SHWN

eWeive
Phase II: Design Report
For eWeive Online Marketplace

Version <2.0>

Table of Contents

Introduction

Below is an overview of the online marketplace app, eWeive, illustrated using a collaboration class diagram. It shows the actors, their possible interactions with the user interface, and what databases we will use.

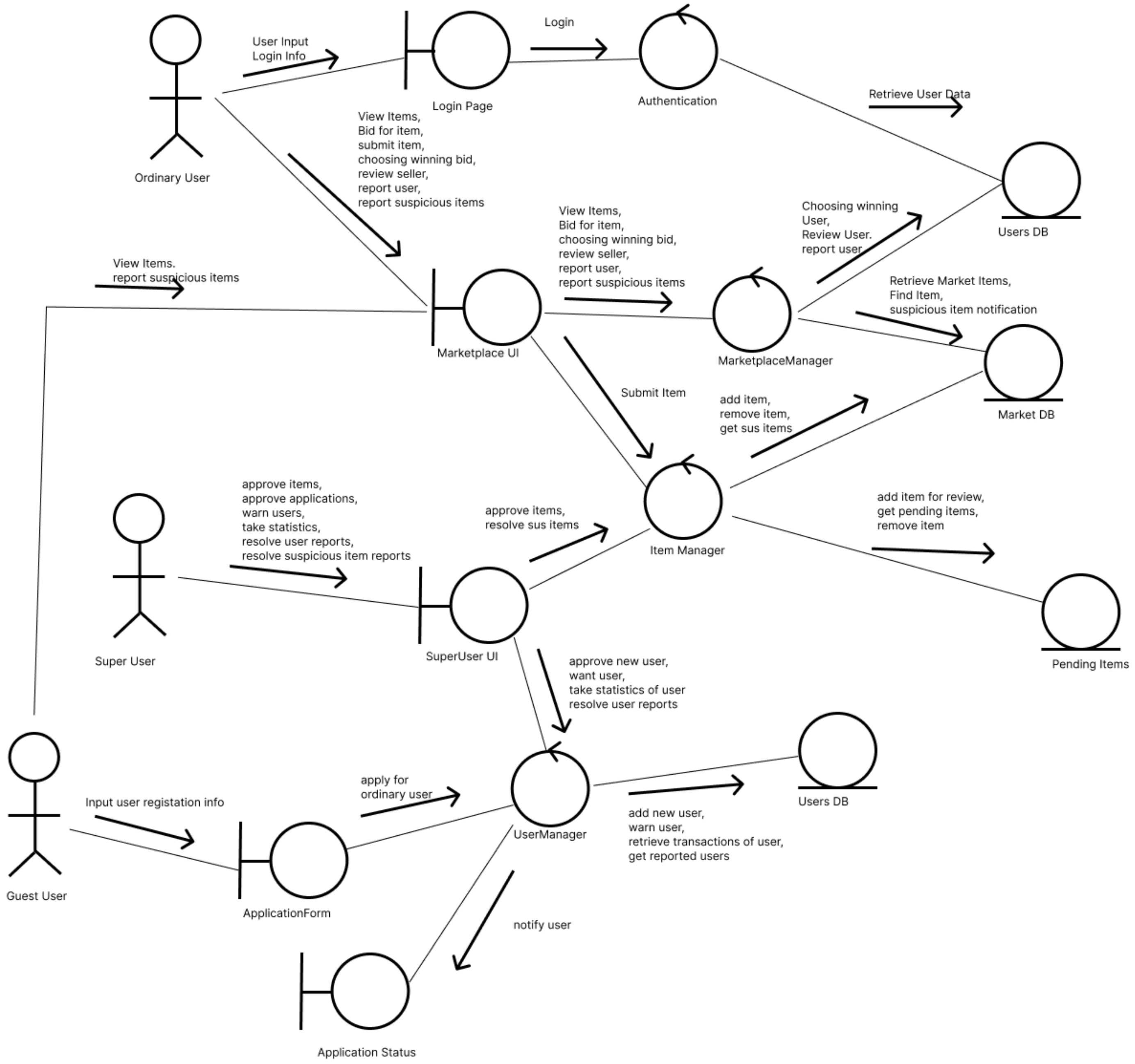


Figure 1. Overview of system with collaboration class diagram

Below are examples of ordered class diagrams for 3 use cases.

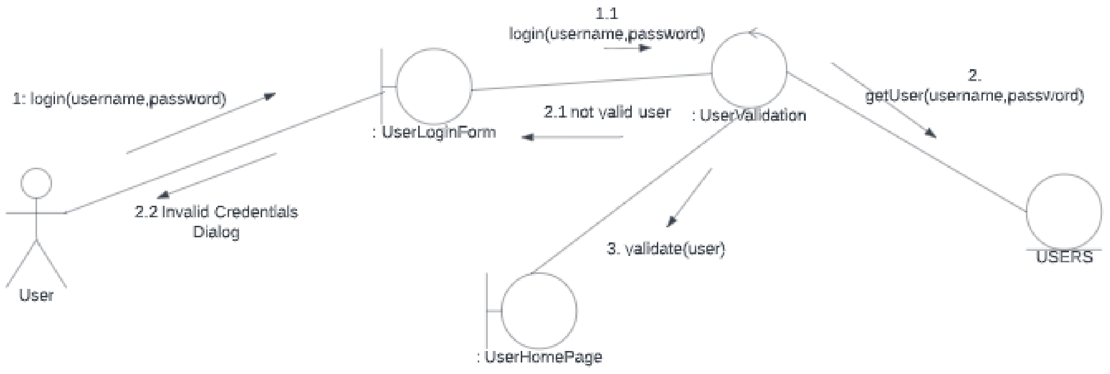


Figure 1.1 Login Use Case Collaboration class diagram

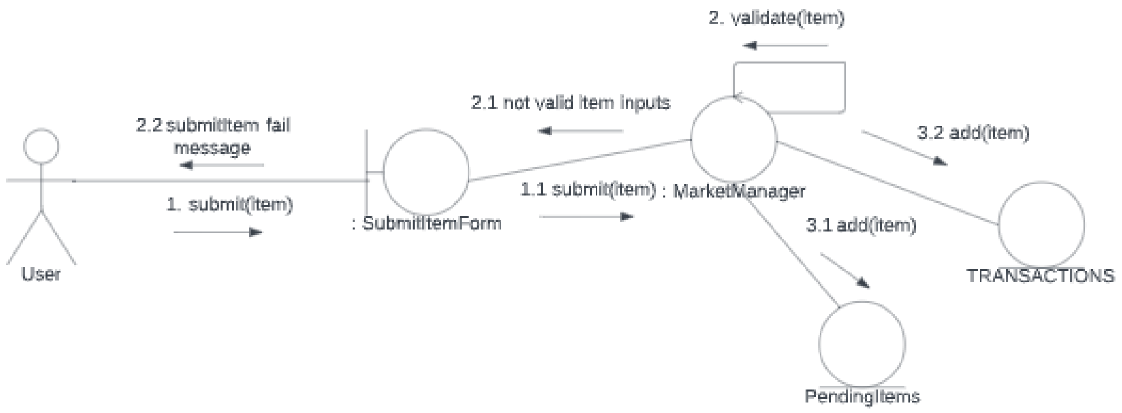


Figure 1.2 Submit Item Use Case Collaboration Class Diagram

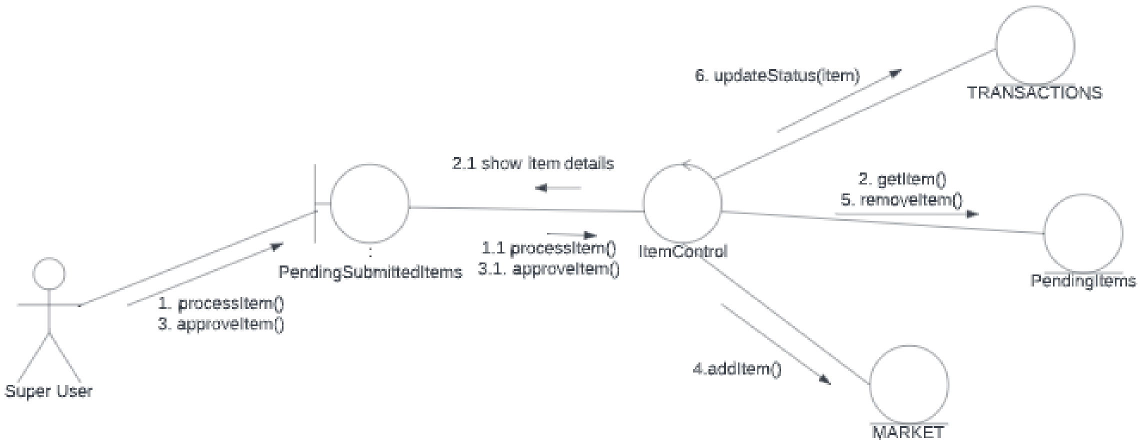


Figure 1.3 Process Item Use Case Collaboration class diagram

Sequence Diagram

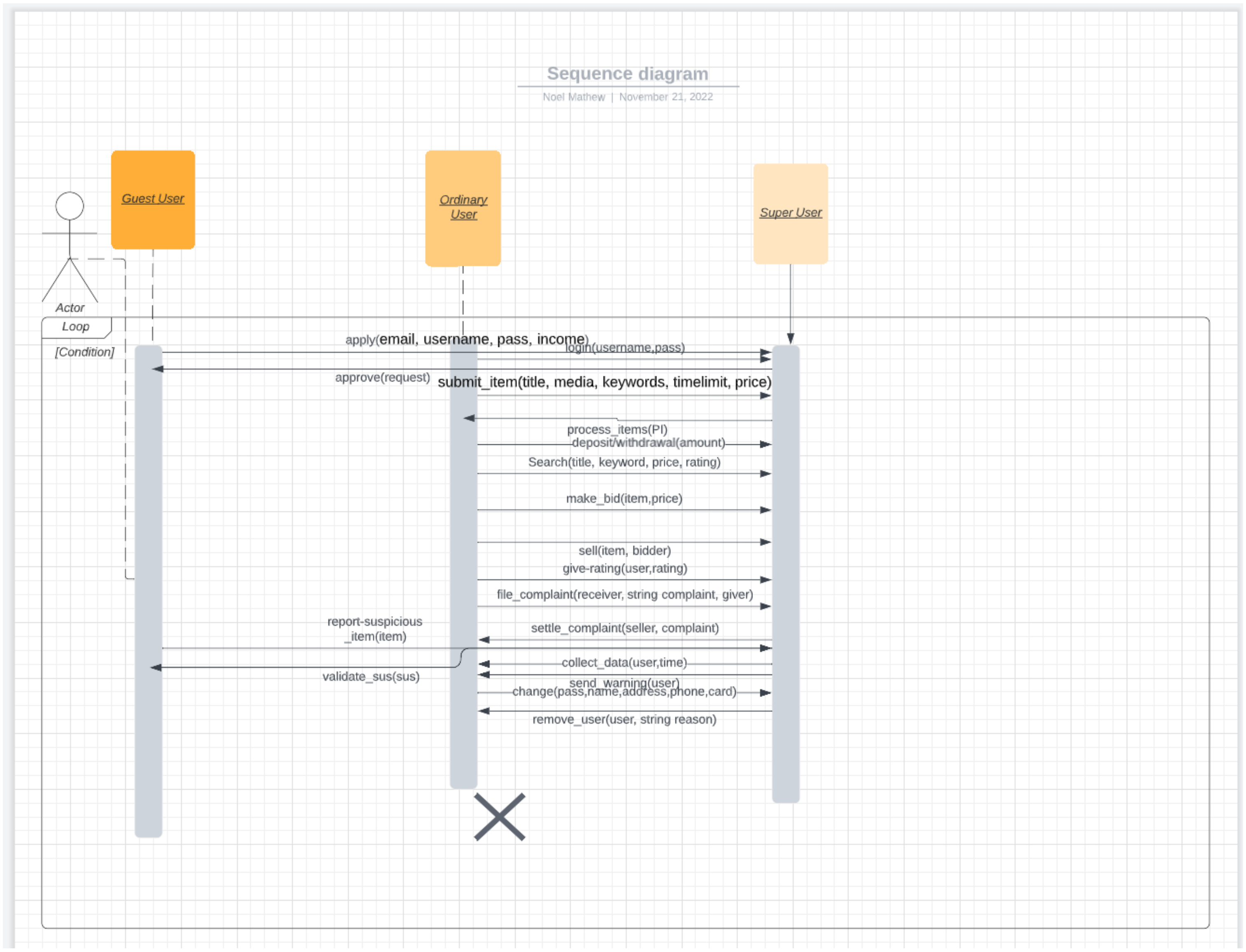

Sequence diagram
Noel Mathew | November 21, 2022
Actor
Guest User
Ordinary User
Super User
Loop
[Condition]
apply(email, username, pass, income)
login(username,pass)
approve(request)
submit_item(title, media, keywords, timelimit, price)
process_items(PI)
deposit/withdrawal(amount)
Search(title, keyword, price, rating)
make_bid(item,price)
sell(item, bidder)
give-rating(user,rating)
file_complaint(receiver, string complaint, giver)
report-suspicious _item(item)
settle_complaint(seller, complaint)
collect_data(user,time)
validate_sus(sus)
send_warning(user)
change(pass,name,address,phone,card)
remove_user(user, string reason)

Figure 2. Sequence diagram of system

All Use Cases Diagrams

- All Use cases normal and exceptional explained:
- A GU applies to be a OU apply(), SU processes application and denies the application
- A GU applies to be a OU apply(), SU processes application and accepts the application, the OU submits information for an item they want to sell, the OU sells the item, the customer must grade the OU after buying the item,

- A GU applies to be a OU apply(), SU processes application and accepts the application, the OU browses in the available items marketplace, the OU submits a bid to buy an available item, their bid is chosen, must grade the seller
- A GU applies to be a OU apply(), SU processes application and accepts the application, the OU browses in the available items marketplace, OU deposits money into user account, the OU submits a bid to buy an available item, their bid is chosen, must grade the seller
- A GU browses available items, reports to SU about a suspicious item, SU removes the item and the OU who posted it and a police report is generated.
- A GU applies to be a OU apply(), SU processes application and accepts the application, the OU browses in the available items marketplace, OU deposits money into user account, the OU submits a bid to buy an available item, their bid is chosen, must grade the seller, buyer files a complaint to SU against the seller because the purchased item has a problem, the SU sends a warning
- A GU applies to be a OU apply(), SU processes application and accepts the application, the OU browses in the available items marketplace, OU deposits money into user account, the OU submits a bid to buy an available item, their bid is chosen, must grade the seller, buyer files a complaint to SU against the seller because the purchased item has a problem, the SU sends a warning, it is the OU's 3rd warning, the OU is removed from the system, the OU is blocked for future re-application
- A GU applies to be a OU apply(), SU processes application and accepts the application, the OU submits information for an item they want to sell, the item is denied
- A GU applies to be a OU apply(), SU processes application and accepts the application, the OU browses in the available items marketplace, OU deposits money into user account, the OU submits a bid to buy an available item, their bid is chosen, must grade the seller, buyer files a complaint to SU against the seller because the purchased item has a problem, the SU sends a warning, the item is removed, the item is blacklisted from the system

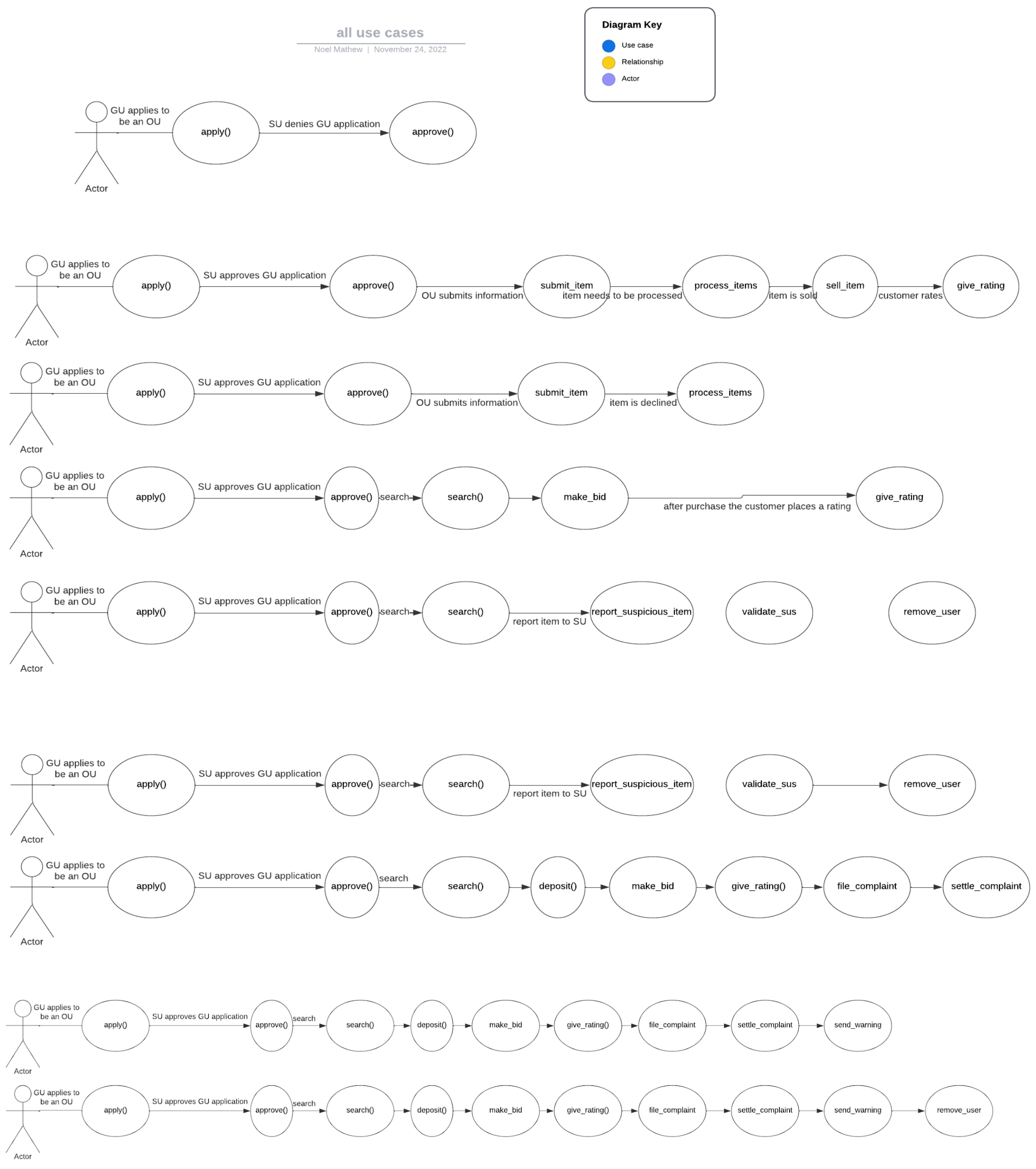


Figure 3. UML Use Case scenarios

Petri Nets for 3 Use Cases

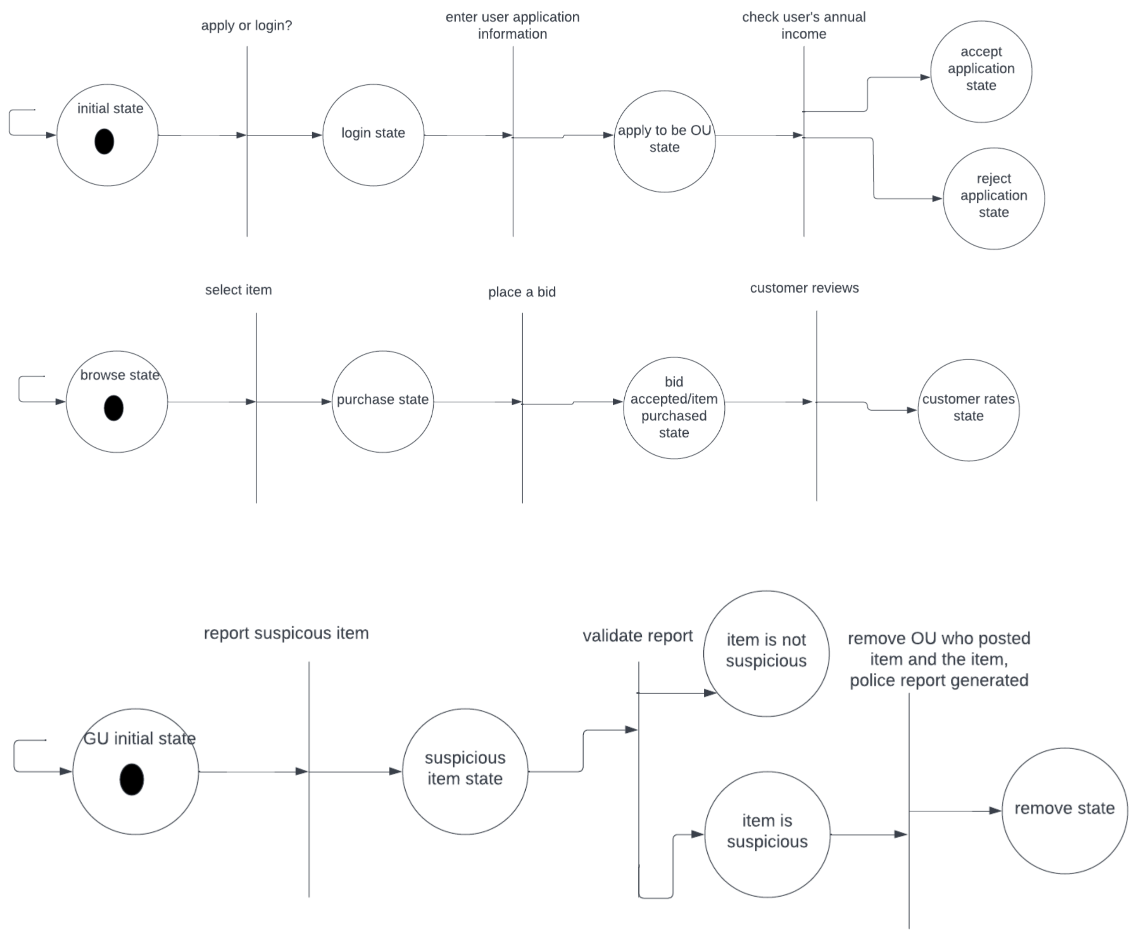


Figure 4. Petri Nets for Use Cases

E-R Diagram of the Entire System: eWeive E/R diagram

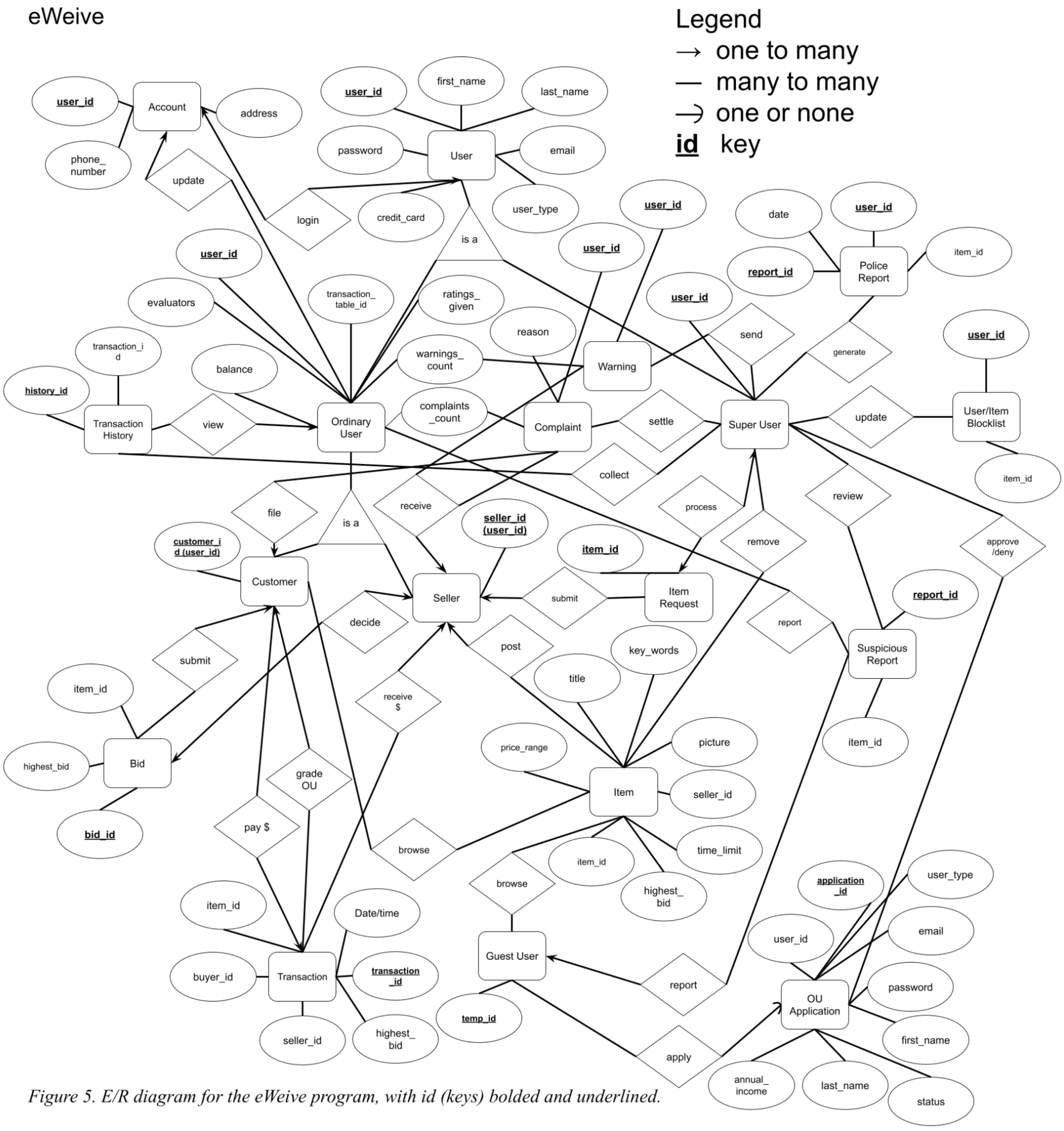


Figure 5. E/R diagram for the eWeive program, with id (keys) bolded and underlined.

Detailed Design

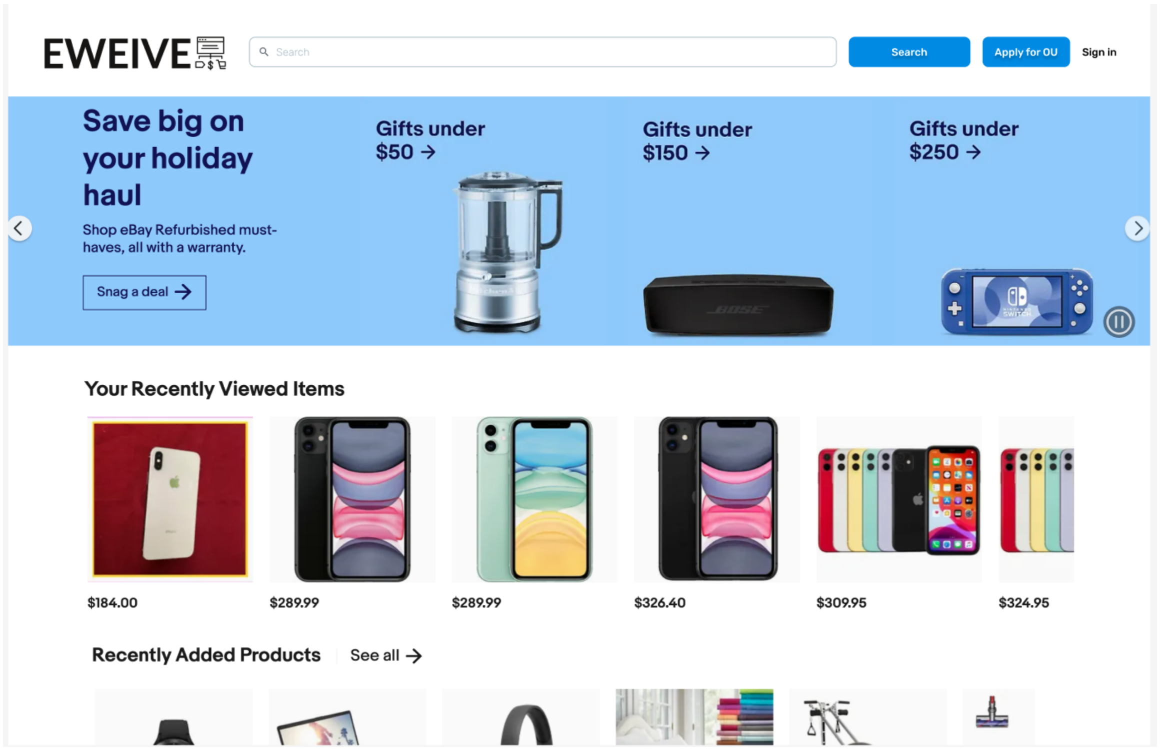
```
def apply(id, email, user_id, first_name, last_name, password,
income, user_type, status):
     “““ Guest Users can apply to be an Ordinary User
     Parameter: id (primary key), email, user_id, first_name,
     last_name, password, income, user_type, status
     Output: string returning “request made”  ”””

     ADD to OU APPLICATIONS TABLE
     return “request made”


def login(username, pass):
     “““ Parameter: username, password
     Output: adds the username to the USERS table and
     authenticates which type of user the user is. ”””

     return username in USERS and pass auth


def approve(request):
     “““ Apply to be an OU
     Parameter: request
     Output: if the user income is above a predetermined amount,
     the user will be added to the USER table, if not the
     request will be denied and deleted from the requests table.
     String returning “accept” or “deny” will be returned
     accordingly.”””

     If req.income < x
          Return “deny”
     Add request to USER table
     Delete from REQS
     return “approve”

def submit_item(title, media, keywords, timelimit, price):
     “““ Submit the information (a picture or
     video,title,keywords,time limit,price range) of the item(s)
     s/he wants to sell
```

```
      Parameter: title, media, keywords, timelimit, price
      Output: this function will take in item information and add
      it to the transactions table and items table, if the item
      was previously on the item blacklist, it will be removed. A
      string is returned.
         - All pending items will be added to the transaction
           table to keep a history of user transactions whether
           accepted or not. """

      PI=pending_item(title, media, keywords,timelimit, price)
      INSERT PI to TRANSACTIONS table
      INSERT PI to ITEMS table
           If (PI = item.blacklist)
                delete from ITEMS table
      Return "submission made"


def process_items(PI):
      """
      Process items the OUs submitted intending to sell, some can
      be publicly available on the system and some may be denied
      and communicated to the OU.
         Parameter: pending item
         Output: if the pending item is less than the price limit
      and the pending item time limit is less than the timelimit
      specified, then the item can be added to the MARKET table.
      If not it will be deleted from the ITEMS table and a string
      will be returned"""

      If (PI.price < Limit) AND (PI.time < timelimit)
           INSERT PI to MARKET table
      Else
           DELETE FROM ITEMS table
           Return "not approved"


def Search(title, keyword, price, rating):
      """
      Browse/search available items based on title, keywords,
      price range, and ratings
        Parameters: title, keyword, price, rating
        Output: if an item matches the filter of specifications
      that was passed in, the item will be displayed on the
      market screen”””

      If (title AND keyword AND price AND rating in MARKET)
           Show item in MARKET
      Return


 def Deposit(amount):
      “““
      Deposit or withdraw money (symbolically, no real money is
      involved in the system) from your own account. After a
      transaction, the buyer’s money is transferred to the
      seller. A bidder cannot place a bid with amount more than
      s/he has in the system
      Parameter: amount
      Output: updates the user balance
      This function takes in the amount that the user wants to
      deposit and updates their balance to reflect that amount
      added”””

      If amount>0
           usr.balance = usr.balance + amount


 def Withdrawal(amount):
      “““
      Deposit or withdraw money (symbolically, no real money is
      involved in the system) from own account. After a
      transaction, the buyer’s money is transferred to the
      seller. A bidder cannot place a bid with amount more than
      s/he has in the system
      Parameter: amount
      Output: updates the user balance
      the user wants to withdraw and updates their balance to
      reflect that amount removed”””

      If amount < usr.balance
           Usr.balance = usr.balance - amount
      return

 def file_complaint(receiver, complaint_comment, giver):
      """ GUs can file complaints against another GU when they
      have an issue with their purchased item, or the seller sees
      one buyer giving 3 1-stars or 3 5-stars to the seller's
      item(s)
      Parameters: string receiver GU, string complaint_comment,
      string giver GU
      Output: new entry in complaints database table """

      connect to MySQL database
      create sqlite3 cursor
      # assumption: complaints database table exists, thus we
      # can insert it with the statement below
      insert_sql = """INSERT INTO COMPLAINTS (receiver, comment,
      giver) VALUES (receiver, complaint_comment, giver)"""
      execute the query insert_sql
      commit the connection to the database
      close the cursor

 def settle_complaints(complaints):
      """ SUs settle complaints by retrieving the database table
      complaints and validating or denying the complaints.
      Parameter: COMPLAINTS table
      Output: the GU seller's complaint count increases by 1
      and/or the complaint is removed """

      # the SU will be able to view the entire complaints table
      # in the GUI, with a button/boolean on whether the SU
      # wants to deny or accept the complaint's validity
      if the complaint is valid:
           seller = complaints.seller_id
           seller.complaint_count += 1
      # in the instance of denying a complaint and after
      # validating a complaint, remove the complaint line from
      # the database
      delete from COMPLAINTS where complaints.comment = …

 def remove_user(user, reason):
      """ SUs can remove users any time with a reason. Removed
      users will not be able to re-apply to be a GU. """

      # add the user to the blocklist
      insert_sql_blocklist = """INSERT INTO blocklist (username,
      email, phone_number) VALUES (user.username, user.email,
      user.phone_number)"""
      # remove the user from the users_table to prohibit login
      delete from users_table

 def send_warning(user):
      """ SUs send warnings if a user has more than 2 complaints
      or a user’s rating is less than 2 from 3 or more
      evaluators. The user is automatically removed from the
      system if they receive two warnings.
      Parameters: user
      Output: user warning count increases with a warning or user
      is removed from the system """

      if user.complaint_count > 2 || (user.average_rating < 2 &&
      user.evaluator_count >= 3):
           if user.warning_count == 1:
                remove_user(user, reason)
           send warning string message to user
           user.warning_count += 1

 def report_suspicious_item(item):
      """ GUs and OUs can report a suspicious item which will be
      added to the SUSPICIOUS_ITEMS SQL table to be reviewed.
      Parameters: item from MARKET table
      Output: a new row is added to suspicious items table """

      insert_sql = """INSERT INTO SUSPICIOUS_ITEMS (seller_id,
      title) VALUES (item.seller_id, item.title)"""


 def validate_suspicious_item(suspicious_items):
      """ The SU views the data table and denies or accepts the
      reports. If the SU accepts the report, the user who posted
      it is removed.
      Parameter: SUSPICIOUS_ITEMS table
      Output: removed user and/or deleted row from table, returns
      a list variable to be used in HTML code to send a popup
      alert ”””

      if the suspicious_item row is valid:
           create a list variable storing the row
           add item to BLACKLIST_ITEMS table
           remove suspicious_items.item from the MARKET
           # below we call the remove user function
           remove_user(suspicious_items.user, “suspicious item”)
           return list
      delete suspicious_item row from SUSPICIOUS_ITEMS

 def make_bid(item, price):
      “““ Ordinary Users can make a bid on a selected item. The
      new bid must be higher than the current highest bid ”””
      Parameter: item to bid for, price to bid for
      Output: updates the items bid if a valid bid was made
      if price > item.high_bid:
           item.high_bid = price

 def sell(item, BIDDER):
      “““ Once time is up for an item, users can choose which
      bidder to sell the item to ”””
      Parameters: item that is being sold, the bidder of choice
      that the item will go to
      Output: item gets sold to the bidder
      if item.time == 0:
           add to TRANSACTIONS
           delete from MARKET

 def give_rating(user, rating):
      “““ Users will give ratings to another user if they have
      bought an item from them  ”””
      Parameters: user to rate, rating to give
      Output: user’s ratings gets updated
      if TRANSACTIONS.seller_id==user.id:
           ratings_given.append(rating)
           sum = sum of ratings
           user.rating = sum/evaluators.length
           check givers ratings to see if complaint filed
```

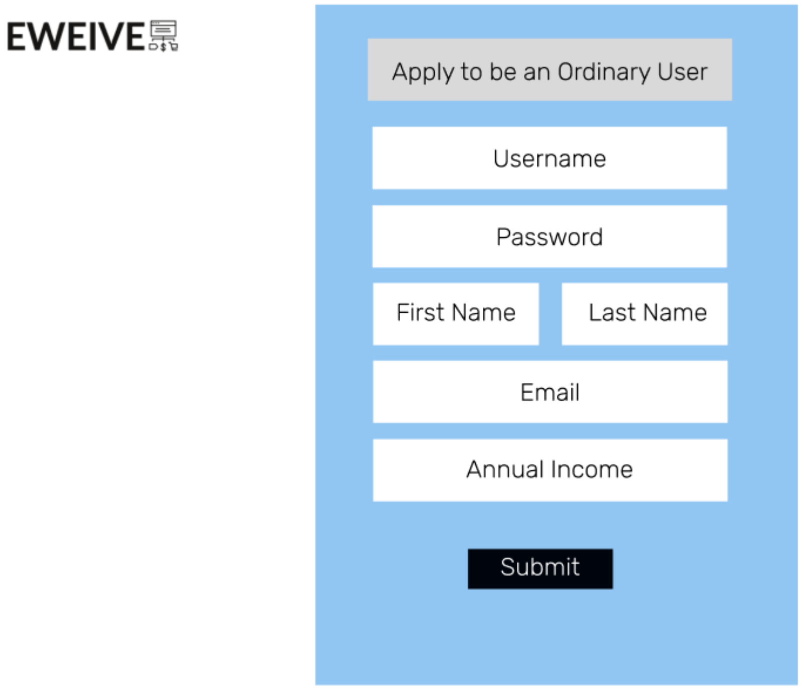
```
def collect_data(user, time):
     “““ Super Users will be able to collect data from a select
     user over a certain time frame ”””
     Parameters: user to collect data from, time range to select
     data from
     Output: the user’s transactions
     select user transactions from given time frame
     return user transactions

def change(name, password, address, number, card):
     “““ Users will be able to update their account
     information ”””
     Parameters: name of the user, password of account, address
     of the user, phone number of the user, card number of the
     user
     Output: user’s account information changed
     user.name = name
     user.pass = password
     user.address = address
     user.number = number
     user.card = card
```

System Screens/Prototype

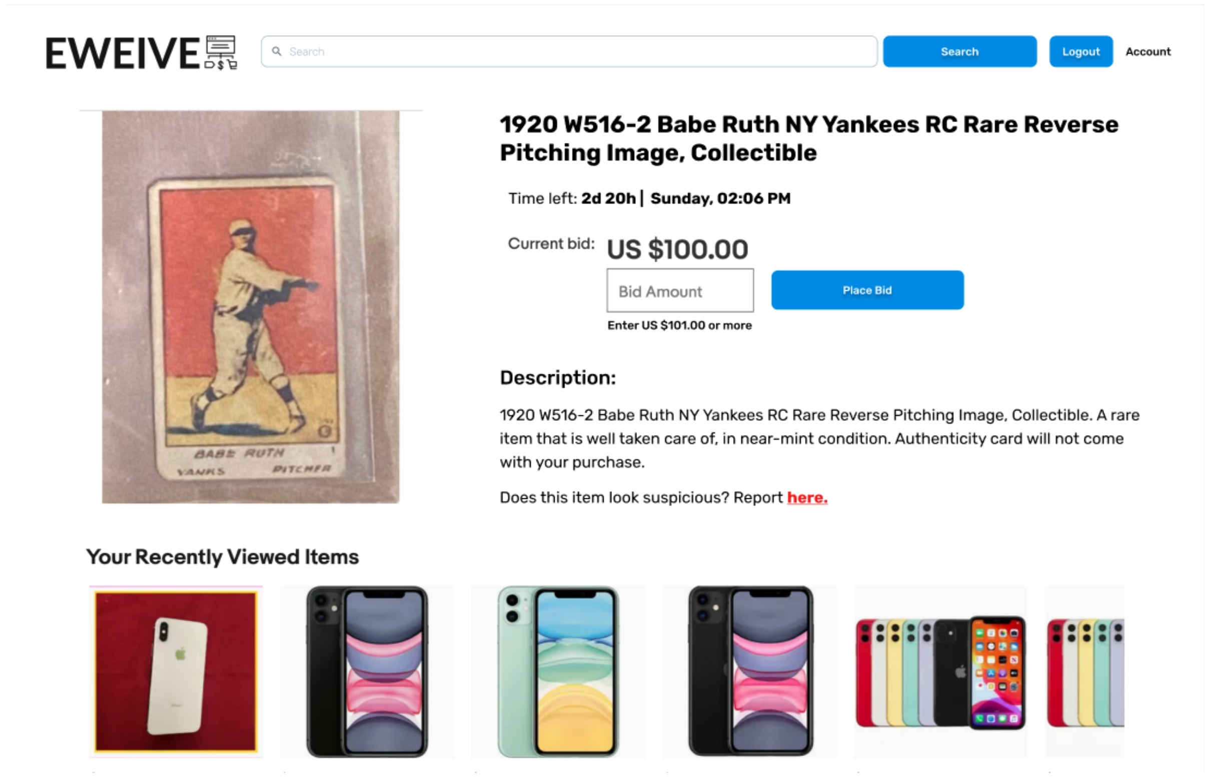
Home Page:



Login Page:



Item Page:



1920 W516-2 Babe Ruth NY Yankees RC Rare Reverse Pitching Image, Collectible

Time left: **2d 20h | Sunday, 02:06 PM**

Current bid: **US $100.00**

Bid Amount — Place Bid

Enter US $101.00 or more

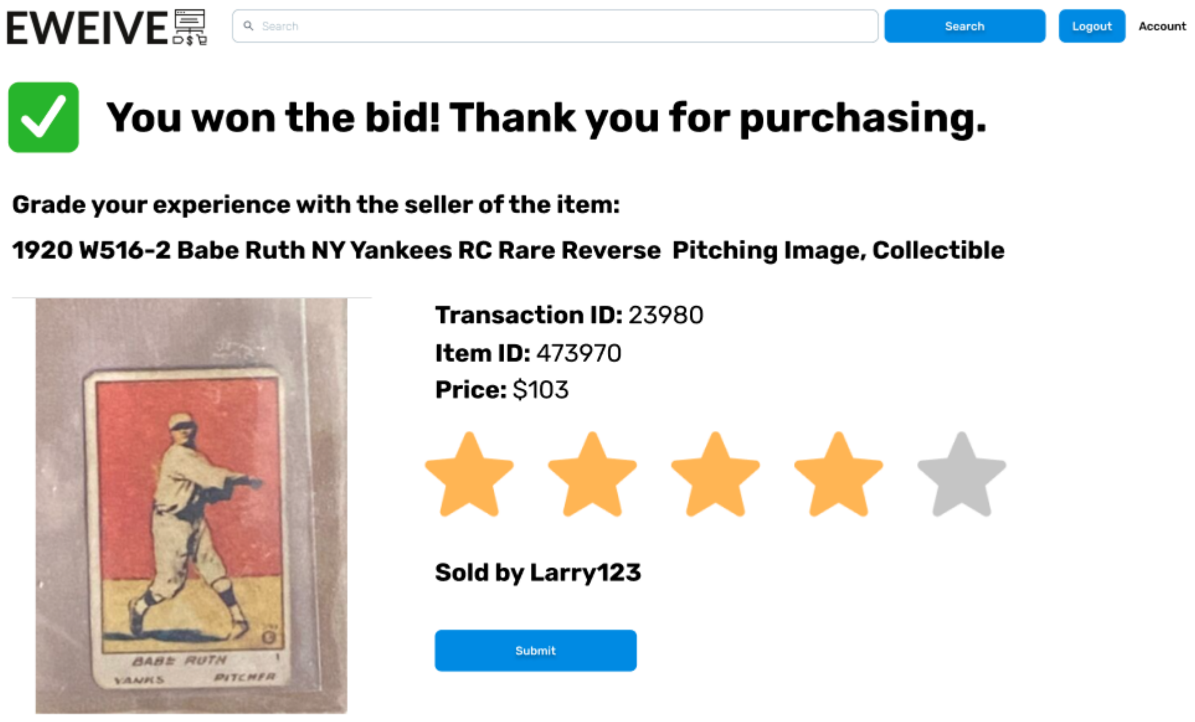
Description:

1920 W516-2 Babe Ruth NY Yankees RC Rare Reverse Pitching Image, Collectible. A rare item that is well taken care of, in near-mint condition. Authenticity card will not come with your purchase.

Does this item look suspicious? Report **here.**

Your Recently Viewed Items

Confirmation Page:



You won the bid! Thank you for purchasing.

Grade your experience with the seller of the item:

1920 W516-2 Babe Ruth NY Yankees RC Rare Reverse Pitching Image, Collectible

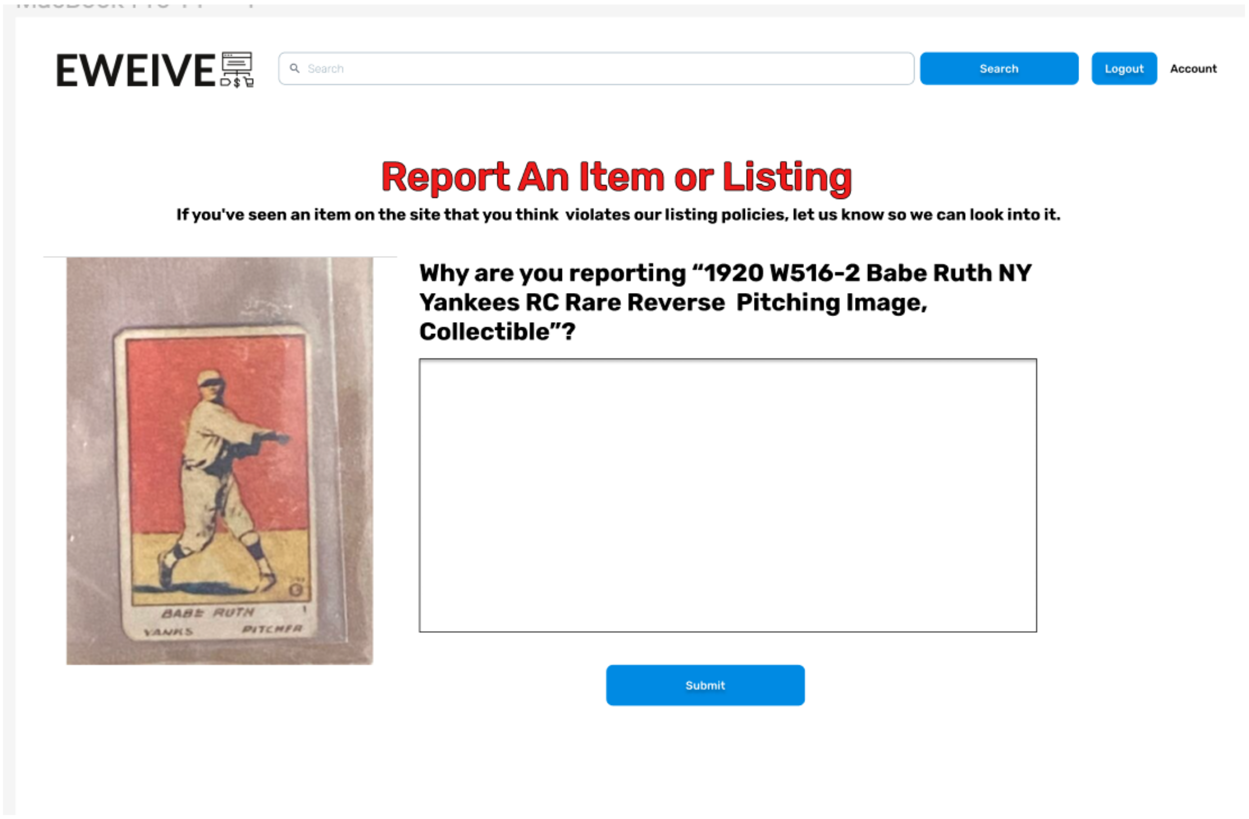
Transaction ID: 23980
Item ID: 473970
Price: $103

★★★★☆

Sold by Larry123

Submit

Report an Item Page:



EWEIVE

Search | Search | Logout | Account

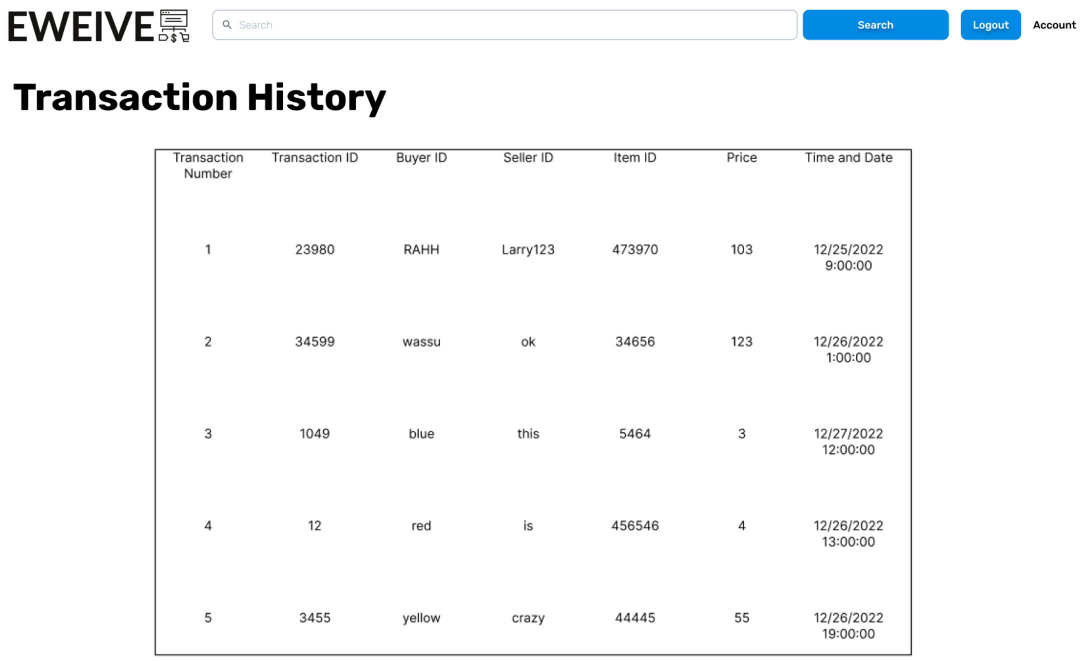
Report An Item or Listing

If you've seen an item on the site that you think violates our listing policies, let us know so we can look into it.

Why are you reporting "1920 W516-2 Babe Ruth NY Yankees RC Rare Reverse Pitching Image, Collectible"?

Submit

Transaction History Page:



EWEIVE

Search | Search | Logout | Account

Transaction History

Transaction Number	Transaction ID	Buyer ID	Seller ID	Item ID	Price	Time and Date
1	23980	RAHH	Larry123	473970	103	12/25/2022 9:00:00
2	34599	wassu	ok	34656	123	12/26/2022 1:00:00
3	1049	blue	this	5464	3	12/27/2022 12:00:00
4	12	red	is	456546	4	12/26/2022 13:00:00
5	3455	yellow	crazy	44445	55	12/26/2022 19:00:00

Group Meeting Memos & Goals

Date	Version	Description	Author(s)
01/11/22	1.0	Updated proposal to include basic technical details and use-case diagram	Siema Alam, Noel Mathew, Hong Wei Chen
10/11/22	2.0	Worked on Phase II: Design Report, wrote out pseudo code for website functions. Delegated how the report should be split up	Siema Alam, Noel Mathew, Hong Wei Chen
10/11/22 - 24/11/22	2.0	Created/updating E/R diagram and database design, Figma screens: item page, review page, home page, partial pseudocode, created goals for future meets	Siema Alam
10/11/22 - 24/11/22	2.0	Created/updating sequence diagram, all use case models, all Petri Nets for three use cases, Figma screen: log in, report page, partial pseudocode	Noel Mathew
10/11/22 - 24/11/22	2.0	Created collaboration class diagram, Figma screen: transaction history table, partial pseudocode	Hong Wei Chen
24/11/22	2.0	Reviewed Phase II Report and submitted the final version of the Design Report.	Siema Alam, Noel Mathew, Hong Wei Chen
26/11/22	3.0	**GOAL**: set-up github repo system with major filing organization/roles, complete login/signup authentication, SQL connections	Siema Alam, Noel Mathew, Hong Wei Chen
30/11/22	3.0	**GOAL:** completed user functions/permissions setup aside from UI, constructed basic UI, decide on 10% bonus feature	TBD
4/12/22	3.0	**GOAL:** update UI to match prototypes and updated functionalities	TBD
7/12/22	3.0	**GOAL:** populate SQL backend with mock users/items/transactions, implement bonus feature	TBD
11/12/22	3.0	**GOAL:** finalize project for demo day, reduce any bugs/errors, update docs	TBD
13/12/22	3.0	**GOAL:** present project in-class	Siema Alam, Noel Mathew, Hong Wei Chen

GitHub Repo

https://github.com/SHWN-Co/eweive