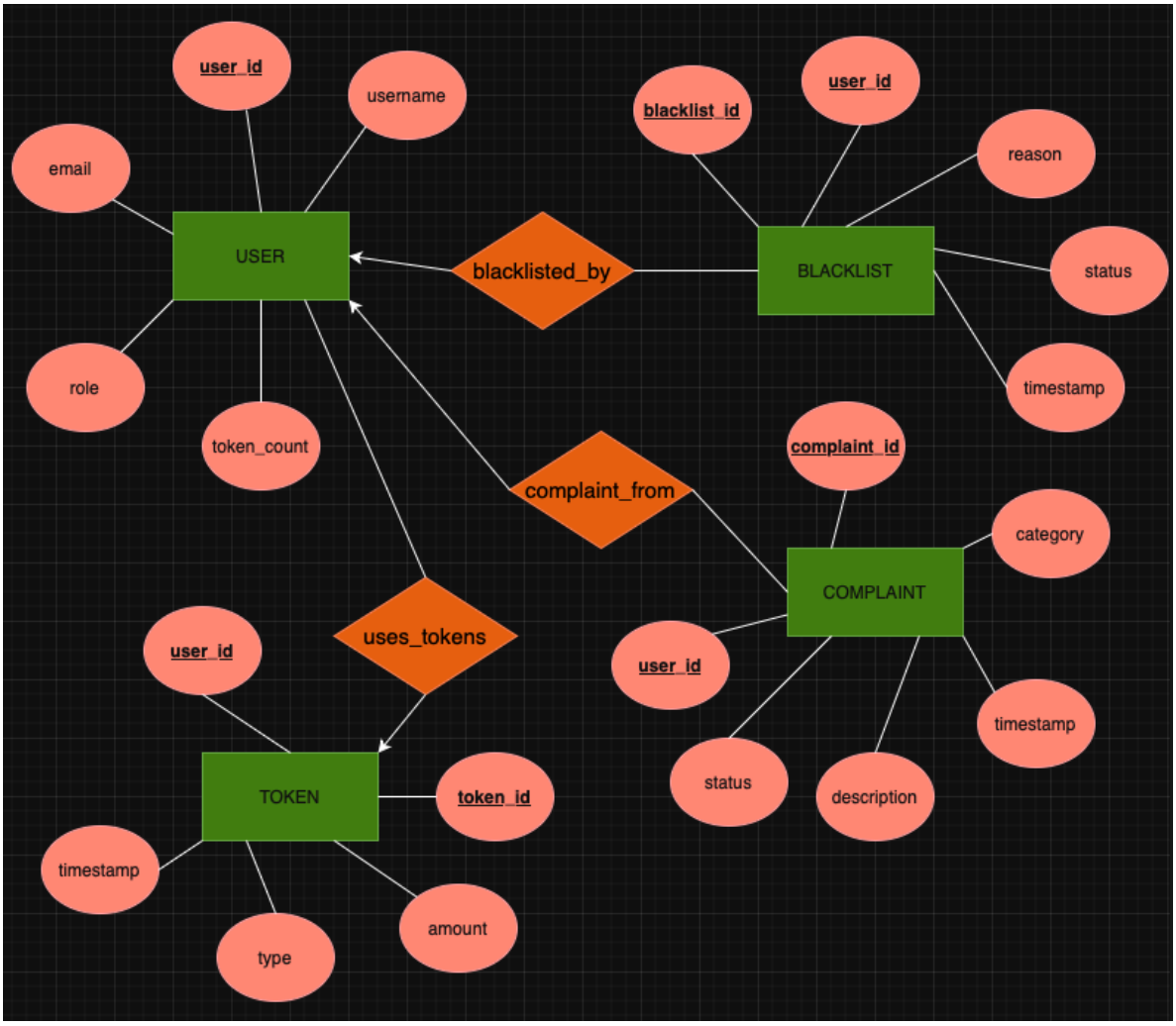
Sequential Diagram for Blacklist Management



3. System Visualization

This section presents the structural, functional, and visual components of the system in detail. It begins with an Entity-Relationship (E/R) diagram that models the system's data structure, including all entities, their attributes, keys, and relationships. Following this, the detailed design subsection outlines the logic behind each method using pseudo-code, specifying inputs, outputs, and core functionalities. Finally, a set of system screens showcases the graphical user interface (GUI), highlighting key screens and including a prototype of a selected functionality to demonstrate the user interaction flow and design approach.

3.1 Entity-Relationship Diagram of the System



3.2 Detailed Design

```
# FILE: models.py (Expanded)
"""
ADDITIONAL MODELS FOR PROJECT REQUIREMENTS

class User(Existing fields +):
  - role: Enum('free', 'paid', 'super')  # User type
  - tokens: Integer  # Available tokens
  - penalty_cooldown: DateTime  # Free user lockout
  - transactions: Relationship -> UserTokenTransaction
```

```
class BlacklistEntry(db.Model):
  - word: String(50), unique
  - status: Enum('pending', 'approved')
  - submitted_by: ForeignKey(User)
  - decided_by: ForeignKey(User)  # Super user

class TextDocument(db.Model):
  - content: Text
  - owner: ForeignKey(User)
  - tokens_charged: Integer
  - collaborators: ManyToMany(User)  # Shared access

class Correction(db.Model):
  - original_text: Text
  - corrected_text: Text
  - correction_type: Enum('self', 'llm')
  - accepted: Boolean
  - tokens_charged: Integer
  - document: ForeignKey(TextDocument)

class Invitation(db.Model):
  - inviter: ForeignKey(User)
  - invitee: ForeignKey(User)
  - document: ForeignKey(TextDocument)
  - status: Enum('pending', 'accepted', 'rejected')

class Complaint(db.Model):
  - complainer: ForeignKey(User)
  - target_user: ForeignKey(User)
  - document: ForeignKey(TextDocument)
  - reason: Text
  - resolution: Enum('pending', 'resolved')
"""

# FILE: routes.py (Expanded)
"""
NEW ROUTES & FUNCTIONALITY

# Token Purchase Endpoint
@bp.route('/purchase-tokens', methods=['POST'])
def purchase_tokens():
   IF user is paid:
      ADD tokens based on payment
      CREATE transaction record
   ELSE:
```

```
ERROR: Only paid users can purchase

# Text Submission Flow
FUNCTION handle_submission(text, user):
    IF user is free:
        IF len(text) > 20 words:
            APPLY 3min cooldown
            REJECT
        IF in cooldown:
            REJECT
    ELSE: # Paid user
        required_tokens = len(text.split())
        IF user.tokens < required_tokens:
            DEDUCT penalty (50% tokens)
            REJECT

    PROCESS blacklist:
        FOR each word in text:
            IF in blacklist:
                REPLACE with *****
                CHARGE tokens per blacklisted word length

    RETURN processed_text

# LLM Correction Workflow
FUNCTION handle_llm_correction(text, user):
    llm_response = CALL_LOCAL_LLM(text)  # Using HuggingFace/Ollama
    DIFF = compare_original_and_corrected(text, llm_response)
    HIGHLIGHT diff in UI
    STORE correction record with 'pending' status

    IF no_diff_found and len(text) > 10:
        AWARD 3 tokens  # Bonus condition

# Collaboration System
FUNCTION handle_invitation(inviter, invitee_email, document):
    invitee = FIND_USER_BY_EMAIL(invitee_email)
    IF invitee is paid:
        CREATE invitation record
        NOTIFY invitee
    ELSE:
        CHARGE inviter 3 tokens  # Reckless invite

# Dispute Resolution
FUNCTION handle_complaint(complainer, target_user, document, reason):
```

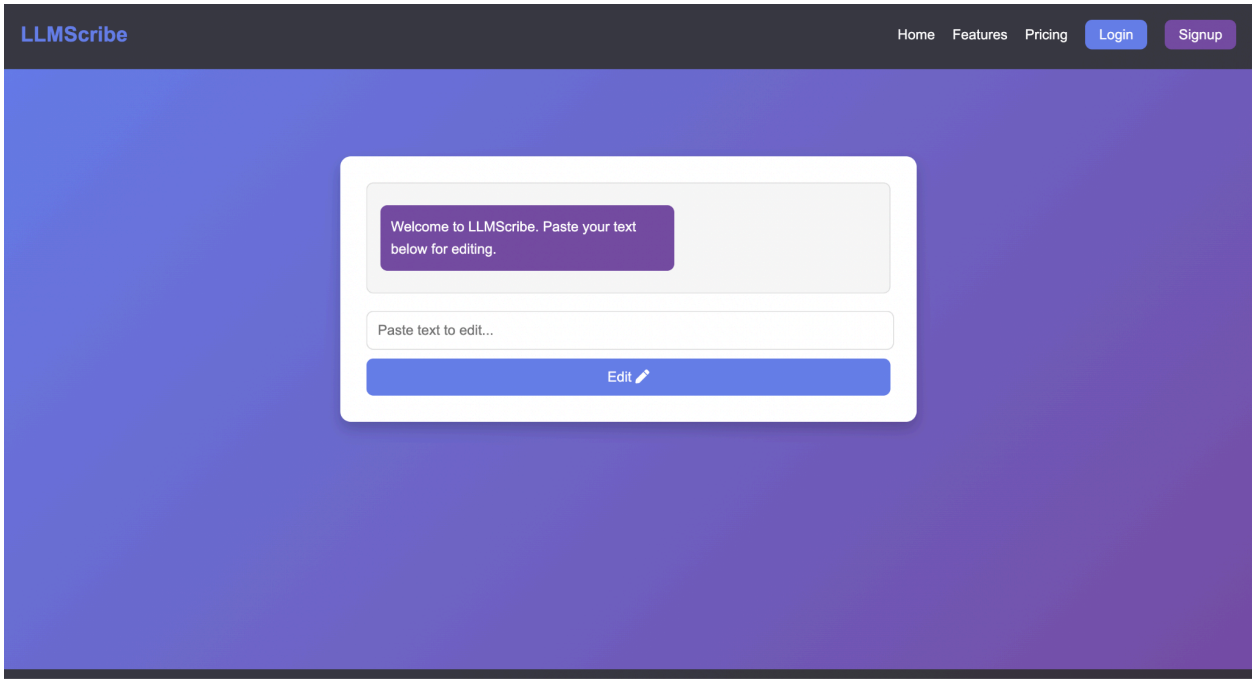
```
CREATE complaint record
   NOTIFY super users
   FREEZE document edits until resolution
   IF super user approves complaint:
      APPLY token penalty to target
   ELSE:
      APPLY penalty to complainer
"""

# PSEUDO-CODE FOR KEY WORKFLOWS

# Text Submission Process
USER_SUBMITS_TEXT ->
SYSTEM:
  VERIFY_USER_STATUS ->
  CHECK_BLACKLIST ->
  CALCULATE_TOKEN_COST ->
  IF ENOUGH_TOKENS:
     PROCESS_CORRECTION ->
     SHOW_HIGHLIGHTS ->
     HANDLE_USER_ACCEPTANCE ->
     UPDATE_TOKENS
  ELSE:
     APPLY_PENALTY ->
     SHOW_ERROR

# Collaboration Workflow
USER_A_INVITES_USER_B ->
SYSTEM:
  VALIDATE_B_IS_PAID ->
  CREATE_INVITATION ->
  IF B_ACCEPTS:
     GRANT_COLLAB_ACCESS ->
     SYNC_EDITS
  ELSE:
     CHARGE_A_RECKLESS_FEE ->
     LOG_INCIDENT

# Creative Feature: Crowdsourced Blacklist
FUNCTION handle_blacklist_submission(word, user):
   IF word not in existing_entries:
      CREATE blacklist_entry(
         word=word,
         status='pending',
         submitted_by=user
```

)

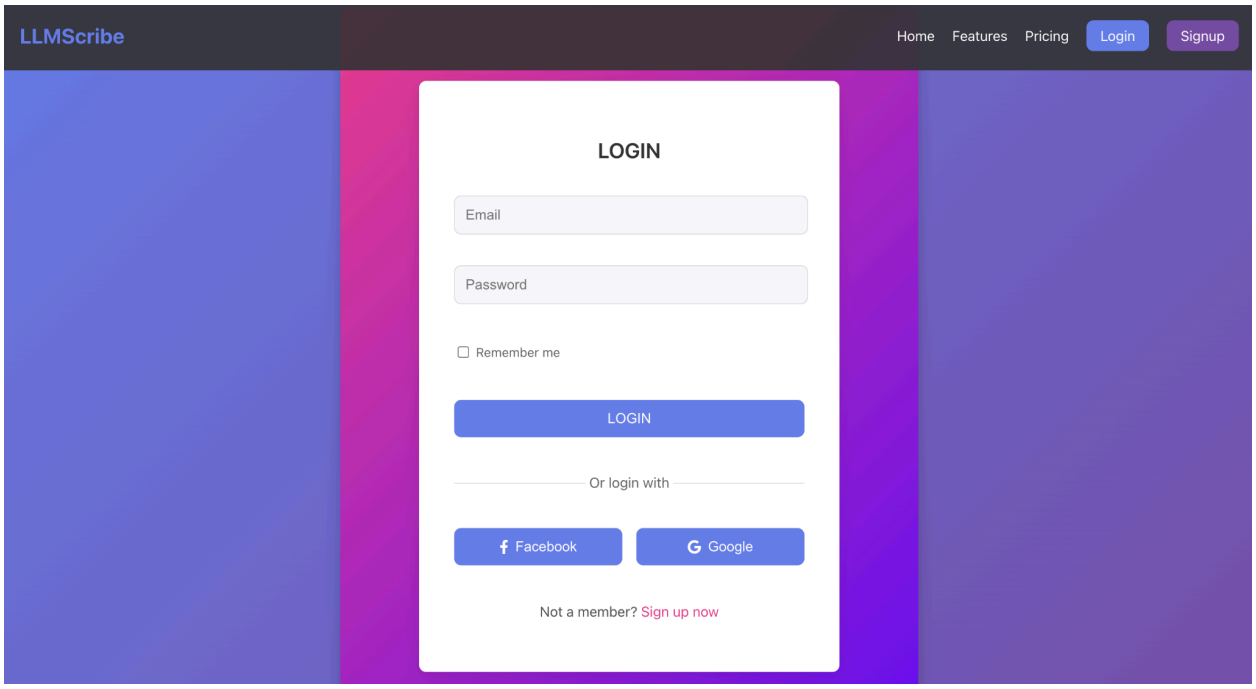
```
NOTIFY_SUPER_USERS
   ELSE:
      IGNORE_DUPLICATE

# Token Management System
CLASS UserTokenTransaction:
  - user: ForeignKey(User)
  - amount: Integer  # + for purchases, - for usage
  - transaction_type: Enum('purchase', 'penalty', 'usage', 'bonus')
  - reference: String  # E.g., "LLM correction doc123"
```

3.3 System Screens

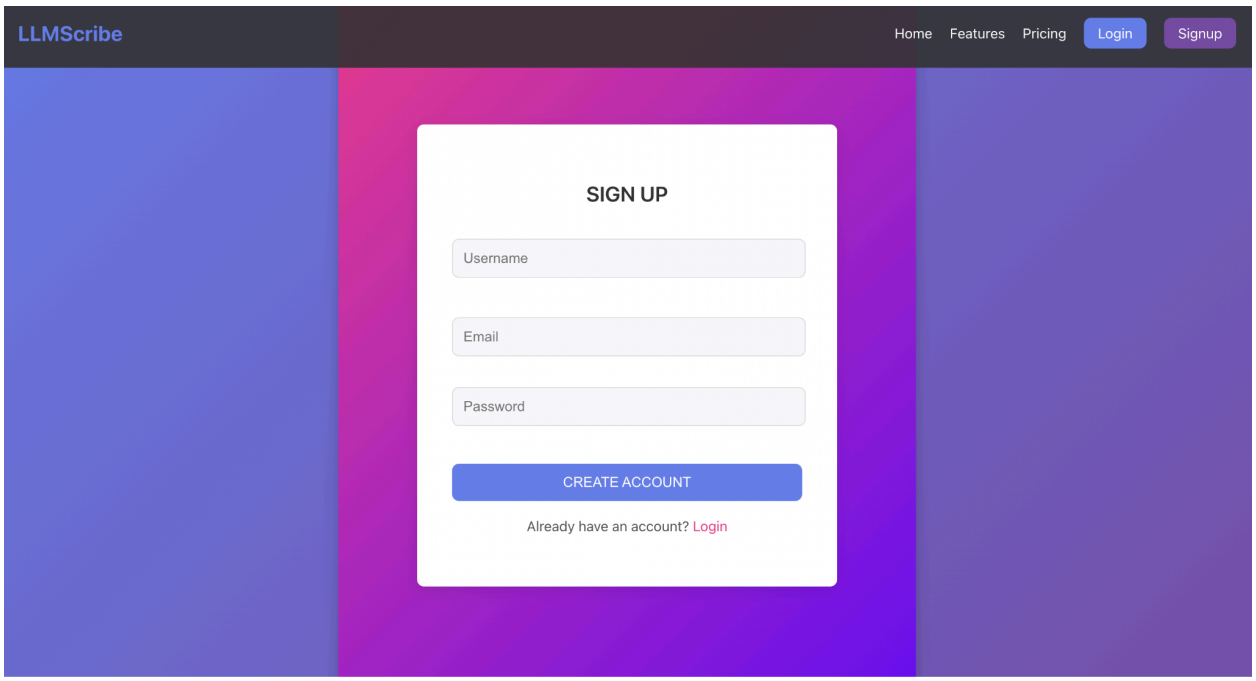
Homepage:



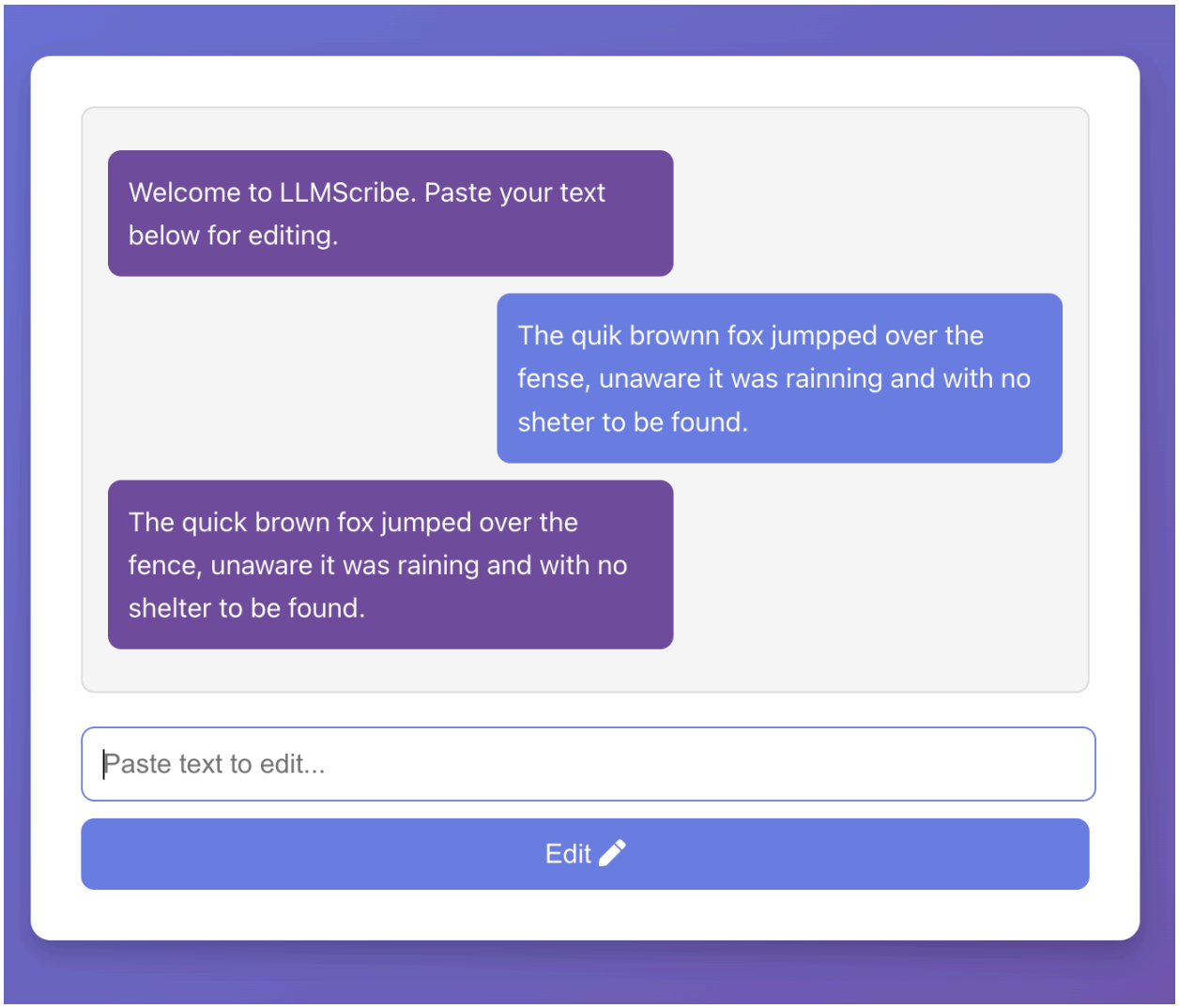
Login Page:



Signup Page:



Sample Prototype:



4. Project Management and Documentation

This section documents the collaborative and organizational aspects of the project. It includes meeting memos and notes that track the progress of the team's discussions, decisions, and action items across various development stages. Any notable concerns or issues related to teamwork or project coordination are also reflected here to provide transparency and insight into the group's dynamics. Additionally, the section provides the URL to the team's Git repository (e.g., GitHub, GitLab, Bitbucket), which contains all relevant project materials, including source code, diagrams, and this report, ensuring centralized and version-controlled access to project deliverables.

4.1 Group Meeting Memos/Teamwork

Week 1 (3/18/2025 - 3/24/2025)

- Team present: All members present
- Project kickoff meeting
- Assigned initial roles:
- Frontend: Nischal
- Backend: Rajiv, Adama
- Full-stack/Documentation: Junaet, Sean
- Concerns raised:
- Sean requested clearer documentation standards

Week 2 (3/25/2025 - 3/31/2025)

- Team present: All members present
- Progress Review:
- Basic authentication system implemented by Rajiv
- Frontend design mockups approved by team
- Action items:
- Junaet to research LLM integration options
- Sean to complete navbar design

Week 3 (4/1/2025 - 4/7/2025)

- Team present: Adama, Sean, Rajiv, Junaet (Nischal absent)
- Major Concern:
- Database schema changes requiring rework
- Progress:
- User authentication completed by Rajiv
- Basic text editing interface implemented by Sean and Junaet
- LLM integration started by Junaet

Week 4 (4/8/2025 - 4/13/2025)

- Team present: Adama, Sean, Rajiv, Junaet (Nischal absent)
- Bug fixes for authentication (Rajiv)
- Text editor refinements (Adama, Sean)
- Design Report Completion (All present members)
- Concerns:
- Nischal's absence affecting progress
- Disagreements on database optimization
- Documentation finalization

Key Challenges Faced:

- Team communication, especially between frontend and backend
- Integration issues with LLM API
- Meeting attendance consistency and absences from certain members
- Documentation standards adherence

- Technical disagreements on implementation approaches

Lessons Learned:

- Need for better initial planning and role definition
- Importance of regular attendance and communication
- Value of clear documentation
- Benefits of early integration testing

4.2 GitHub Repository

https://github.com/Jdawg0309/LLM