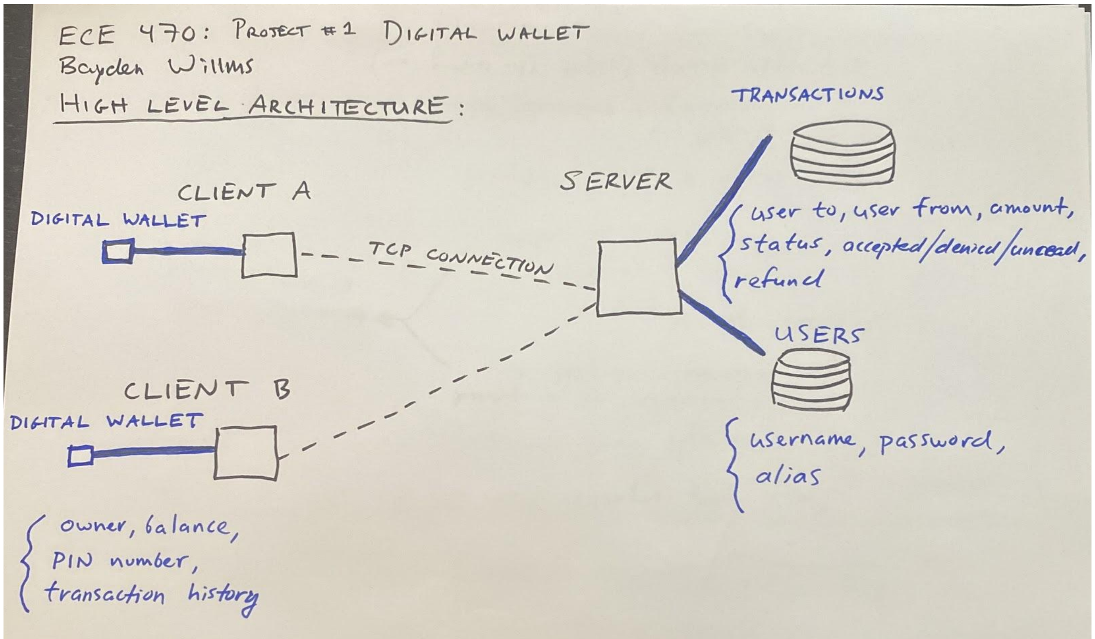
ECE 470: Project 1 Digital Wallet

Bayden Willms

Project Description:

This is a client/server style TCP Digital Wallet service, which is implemented using Python. Users maintain a Digital Wallet on their local device, which contains all the necessary information to use the service, such as a balance, PIN number, and a history of transactions. To view incoming messages and send payments to other users, login is required before a connection is established with the server. The Top Level Menu is given to the client locally without a connection to the server, and has options to register an account, login, open your wallet, and exit. The digital wallet can be accessed locally without server connection, and requires a username and correct PIN number. Once opened, the wallet menu allows the client to check their balance, deposit money, and view history of their transactions. A login requires a username and password, and once a login request is successful, then a TCP connection begins. Once connected, users are given the Main Menu, and can send payment requests to other users, cancel a payment request that they made, or refund a payment that they made to another user, as well as going back to the previous menu and exiting the program. Additionally, users have a message inbox, which informs them if they have been paid or if someone is requesting a refund or payment to the user. The server maintains two databases which store the information on completed and pending transactions in the system, which includes the amount, the user it was sent by, the user it is being sent to, if it has been completed or is still pending, if it is a refund, and if a payment has been accepted or denied and is still waiting to be paid, and another database which stores information on each user, which includes a username, password, and alias. Each time the server receives a message, it scans the transaction and user databases and creates dictionaries for the set of all transactions and users, respectively. In several scenarios, a request is made by the client and the server adjusts the transaction accordingly, and then when ready, the client's protocol code automatically sends a request for that updated transaction to be placed into the database (in this case, a text file). The text file is edited by copying each line into a list, but if a match is found the transaction is updated accordingly in the list, and the transaction file is rewritten using this updated list of transactions. For the scenarios which require editing or reading the transactions, such as getting new messages, refunding, or canceling a payment, the server handles the request according to the message type, and reads for the parameters that are required for a given scenario to be successful. For example, a client receiving a message that someone paid them requires a transaction to exist in the database that was sent by them, still has a pending status, and has been accepted. Multiple current users cannot be accepted by this service. Clients send transaction requests to the server, and are received at another time when another client logs on. When a new client wants to register to the service, they create a user profile with a username, password, and a Digital Wallet PIN number, and they are added to the server and a Digital Wallet is automatically created for them on their local device. The server continues to wait for incoming messages from other clients when nobody is connected to the server. Therefore, these clients have no interaction with each other, so to use the service the client must know the username of the person they would like to send a request to.

High Level Architecture:


ECE 470: PROJECT #1 DIGITAL WALLET
Bayden Willms
HIGH LEVEL ARCHITECTURE:
CLIENT A
DIGITAL WALLET
TCP CONNECTION
SERVER
TRANSACTIONS
user to, user from, amount, status, accepted/denied/unread, refund
USERS
username, password, alias
CLIENT B
DIGITAL WALLET
owner, balance, PIN number, transaction history

Scenarios:

Scenario 1: Register a new account

- Client runs the application and the Top Level menu is displayed. When register an account is selected, a register request is sent to the server, and the server requests a username, password, an initial deposit, and Digital Wallet PIN number for the new account. The username and password is added to the Users database if an existing user with that username does not already exist, and an alias (number between 1,000-10,000) is also added for that user. Additionally, a Digital Wallet is created on the new user's local device, which contains the owner, the PIN number, and an empty list of their transaction history. They are returned to the Top Level Menu if the register request fails, or are given a confirmation of successful account creation and returned to the Top Level Menu.

Scenario 2: Digital Wallet Login

- A user can login to their Digital Wallet without connecting to the server, or login to their wallet while connected to the server. Either scenario follows the same structure. Since the Digital Wallet is maintained on the client side for security, a user logs into their Digital Wallet with their username and PIN number. If the username or PIN number is incorrect, they are returned to the login/register menu. If it is correct for a wallet on that local device, they are logged into it and can perform scenarios 3-5.

Scenario 3: Deposit money

- A user can login to their Digital Wallet without connecting to the server, or login to their wallet while connected to the server. Either scenario follows the same structure. Since the Digital Wallet is maintained on the client side for security, a user logs into their Digital Wallet with their username and PIN number, and can choose to deposit an amount of money. The Digital Wallet updates its balance locally.

Scenario 4: View Transaction History

- A user can login to their Digital Wallet without connecting to the server, or login to their wallet while connected to the server. Either scenario follows the same structure. Since the Digital Wallet is maintained on the client side for security, a user logs into their Digital Wallet with their username and PIN number, and can choose to view their transaction history, and the previous transactions completed by that user are displayed. The Digital Wallet gets this information locally.

Scenario 5: Check Balance

- A user can login to their Digital Wallet without connecting to the server, or login to their wallet while connected to the server. Either scenario follows the same structure. Since the Digital Wallet is maintained on the client side for security, a user logs into their Digital Wallet with their username and PIN number, and can choose to view their wallet balance, and their balance is displayed. The Digital Wallet gets this information locally.

Scenario 6: Login

- A user can request to login from the Top Level Menu. A login request is sent to the server, and the server requests a username and password from the client. The login username and password is received by the server and the Users database is checked to verify the credentials. If there is a problem with verifying the credentials, an error message is sent back to the client and they are returned to the Top Level Menu. If the login credentials match an existing user, a connection is established between the client and server, and the Main Menu is given to the client.

Scenario 7: Logout / Exit

- A logout or exit can be performed in any menu after a connection has been established after a successful login. A logout message is sent to the server, and the client is disconnected.

Scenario 8: Check Incoming Messages

- From the Main Menu (once logged in), a user can check their incoming messages. A check messages request is sent to the server, and the server returns the number of new messages for that user. If there are no new messages, the client is returned to the Main Menu and told that they do not have any messages. If there are new messages, the number of messages is sent to the user, and the client automatically requests to view the first new message in the transaction database. If the message is informing the client that they have been paid, either by a payment request or by a returning refund, they are informed and their wallets are updated, and then are sent back to the Main Menu. If the message is an incoming payment or refund request, that request can either be Accepted or Denied by the client. If accepted, the amount is subtracted from the client's wallet, and the transaction is updated by the server in the transactions database. If denied, the transaction is just updated by the server.

Here are the requirements for a new message:

- Client is paid after a payment request
    - A transaction must exist such that the user requesting messages sent the transaction, that the status is still pending, and that it has been accepted.
- Client is paid back after a refund request
    - A transaction must exist such that the user requesting messages is the user who the request was sent from (because this is a refund of a previous transaction which the client paid), that the status is pending, and that it has been marked as a refund.
- Client is receiving a payment request
    - A transaction must exist such that the user requesting messages is the user that the transaction was sent to, that the status is pending, and that it has not been marked as accepted or denied yet by that current user.
- Client is receiving a request a previous payment made
    - A transaction must exist such that the user requesting is the user that originally sent the transaction, the status is pending, and it has been marked as a refund

Scenario 9: Send Payment Request

- From the Main Menu (once logged in), a user can request a payment from another user. A payment request is sent to the server, and the server prompts the user for a recipient's username, and the payment amount. That information is sent to the server, and a new transaction is created in the transactions database, as long as there is a user in the server that exists with that username. Then the user logs out. Another user logs on to the service, and when they check their messages (Scenario 8) they see the payment request.

Scenario 10: Send Refund Request

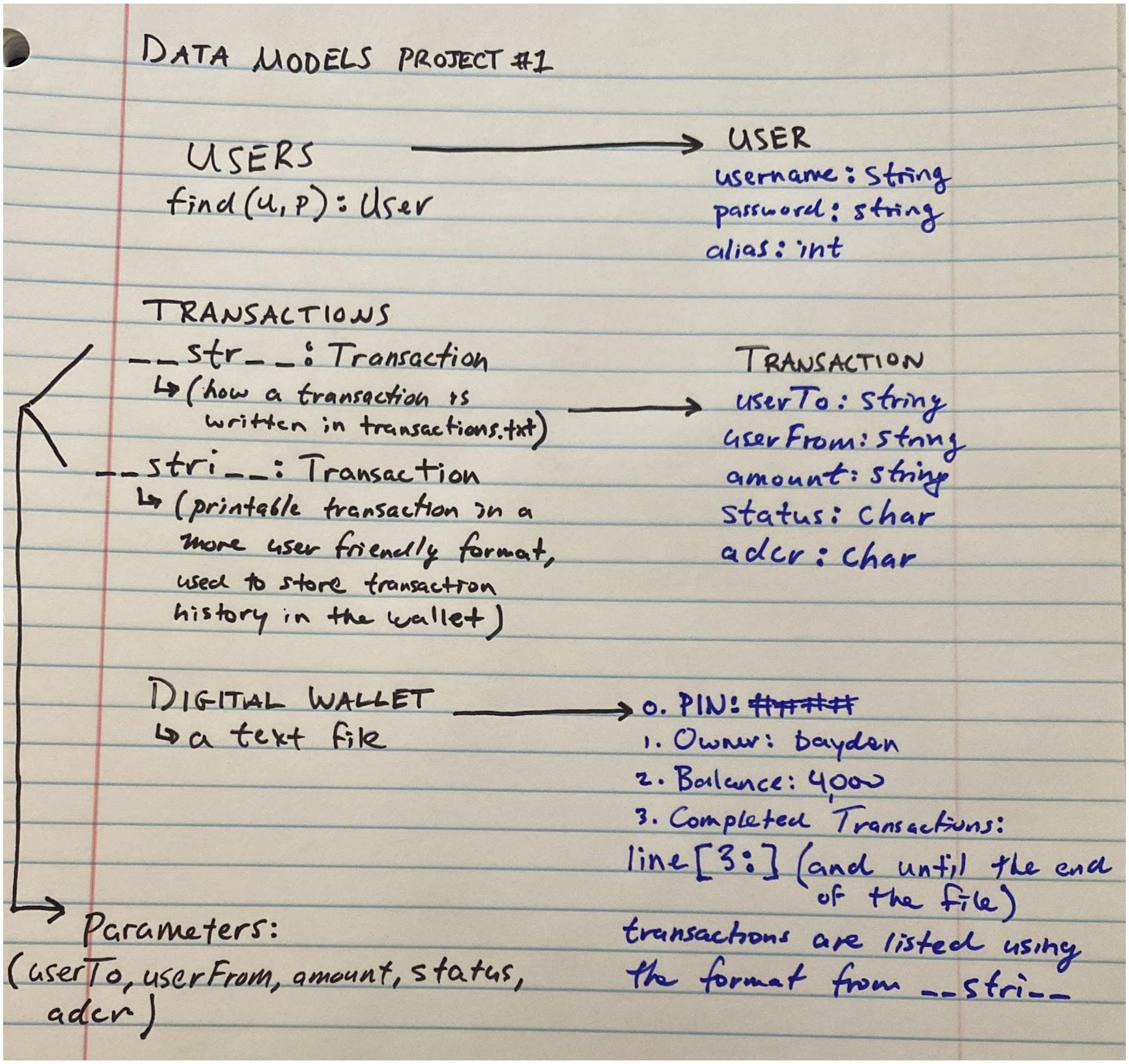
- From the Main Menu (once logged in), a user can request a refund from a payment previously made to another user. To use this, the user should check the transaction history of their wallet and get the Transaction ID number (TID) of the transaction that they would like to refund (see Scenarios 2 and 4). In the Main Menu, a refund request is selected and the request is sent to the server. The server prompts the user for a TID of the payment they would like to refund, and checks that TID to the existing transactions in the transactions database. If that TID matches a completed transaction, then the transaction is changed to a refund and the other user will see that a refund request was made for that payment when they check their messages (see Scenario 8). If the TID does not match an existing payment, or the payment is still pending, then the user is returned to the Main Menu.

Scenario 11: Cancel an Outgoing Payment Request

- From the Main Menu (once logged in), a user can request to cancel an outgoing payment request of a previous payment request. The user selects to cancel a payment, and the cancel request is sent to the server. The server prompts the user for the recipient of the payment that you want to cancel. The recipient's username is inputted, and if a pending transaction between those two users exists in the transactions database, the transaction ID is changed to 0 (it will never be found) and its content is updated to null. This ensures that the recipient user will never see the request when checking their messages (see Scenario 8). If a transaction does not exist where these two users are the sender and recipient and the transaction is still pending, then the user is returned to the Main Menu.

Detailed Design:

Data Models:



DATA MODELS PROJECT #1

USERS
find(u, p): User

USER
username: string
password: string
alias: int

TRANSACTIONS
__str__: Transaction
(how a transaction is written in transactions.txt)
__stri__: Transaction
(printable transaction in a more user friendly format, used to store transaction history in the wallet)

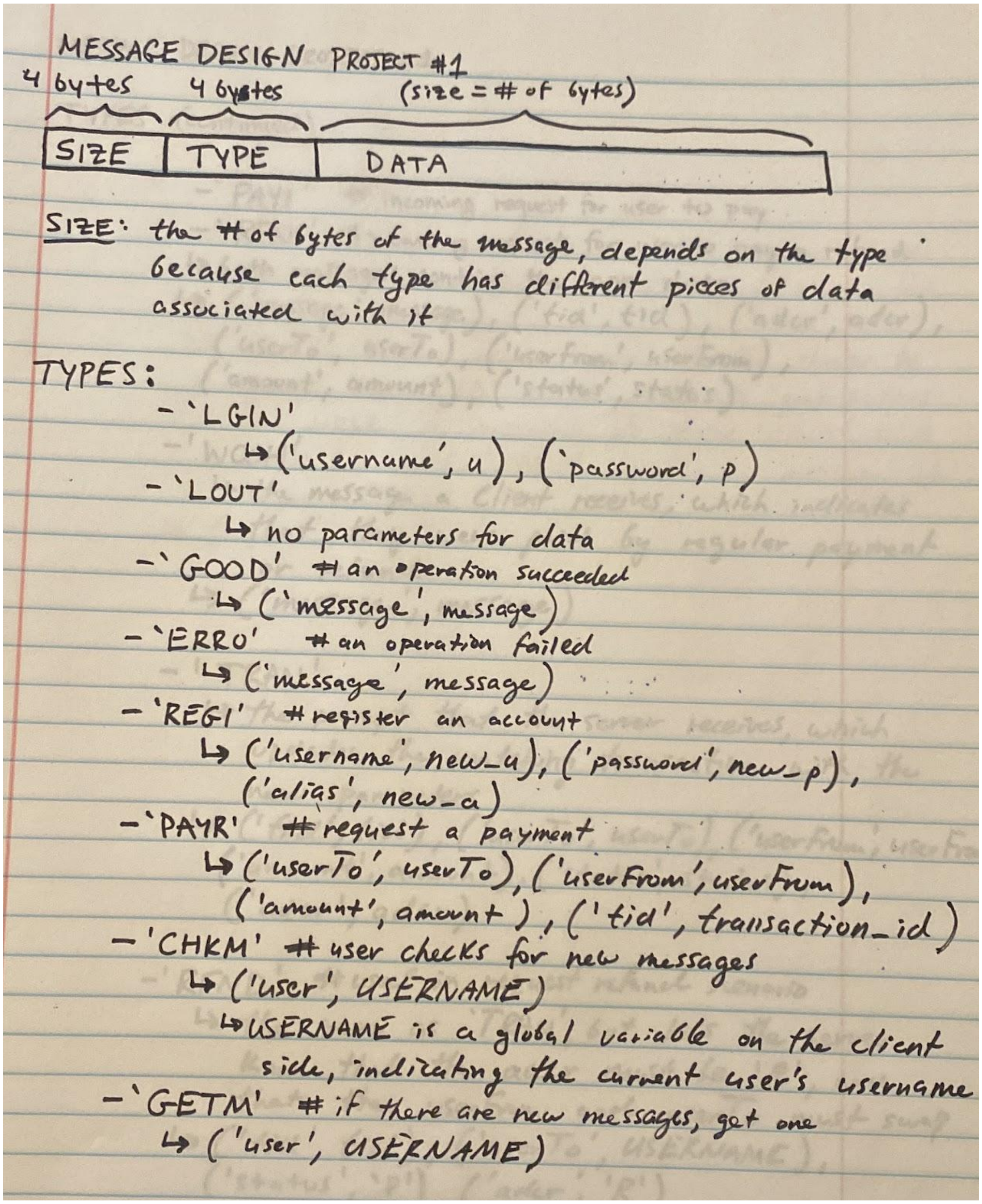
TRANSACTION
userTo: string
userFrom: string
amount: string
status: char
adcr: char

DIGITAL WALLET
a text file

0. PIN: ~~#######~~
1. Owner: bayden
2. Balance: 4,000
3. Completed Transactions:
line[3:] (and until the end of the file)
transactions are listed using the format from __stri__

Parameters:
(userTo, userFrom, amount, status, adcr)

Messages:



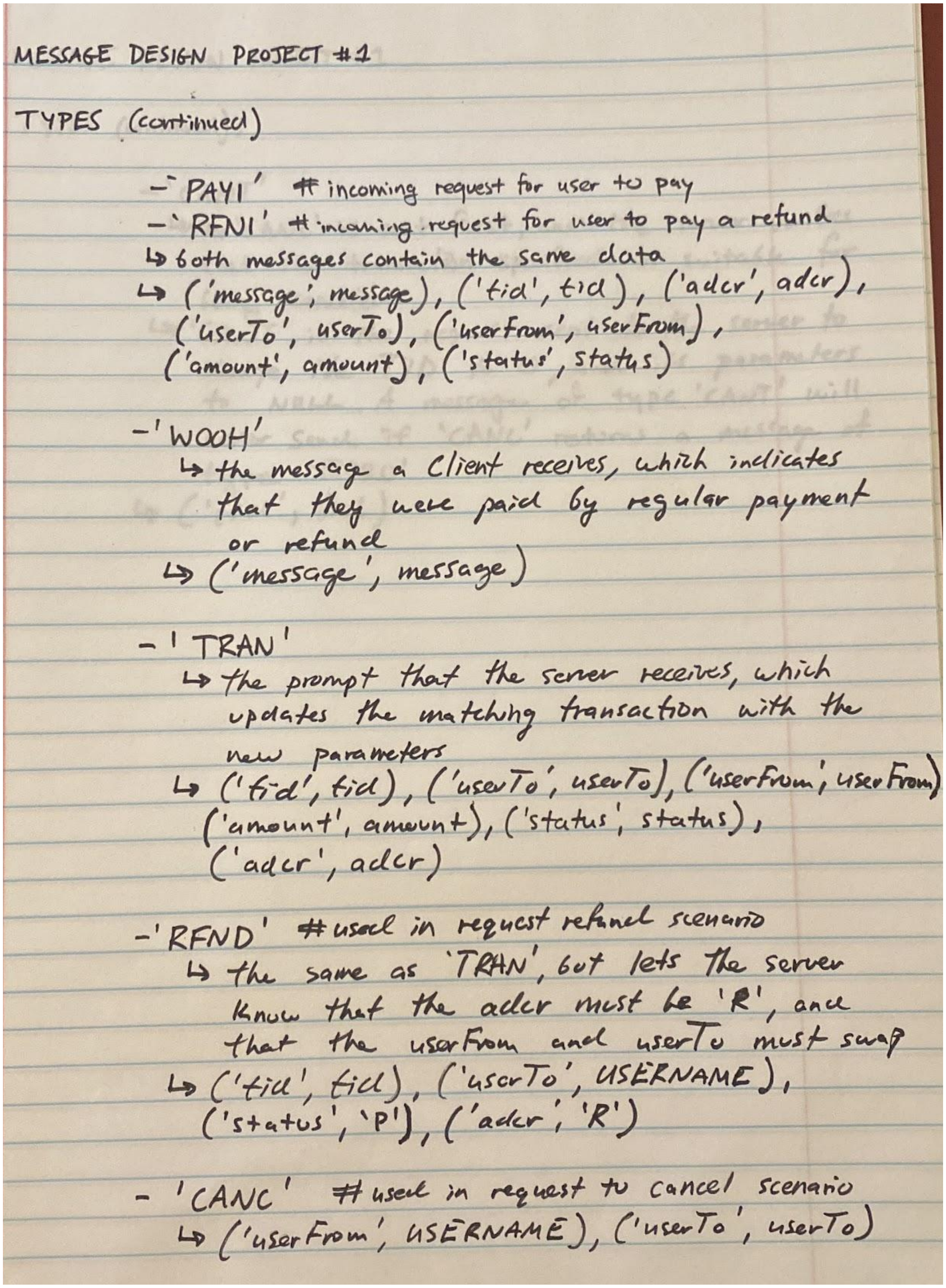
MESSAGE DESIGN PROJECT #1

4 bytes	4 bytes	(size = # of bytes)
SIZE	TYPE	DATA

SIZE: the # of bytes of the message, depends on the type because each type has different pieces of data associated with it

TYPES:

- 'LGIN'
  - ↳ ('username', u), ('password', p)
- 'LOUT'
  - ↳ no parameters for data
- 'GOOD' # an operation succeeded
  - ↳ ('message', message)
- 'ERRO' # an operation failed
  - ↳ ('message', message)
- 'REGI' # register an account
  - ↳ ('username', new_u), ('password', new_p), ('alias', new_a)
- 'PAYR' # request a payment
  - ↳ ('userTo', userTo), ('userFrom', userFrom), ('amount', amount), ('tid', transaction_id)
- 'CHKM' # user checks for new messages
  - ↳ ('user', USERNAME)
    - ↳ USERNAME is a global variable on the client side, indicating the current user's username
- 'GETM' # if there are new messages, get one
  - ↳ ('user', USERNAME)



MESSAGE DESIGN PROJECT #1

TYPES (continued)

- 'PAYI' # incoming request for user to pay
- 'RFNI' # incoming request for user to pay a refund

↳ both messages contain the same data

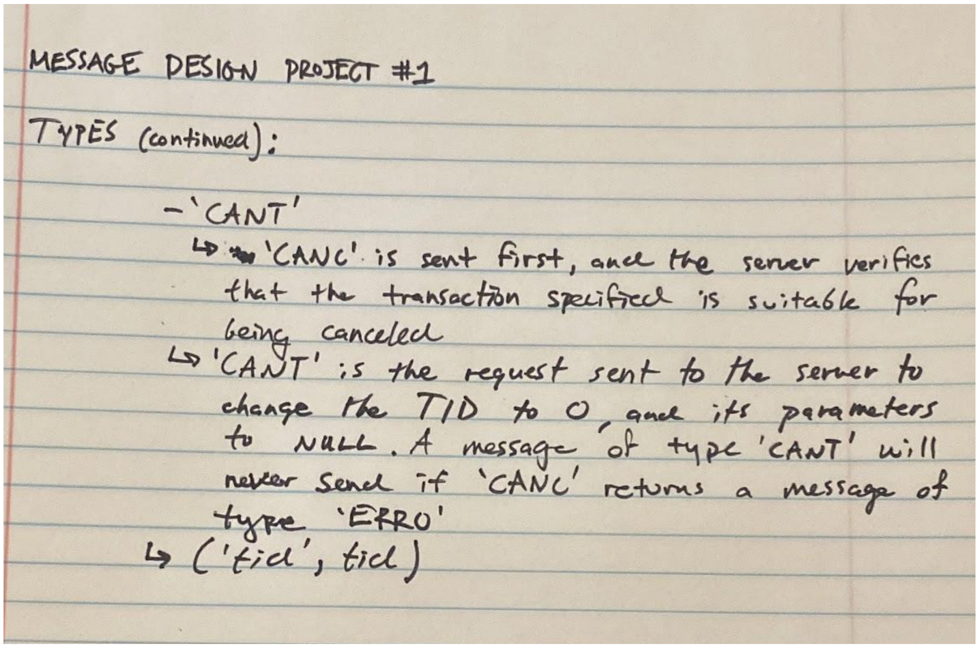
↳ ('message', message), ('tid', tid), ('adcr', adcr), ('userTo', userTo), ('userFrom', userFrom), ('amount', amount), ('status', status)

- 'WOOH'
  - ↳ the message a Client receives, which indicates that they were paid by regular payment or refund
  - ↳ ('message', message)

- 'TRAN'
  - ↳ the prompt that the server receives, which updates the matching transaction with the new parameters
  - ↳ ('tid', tid), ('userTo', userTo), ('userFrom', userFrom) ('amount', amount), ('status', status), ('adcr', adcr)

- 'RFND' # used in request refund scenario
  - ↳ the same as 'TRAN', but lets the server know that the adcr must be 'R', and that the userFrom and userTo must swap
  - ↳ ('tid', tid), ('userTo', USERNAME), ('status', 'P'), ('adcr', 'R')

- 'CANC' # used in request to cancel scenario
  - ↳ ('userFrom', USERNAME), ('userTo', userTo)



MESSAGE DESIGN PROJECT #1

TYPES (continued):

- 'CANT'

↳ 'CANC' is sent first, and the server verifies that the transaction specified is suitable for being canceled

↳ 'CANT' is the request sent to the server to change the TID to 0, and its parameters to NULL. A message of type 'CANT' will never send if 'CANC' returns a message of type 'ERRO'

↳ ('tid', tid)

<u>Client/Server Operations:</u>

File	Comment	Client	Server
cuser.py	User class		Y
transaction.py	Transaction class	Y	Y
cmessage.py	Cmessage class	Y	Y
cprotocol.py	Cprotocol class	Y	Y
cserverops.py	Server operations		Y
cclientops.py	Client Operations	Y	
tcp_server.py	Server		Y
tcp_client.py	Client	Y	
users.txt	Data for users		Y
transactions.txt	Data for transactions		Y
<USERNAME>DigitalWallet.txt	Digital wallet data	Y	

Client/Server Operations:

Client Operations:

tcp_client.py:

1. Runs the client operations file, detailed below

cclientops.py:

1. def __init__(self):
    a. Constructor
2. def _connect(self):
    a. Establishes TCP connection with server
3. def _doLogin(self):
    a. Sends a message of type ‘LGIN’, connection established if successful
4. def _doRegister(self):
    a. Sends a message of type ‘REGI’, creates a digital wallet locally if successfully added to the users text file by the server
5. def _doLogout(self):
    a. Sends a message of type ‘LOUT’, logs out the current user and ends connection
6. def _shutdown(self):
    a. Shuts down the connection either by logout or exit
7. def _doSignInWallet(self):
    a. Signs in a user to their wallet, verifying that the username and PIN match an existing wallet on their local device. Once logged in, wallet operations are available
8. def _doCheckBalance(self):
    a. Checks the balance of the digital wallet of the current user by reading the text file
9. def _doMakeDeposit(self):
    a. Takes the current balance of the wallet and adds the deposit amount, then rewrites the Digital Wallet text file using a loop to store the changes and write them
10. def _doCheckTransactions(self):
    a. Since transactions are appended to the end of the list, every line after the 4th line of a digital wallet contains a set of completed transactions. This function reads these lines, creates transaction objects out of them, and prints them
11. def _doPayRequest(self):
    a. Sends a message of type ‘PAYR’, generates a transaction ID for this new transaction. An error is received if the recipient user does not exist in users.txt
12. def _doCheckM(self):

   a. First sends a message of type ‘CHKM’, and the number of new messages are displayed for the user, and if the first message found was a message informing the client that they were paid by a pay request or refund. If there are no new messages, the client is informed of this and returned to the Main Menu. Otherwise, the next step proceeds:
   b. Then, a message of type ‘GETM’ is sent, and the transaction that triggered a message for that client is returned. The client makes changes to this transaction depending on the scenario (i.e. an incoming pay request, refund)
   c. Finally, a message of type ‘TRAN’ is sent, and the updated transaction’s parameters are sent to the server, and the transaction is updated in transactions.txt and in the dictionary of that file which the server maintains to run operations.

13. def _doRefund(self):
    a. Sends a message of type ‘RFND’, only if the refund is a completed transaction found in the client’s Digital wallet. The server interprets this message type by updating the matching transaction in the transactions.txt file.
14. def _doCancel(self):
    a. Sends a message of type ‘CANC’. If the parameters match those of a pending transaction that has not been accepted yet, then the function continues to the next step:
    b. Sends a message of type ‘CANT’. This message type indicates to the server to change the matching transaction’s tid to 0, and the rest of the parameters to null.
15. def _openWalletOperations(self):
    a. The menu for wallet operations, only after successful wallet login.
16. def _openWalletMenu(self):
    a. The menu for signing into a digital wallet. Requires a correct PIN number and username to perform wallet operations available in function 15
17. def _doTopLevelMenu(self):
    a. The client side menu, which allows the user to login to an existing account, register a new account, or exit the application. Successful login establishes connection and gives the Main Menu.
18. def _doMainMenu(self):
    a. Once connected, this menu is displayed, and allows the user to open their inbox, send payment, refund, and cancel requests, open their digital wallet, go back to the Top Level Menu, or exit the application
19. def run(self):
    a. Runs the cclientops.py file, gives the Top Level Menu for users that are not logged in, and the Main Menu for currently logged in users, by default. This default can be changed if the user goes back while in one of these menus.

Server Operations:

tcp_server.py:

1. Continuously waits for new connections. Once it accepts an incoming connection, it runs cserverops.py, the server operations, detailed below:

cserverops.py:

1. def run(self):
    a. The connection is already established, and the server loads 2 dictionaries of users and transactions from the text files “users.txt” and “transactions.txt”. This ensures that the transactions and users in the dictionaries are the same as what is contained in the text files, so that when operations are performed they can be successful. These dictionaries are also updated each time the server gets a message from the client. Each time the server gets a message, it sends the message to self._process(), where the server performs operations depending on the message type received. When a connection terminates, the server operations shut down.
    b. Routing (the message type received and the function invoked to process it):
        i. 'LGIN': self._doLogin,
        ii. 'LOUT': self._doLogout,
        iii. 'SRCH': self._doSearch,
        iv. 'REGI': self._doRegister,
        v. 'PAYR': self._doPayRequest,
        vi. 'CHKM': self._doCheckM,
        vii. 'GETM': self._doGetM,
        viii. 'TRAN': self._doUpdateTransactions,
        ix. 'RFND': self._doUpdateTransactions,
        x. 'CANC': self._doCancel,
        xi. 'CANT': self._doUpdateTransactions

2. def __init__(self):
    a. Constructor for the user and transaction dictionaries. This function also routes each message type to its appropriate function. The entire message is sent to each function.
3. def load(self, uname: str, cname: str):
    a. Opens and reads the transaction and user text files, and creates the two dictionaries using the lines of the text file. These two dictionaries are essential in many functions throughout the server operations.
4. def _doLogin(self, req: Cmessage) -> Cmessage:
    a. Verifies that the user and password match in the users dictionary, continue connection if successful. Disconnect the client if not successful.

5. def _doRegister(self, req: Cmessage) -> Cmessage:
    a. Verifies that the new user request does not match a username that is in the users dictionary. If it does not, then the new user object is written to the users.txt file.
6. def _doLogout(self, req: Cmessage) -> Cmessage:
    a. Terminates the connection
7. def _doCancel(self, req: Cmessage) -> Cmessage:
    a. There must be an existing transaction that was completed between the user requesting the cancellation and the recipient that has not been accepted or denied by the recipient yet. If this is the case, a message of type 'GOOD' is sent back to the client, indicating that the client can send a message to update this transaction.
8. def _doPayRequest(self, req: Cmessage) -> Cmessage:
    a. Verifies that the recipient user of this payment request exists in the users dictionary, and if they do, send a message of type 'GOOD' to the client indicating that the payment request's transaction object can be written to transactions.txt.
9. def _doCheckM(self, req: Cmessage) -> Cmessage:
    a. Counts the number of new messages for the user requesting to check their inbox. If there are no new messages, a message of type 'ERRO' is sent to the client, with a message saying there are no new messages. If there are new messages, the server tells the client how many there are in the message with a message type of 'GOOD'.
10. def _doGetM(self, req: Cmessage) -> Cmessage:
    a. Only happens when def _doCheckM() sends a 'GOOD' message to the client (or that there are new messages). This function checks to see what type of new message it is, and sends the incoming request message type back to the client, with the parameters being the transaction. Additionally, this function adds money to a client's wallet if their message is a notification that another client payed them.
11. def _process(self, req: Cmessage) -> Cmessage:
    a. Gets the incoming message from the client, and inputs the type as the parameter for the route. Each expected message type has an associated function in the server operations to handle it, and this function sends the incoming message to that corresponding function.
12. def shutdown(self):
    a. Shuts down the server operations and the connection

Class Operations:

transaction.py:

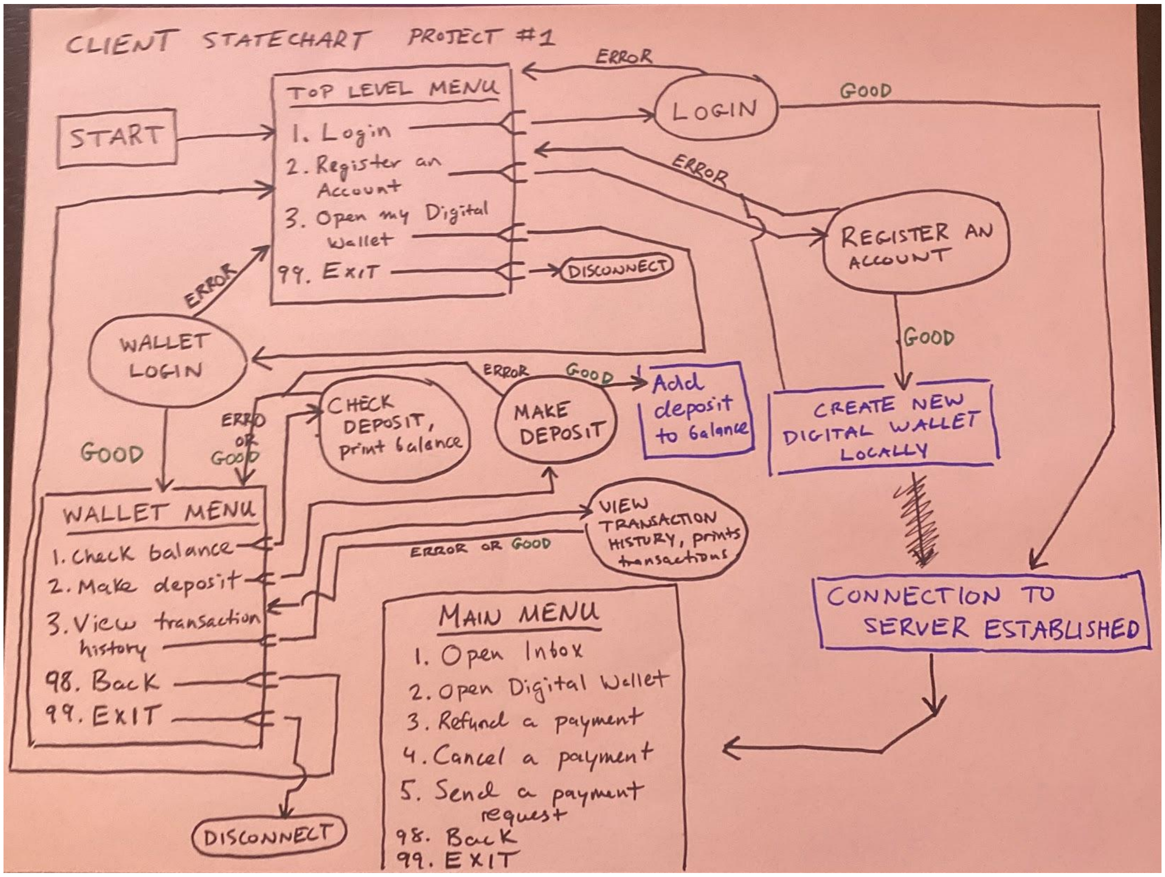
1. def __str__(self):
    a. Returns a newline-seperated string of each parameter of a transaction. Used to print the transactions in transactions.txt.
2. def __stri__(self):
    a. Returns a newline-seperated string of each parameter of a transaction, with a label for that parameter (i.e. Transaction ID: #####\n). Used to print the history of transactions in a Digital Wallet in a user friendly format.

cuser.py:

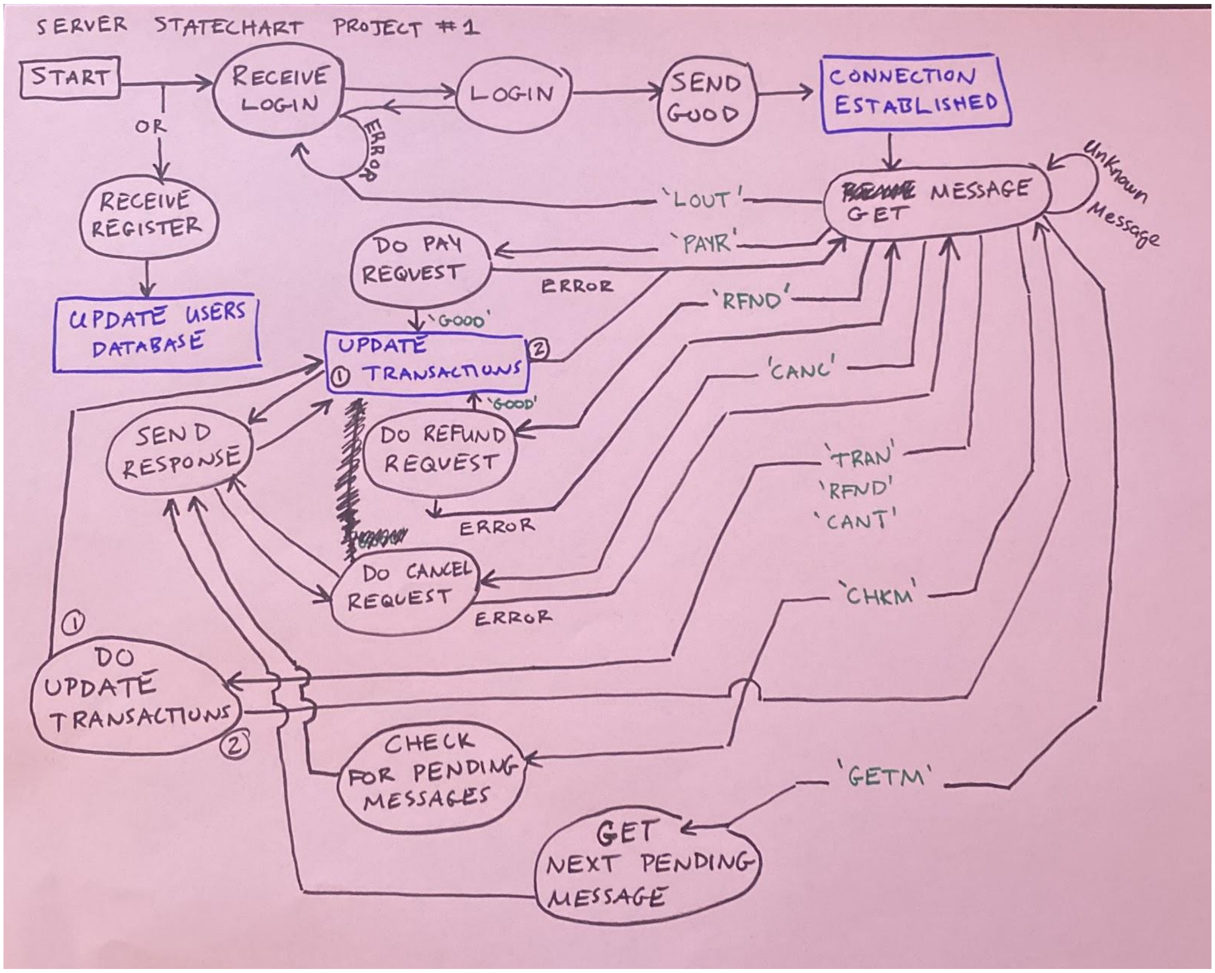
1. def login(self, u: str, p: str) -> bool:
    a. Determines if an attempted login matches the username and password in the users dictionary

<u>Client Statechart</u>

Note: the logic and message types detailed above explain how each state changes. The determination of the state change is not copied to the statechart.

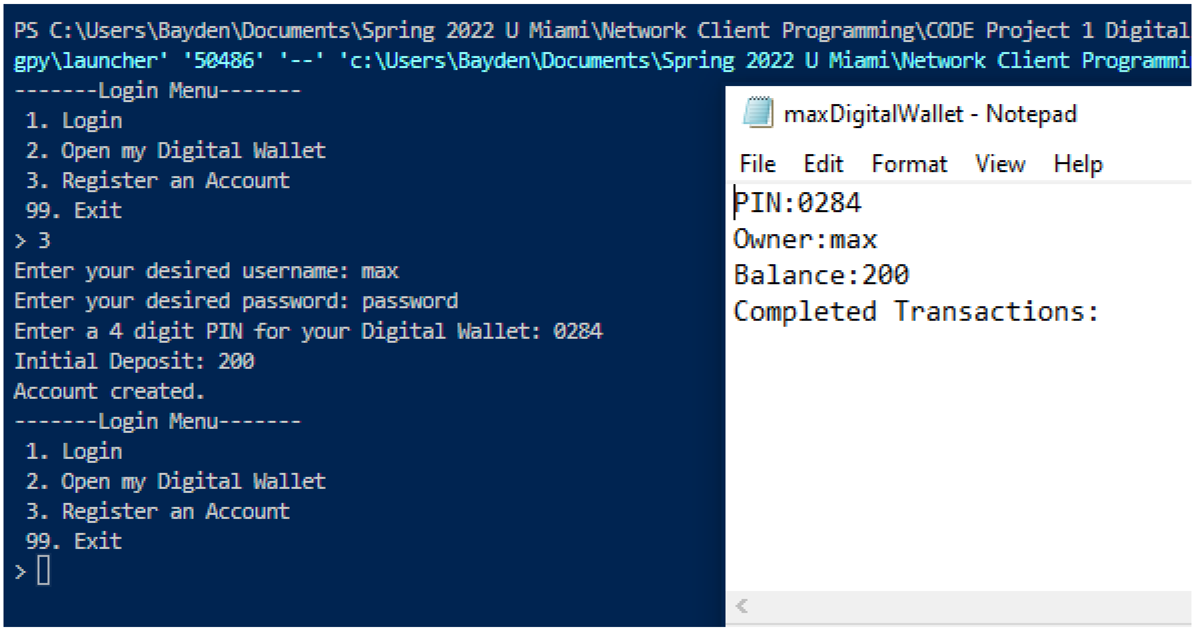


Server Statechart



Verification (Sample Scenarios and Screenshots):

Scenario 1: Register a new account

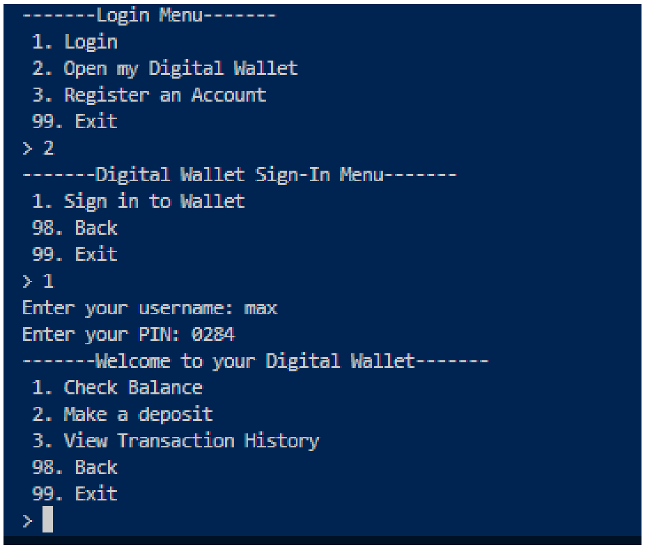


```
PS C:\Users\Bayden\Documents\Spring 2022 U Miami\Network Client Programming\CODE Project 1 Digital
gpy\launcher' '50486' '--' 'c:\Users\Bayden\Documents\Spring 2022 U Miami\Network Client Programmi
-------Login Menu-------
 1. Login
 2. Open my Digital Wallet
 3. Register an Account
 99. Exit
> 3
Enter your desired username: max
Enter your desired password: password
Enter a 4 digit PIN for your Digital Wallet: 0284
Initial Deposit: 200
Account created.
-------Login Menu-------
 1. Login
 2. Open my Digital Wallet
 3. Register an Account
 99. Exit
>
```

maxDigitalWallet - Notepad

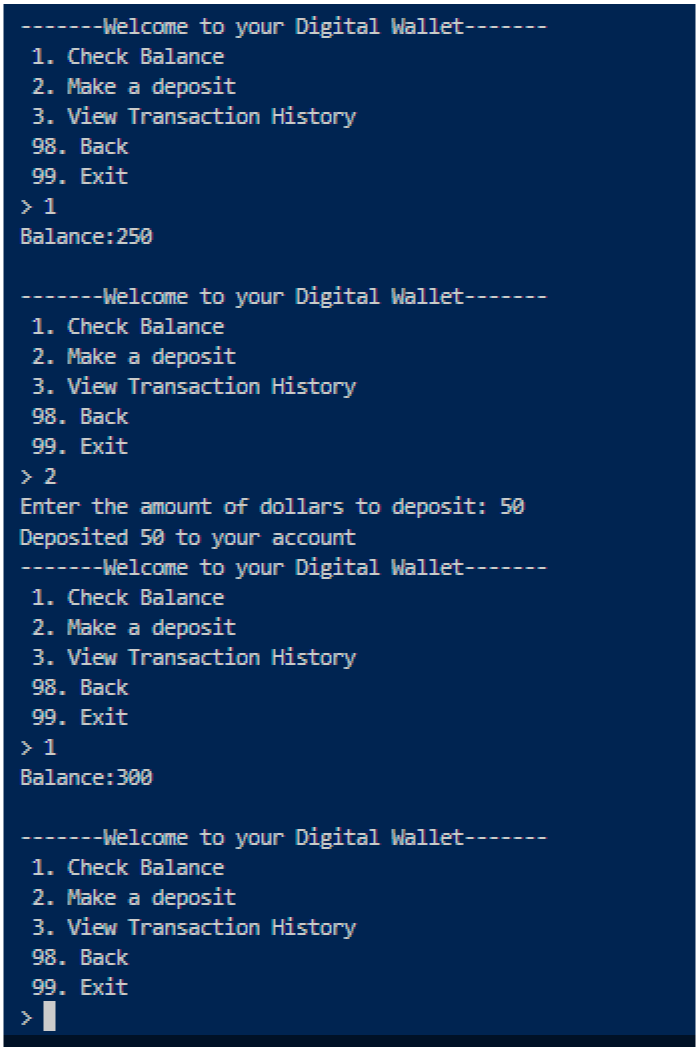
```
PIN:0284
Owner:max
Balance:200
Completed Transactions:
```

Scenario 2: Digital Wallet Login



```
-------Login Menu-------
 1. Login
 2. Open my Digital Wallet
 3. Register an Account
 99. Exit
> 2
-------Digital Wallet Sign-In Menu-------
 1. Sign in to Wallet
 98. Back
 99. Exit
> 1
Enter your username: max
Enter your PIN: 0284
-------Welcome to your Digital Wallet-------
 1. Check Balance
 2. Make a deposit
 3. View Transaction History
 98. Back
 99. Exit
>
```

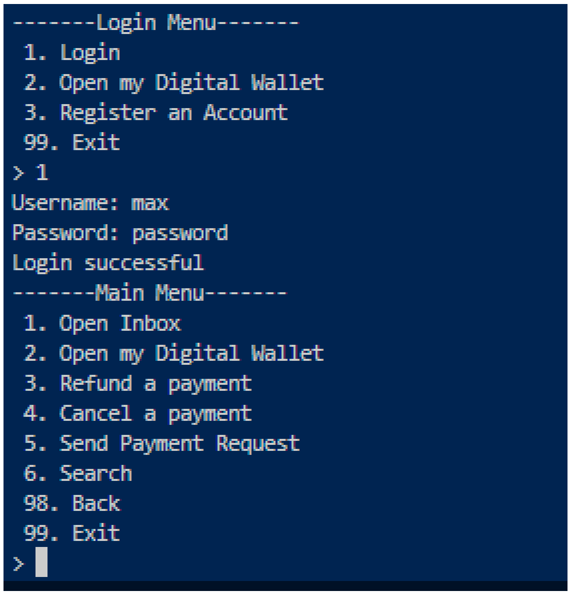
Scenario 3: Deposit money
Scenario 5: Check Balance



```
-------Welcome to your Digital Wallet-------
 1. Check Balance
 2. Make a deposit
 3. View Transaction History
 98. Back
 99. Exit
> 1
Balance:250

-------Welcome to your Digital Wallet-------
 1. Check Balance
 2. Make a deposit
 3. View Transaction History
 98. Back
 99. Exit
> 2
Enter the amount of dollars to deposit: 50
Deposited 50 to your account
-------Welcome to your Digital Wallet-------
 1. Check Balance
 2. Make a deposit
 3. View Transaction History
 98. Back
 99. Exit
> 1
Balance:300

-------Welcome to your Digital Wallet-------
 1. Check Balance
 2. Make a deposit
 3. View Transaction History
 98. Back
 99. Exit
> 
```

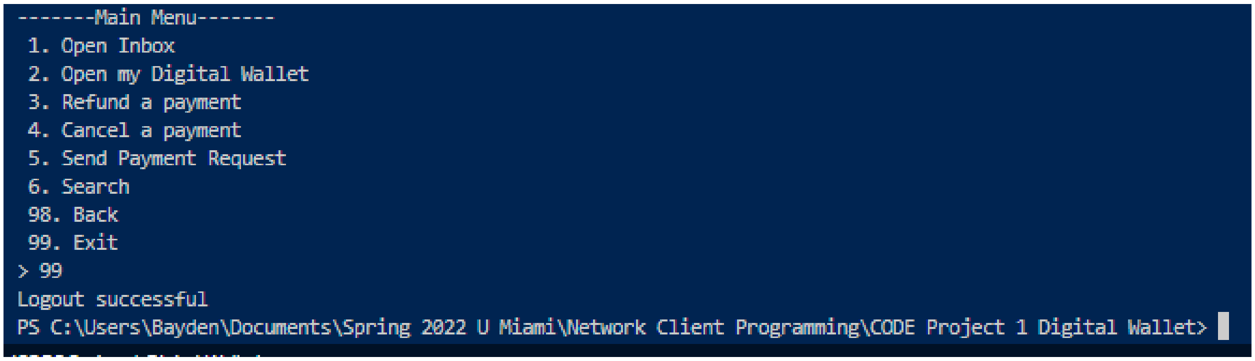
Scenario 4: View Transaction History

Scenario 6: Login



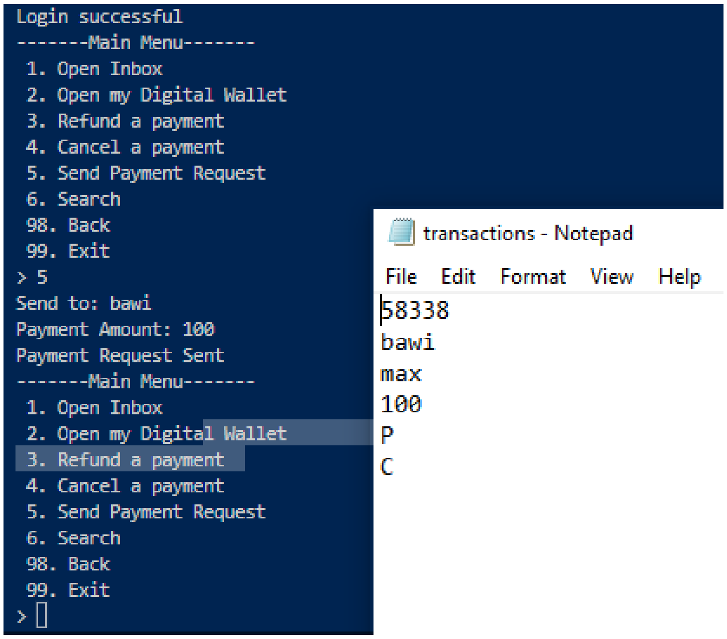
```
-------Login Menu-------
 1. Login
 2. Open my Digital Wallet
 3. Register an Account
 99. Exit
> 1
Username: max
Password: password
Login successful
-------Main Menu-------
 1. Open Inbox
 2. Open my Digital Wallet
 3. Refund a payment
 4. Cancel a payment
 5. Send Payment Request
 6. Search
 98. Back
 99. Exit
>
```

Scenario 7: Logout / Exit



```
-------Main Menu-------
 1. Open Inbox
 2. Open my Digital Wallet
 3. Refund a payment
 4. Cancel a payment
 5. Send Payment Request
 6. Search
 98. Back
 99. Exit
> 99
Logout successful
PS C:\Users\Bayden\Documents\Spring 2022 U Miami\Network Client Programming\CODE Project 1 Digital Wallet>
```

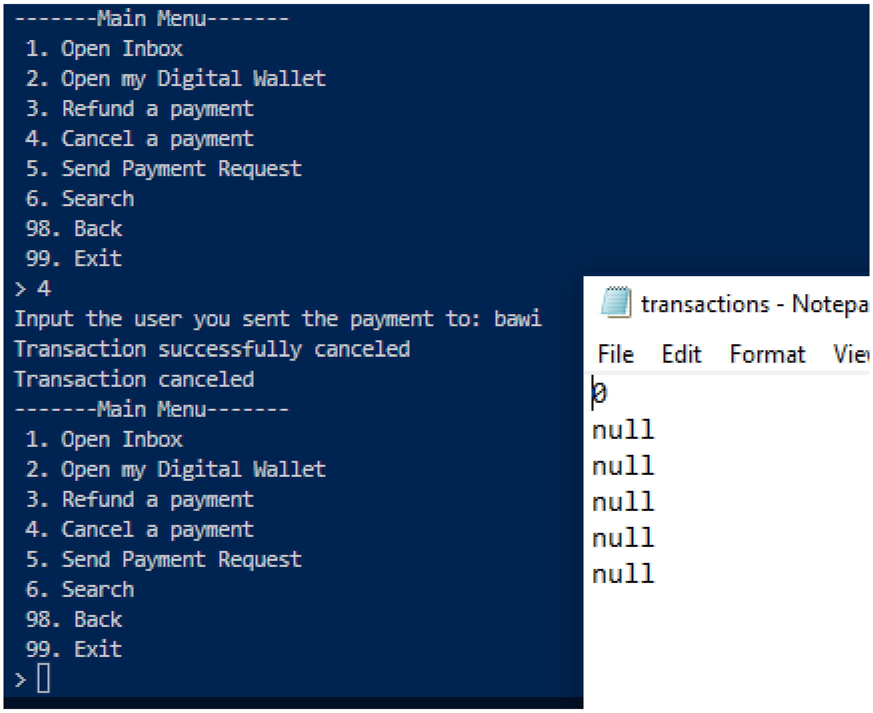
Scenario 9: Send Payment Request



```
Login successful
-------Main Menu-------
 1. Open Inbox
 2. Open my Digital Wallet
 3. Refund a payment
 4. Cancel a payment
 5. Send Payment Request
 6. Search
 98. Back
 99. Exit
> 5
Send to: bawi
Payment Amount: 100
Payment Request Sent
-------Main Menu-------
 1. Open Inbox
 2. Open my Digital Wallet
 3. Refund a payment
 4. Cancel a payment
 5. Send Payment Request
 6. Search
 98. Back
 99. Exit
>
```

```
58338
bawi
max
100
P
C
```

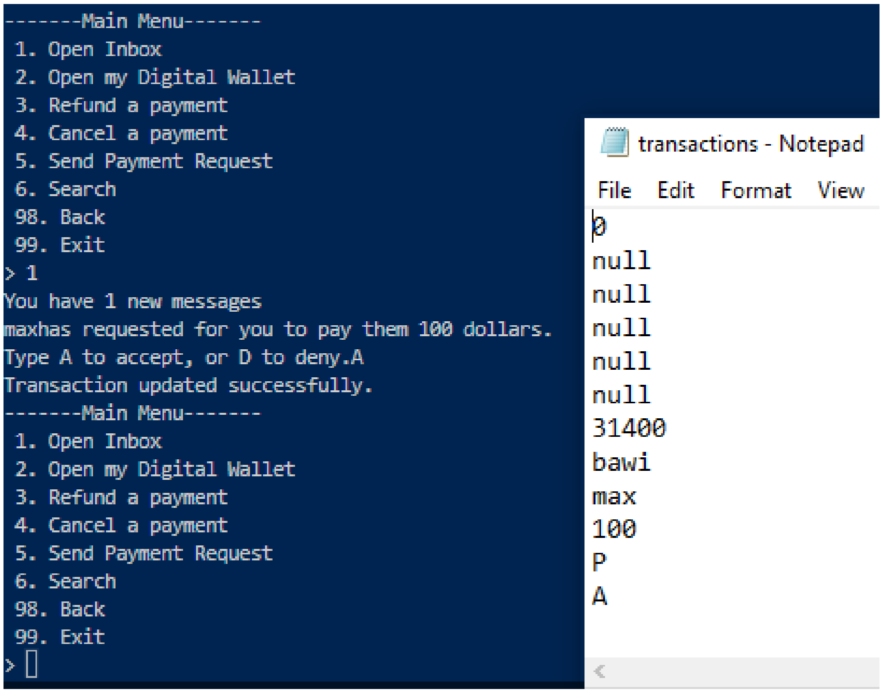
Scenario 11: Cancel an Outgoing Payment Request



```
-------Main Menu-------
 1. Open Inbox
 2. Open my Digital Wallet
 3. Refund a payment
 4. Cancel a payment
 5. Send Payment Request
 6. Search
 98. Back
 99. Exit
> 4
Input the user you sent the payment to: bawi
Transaction successfully canceled
Transaction canceled
-------Main Menu-------
 1. Open Inbox
 2. Open my Digital Wallet
 3. Refund a payment
 4. Cancel a payment
 5. Send Payment Request
 6. Search
 98. Back
 99. Exit
>
```

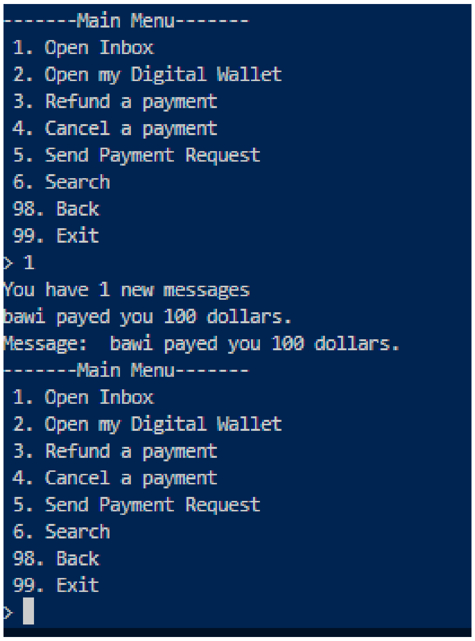
```
0
null
null
null
null
null
```

Scenario 8: Check Incoming Messages



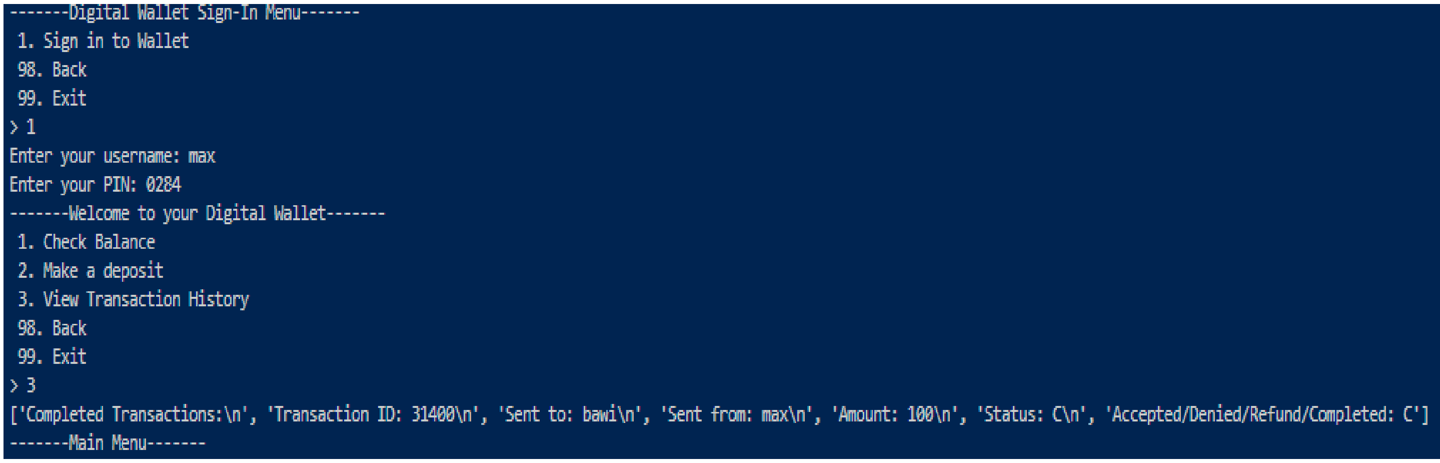
```
-------Main Menu-------
 1. Open Inbox
 2. Open my Digital Wallet
 3. Refund a payment
 4. Cancel a payment
 5. Send Payment Request
 6. Search
 98. Back
 99. Exit
> 1
You have 1 new messages
maxhas requested for you to pay them 100 dollars.
Type A to accept, or D to deny.A
Transaction updated successfully.
-------Main Menu-------
 1. Open Inbox
 2. Open my Digital Wallet
 3. Refund a payment
 4. Cancel a payment
 5. Send Payment Request
 6. Search
 98. Back
 99. Exit
>
```

```
0
null
null
null
null
null
31400
bawi
max
100
P
A
```



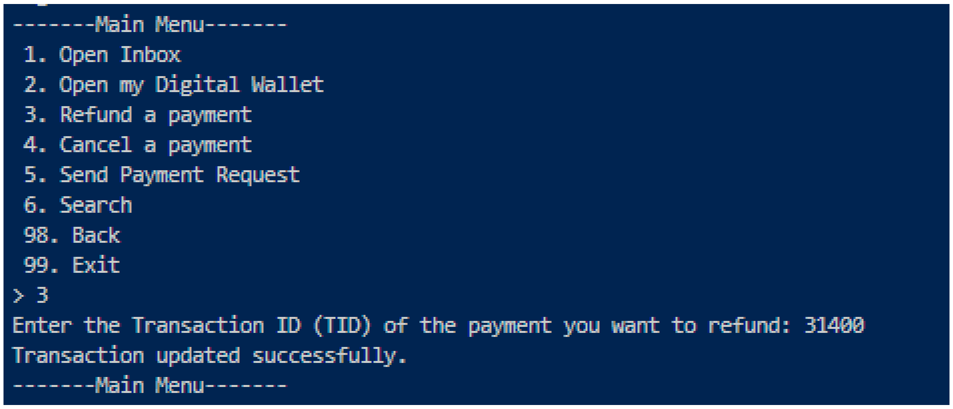
```
-------Main Menu-------
 1. Open Inbox
 2. Open my Digital Wallet
 3. Refund a payment
 4. Cancel a payment
 5. Send Payment Request
 6. Search
 98. Back
 99. Exit
> 1
You have 1 new messages
bawi payed you 100 dollars.
Message:  bawi payed you 100 dollars.
-------Main Menu-------
 1. Open Inbox
 2. Open my Digital Wallet
 3. Refund a payment
 4. Cancel a payment
 5. Send Payment Request
 6. Search
 98. Back
 99. Exit
>
```

Scenario 4: View Transaction History

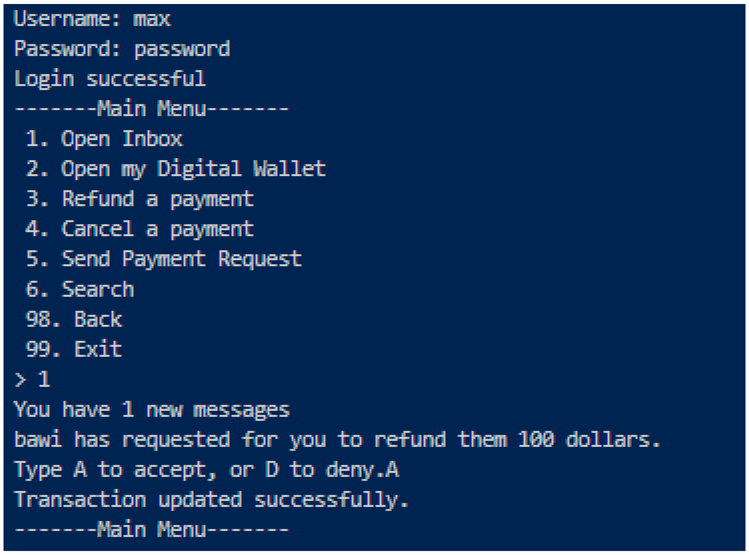


```
-------Digital Wallet Sign-In Menu-------
 1. Sign in to Wallet
 98. Back
 99. Exit
> 1
Enter your username: max
Enter your PIN: 0284
-------Welcome to your Digital Wallet-------
 1. Check Balance
 2. Make a deposit
 3. View Transaction History
 98. Back
 99. Exit
> 3
['Completed Transactions:\n', 'Transaction ID: 31400\n', 'Sent to: bawi\n', 'Sent from: max\n', 'Amount: 100\n', 'Status: C\n', 'Accepted/Denied/Refund/Completed: C']
-------Main Menu-------
```

Scenario 10: Send Refund Request



```
-------Main Menu-------
 1. Open Inbox
 2. Open my Digital Wallet
 3. Refund a payment
 4. Cancel a payment
 5. Send Payment Request
 6. Search
 98. Back
 99. Exit
> 3
Enter the Transaction ID (TID) of the payment you want to refund: 31400
Transaction updated successfully.
-------Main Menu-------
```



```
Username: max
Password: password
Login successful
-------Main Menu-------
 1. Open Inbox
 2. Open my Digital Wallet
 3. Refund a payment
 4. Cancel a payment
 5. Send Payment Request
 6. Search
 98. Back
 99. Exit
> 1
You have 1 new messages
bawi has requested for you to refund them 100 dollars.
Type A to accept, or D to deny.A
Transaction updated successfully.
-------Main Menu-------
```

Conclusion:

In this project I implemented a client/server Python program which performs a Digital Wallet service. Clients can register, send payments to one another, refund or cancel those payments, view and update their wallet, and check their messages. I used text files instead of noSQL because I have not had a lot of experience in databases yet and I was under a deadline to complete this project. The service does not allow for multiple clients to be connected at the same time, so updates in the server happen one at a time, per active session. I got more practice implementing dictionaries, creating classes, and working with files in Python, and I learned how to create and work with TCP messaging across a network. Additionally, I gained practice designing a complicated program on paper almost entirely before writing any code, which is what ultimately was responsible for completing the program by the required date, because debugging programs which communicate with one another through messages over the internet can get very complicated without a good plan.

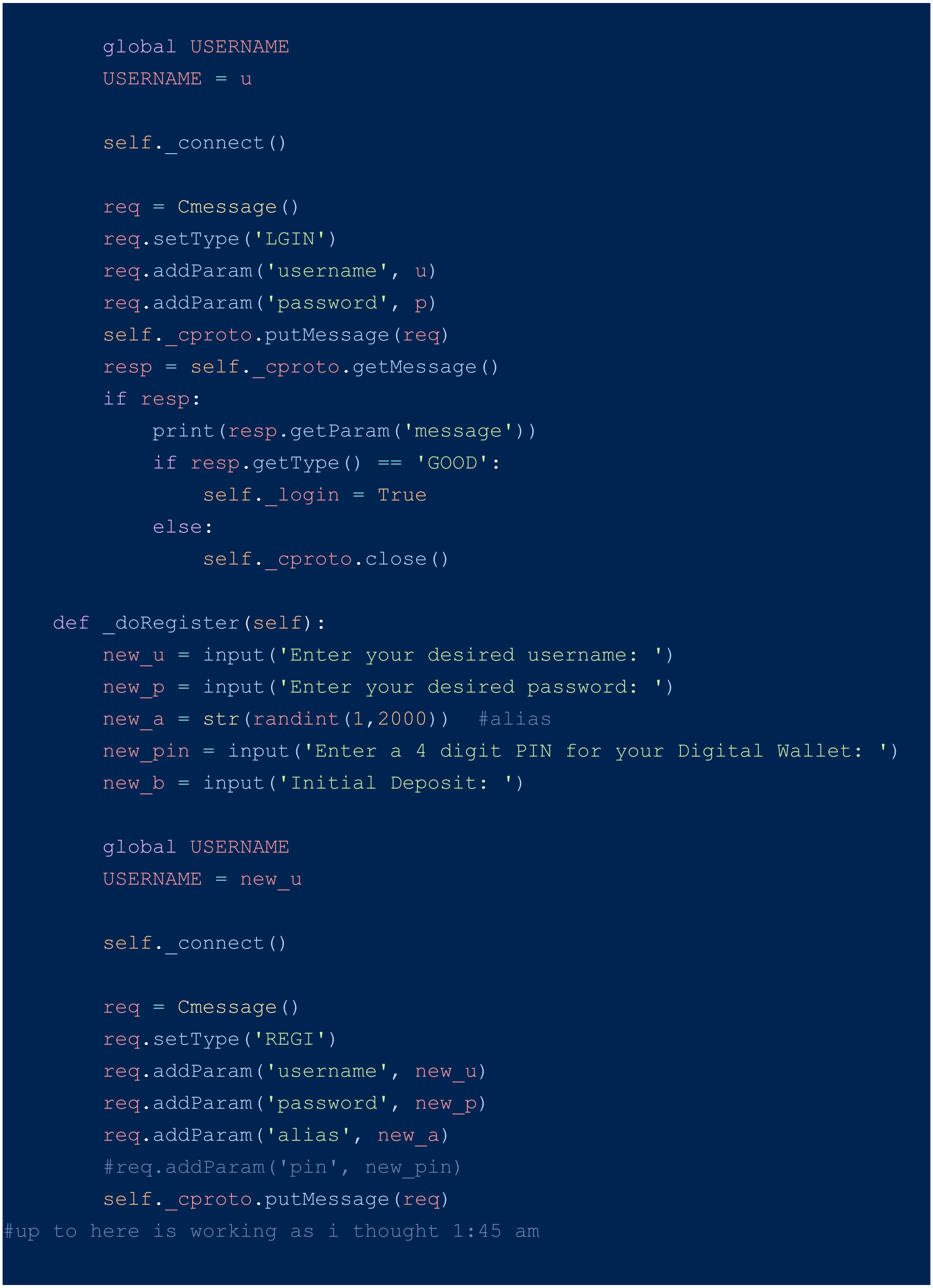
Appendix:
(code, do this in Microsoft Word to preserve formatting from Visual Studio)

tcp_client.py

```
'''
Created on Feb 19, 2021

@author: bayden
'''

from cclientops import Cclientops
if __name__ == '__main__':
    cops = Cclientops()
    cops.run()
```

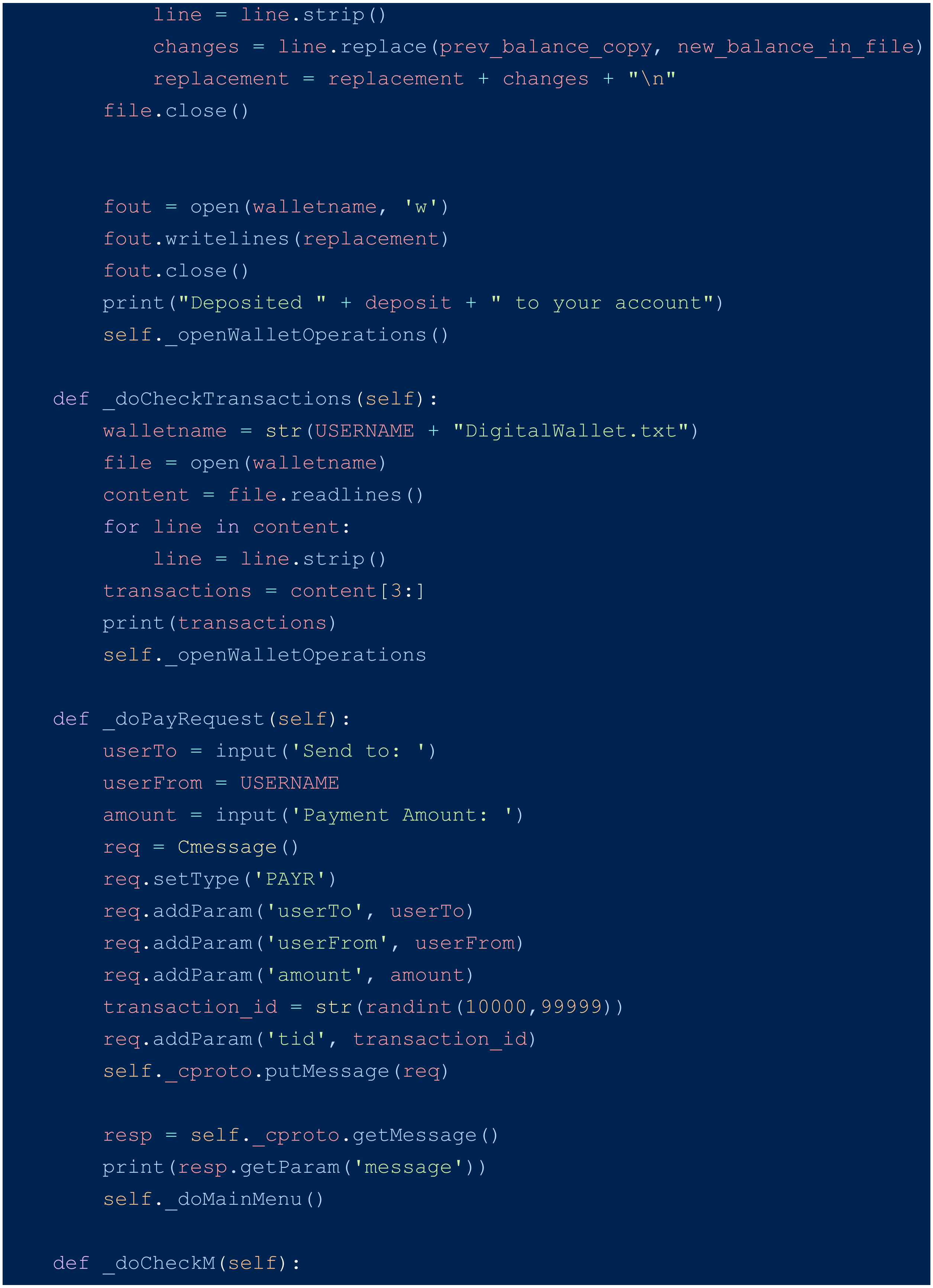
cclientops.py

```
'''
Created on Mar 3, 2022

@author: bayden
'''
from email import message
import socket
from cprotocol import Cprotocol
from cmessage import Cmessage
from random import randint
from transaction import Transaction
import sys

class Cclientops(object):
    '''
    classdocs
    '''

    def __init__(self):
        '''
        Constructor
        '''
        self._cproto = Cprotocol()
        self._login = False
        self._done = False
        self._debug = True

    def _debugPrint(self, m: str):
        if self._debug:
            print(m)

    def _connect(self):
        commsoc = socket.socket()
        commsoc.connect(("localhost",50001))
        self._cproto = Cprotocol(commsoc)

    def _doLogin(self):
        u = input('Username: ')
        p = input('Password: ')
```

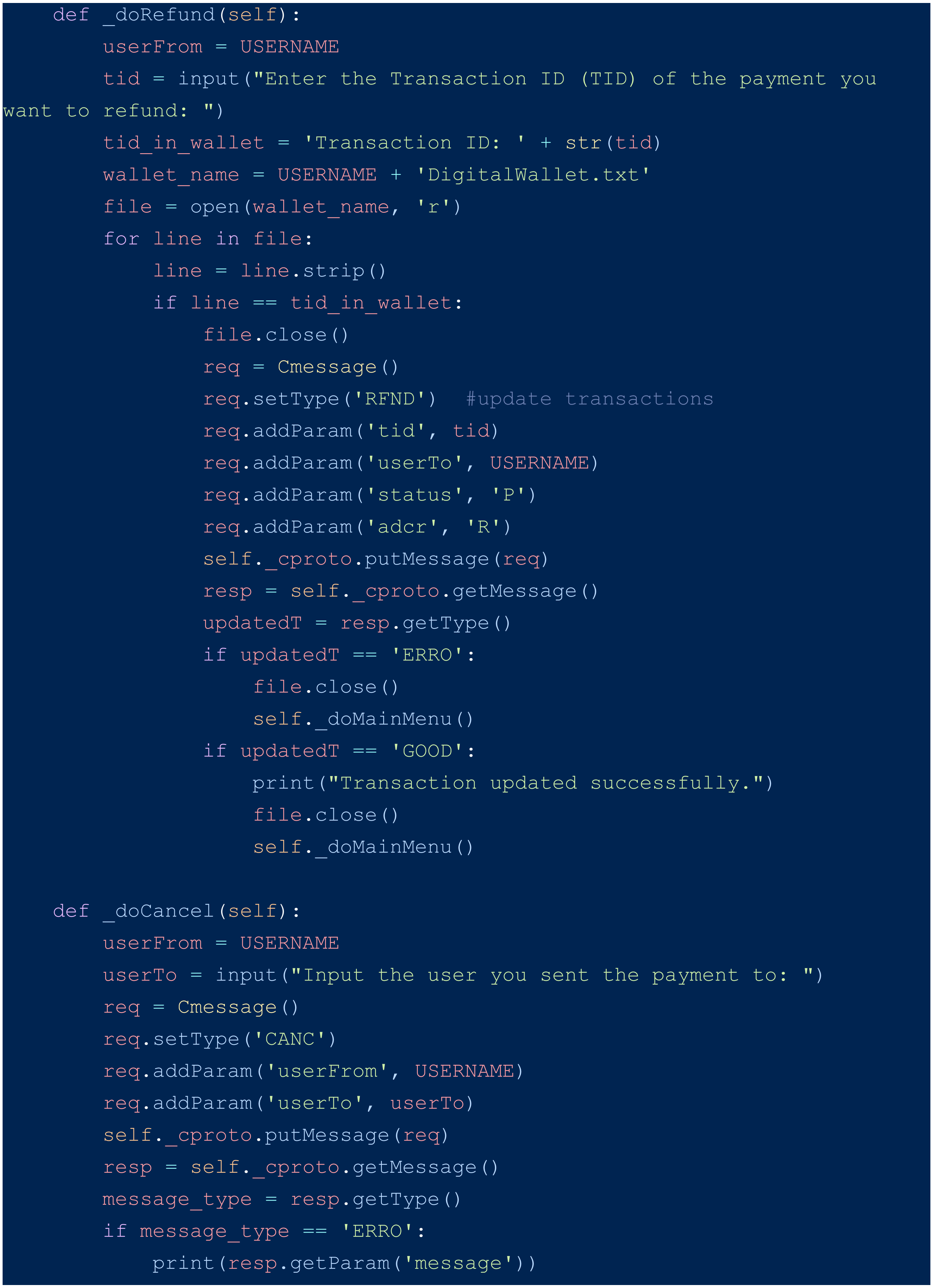


```
        global USERNAME
        USERNAME = u

        self._connect()

        req = Cmessage()
        req.setType('LGIN')
        req.addParam('username', u)
        req.addParam('password', p)
        self._cproto.putMessage(req)
        resp = self._cproto.getMessage()
        if resp:
            print(resp.getParam('message'))
            if resp.getType() == 'GOOD':
                self._login = True
            else:
                self._cproto.close()

    def _doRegister(self):
        new_u = input('Enter your desired username: ')
        new_p = input('Enter your desired password: ')
        new_a = str(randint(1,2000))  #alias
        new_pin = input('Enter a 4 digit PIN for your Digital Wallet: ')
        new_b = input('Initial Deposit: ')

        global USERNAME
        USERNAME = new_u

        self._connect()

        req = Cmessage()
        req.setType('REGI')
        req.addParam('username', new_u)
        req.addParam('password', new_p)
        req.addParam('alias', new_a)
        #req.addParam('pin', new_pin)
        self._cproto.putMessage(req)
#up to here is working as i thought 1:45 am
```

```
        resp = self._cproto.getMessage()
        if resp:
            print(resp.getParam('message'))
            if resp.getType() == 'GOOD':
                #create the new user's digital wallet:
                filename = str((new_u) + 'DigitalWallet.txt')
                with open(filename, 'a') as f:
                    pin_to_write = str('PIN:' + new_pin + '\n')
                    f.write(pin_to_write)
                    owner_to_write = str("Owner:" + new_u + '\n')
                    f.write(owner_to_write)
                    balance_to_write = str("Balance:" + new_b + '\n')
                    f.write(balance_to_write)
                    f.write('Completed Transactions:' + '\n')
                    f.close()

                self._cproto.close()
            else:
                self._cproto.close()

    def _doLogout(self):
        req = Cmessage()
        req.setType('LOUT')
        self._cproto.putMessage(req)
        resp = self._cproto.getMessage()
        if resp:
            print(resp.getParam('message'))
        self._login = False

    def _doSearch(self):
        cid = input('Course id:')

        req = Cmessage()
        req.setType('SRCH')
        req.addParam('cid', cid)
        self._cproto.putMessage(req)
        resp = self._cproto.getMessage()
        if resp:
            if resp.getType() == 'DATA':
                print('Course id: {}\nName: {}\nCredits:
{}\n'.format(resp.getParam('cid'),

resp.getParam('name'),

resp.getParam('credits')))
            else:
                print(resp.getParam('message'))

    def _shutdown(self):
        if self._login:
            self._doLogout()
            self._cproto.close()
        self._login = False
        self._done = True

    def _doSignInWallet(self):
        username = input('Enter your username: ')
        pin = input('Enter your PIN: ')

        global USERNAME
        USERNAME = username

        filename = str(username + 'DigitalWallet.txt')
        try:
            with open(filename) as f:
                lines = f.readlines()
                pin_entered = str('PIN:' + pin)
                pin_in_file = lines[0]
                pin_in_file = pin_in_file.strip()
                pin_entered = pin_entered.strip()

                if pin_in_file == pin_entered:
                    self._openWalletOperations()
                    #lines = f.readlines()
                    #for line in lines:
                        #print(line)
                else:
                    self._openWalletMenu()
        except IOError as e:
            self._openWalletMenu()

    def _doCheckBalance(self):
        walletname = str(USERNAME + "DigitalWallet.txt")
        file = open(walletname)
        content = file.readlines()
        balance = content[2]
        print(balance)
        self._openWalletOperations()

    def _doMakeDeposit(self):
        deposit = input("Enter the amount of dollars to deposit: ")
        deposit_in_file = str('Balance:' + deposit)
        walletname = str(USERNAME + 'DigitalWallet.txt')
        file = open(walletname)
        content = file.readlines()
        file.close()
        prev_balance = content[2]
        #prev_balance_copy = prev_balance
        prev_balance = prev_balance.strip()
        prev_balance_copy = prev_balance

        # balance_add = 0
        # actual_balance_number = [int(i) for i in prev_balance.split() if
i.isdigit()]
        # for i in actual_balance_number:
        #     i = int(i)
        #     balance_add = i + balance_add

        #prev_balance = prev_balance.replace(content[2],'')
        #int(prev_balance)
        balance_a = prev_balance[8:]
        balance_add = int(balance_a)
        new_balance = int(int(deposit) + balance_add)
        new_balance = str(new_balance)
        new_balance_in_file = str('Balance:' + new_balance)

        file = open(walletname, "r")
        replacement = ""
        for line in file:
```



```
            line = line.strip()
            changes = line.replace(prev_balance_copy, new_balance_in_file)
            replacement = replacement + changes + "\n"
        file.close()


        fout = open(walletname, 'w')
        fout.writelines(replacement)
        fout.close()
        print("Deposited " + deposit + " to your account")
        self._openWalletOperations()

    def _doCheckTransactions(self):
        walletname = str(USERNAME + "DigitalWallet.txt")
        file = open(walletname)
        content = file.readlines()
        for line in content:
            line = line.strip()
        transactions = content[3:]
        print(transactions)
        self._openWalletOperations

    def _doPayRequest(self):
        userTo = input('Send to: ')
        userFrom = USERNAME
        amount = input('Payment Amount: ')
        req = Cmessage()
        req.setType('PAYR')
        req.addParam('userTo', userTo)
        req.addParam('userFrom', userFrom)
        req.addParam('amount', amount)
        transaction_id = str(randint(10000,99999))
        req.addParam('tid', transaction_id)
        self._cproto.putMessage(req)

        resp = self._cproto.getMessage()
        print(resp.getParam('message'))
        self._doMainMenu()

    def _doCheckM(self):
        #first see if there are any new messages
        req = Cmessage()
        req.setType('CHKM')
        req.addParam('user', USERNAME)
        self._cproto.putMessage(req)
        resp = self._cproto.getMessage()
        m_type = resp.getType()
        message = resp.getParam('message')
        print(message)
        #display if new messages
        #if no new messages, then go back to previous menu
        if m_type == 'ERRO':
            self._doMainMenu()
        else:
            req = Cmessage()
            req.setType('GETM')
            req.addParam('user', USERNAME)
            self._cproto.putMessage(req)

            resp = self._cproto.getMessage()
            message_type = resp.getType()
            message = resp.getParam('message')
            tid = resp.getParam('tid')
            userFrom = resp.getParam('userFrom')
            userTo = resp.getParam('userTo')
            adcr = resp.getParam('adcr')
            status = resp.getParam('status')
            amount = resp.getParam('amount')

            t = Transaction(tid, userTo, userFrom, amount, status, adcr)
            t_copy = Transaction(tid, userTo, userFrom, amount, status,
adcr)

            if message_type == 'WOOH': #GPAY get payment, someone payed
you
                print(message)
                t.adcr = 'C'
                t.status = 'C'
            else: #PAYI means Pay in
            # if you are receiving a payment request
                print(message)
                user_input = input("Type A to accept, or D to deny.")
                if user_input == 'A':
                    #subtract amount from my wallet
                    #change ADCR of transaction

                    if message_type == 'PAYI':
                        t.adcr = 'A'
                        t.status = 'P'

                    if message_type == 'RFNI':
                        t.adcr = 'Z'
                        t.status = 'C'

                #subtracting from wallet
                    amount = int(t.amount) * -1
                    amount = str(amount)
                    walletname = str(USERNAME + 'DigitalWallet.txt')
                    file = open(walletname)
                    content = file.readlines()
                    file.close()
                    prev_balance = content[2]
                    prev_balance = prev_balance.strip()
                    prev_balance_copy = prev_balance
                    balance_a = prev_balance[8:]
                    balance_add = int(balance_a)
                    new_balance = int(int(amount) + balance_add)
                    new_balance = str(new_balance)
                    new_balance_in_file = str('Balance:' + new_balance)

                    file = open(walletname, "r")
                    replacement = ""
                    for line in file:
                        line = line.strip()
                        changes = line.replace(prev_balance_copy,
new_balance_in_file)
                        replacement = replacement + changes + "\n"
                    file.close()
                    fout = open(walletname, 'w')
                    fout.writelines(replacement)
                    fout.close()

                    file = open(walletname, "a")
                    transaction_add = ""
                    new_n_t = Transaction(t.tid, t.userTo, t.userFrom,
t.amount, 'C', 'C')
                    transaction_add = str(new_n_t.__stri__())
                    file.write(transaction_add)
                    file.close()

                if user_input == 'D':
                    #change the ADCR of the transaction in transactions
                    t.adcr = 'D'
                    t.status = 'C'


                req = Cmessage()
                req.setType('TRAN')  #update transactions
                req.addParam('tid', t.tid)
                req.addParam('userTo', t.userTo)
                req.addParam('userFrom', t.userFrom)
                req.addParam('amount', t.amount)
                req.addParam('status', t.status)
                req.addParam('adcr', t.adcr)
                self._cproto.putMessage(req)
                resp = self._cproto.getMessage()
                updatedT = resp.getType()
                if updatedT == 'ERRO':
                    self._doMainMenu()
                if updatedT == 'GOOD':
                    print("Transaction updated successfully.")
                self._doMainMenu()

        print("Message: ", message)
        self._doMainMenu()
```



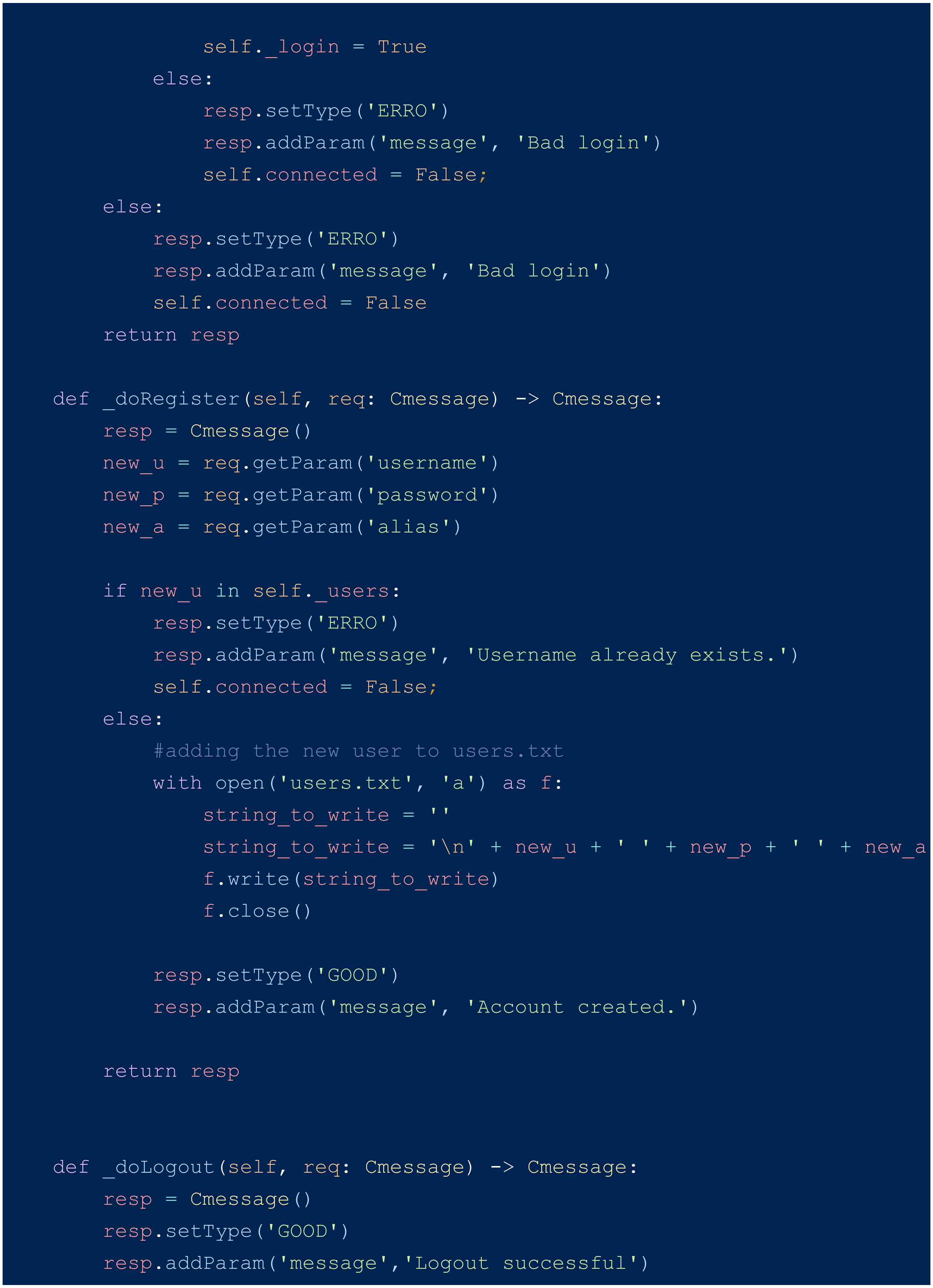
```
    def _doRefund(self):
        userFrom = USERNAME
        tid = input("Enter the Transaction ID (TID) of the payment you
want to refund: ")
        tid_in_wallet = 'Transaction ID: ' + str(tid)
        wallet_name = USERNAME + 'DigitalWallet.txt'
        file = open(wallet_name, 'r')
        for line in file:
            line = line.strip()
            if line == tid_in_wallet:
                file.close()
                req = Cmessage()
                req.setType('RFND')  #update transactions
                req.addParam('tid', tid)
                req.addParam('userTo', USERNAME)
                req.addParam('status', 'P')
                req.addParam('adcr', 'R')
                self._cproto.putMessage(req)
                resp = self._cproto.getMessage()
                updatedT = resp.getType()
                if updatedT == 'ERRO':
                    file.close()
                    self._doMainMenu()
                if updatedT == 'GOOD':
                    print("Transaction updated successfully.")
                    file.close()
                    self._doMainMenu()

    def _doCancel(self):
        userFrom = USERNAME
        userTo = input("Input the user you sent the payment to: ")
        req = Cmessage()
        req.setType('CANC')
        req.addParam('userFrom', USERNAME)
        req.addParam('userTo', userTo)
        self._cproto.putMessage(req)
        resp = self._cproto.getMessage()
        message_type = resp.getType()
        if message_type == 'ERRO':
            print(resp.getParam('message'))
            self._doMainMenu()
        if message_type == 'GOOD':
            print(resp.getParam('message'))
            tid = resp.getParam('tid')
            req = Cmessage()
            req.setType('CANT') #cant is cancel transaction in
trasactions.txt
            req.addParam('tid', tid)
            self._cproto.putMessage(req)
            resp = self._cproto.getMessage()
            print("Transaction canceled")

        self._doMainMenu()

    def _openWalletOperations(self):
        print("-------Welcome to your Digital Wallet-------")
        menu = [' 1. Check Balance', ' 2. Make a deposit',' 3. View
Transaction History',' 98. Back', ' 99. Exit']
        choices = {'1': self._doCheckBalance, '2': self._doMakeDeposit,
                    '3': self._doCheckTransactions, '98':
self._openWalletMenu, '99': self._shutdown}
        print('\n'.join(menu))
        choice = input('> ')
        if choice in choices:
            m = choices[choice]
            m()

    def _openWalletMenu(self):
        print("-------Digital Wallet Sign-In Menu-------")
        menu1 = [' 1. Sign in to Wallet',' 98. Back', ' 99. Exit']
        if self._login == True:
            choices = {'1': self._doSignInWallet, '2':
self._doMakeDeposit, '98': self._doMainMenu, '99': self._shutdown}
        if self._login == False:
            choices = {'1': self._doSignInWallet, '2':
self._doMakeDeposit, '98': self._doTopLevelMenu, '99': self._shutdown}
        print('\n'.join(menu1))
        choice = input('> ')
        if choice in choices:
            m = choices[choice]
            m()

    #client side menu before connecting to server
    # edited successfully for adding register account
    def _doTopLevelMenu(self):
        print("-------Login Menu-------")
        menu = [' 1. Login', ' 2. Open my Digital Wallet', ' 3. Register an Account',' 99. Exit']
        choices = {'1': self._doLogin, '2': self._openWalletMenu, '3': self._doRegister, '99': self._shutdown}
        print('\n'.join(menu))
        choice = input('> ')
        if choice in choices:
            m = choices[choice]
            m()

    def _doMainMenu(self):
        print("-------Main Menu-------")
        menu = [' 1. Open Inbox', ' 2. Open my Digital Wallet', ' 3. Refund a payment', ' 4. Cancel a payment', ' 5. Send Payment Request', ' 6. Search', ' 98. Back', ' 99. Exit']
        choices = {'1': self._doCheckM, '2': self._openWalletMenu, '3': self._doRefund, '4': self._doCancel, '5': self._doPayRequest, '6': self._doSearch, '98': self._doLogout, '99': self._shutdown}
        print('\n'.join(menu))
        choice = input('> ')
        if choice in choices:
            m = choices[choice]
            m()

    def run(self):
        while (self._done == False):
            if (self._login == False):
                self._doTopLevelMenu()
            else:
                self._doMainMenu()
        self._shutdown()
```

tcp_server.py

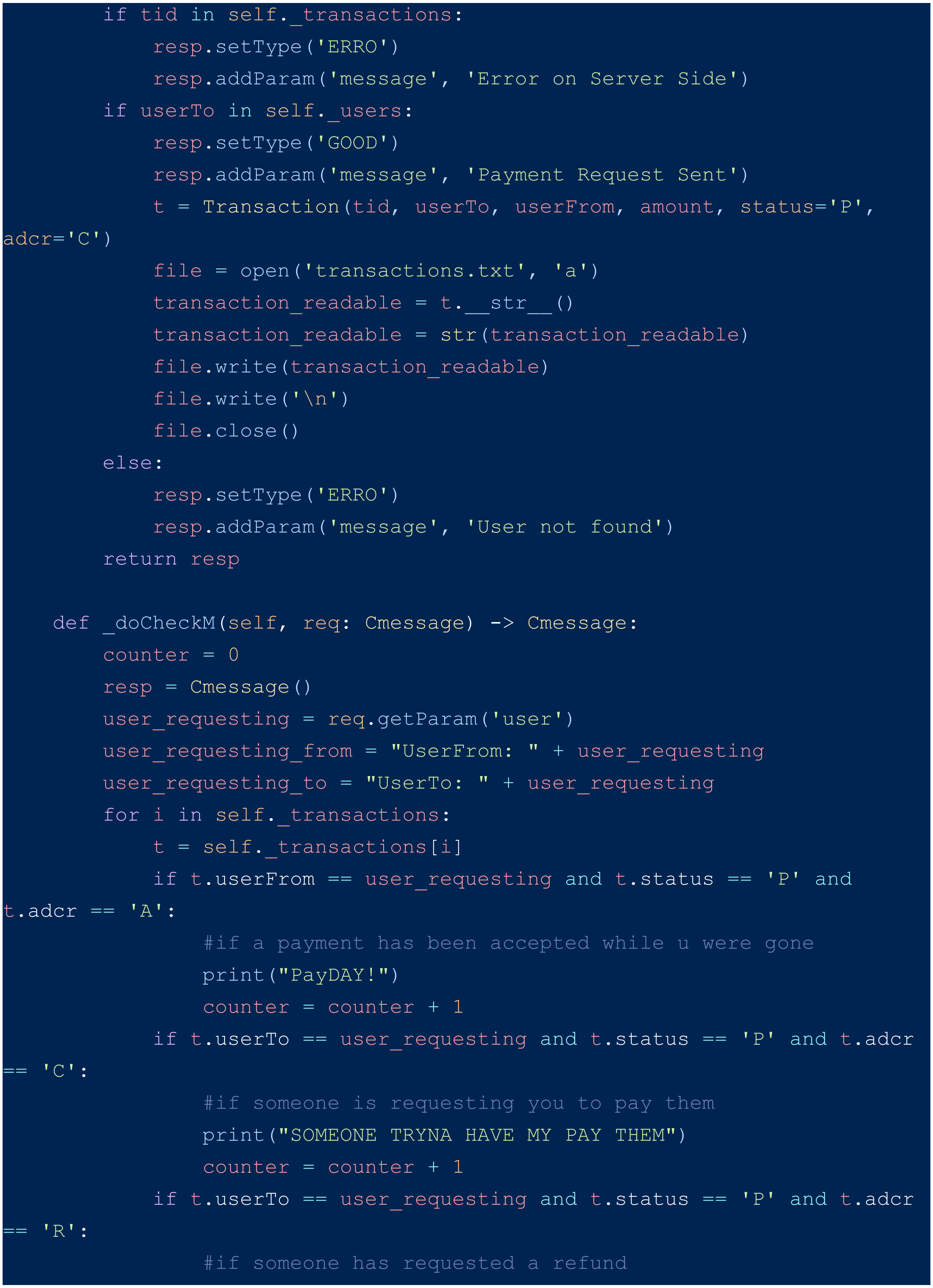
```
'''
Created on Feb 19, 2021

@author: bayden
'''

import socket
from cserverops import Cserverops
from cprotocol import Cprotocol

if __name__ == '__main__':
    # create the server socket
    #  defaults family=AF_INET, type=SOCK_STREAM, proto=0, filno=None
    serversoc = socket.socket()

    # bind to local host:50000
    port = 50001
    serversoc.bind(("localhost",port))

    # make passive with backlog=5
    serversoc.listen(5)

    # wait for incoming connections
    n = 0
    while n < 100:
        print("Listening on ", port)

        # accept the connection
        commsoc, raddr = serversoc.accept()

        # run the serverops
        sops = Cserverops()
        sops.sproto = Cprotocol(commsoc)
        sops.connected = True
        sops.run()
        n = n + 1



    # close the server socket
    serversoc.close()
```

cserverops.py

```
'''
Created on Mar 3, 2022

@author: bayden
'''
from contextlib import nullcontext
from cmessage import Cmessage
from cprotocol import Cprotocol
from cuser import Cuser
from transaction import Transaction

class Cserverops(object):
    '''
    classdocs
    '''


    def __init__(self):
        '''
        Constructor
        '''
        self._users = {}
        self._transactions = {}
        self.sproto = Cprotocol()
        self.connected = False
        self._login = False
        self._route = {'LGIN': self._doLogin,
                       'LOUT': self._doLogout,
                       'SRCH': self._doSearch,
                       'REGI': self._doRegister,
                       'PAYR': self._doPayRequest,
                       'CHKM': self._doCheckM,
                       'GETM': self._doGetM,
                       'TRAN': self._doUpdateTransactions,
                       'RFND': self._doUpdateTransactions,
                       'CANC': self._doCancel,
                       'CANT': self._doUpdateTransactions}
        self._debug = True

    def _debugPrint(self, m: str):
        if self._debug:
            print(m)

    def load(self, uname: str, cname: str):
        with open(uname) as fp:
            for line in fp:
                line = line.strip()
                values = line.split()
                user = Cuser(values[0],values[1],values[2])
                self._users[values[0]] = user

        with open(cname) as fp:
            for line in fp:
                line = line.strip()
                #tid = fp.readline().strip()
                userTo = fp.readline().strip()
                userFrom = fp.readline().strip()
                amount = fp.readline().strip()
                status = fp.readline().strip()
                adcr = fp.readline().strip()
                t = Transaction(line, userTo, userFrom, amount, status,
adcr)
                self._transactions[line] = t

    def _doLogin(self, req: Cmessage) -> Cmessage:
        resp = Cmessage()
        u = req.getParam('username')
        p = req.getParam('password')
        if u in self._users:
            if self._users[u].login(u,p):
                resp.setType('GOOD')
                resp.addParam('message', 'Login successful')

                global CURRENT_USER
                CURRENT_USER = u
```



```
                self._login = True
            else:
                resp.setType('ERRO')
                resp.addParam('message', 'Bad login')
                self.connected = False;
        else:
            resp.setType('ERRO')
            resp.addParam('message', 'Bad login')
            self.connected = False
        return resp

    def _doRegister(self, req: Cmessage) -> Cmessage:
        resp = Cmessage()
        new_u = req.getParam('username')
        new_p = req.getParam('password')
        new_a = req.getParam('alias')

        if new_u in self._users:
            resp.setType('ERRO')
            resp.addParam('message', 'Username already exists.')
            self.connected = False;
        else:
            #adding the new user to users.txt
            with open('users.txt', 'a') as f:
                string_to_write = ''
                string_to_write = '\n' + new_u + ' ' + new_p + ' ' + new_a
                f.write(string_to_write)
                f.close()

            resp.setType('GOOD')
            resp.addParam('message', 'Account created.')

        return resp


    def _doLogout(self, req: Cmessage) -> Cmessage:
        resp = Cmessage()
        resp.setType('GOOD')
        resp.addParam('message','Logout successful')
        self._login = False
        self.connected = False
        return resp

    def _doCancel(self, req: Cmessage) -> Cmessage:
        resp = Cmessage()
        userFrom = req.getParam('userFrom')
        userTo = req.getParam('userTo')
        for i in self._transactions:
            t = self._transactions[i]
            if t.userFrom == userFrom and t.userTo == userTo and t.status
== 'P' and t.adcr == 'C':
                tid = t.tid
                resp.addParam('tid', tid)
                resp.addParam('message', 'Transaction successfully
canceled')
                resp.setType('GOOD')
        return resp


    def _doSearch(self, req: Cmessage) -> Cmessage:
        resp = Cmessage()
        cid = req.getParam('cid')
        if cid in self._courses:
            c = self._courses[cid]
            resp.setType('DATA')
            resp.addParam('cid', c.cid)
            resp.addParam('name', c.name)
            resp.addParam('credits', c.credits)
        else:
            resp.setType('ERRO')
            resp.addParam('message', 'course not found')
        return resp

    def _doPayRequest(self, req: Cmessage) -> Cmessage:
        resp = Cmessage()
        userTo = req.getParam('userTo')
        userFrom = req.getParam('userFrom')
        amount = req.getParam('amount')
        tid = req.getParam('tid')
```

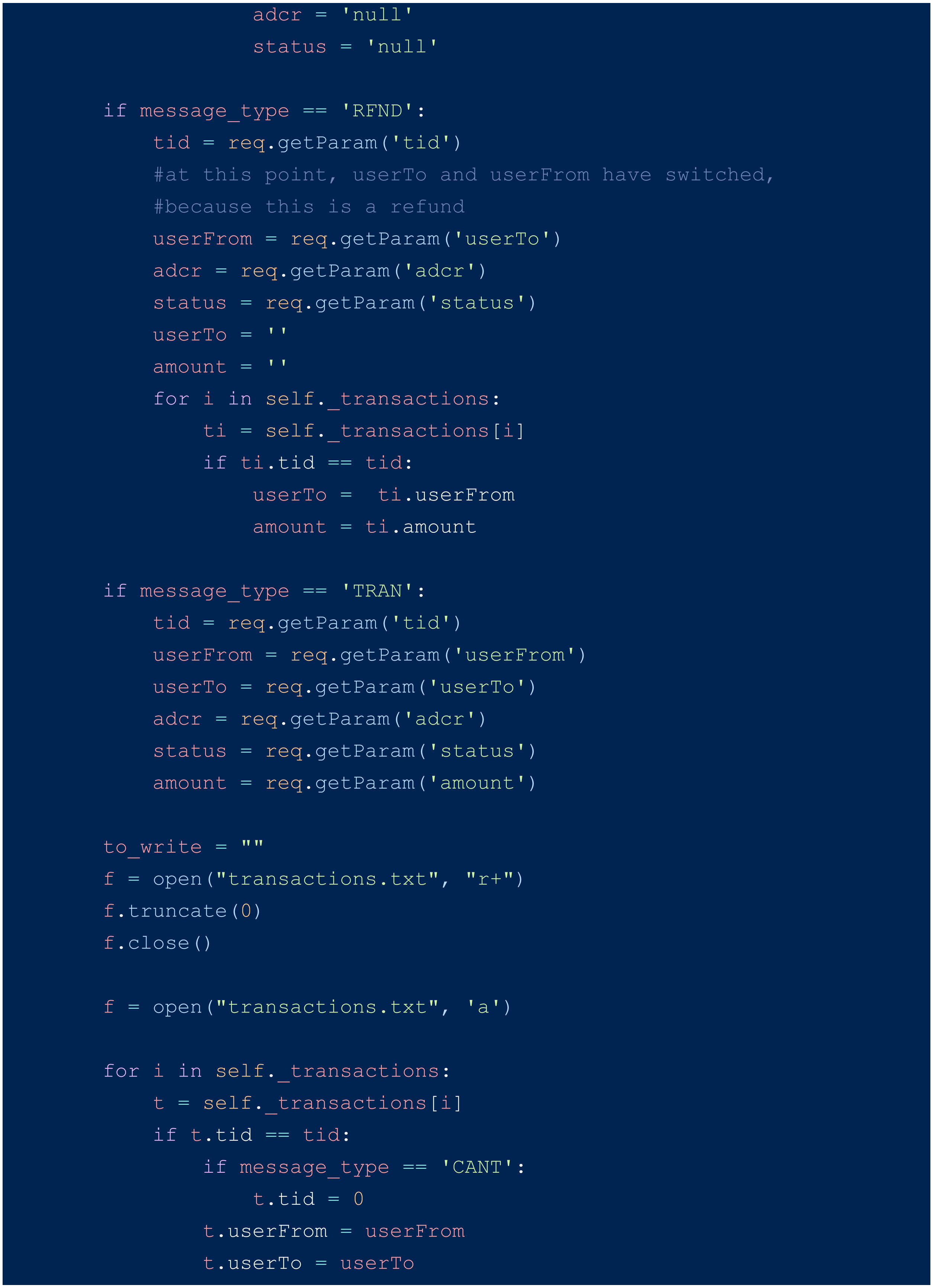


```
        if tid in self._transactions:
            resp.setType('ERRO')
            resp.addParam('message', 'Error on Server Side')
        if userTo in self._users:
            resp.setType('GOOD')
            resp.addParam('message', 'Payment Request Sent')
            t = Transaction(tid, userTo, userFrom, amount, status='P',
adcr='C')
            file = open('transactions.txt', 'a')
            transaction_readable = t.__str__()
            transaction_readable = str(transaction_readable)
            file.write(transaction_readable)
            file.write('\n')
            file.close()
        else:
            resp.setType('ERRO')
            resp.addParam('message', 'User not found')
        return resp

    def _doCheckM(self, req: Cmessage) -> Cmessage:
        counter = 0
        resp = Cmessage()
        user_requesting = req.getParam('user')
        user_requesting_from = "UserFrom: " + user_requesting
        user_requesting_to = "UserTo: " + user_requesting
        for i in self._transactions:
            t = self._transactions[i]
            if t.userFrom == user_requesting and t.status == 'P' and
t.adcr == 'A':
                #if a payment has been accepted while u were gone
                print("PayDAY!")
                counter = counter + 1
            if t.userTo == user_requesting and t.status == 'P' and t.adcr
== 'C':
                #if someone is requesting you to pay them
                print("SOMEONE TRYNA HAVE MY PAY THEM")
                counter = counter + 1
            if t.userTo == user_requesting and t.status == 'P' and t.adcr
== 'R':
                #if someone has requested a refund
                print("SOMEONE REQUESTING REFUND")
                counter = counter + 1
            if t.userFrom == user_requesting and t.status == 'C' and
t.adcr == 'Z':
                print("PAYDAY! (Refund)")
                counter = counter + 1
                #someone payed your refund request

        if counter == 0:
            resp.setType('ERRO')
            counter = str(counter)
            message = "You have " + counter + " new messages"
            resp.addParam('message', message)
        else:
            counter = str(counter)
            resp.setType('GOOD')
            message = "You have " + counter + " new messages"
            resp.addParam('message', message)
        return resp

    def _doGetM(self, req: Cmessage) -> Cmessage:
        refund_or_pay = 0
        #refund = 0, pay = 1
        resp = Cmessage()
        print('got to the doGetM')
        user_requesting = req.getParam('user')
        for i in self._transactions:
            t = self._transactions[i]
            if(t.userFrom == user_requesting and t.status == 'P' and
t.adcr == 'A') or (t.userFrom == user_requesting and t.status == 'C' and
t.adcr == 'Z'):
                refund_or_pay = 1
                resp.setType('WOOH')
                if t.userFrom == user_requesting and t.status == 'C' and
t.adcr == 'Z':
                #someone payed me my refund request while I was gone
                    refund_or_pay = 0
                    message = t.userFrom + " has payed you a refund of " +
t.amount + " dollars."
                    resp.addParam('message', message)
                #if a payment has been accepted while u were gone
                walletname = ''
                walletname = user_requesting + "DigitalWallet.txt"
                f = open(walletname, "r")
                content = f.readlines()
                balance = content[2]
                balance = balance.strip()
                prev_balance_copy = balance
                balance1 = balance
                balance1 = balance[8:]
                balance1 = int(balance1)
                balance3 = t.amount.strip()
                balance3 = t.amount
                add = int(balance3)
                balance2 = balance1 + add
                balance2 = str(balance2)
                new_balance_in_file = str("Balance:" + balance2)
                f.close()

                file = open(walletname, "r")
                replacement = ''
                for line in file:
                    line = line.strip()
                    changes = line.replace(prev_balance_copy,
new_balance_in_file)
                    replacement = replacement + changes + "\n"
                file.close()
                fout = open(walletname, 'w')
                fout.writelines(replacement)
                fout.close()

                new_t = Transaction(t.tid, t.userTo, t.userFrom, t.amount,
'C', 'C')

                if refund_or_pay == 1:
                    message = ''
                    add = str(add)
                    message = t.userTo + " payed you " + add + " dollars."
```

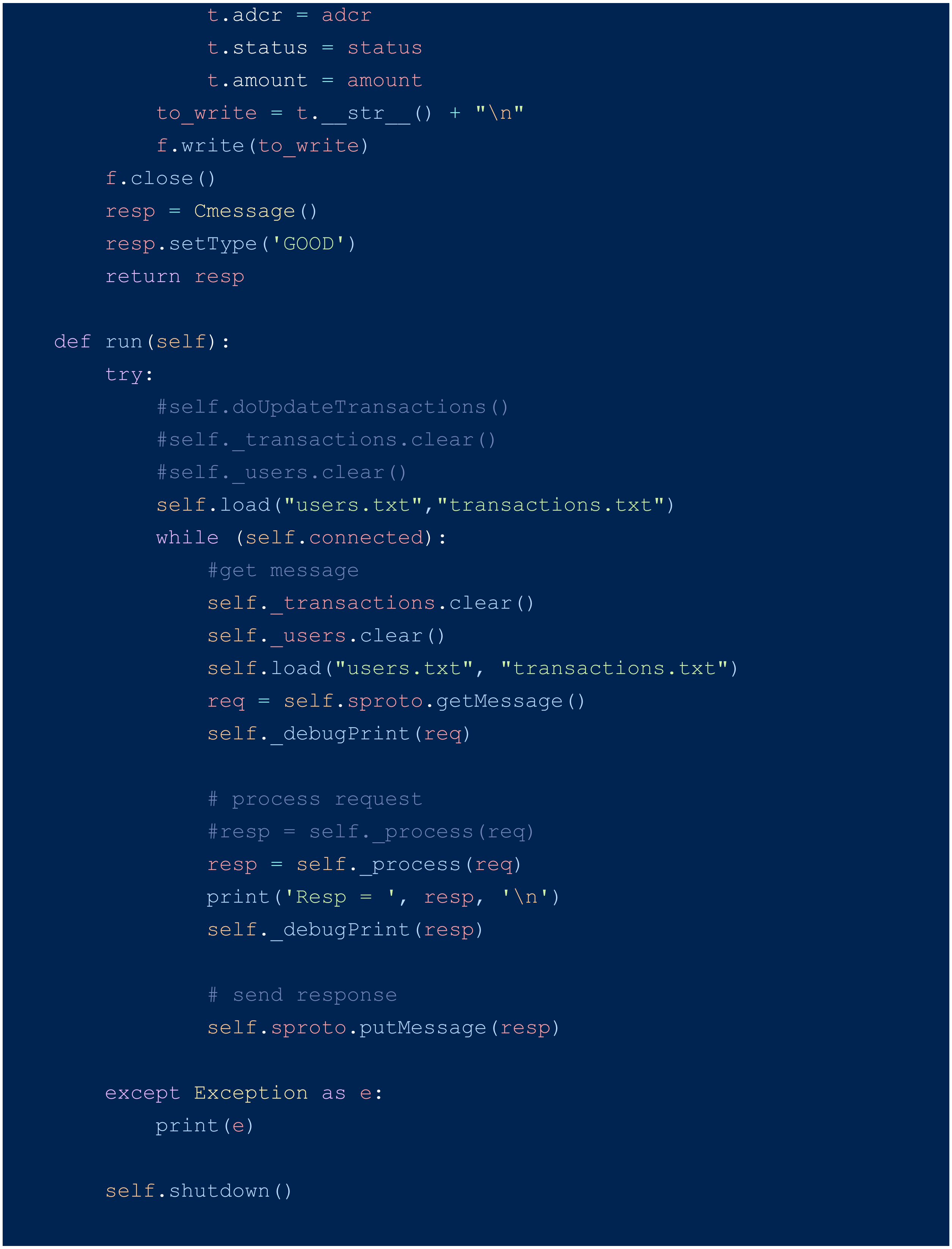
```
        resp.addParam('message', message)

    fadd = open(walletname, 'a')

    transaction_readable = new_t.__stri__()
    transaction_readable = str(transaction_readable)

    fadd.write(transaction_readable)
    fadd.close()

    #update transactions
    file = open("transactions.txt")
    replacement = ""
    changes = ""
    marker = 0
    marker2 = 0
    initial = 0
    for line in file:
        line = line.strip()
        if marker != 0:
            marker2 = marker2 + 1
        if marker2 == 4:
            changes = line.replace(t.status, new_t.status)
            replacement = replacement + changes + "\n"
        if marker2 == 5:
            changes == line.replace(t.adcr, new_t.adcr)
            replacement = replacement + changes + "\n"
            marker = 0

        if line == t.tid:
            marker = 1

        if marker2 != 4 and marker2 != 5:
            if initial == 1:
                replacement = replacement + line + "\n"

            if initial == 0:
                replacement = line + "\n"
                initial = 1
```

```
                    if marker2 == 5:
                        marker2 = 0
                file.close()
                fout = open("transactions.txt", 'w')
                fout.writelines(replacement)
                fout.close()

                break
            #if a payment has been accepted while u were gone


            if t.userTo == user_requesting and t.status == 'P' and t.adcr
== 'R':
                #if someone has requested a refund
                message = t.userFrom + " has requested for you to refund
them " + t.amount + " dollars."
                resp.addParam('message', message)
                resp.setType('RFNI') #refund request coming in
                break

            if t.userTo == user_requesting and t.status == 'P' and t.adcr
== 'C':
            #if someone is requesting you to pay them
                message = t.userFrom + "has requested for you to pay them
" + t.amount + " dollars."
                resp.addParam('message', message)
                resp.setType('PAYI')
                break



            # if t.userFrom == user_requesting and t.status == 'P' and
t.adcr == 'R':
            #     #if someone has requested a refund
            #     print("SOMEONE REQUESTING REFUND")
            #     message = t.userTo + " is requesting a refund of " +
t.amount + " dollars from previous payment ID: " + t.tid + "."
            #     resp.addParam('message', message)
            #     resp.setType('RPAY')
            #     break
```

```
        _tid = t.tid
        _userFrom = t.userFrom
        _userTo = t.userTo
        _adcr = t.adcr
        _status = t.status
        _amount = t.amount
        resp.addParam('tid', _tid)
        resp.addParam('userFrom', _userFrom)
        resp.addParam('userTo', _userTo)
        resp.addParam('adcr', _adcr)
        resp.addParam('status', _status)
        resp.addParam('amount', _amount)



        return resp

    def _process(self, req: Cmessage) -> Cmessage:
        m = self._route[req.getType()]
        return m(req)

    def shutdown(self):
        self.sproto.close()
        self.connected = False
        self._login = False

    def _doUpdateTransactions(self, req: Cmessage) -> Cmessage:
        #loop with _transactions[] updating transactions file


        message_type = req.getType()
        if message_type == 'CANT':
            tid = req.getParam('tid')
            for i in self._transactions:
                tii = self._transactions[i]
                if tii.tid == tid:
                    tid_null = 0
                    userTo = 'null'
                    userFrom = 'null'
                    amount = 'null'
```



```
                    adcr = 'null'
                    status = 'null'

        if message_type == 'RFND':
            tid = req.getParam('tid')
            #at this point, userTo and userFrom have switched,
            #because this is a refund
            userFrom = req.getParam('userTo')
            adcr = req.getParam('adcr')
            status = req.getParam('status')
            userTo = ''
            amount = ''
            for i in self._transactions:
                ti = self._transactions[i]
                if ti.tid == tid:
                    userTo =  ti.userFrom
                    amount = ti.amount

        if message_type == 'TRAN':
            tid = req.getParam('tid')
            userFrom = req.getParam('userFrom')
            userTo = req.getParam('userTo')
            adcr = req.getParam('adcr')
            status = req.getParam('status')
            amount = req.getParam('amount')

        to_write = ""
        f = open("transactions.txt", "r+")
        f.truncate(0)
        f.close()

        f = open("transactions.txt", 'a')

        for i in self._transactions:
            t = self._transactions[i]
            if t.tid == tid:
                if message_type == 'CANT':
                    t.tid = 0
                t.userFrom = userFrom
                t.userTo = userTo
```



```
                t.adcr = adcr
                t.status = status
                t.amount = amount
            to_write = t.__str__() + "\n"
            f.write(to_write)
        f.close()
        resp = Cmessage()
        resp.setType('GOOD')
        return resp

    def run(self):
        try:
            #self.doUpdateTransactions()
            #self._transactions.clear()
            #self._users.clear()
            self.load("users.txt","transactions.txt")
            while (self.connected):
                #get message
                self._transactions.clear()
                self._users.clear()
                self.load("users.txt", "transactions.txt")
                req = self.sproto.getMessage()
                self._debugPrint(req)

                # process request
                #resp = self._process(req)
                resp = self._process(req)
                print('Resp = ', resp, '\n')
                self._debugPrint(resp)

                # send response
                self.sproto.putMessage(resp)

        except Exception as e:
            print(e)

        self.shutdown()
```

cprotocol.py

```python
'''
Created on Feb 24, 2022

@author: bayden
'''
import socket
from cmessage import Cmessage

class Cprotocol(object):
    '''
    classdocs
    '''
    BUFSIZE = 8196

    def __init__(self, s: socket=-1):
        '''
        Constructor
        '''
        self._sock = s

    def _loopRecv(self, size: int):
        data = bytearray(b" "*size)
        mv = memoryview(data)
        while size:
            rsize = self._sock.recv_into(mv,size)
            mv = mv[rsize:]
            size -= rsize
        return data

    def putMessage(self, m: Cmessage):
        data = m.marshal()
        self._sock.sendall(data.encode('utf-8'))

    def getMessage(self) -> Cmessage:
        try:
            m = Cmessage()
            size = int(self._loopRecv(4).decode('utf-8'))
            mtype = self._loopRecv(4).decode('utf-8')
            params = self._loopRecv(size).decode('utf-8')
            m.unmarshal(params)
            m.setType(mtype)
        except Exception:
            raise Exception('bad getMessage')
        else:
            return m

    def close(self):
        self._sock.close()
```

cmessage.py

```
'''
Created on Feb 24, 2022

@author: bayden
'''
from enum import Enum

class Cmessage(object):
    '''
    classdocs
    '''
    # Constance
    MCMDS = Enum('MCMDS', {'LGIN': 'LGIN', 'LOUT': 'LOUT','SRCH': 'SRCH',
                           'DATA': 'DATA', 'GOOD': 'GOOD', 'ERRO': 'ERRO',
                           'REGI': 'REGI', 'PAYR': 'PAYR', 'CHKM': 'CHKM',
                           'GETM': 'GETM', 'PAYI': 'PAYI', 'RPAY': 'RPAY',
                           'WOOH': 'WOOH', 'TRAN': 'TRAN', 'RFND': 'RFND',
                           'RFNI': 'RFNI', 'CANC': 'CANC', 'CANT':
'CANT'})

    PJOIN = '&'
    VJOIN = '{}={}'
    VJOIN1 = '='

    def __init__(self):
        '''
        Constructor
        '''
        self._type = Cmessage.MCMDS['GOOD']
        self._params = {}

    def __str__(self) -> str:
        '''
        Stringify - marshal
        '''
        return self.marshal()

    def reset(self):
        self._type = Cmessage.MCMDS['GOOD']
        self._params.clear()
        self._params = {}

    def setType(self, mtype: str):
        self._type = Cmessage.MCMDS[mtype]

    def getType(self) -> str:
        return self._type.value

    def addParam(self, name: str, value: str):
        self._params[name] = value;

    def getParam(self, name: str) -> str:
        return self._params[name]

    def marshal(self) -> str:
        size = 0
        params = ''
        if (self._params):
            pairs = [Cmessage.VJOIN.format(k,v) for (k, v) in
self._params.items()]
            params = Cmessage.PJOIN.join(pairs)
            size = len(params)
        return '{:04}{}{}'.format(size,self._type.value,params)

    def unmarshal(self, value: str):
        self.reset()
        if value:
            params = value.split(Cmessage.PJOIN)
            for p in params:
                k,v = p.split(Cmessage.VJOIN1)
                self._params[k] = v
```

cuser.py

```
'''
Created on Feb 24, 2022

@author: bayden
'''

class Cuser(object):
    '''
    classdocs
    '''

    def __init__(self, u: str, p: str, a: str):
        '''
        Constructor
        '''
        self._username = u
        self._password = p
        self._alias = a

    def __str__(self):
        return
'Alias:{}\nUsername:{}\n'.format(self._alias,self._username)



    def login(self, u: str, p: str) -> bool:
        if ((self._username == u) and (self._password == p)):
            return True
        else:
            return False
```

transaction.py

```python
'''
Created on Feb 24, 2022

@author: bayden
'''

class Transaction(object):
    '''
    classdocs
    '''

    def __init__(self, tid: str, userTo: str, userFrom: str, amount: str, status: str, adcr: str):
        '''
        Constructor
        '''
        self.tid = tid
        self.userTo = userTo
        self.userFrom = userFrom
        self.amount = amount
        self.status = status
        self.adcr = adcr
        #ADC = Accepted, Denied, or Completed
        #Completed means that it is either fully complete, or
        # or that userTo hasnt Accepted or Denied yet

    def __str__(self):
        #return 'TID: {}\nUserTo: {}\nUserFrom: {}\nAmount: {}\nStatus: {}\nadcr: {}'.format(self.tid,self.userTo,self.userFrom,self.amount,self.status,self.adcr)
        return '{}\n{}\n{}\n{}\n{}\n{}'.format(self.tid,self.userTo,self.userFrom,self.amount,self.status,self.adcr)

    def __stri__(self):
        return 'Transaction ID: {}\nSent to: {}\nSent from: {}\nAmount: {}\nStatus: {}\nAccepted/Denied/Refund/Completed:
```

```
{}'.format(self.tid,self.userTo,self.userFrom,self.amount,self.status,self
.adcr)
```