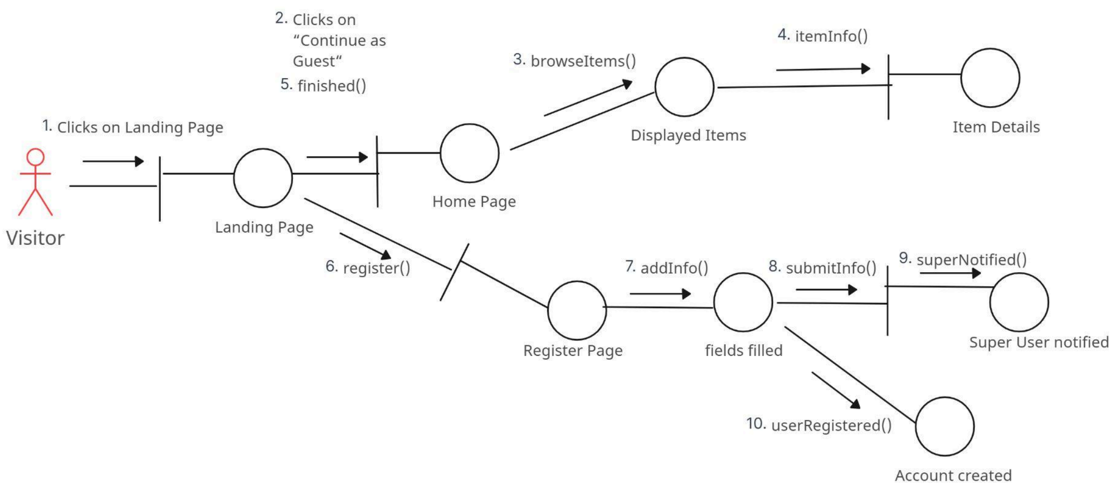
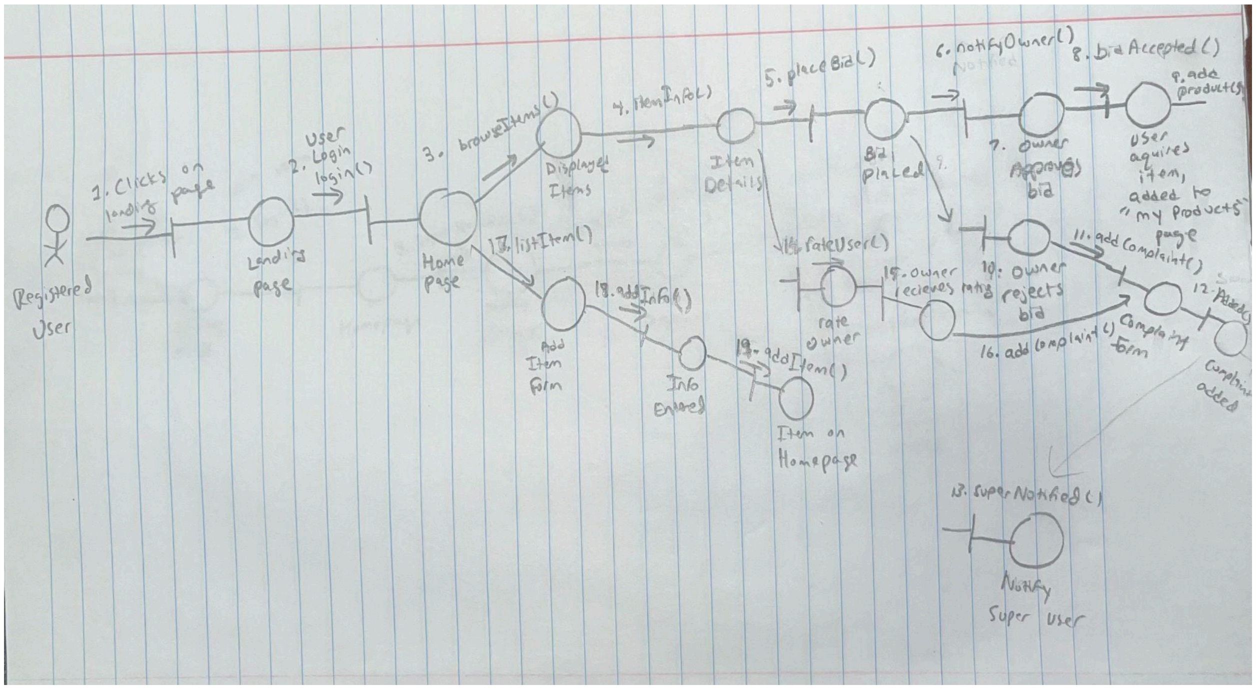
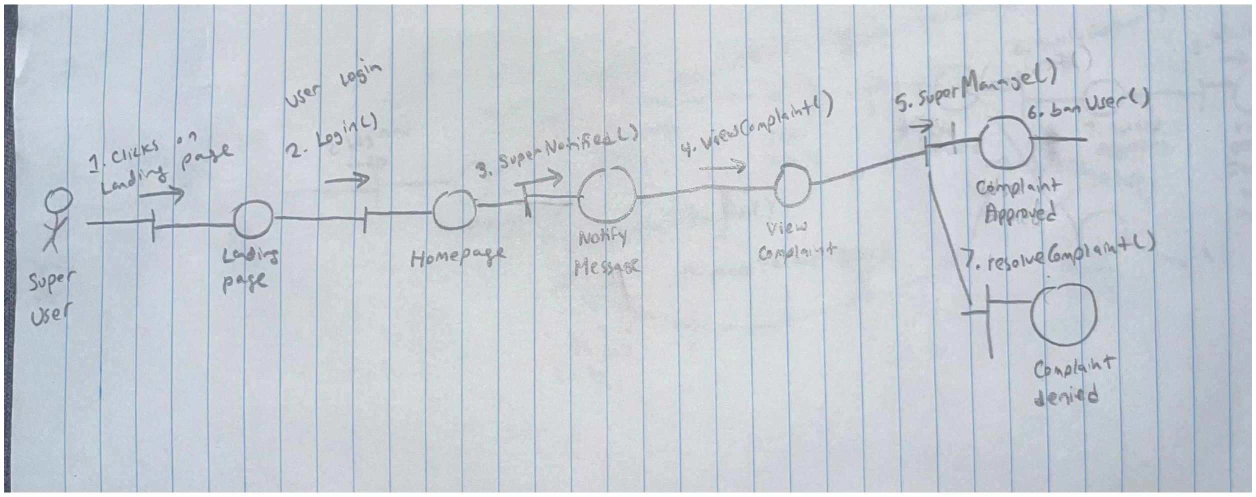
Brandon B,
Ratul Bin Rasul,
Mamuna Chaudhry,
Kalelo Dukuray

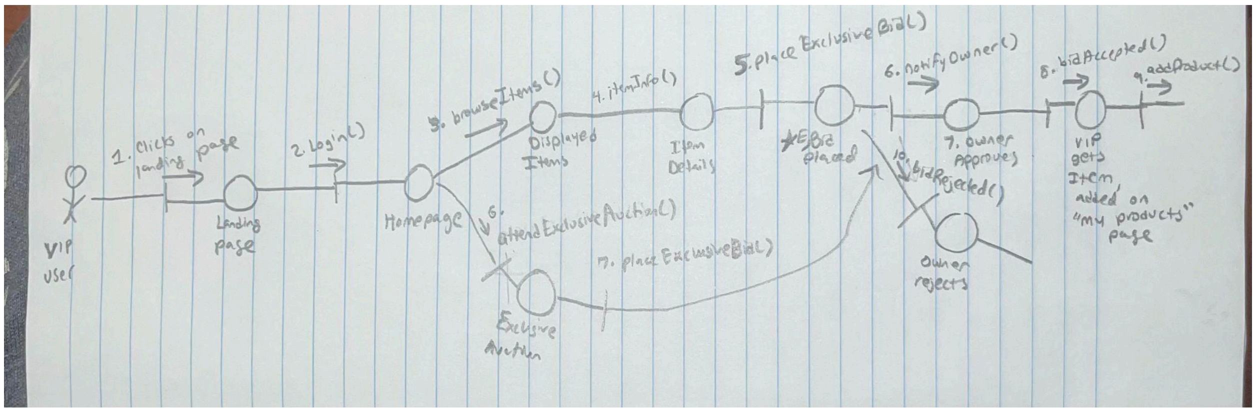
<E-BIDDING PROJECT >
Phase 2 Report

Part 1) Introduction with collaboration-class diagram









Part 2) Use-Case Diagram

Use Case Scenarios

2.1 Visitor Browsing and Registration

Normal Scenario:

- A visitor (V) browses items.
- To participate, V applies to become a User (U).
- V is prompted with a random arithmetic question to verify human authenticity.
- Super-User (S) reviews and approves registration.

Exceptional Scenario:

- V provides incorrect answers repeatedly, leading to a temporary block on registration.

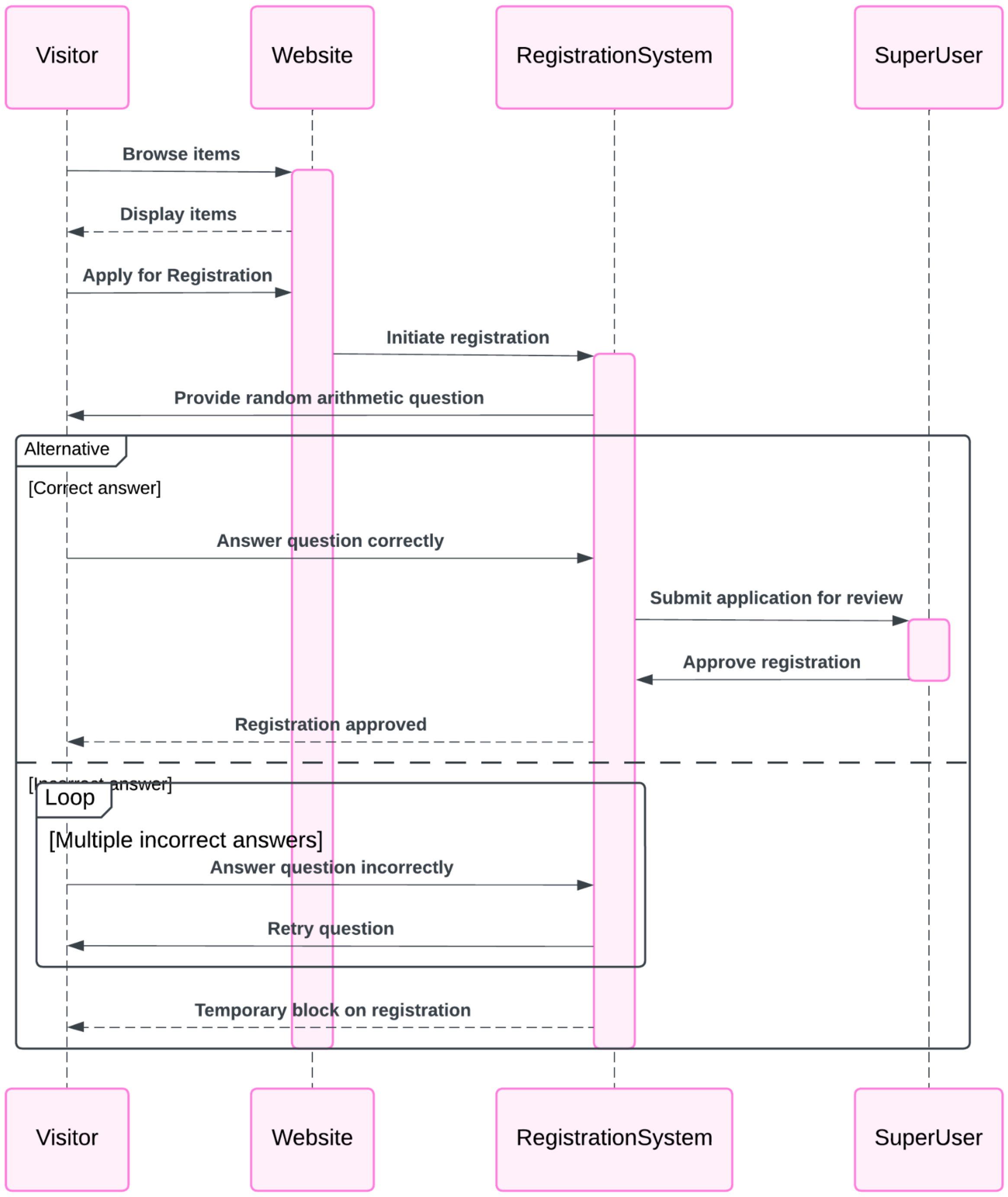


Diagram 1: Sequence Diagram (Visitor Browsing and Registration)

2.2 Item Listing

Normal Scenario:
- User (U) logs in and lists items/services for sale or rent, setting the asking price.
- Item successfully posted to the listing.

Exceptional Scenario:
- U attempts to list an item without login, receiving an error prompt.

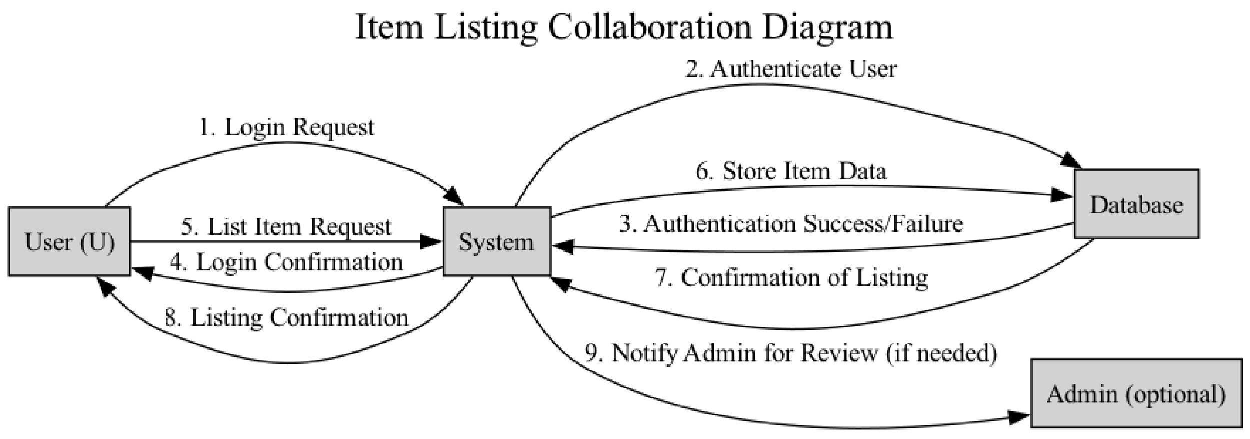


Diagram 2: Collaboration Diagram (Item Listing)

2.3 Bidding on Items
Normal Scenario:
- U places a bid on a listed item with adequate account funds.
- Bid is registered, and a notification is sent to the item's owner.

Exceptional Scenario:
- Insufficient funds trigger an alert, preventing the bid from being placed.

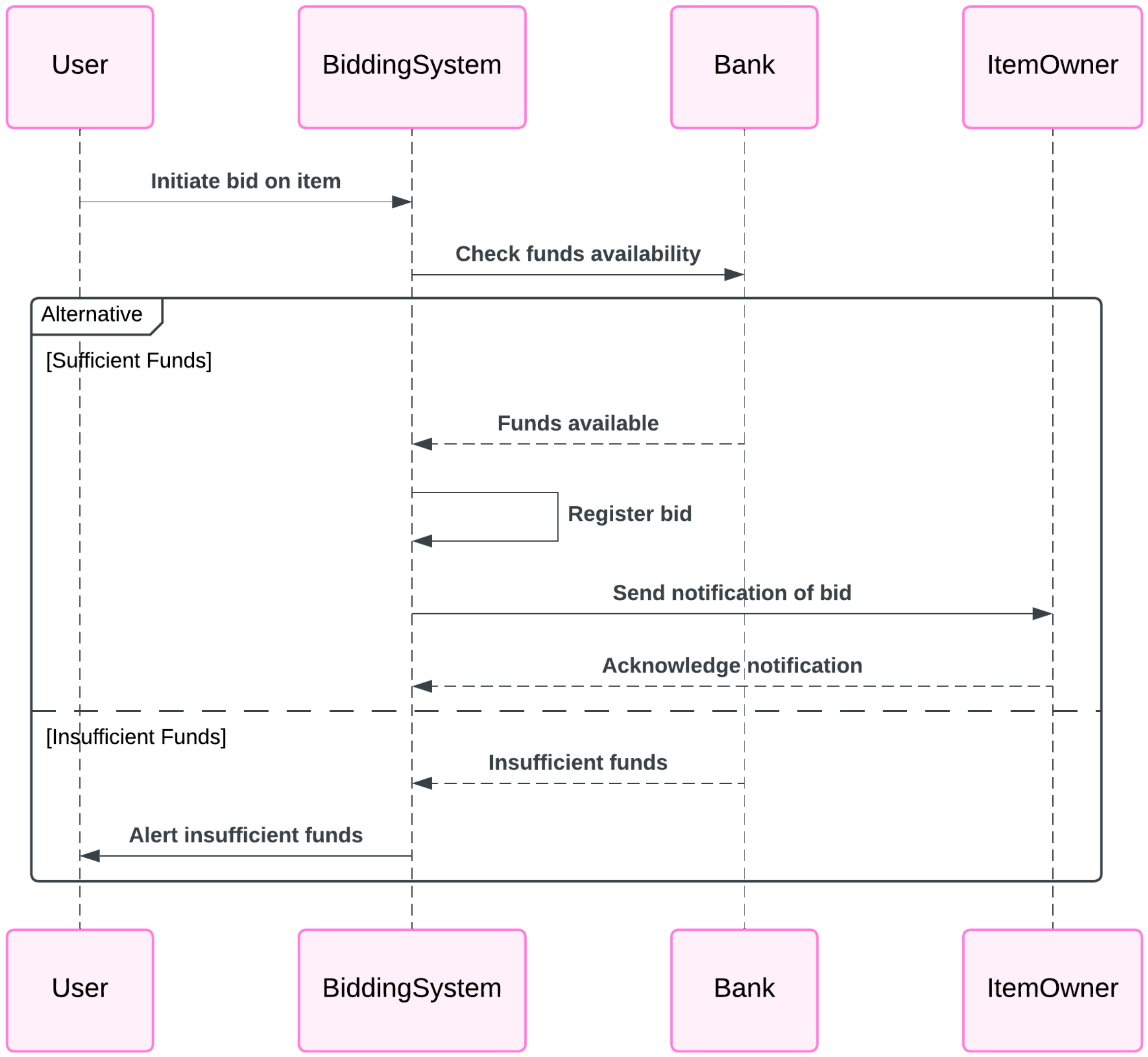


Diagram 3: Sequence Diagram (Bidding on Items)

2.4 Rating and Complaints
Normal Scenario:
- Post-transaction, the buyer and seller rate each other anonymously.
- Ratings are recorded and impact future privileges or suspensions.

Exceptional Scenario:
- One party attempts to rate outside of transaction context, which is disallowed.

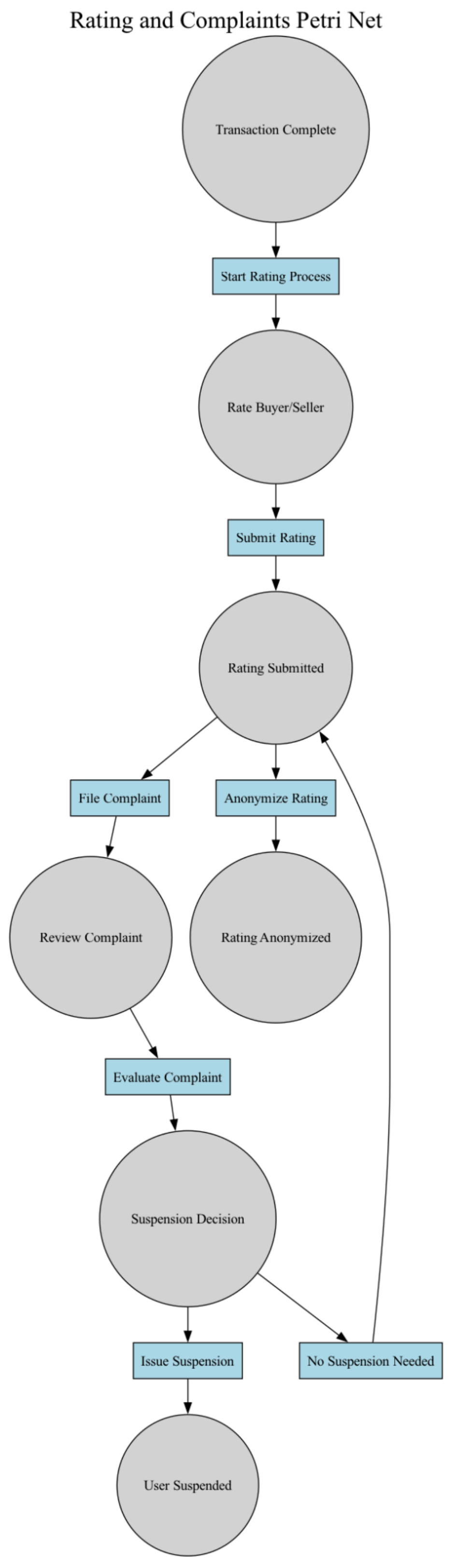


Diagram 4: Petri-Net Diagram (Rating and Complaints)

2.5 VIP Live Bidding
Normal Scenario:
- VIP users join a live bidding session for exclusive items.
- Bidding is time-sensitive, and the highest bid wins when the timer ends.

Exceptional Scenario:
- VIP status revoked due to lack of funds or complaints, excluding the user from VIP auctions.

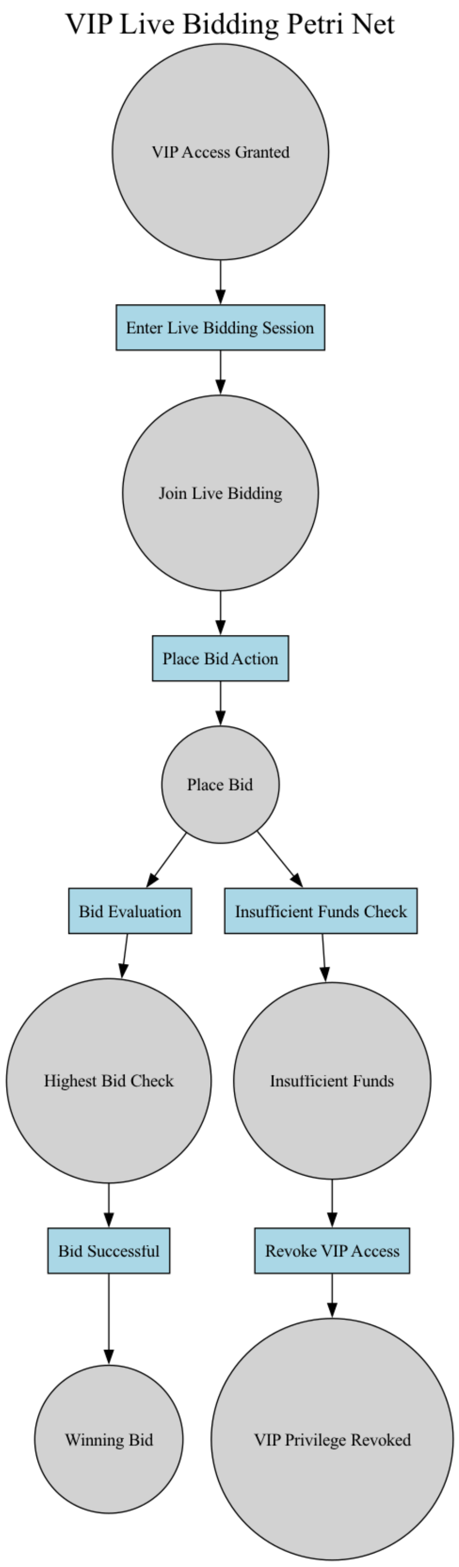


Diagram 5: Petri-Net Diagram (VIP Live Bidding)

2.6 Transaction Handling and Account Management

Normal Scenario:

- U completes a transaction, and funds are transferred from buyer to seller.
- Both parties receive transaction confirmations.

Exceptional Scenario:

- Transaction fails due to technical errors or insufficient balance, triggering a rollback.

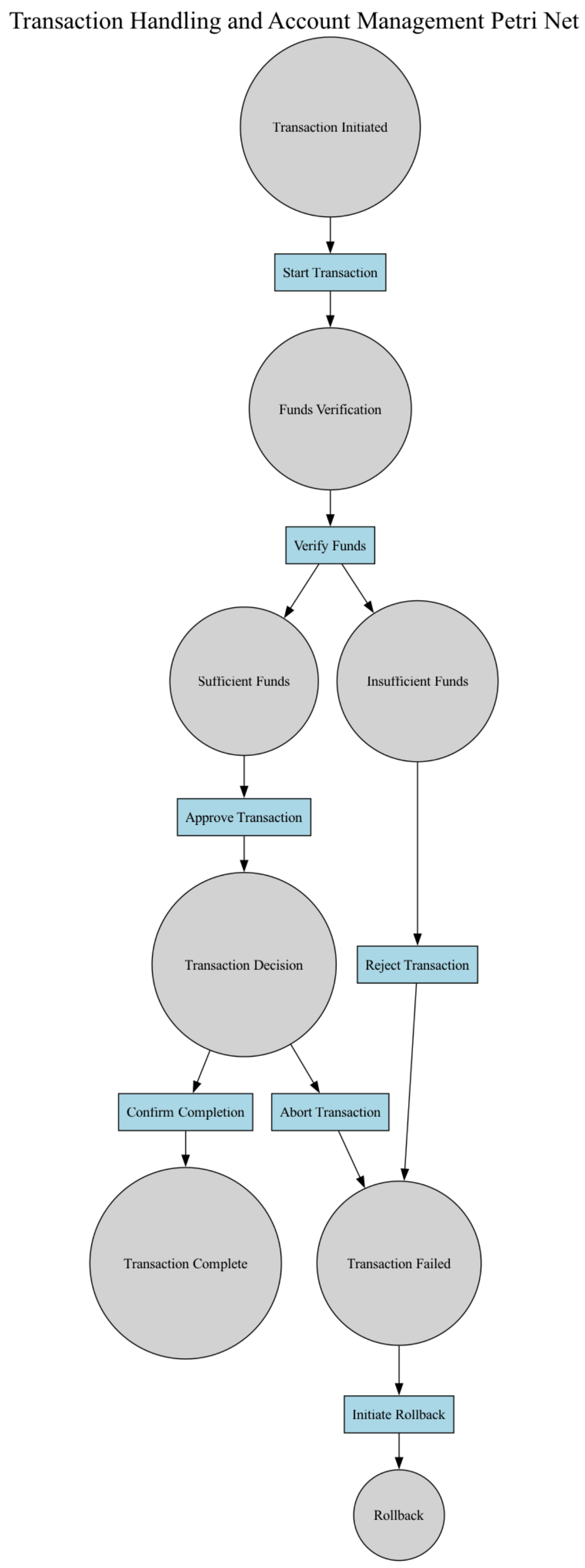
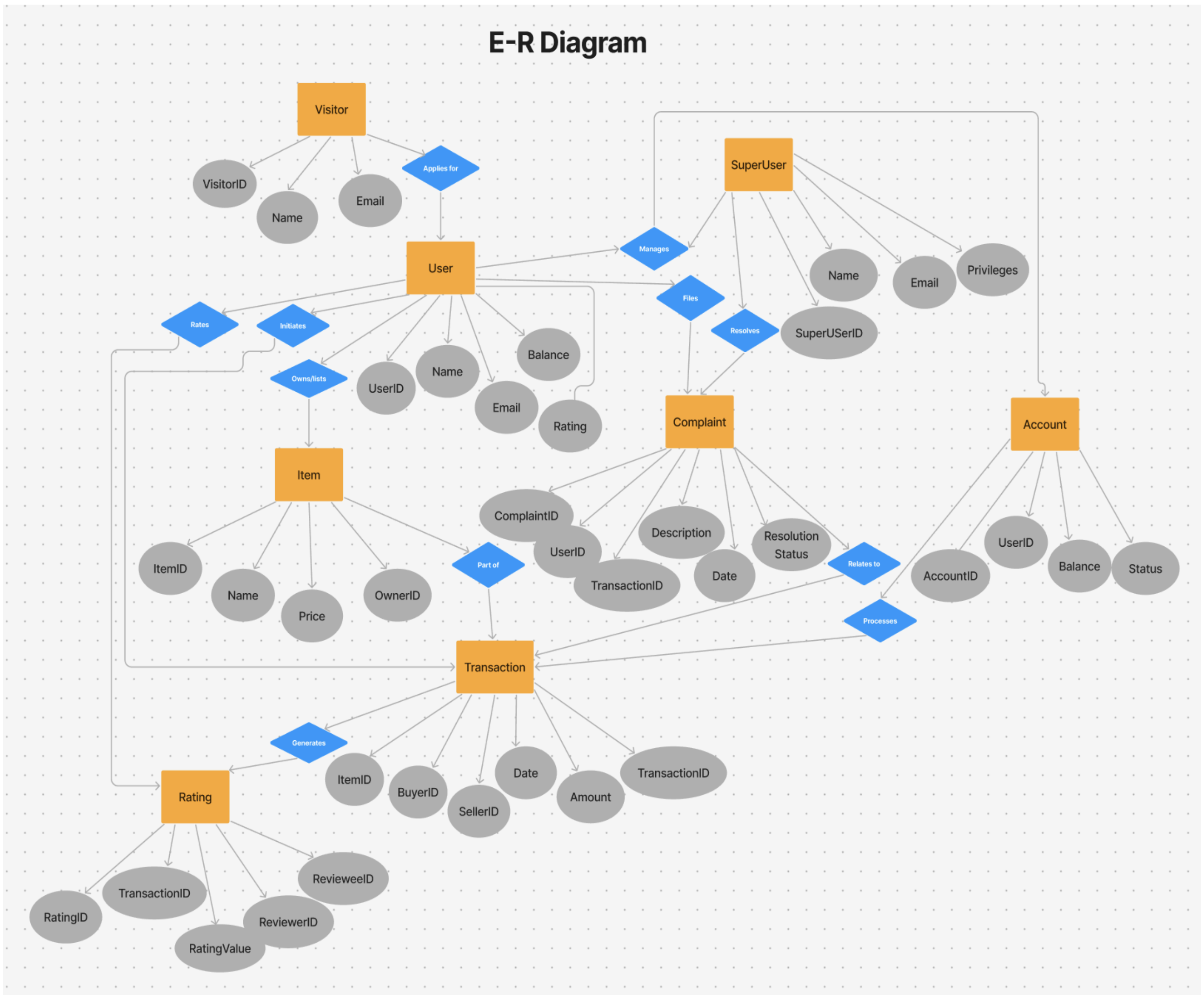
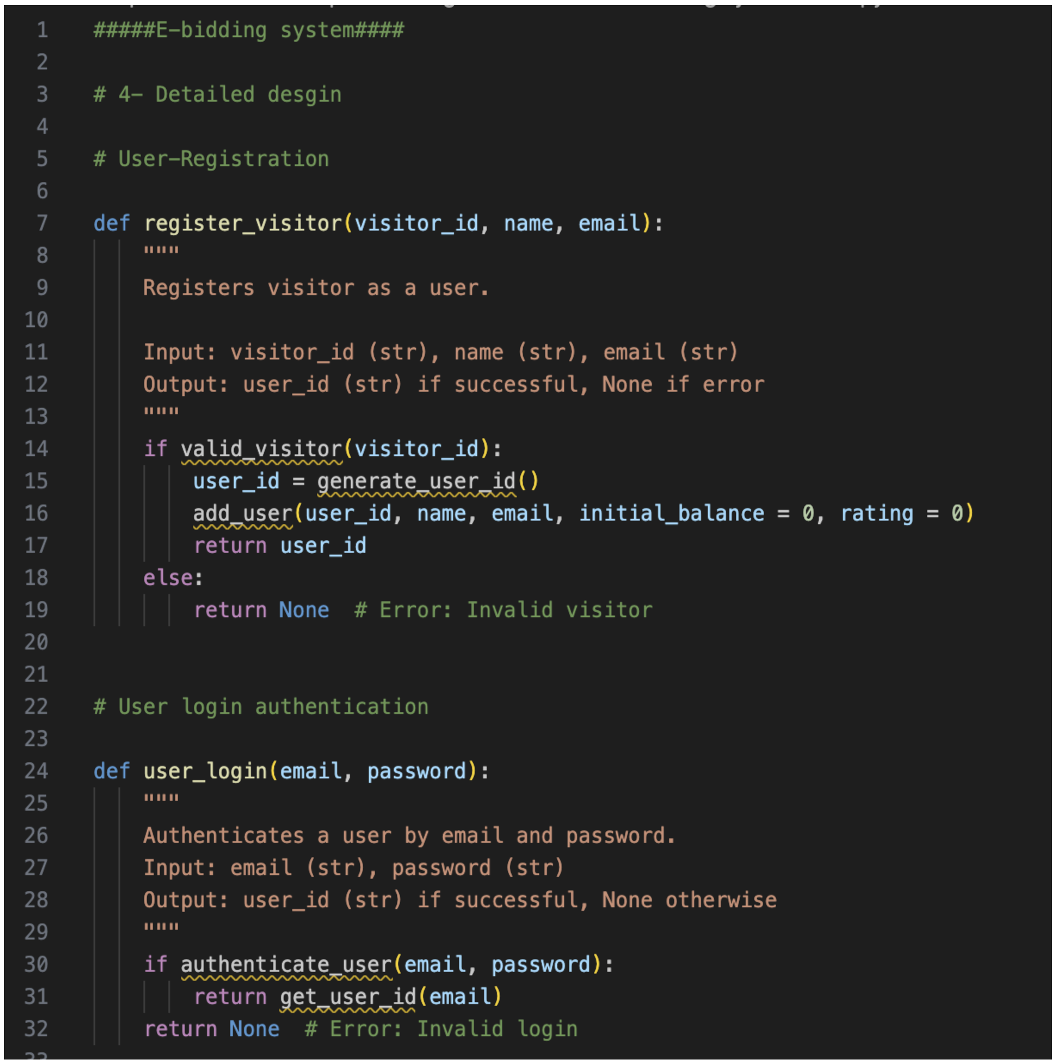


Diagram 6: Petri-Net Diagram (Transaction Handling and Account Management)

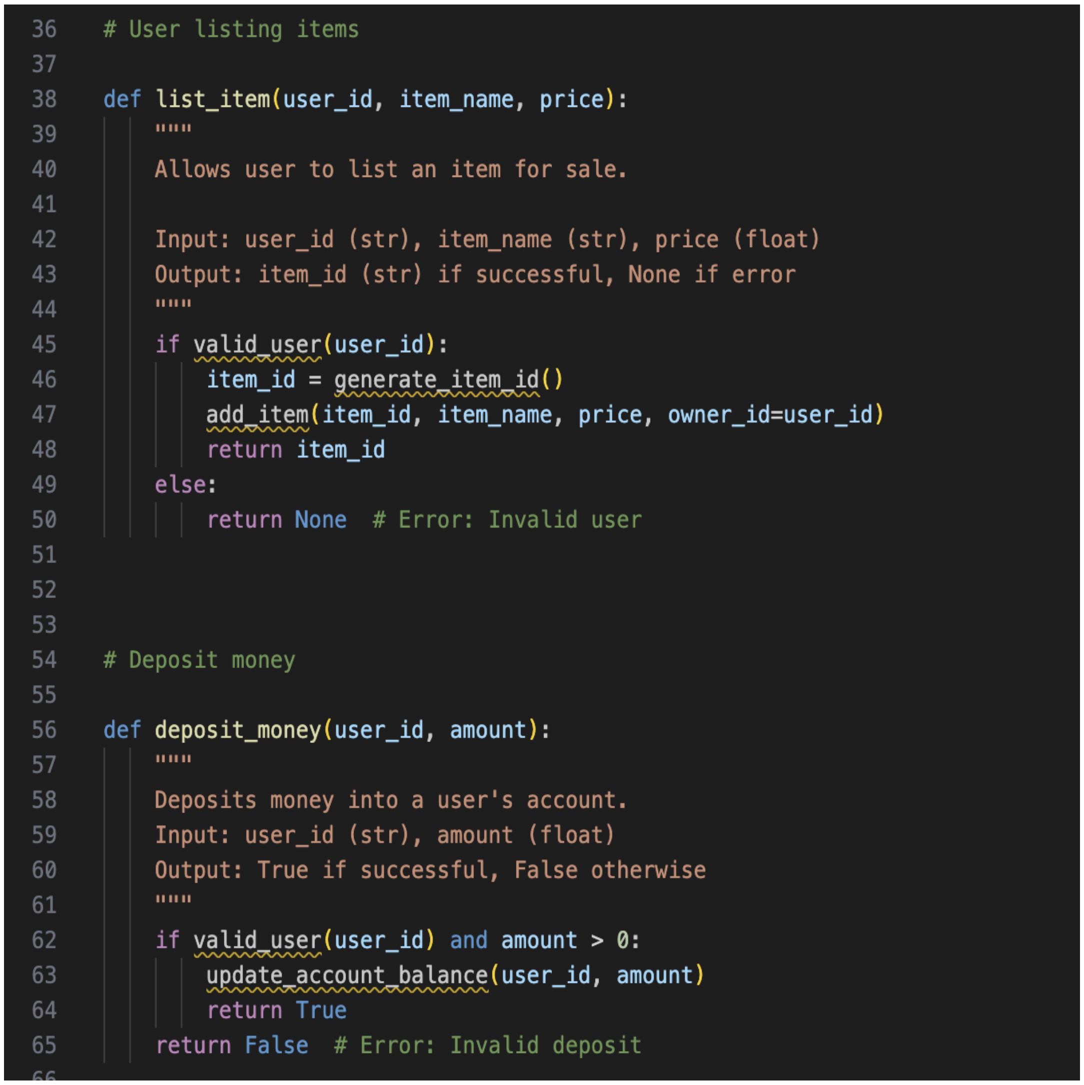
Part 3) E-R Diagram for the entire system-



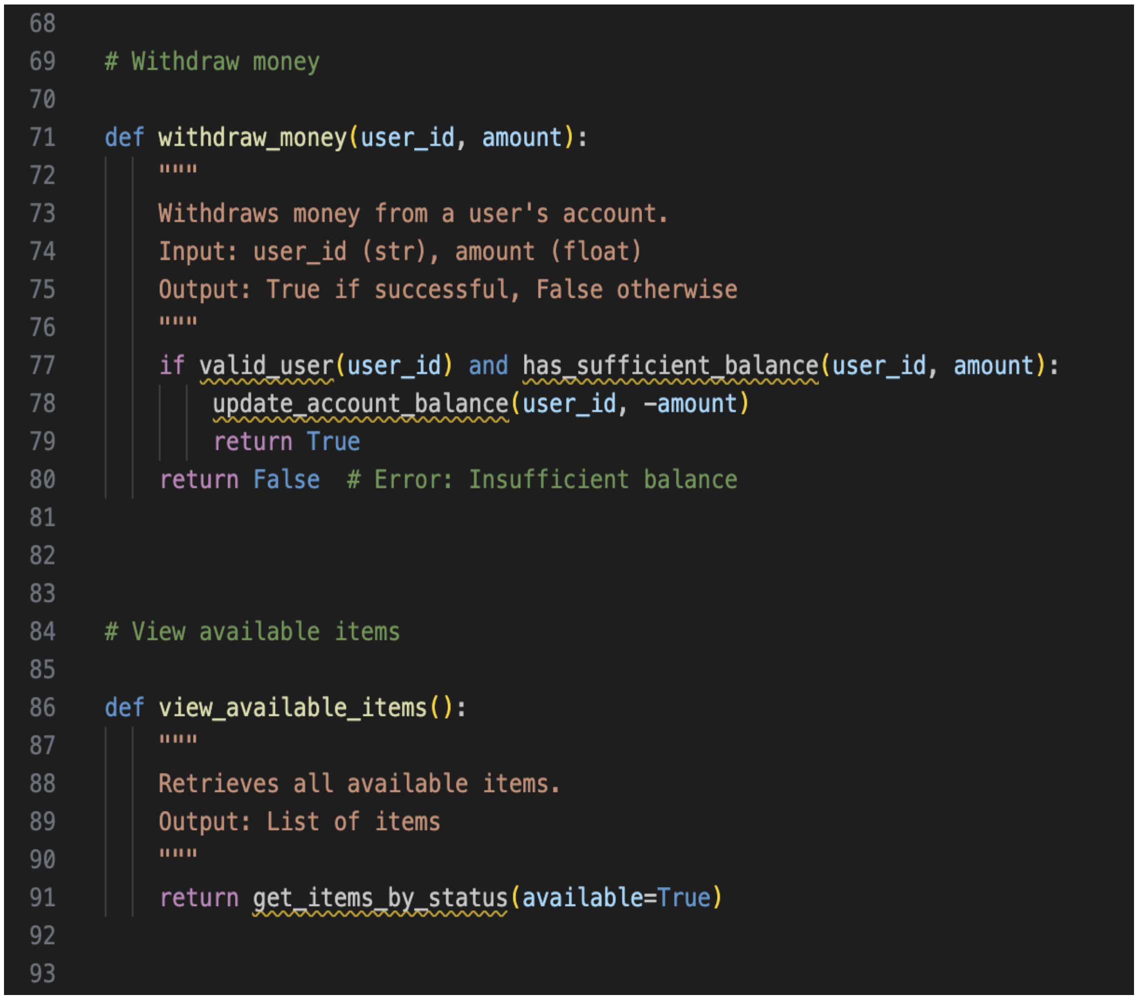
Part 4) Detailed Design-



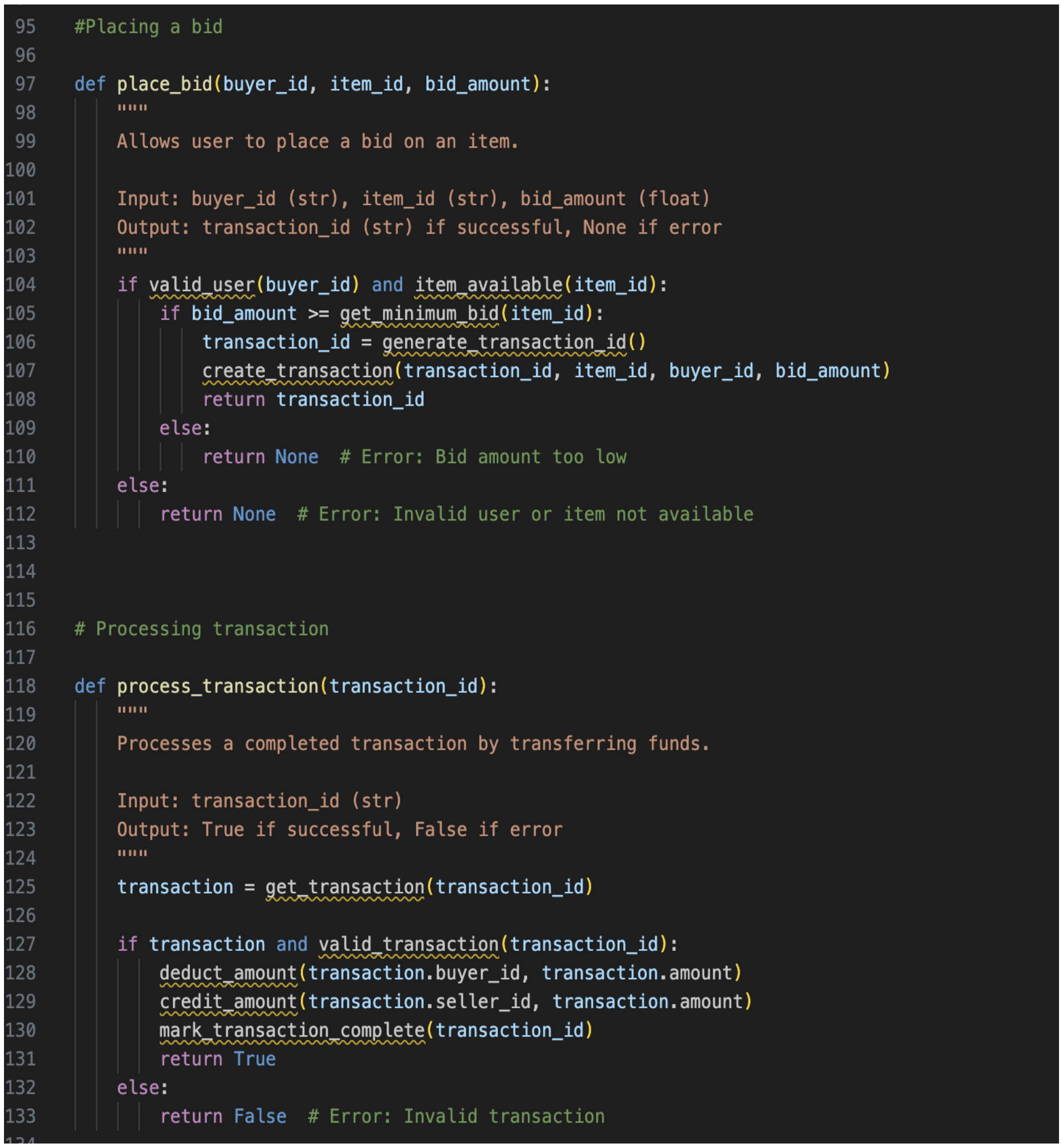
```python
#####E-bidding system####

# 4- Detailed desgin

# User-Registration

def register_visitor(visitor_id, name, email):
    """
    Registers visitor as a user.

    Input: visitor_id (str), name (str), email (str)
    Output: user_id (str) if successful, None if error
    """
    if valid_visitor(visitor_id):
        user_id = generate_user_id()
        add_user(user_id, name, email, initial_balance = 0, rating = 0)
        return user_id
    else:
        return None  # Error: Invalid visitor


# User login authentication

def user_login(email, password):
    """
    Authenticates a user by email and password.
    Input: email (str), password (str)
    Output: user_id (str) if successful, None otherwise
    """
    if authenticate_user(email, password):
        return get_user_id(email)
    return None  # Error: Invalid login
```



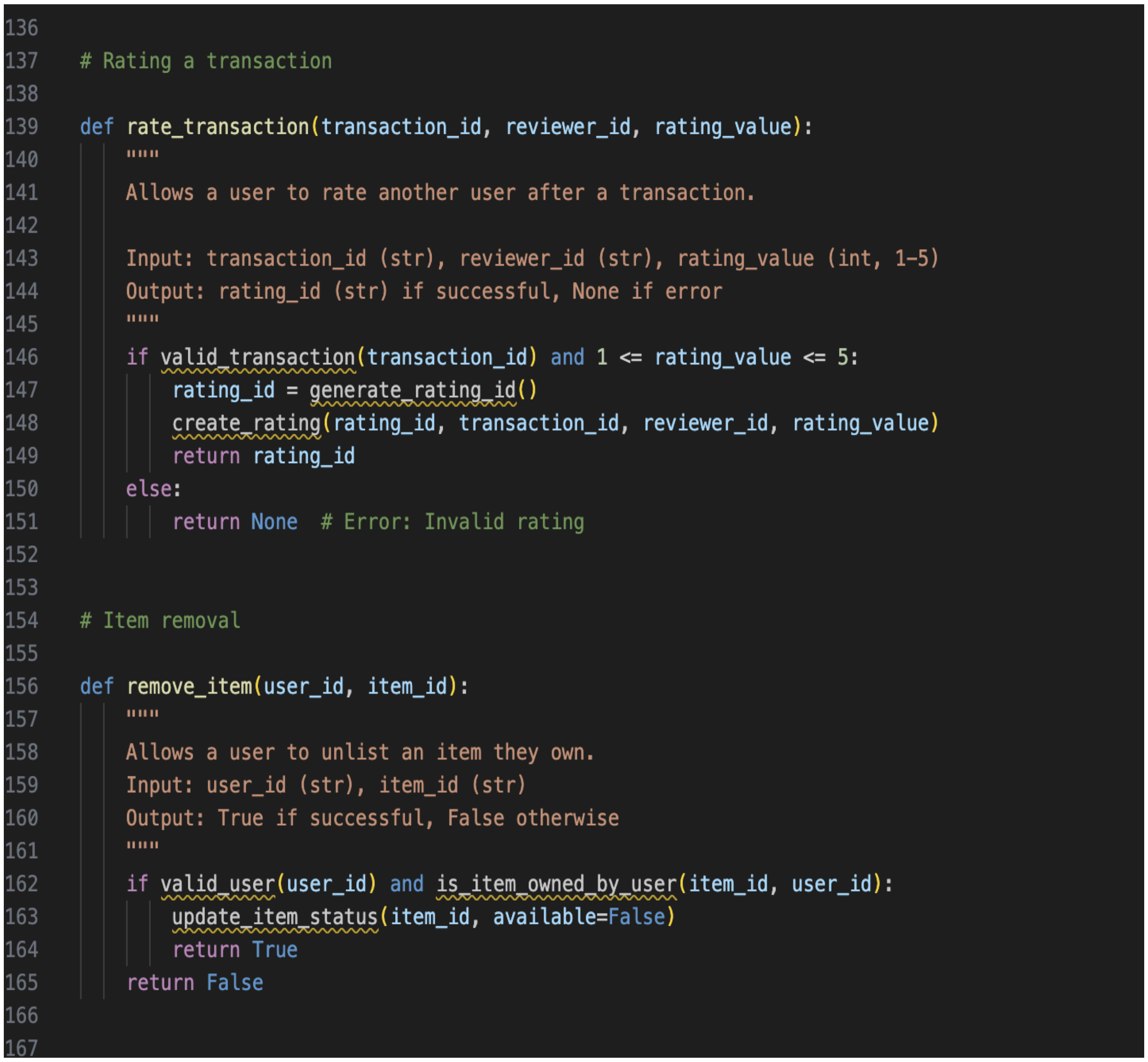
```python
# User listing items

def list_item(user_id, item_name, price):
    """
    Allows user to list an item for sale.

    Input: user_id (str), item_name (str), price (float)
    Output: item_id (str) if successful, None if error
    """
    if valid_user(user_id):
        item_id = generate_item_id()
        add_item(item_id, item_name, price, owner_id=user_id)
        return item_id
    else:
        return None  # Error: Invalid user



# Deposit money

def deposit_money(user_id, amount):
    """
    Deposits money into a user's account.
    Input: user_id (str), amount (float)
    Output: True if successful, False otherwise
    """
    if valid_user(user_id) and amount > 0:
        update_account_balance(user_id, amount)
        return True
    return False  # Error: Invalid deposit
```



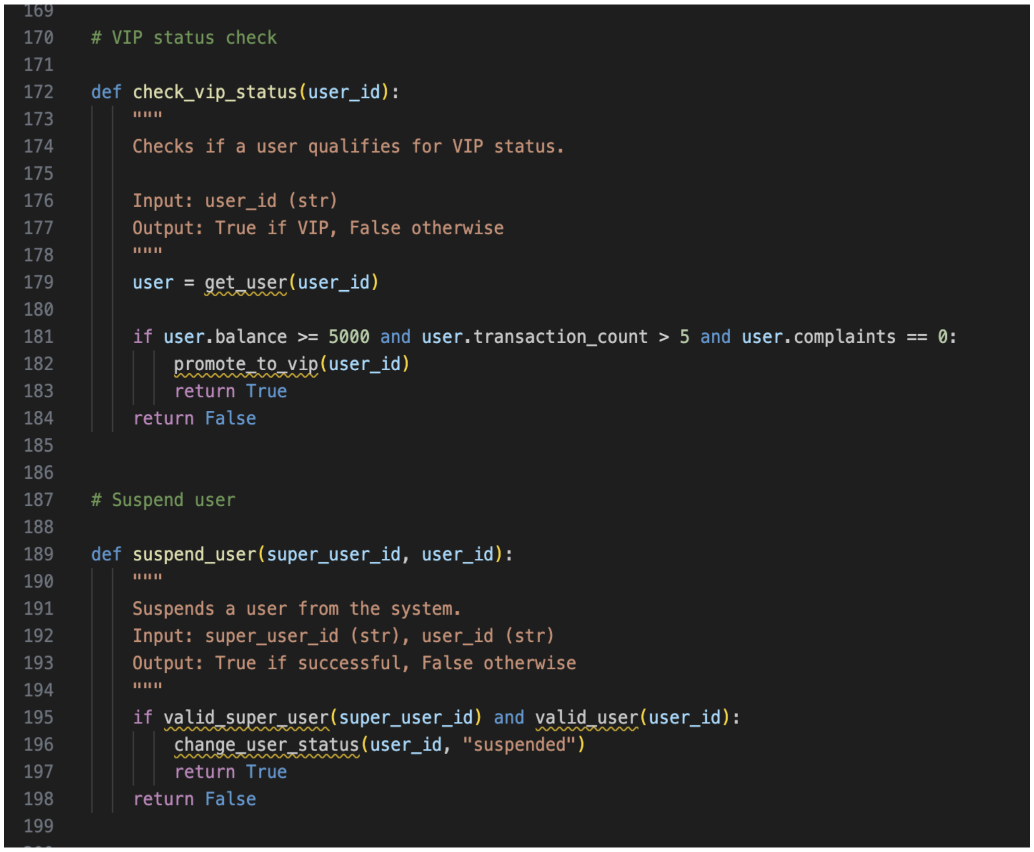
```python

# Withdraw money

def withdraw_money(user_id, amount):
    """
    Withdraws money from a user's account.
    Input: user_id (str), amount (float)
    Output: True if successful, False otherwise
    """
    if valid_user(user_id) and has_sufficient_balance(user_id, amount):
        update_account_balance(user_id, -amount)
        return True
    return False  # Error: Insufficient balance



# View available items

def view_available_items():
    """
    Retrieves all available items.
    Output: List of items
    """
    return get_items_by_status(available=True)


```



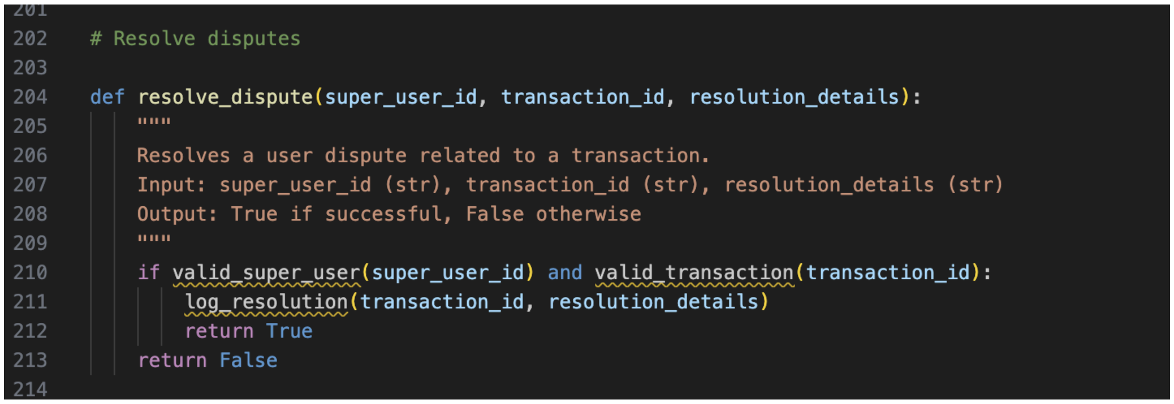
```python
#Placing a bid

def place_bid(buyer_id, item_id, bid_amount):
    """
    Allows user to place a bid on an item.

    Input: buyer_id (str), item_id (str), bid_amount (float)
    Output: transaction_id (str) if successful, None if error
    """
    if valid_user(buyer_id) and item_available(item_id):
        if bid_amount >= get_minimum_bid(item_id):
            transaction_id = generate_transaction_id()
            create_transaction(transaction_id, item_id, buyer_id, bid_amount)
            return transaction_id
        else:
            return None  # Error: Bid amount too low
    else:
        return None  # Error: Invalid user or item not available



# Processing transaction

def process_transaction(transaction_id):
    """
    Processes a completed transaction by transferring funds.

    Input: transaction_id (str)
    Output: True if successful, False if error
    """
    transaction = get_transaction(transaction_id)

    if transaction and valid_transaction(transaction_id):
        deduct_amount(transaction.buyer_id, transaction.amount)
        credit_amount(transaction.seller_id, transaction.amount)
        mark_transaction_complete(transaction_id)
        return True
    else:
        return False  # Error: Invalid transaction
```



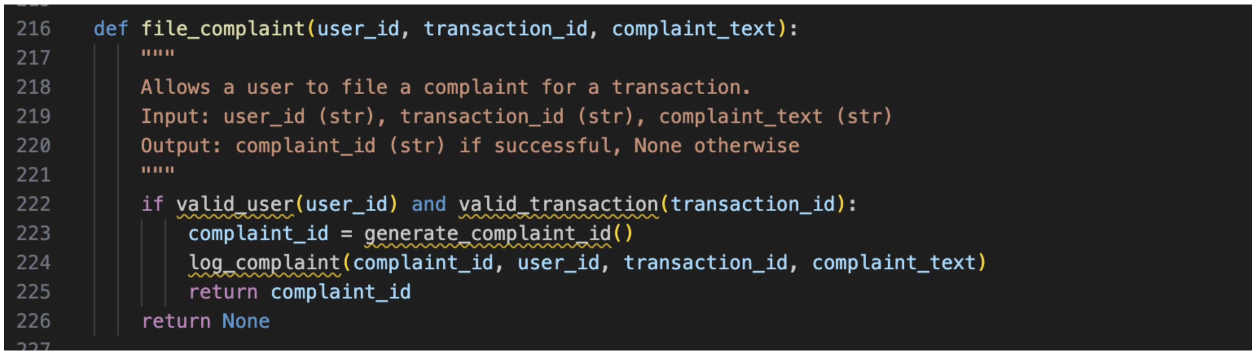
```python

# Rating a transaction

def rate_transaction(transaction_id, reviewer_id, rating_value):
    """
    Allows a user to rate another user after a transaction.

    Input: transaction_id (str), reviewer_id (str), rating_value (int, 1-5)
    Output: rating_id (str) if successful, None if error
    """
    if valid_transaction(transaction_id) and 1 <= rating_value <= 5:
        rating_id = generate_rating_id()
        create_rating(rating_id, transaction_id, reviewer_id, rating_value)
        return rating_id
    else:
        return None  # Error: Invalid rating


# Item removal

def remove_item(user_id, item_id):
    """
    Allows a user to unlist an item they own.
    Input: user_id (str), item_id (str)
    Output: True if successful, False otherwise
    """
    if valid_user(user_id) and is_item_owned_by_user(item_id, user_id):
        update_item_status(item_id, available=False)
        return True
    return False

```



```python
# VIP status check

def check_vip_status(user_id):
    """
    Checks if a user qualifies for VIP status.

    Input: user_id (str)
    Output: True if VIP, False otherwise
    """
    user = get_user(user_id)

    if user.balance >= 5000 and user.transaction_count > 5 and user.complaints == 0:
        promote_to_vip(user_id)
        return True
    return False


# Suspend user

def suspend_user(super_user_id, user_id):
    """
    Suspends a user from the system.
    Input: super_user_id (str), user_id (str)
    Output: True if successful, False otherwise
    """
    if valid_super_user(super_user_id) and valid_user(user_id):
        change_user_status(user_id, "suspended")
        return True
    return False

```

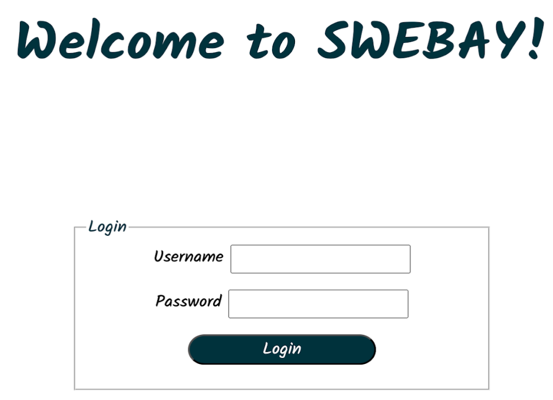


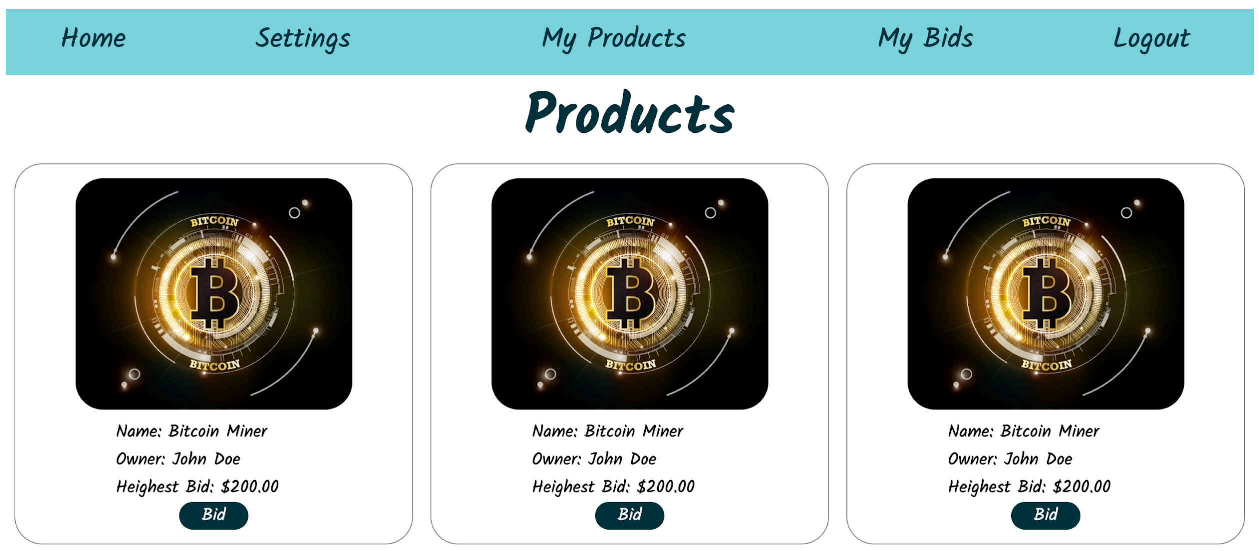
```python
# Resolve disputes

def resolve_dispute(super_user_id, transaction_id, resolution_details):
    """
    Resolves a user dispute related to a transaction.
    Input: super_user_id (str), transaction_id (str), resolution_details (str)
    Output: True if successful, False otherwise
    """
    if valid_super_user(super_user_id) and valid_transaction(transaction_id):
        log_resolution(transaction_id, resolution_details)
        return True
    return False

```



```python
def file_complaint(user_id, transaction_id, complaint_text):
    """
    Allows a user to file a complaint for a transaction.
    Input: user_id (str), transaction_id (str), complaint_text (str)
    Output: complaint_id (str) if successful, None otherwise
    """
    if valid_user(user_id) and valid_transaction(transaction_id):
        complaint_id = generate_complaint_id()
        log_complaint(complaint_id, user_id, transaction_id, complaint_text)
        return complaint_id
    return None
```

Part 5) GUI Mockups & Prototype-


Welcome to SWEBAY!



Products

Name: Bitcoin Miner
Owner: John Doe
Heighest Bid: $200.00

Name: Bitcoin Miner
Owner: John Doe
Heighest Bid: $200.00

Name: Bitcoin Miner
Owner: John Doe
Heighest Bid: $200.00

Part 6) Memos of group meetings

10/8/24 - 3:15 pm

Meeting 1- Discussing roles for group members

In our first group meeting, we focused on defining roles and responsibilities to ensure an organized and efficient workflow throughout the project. With four members, we divided tasks based on individual strengths and interests. One member took the role of **Project Manager**, ensuring deadlines are met, organizing meetings, and tracking progress. Another member took the **System Designer role**, responsible for creating the Chen-style E-R diagram and designing the overall system architecture. The third member took the role of **Developer**, implementing key functionalities like user registration, bidding, and transaction processing, as outlined in the requirements. Finally, the fourth member took charge of the backend of the project making sure that all data is stored correctly as well as testing the system for errors and ensuring the personalized features work correctly. By distributing these roles, we'll cover all aspects of the project while fostering accountability and collaboration. Overall we agreed to help each other out if we were stuck in anything complicated so even though our roles were defined it didn't mean that it was finite as others could also jump in to help with the structure diagrams or the frontend of things. Our goal was to make the best project possible as a team.

10/14/24- 3:30 pm

Meeting 2- Working on Phase 1 report

In our second meeting, we focused on discussing the **Phase 1 Report** and strategizing how to effectively divide the workload among our four team members. We began by reviewing the Software Requirements Specification (SRS) template, which outlines sections such as **Introduction**, **Purpose**, **Scope**, **Use-Case Model Survey**, **Specific Requirements**, and **Supporting Information**. To ensure an even distribution of tasks, we assigned each member a specific section: one member took responsibility for the **Introduction, Purpose, and Scope**, defining the overall goals and context of the system. Another member focused on the **Use-Case Model Survey and Assumptions**, identifying key use cases and dependencies. The third member worked on the **Specific Requirements**, detailing the functional and non-functional requirements to guide the system's design and implementation. Finally, the fourth member handled the **Supporting Information and Supplementary Requirements**, compiling additional details such as indexes, appendices, and diagrams. Throughout the meeting, we emphasized the importance of collaboration and maintaining a consistent tone and structure across the document. We agreed to regularly review each other's work to ensure quality and coherence.

11/5/24

Meeting 3- Working on Phase 2 report

In our third meeting, we focused on planning and dividing the tasks for the **Phase 2 Report**. This phase required us to provide a detailed design of the system, including the **Collaboration Class Diagram**, **E-R Diagram**, **pseudo-code for all methods**, and **GUI prototypes**. To ensure efficient progress, we split the workload strategically among our four members. One member took responsibility for creating the **Collaboration Class Diagram** and ensuring it accurately reflected the interactions between system components. Another member worked on the **E-R Diagram**, incorporating attributes and keys for each class, and ensuring it aligned with the Chen notation we discussed earlier. The third member focused on writing the **pseudo-code** for all methods, providing clear input, output, and logic for each functionality. Lastly, the fourth member handled the **GUI screens**, designing major system interfaces and creating a sample prototype to demonstrate key functionalities. We also assigned each member the task of documenting their work and summarizing key points for the meeting memos. By the end of the meeting, everyone had a clear understanding of their responsibilities, and we set a timeline to review and integrate our work into a cohesive report.

Part 7) GitHub Repo:

https://github.com/kdukuray/ecom-bidding-csc322