**Program Part 2 Checklist**

**Requirements:**

1. implement all commands from this and previous assignments: BUY, SELL, ~~DEPOSIT,~~

~~BALANCE~~, LIST, ~~QUIT, SHUTDOWN~~, ~~LOGIN, LOGOUT, WHO, LOOKUP~~.

2. all users share the same data source (one database or one text file). The user’s information should be maintained by the server.

**Not complete? (Not sure if he wants us to use these userIDs or the existing ones we already had)**

You must have the following users (lower case) in your system:

|  |  |
| --- | --- |
| **UserID** | **Password** |
| **root** | **root01** |
| mary | mary01 |
| john | john01 |
| moe | moe01 |

|  |  |
| --- | --- |
| **UserID** | **Password** |
| root | root01 |
| j\_doe | Passwrd4 |
| j\_smith | Pass456 |
| c\_brown | Snoopy |
| l\_van | Football |
| l\_blanket | Security23 |

**3. make sure that you do sufficient error handling such that a user can't crash your server. For instance, what will you do if a user provides invalid input?**

**LOGIN command may need this? When a user enters invalid username or password, we will display the error message first then close the client? (Optional – added this already but we can remove it if unnecessary)**

~~4. The client should be able to connect to the server running on any machine. Therefore, the~~

~~server IP address should be a command line parameter for the client program.~~

~~5. the server should print out all messages received from clients on the screen.~~

~~6. when the previous client exits, the server should allow the next client to connect.~~

~~7. your source codes must be well commented.~~

**README file** **90% Complete (uploaded to GitHub called ReadMe2.md**

- missing screenshots of: (Server and client 1 & Server, with at least two clients: client 1 and client 2).

-and report if we have any bugs

8. include a **README file** in your submission. A document with instructions, screenshots

and what was each student’s role in this assignment: ~~Introduction~~, ~~Running Instructions, Each Student’s role~~, Bugs in the code, ~~Test cases~~, Screenshots (Server and client 1 & Server, with at least two clients: client 1 and client 2).

**9. Video Recording**

**Commands:**

**LOGIN (Works)**

Login the user to the remote server. A client that wants to login should begin by sending the

ASCII string “LOGIN" followed by a space, followed by a UserID, followed by a space,

followed by a Password, and followed by the newline character (i.e., '\n').

Your server should be initialized with the UserIDs and Passwords of at least four users who will

be allowed to execute the commands on the server. However, a non-root user is allowed to

execute the QUIT commands anonymously (without login).

When the server receives a LOGIN command from a client, it should check if the UserID and

password are correct and match what the server stores. If login info is not correct or doesn’t

exist, the server should return the string “403 Wrong UserID or Password,” otherwise the server

should return the “200 OK” message. This command will result in a server creating a new thread for this client.

A client-server interaction with the LOGIN command thus looks like:

Client sends: LOGIN john john01

Server sends: 200 OK

If Invalid Login credentials: “403 Wrong UserID or Password,”

A black screen with white text

Description automatically generated A screen shot of a computer

Description automatically generated

**A black screen with white text

Description automatically generated**

**LIST (Not complete)**

A LIST command issued by user John should return only John’s records. A client

server interaction looks like this:

Client sends: LIST

Server sends: 200 OK

The list of records in the Pokémon cards table for current user, John:

ID Card Name Type Rarity Count OwnerID

1 Pikachu Electric Common 2 John

5 Charizard Fire Rare 6 John

Scenario 2, if a root user is currently logged in

Client sends: LIST

Server sends: 200 OK

The list of records in the cards database:

ID Card Name Type Rarity Count OwnerID

1 Pikachu Electric Common 2 John

2 Charizard Fire Rare 1 Jane

5 Charizard Fire Rare 6 John

**Root User:** Only a root user can list ALL records for ALL users.

– What we need to fix:

* I currently have it displaying all records for all users, but the last record gets cut off
* Also, we need to display users first\_name for OwnerID

A screen shot of a computer

Description automatically generated 

**User**: Only a root user can list ALL records for ALL users.

– What we need to fix:

* We need to display users first\_name for OwnerID (and remove users first name from the result)

A black background with white text

Description automatically generated

**LOGOUT (WORKS/COMPLETE)**

Logout from the server. A client sends the ASCII string “LOGOUT" followed by a name

followed by the newline character (i.e., '\n').

A user is not allowed to send BUY, SELL, LIST, BALANCE, and SHUTDOWN commands after logout, but it can still send the QUIT commands. This command should result in the server terminating the allocated socket and thread

for this client.

A client-server interaction with the LOGOUT command looks like:

Client sends: LOGOUT

Server sends: 200 OK

A black background with white text

Description automatically generatedA close up of a sign

Description automatically generated

A black background with white text

Description automatically generated A black screen with white text

Description automatically generated

**WHO (WORKS- still need to test server with 2 clients)**

List all active users, including the UserID and the user’s IP addresses. A client sends the ASCII

string “WHO" followed by the newline character (i.e., '\n'). T

his command is only allowed for the root user.

A client-server interaction with the WHO command thus looks like:

Client sends: WHO

Server sends: 200 OK

The list of the active users:

John 141.215.69.202

root 127.0.0.1

Root user vs User outputs:

A black background with white text

Description automatically generated A black background with white text

Description automatically generated A black screen with white text

Description automatically generated A black rectangular sign with white text

Description automatically generated

**LOOKUP (WORKS/COMPLETE)**

Look up a Pokémon card name in the list. Display a complete card record for the logged in user

that matched full or partial name of the card’s name. A client sends the ASCII string

“LOOKUP" followed by a space, followed by a name followed by the newline character (i.e.,

'\n').

When the server receives a LOOKUP command from a client, it will look up the card’s name

and returns the matched record for that logged in user. When there is a match, it returns the “200

OK” message and all matched record(s). If there is no match, it returns the “404 Your search did

not match any records”. Partial or full card name are both accepted.

A client-server interaction with the LOOKUP command thus looks like:

Client sends: LOOKUP Fire

Server sends: 200 OK

Found 1 match

ID Card Name Type Rarity Count Owner

2 Charizard Fire Rare 1 John

Client sends: LOOKUP random\_card

Server sends: 404 Your search did not match any records.

A screenshot of a computer

Description automatically generated

**DEPOSIT (WORKS/COMPLETE)**

Deposit USD to the user’s account/record. A user can deposit an amount of USD into their

account. A client that wants to deposit an amount of USD should begin by sending the ASCII

string “DEPOSIT" followed by a space, followed by a USD amount, followed by a space,

followed the newline character (i.e., '\n').

A client-server interaction with the LOOKUP command thus looks like:

Client sends: DEPOSIT 12.34

Server reply: deposit successfully. New User balance $112.34

A black screen with white text

Description automatically generatedA black background with white text

Description automatically generated

**SHUTDOWN (WORKS/COMPLETE)**

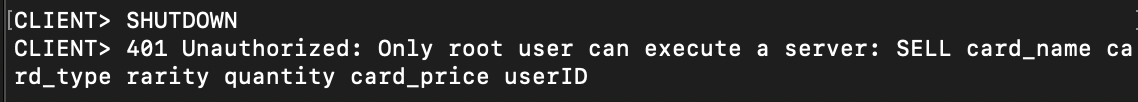
If a client sends a shutdown command to the server, it will terminate all connected clients and

shut down the server. However, only the root user has the authority to execute a server

shutdown. If a non-root user attempts to send a shutdown command, the server should refuse it

and send a 401 status code, and an error message accordingly.

When non-root users try shutdown command:

 A close up of a sign

Description automatically generated

When root users shutdown:

A black background with white text

Description automatically generated A black screen with white text

Description automatically generated

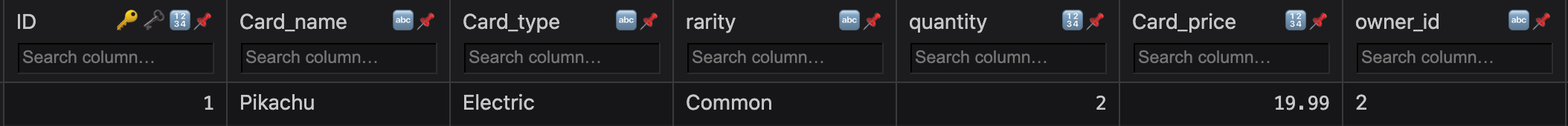
**Program 1:**

**BUY (Not working?):**

* I added the correct parameters, but I keep getting the missing information error.

A black background with white text

Description automatically generated



//clientData Holds:

//Pikachu: clientData[0] -name

//electric: clientData[1] -type

//common: clientData[2] -rarity

//2: clientData[3] -Quantity

//19.99: clientData[4] -COST of Card

//3: clientData[5] - buy FROM

//clientData[0]: card\_name

//clientData[1]: card\_type

//clientData[2]: rarity

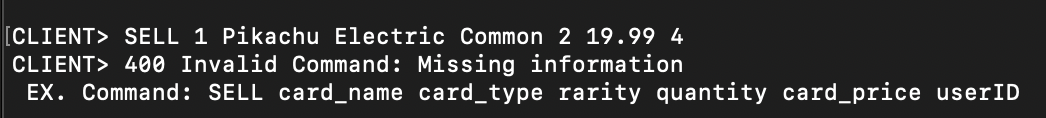
//clientData[3]: quantity

//clientData[4]: card\_price

//clientData[5]: owner\_id

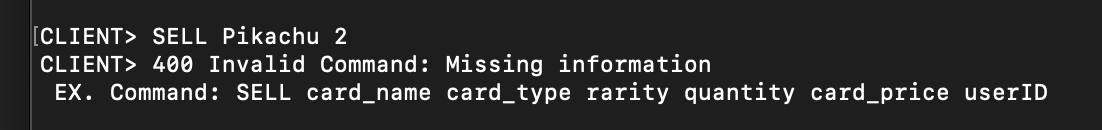
**SELL (Not working?):**

* I added the correct parameters, but I keep getting the missing information error.

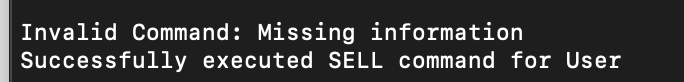
****

Sell command outputs:

When user enters invalid parameters:



Server:



**BALANCE (complete) :**

Display the USD balance for the selected user  
A client-server interaction with the BALANCE command looks like:  
c: BALANCE 3  
s: Received: BALANCE 3  
c: 200 OK  
Balance for user Jane Smith: $10.00

A black background with white text

Description automatically generatedA black background with white text

Description automatically generated

**QUIT (complete):**

Only terminate the client. The client exits when it receives the confirmation message from the

server.

A client-server interaction with the QUIT command looks like:

c: QUIT

c: 200 OK

A black background with white text

Description automatically generated A black screen with white text

Description automatically generated

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test # | Description of Test | Input Value | Expected Output | Actual Output | Test Pass/Fail |
| 1 | LOGIN: Test valid user/password | LOGIN j\_doe Passwrd4 | 200 OK | 200 OK | Pass |
| 2 | LOGIN: Test Invalid input | LOGIN j\_doe Passw11 | 403 Wrong UserID or Password | 403 Wrong UserID or Password | Pass |
| 3 | Test a successful LOGOUT command | LOGOUT | 200 OK | 200 OK | Pass |
| 4 | Test a WHO command by root user | WHO | 200 OK The list of the active users: (User list) | 200 OK  The list of the active users:  John 141.215.69.202  root 127.0.0.1 | Pass |
| 5 | Test a WHO command by non-root user | WHO | 401 Unauthorized: This command is only allowed for the root user. | 401 Unauthorized: This command is only allowed for the root user. | Pass |
| 6 | Test a successful LOOKUP command with a full card name match | LOOKUP Charizard | 200 OK Found 1 match (Card details) | 200 OK  Found 1 match  ID Card Name Type Rarity Count Owner  2 Charizard Fire Rare 1 John | Pass |
| 7 | Test a LOOKUP command with no card name match | LOOKUP Charmander | 404 Your search did not match any records | 404 Your search did not match any records | Pass |
| 8 | Test a successful DEPOSIT command | DEPOSIT 40.50 | Deposit successfully. New User balance $140.50 | 200 OK  Deposit successfully. New User balance for john: $140.50 | Pass |
| 9 | Test a successful SHUTDOWN command by a root user | SHUTDOWN | Server sends: 200 OK and terminates | 200 OK and terminates | Pass |
| 10 | Test an unsuccessful SHUTDOWN command by a non-root user | SHUTDOWN | 401 Unauthorized: Only root user can execute a server SHUTDOWN. | 401 Unauthorized: Only root user can execute a server SHUTDOWN. | Pass |
| 11 | Test a successful BUY command | BUY Pikachu Electric Common 19.99 1 2 | 200 OK  BOUGHT: New balance: 1 Pikachu. User USD balance $50.01 | 200 OK  BOUGHT: New balance: 1 Pikachu. User USD balance $50.01 | Pass |
| 12 | Test a failed BUY command with insufficient user balance | BUY Squirtle Water Uncommon 30 4 | 400 Invalid Command: Not enough balance | 400 Invalid Command: Not enough balance | Pass |
| 13 | Test a failed BUY command with invalid parameters | BUY Pikachu 19.99 1 | 400 Invalid Command: Missing information  EX. Command: BUY card\_name card\_type rarity quantity card\_cost userID | 400 Invalid Command: Missing information  EX. Command: BUY card\_name card\_type rarity quantity card\_cost userID | Pass |
| 14 | Test a successful SELL command | SELL Pikachu 1 19.99 4 | 200 OK  SOLD: New balance: 1 Pikachu. User's balance USD $119.99 | 200 OK  SOLD: New balance: 1 Pikachu. User's balance USD $119.99 | Pass |
| 15 | Test a successful LIST command for a specific owner (non-root user) | LIST | 200 OK  The list of records in the Pokémon cards table for current user, user 1: (Card list) | 200 OK  The list of records in the Pokémon cards table for charlie,  ID Card Name Type Rarity Count OwnerID  3 Bulbasaur Grass Common 50 Charlie | Pass |
| 16 | Test a successful LIST command for a specific owner (root user) | LIST | 200 OK  The list of records in the cards database:  (all users Pokémon Card records) | 200 OK  The list of records in the cards database:  ID Card Name Type Rarity Count OwnerID  1 Pikachu Electric Common 2 John  2 Charizard Fire Rare 1 Jane  3 Bulbasaur Grass Common 50 Charlie  4 Squirtle Water Uncommon 30 Lucy  5 Jigglypuff Normal Common 3 Linus | Pass |
| 17 | Test a successful BALANCE command | BALANCE | 200 OK  Balance for user Jane Smith: $90.00 | 200 OK  Balance for user jane: $90.00 | Pass |
| 18 | Test a successful QUIT command | QUIT | 200 OK | 200 OK | Pass |

WRITE 2 PROGRAMS-

~~Client~~

~~Server~~

~~Matching Port Numbers~~

~~Client cannot connect w/out IP input~~

~~SQLITE Database~~

~~2 Tables, Initially EMPTY~~

~~Table 1: Users~~

~~Table 2: Pokemon\_cards~~

~~Each User record in the users table should have the following fields:~~

~~• ID int NOT NULL AUTO\_INCREMENT,~~

~~• email varchar(255) NOT NULL,~~

~~• first\_name varchar(255),~~

~~• last\_name varchar(255),~~

~~• user\_name varchar(255) NOT NULL,~~

~~• password varchar(255),~~

~~• usd\_balance DOUBLE NOT NULL,~~

Teacher put usd\_balance 2x, but SQLITE says invalid with same name. No balance in Pokemon\_cards->Must have meant that since need balance

~~Each Pokemon record in the Pokemon\_Cards table should have the following fields:~~

~~• ID int NOT NULL AUTO\_INCREMENT,~~

~~• card\_name Text NOT NULL,~~

~~• card\_type Text NOT NULL,~~

~~• rarity Text NOT NULL,~~

~~• count Integer,~~

~~• owner\_id Text,~~

~~+ CARD BALANCE!~~

COMMANDS-

~~LOGIN~~

~~BUY~~

~~SELL~~

~~BALANCE~~

~~LIST~~

~~SHUTDOWN~~

~~QUIT~~

BUY: (\*Works! But need to have all info from table)

c:BUY card\_name card\_type rarity card\_price quantity user\_id

S: Received: echoed

c: 200 OK

to\_c: BOUGHT: New Balance: quantity card\_name. User USD balance card\_price

SELL: (WORKS! But need to remove from table once sold-> i.e delete where card\_balance = 0.0)

c: SELL card\_name quantity card\_price user\_id

s: Received: echoed

c: 200 OK

to\_c: SOLD: New balance: quantity card\_name. User's balance USE usd\_balance

LIST: (Works! But need owner\_id as input, once logged in wasn’t requiring them to keep putting their id)

c: LIST owner\_id

s: Recieved: echoed

c: 200 OK

to\_c: The list of records in the Pokémon cards table for current user, user user\_id

BALANCE:

c: BALANCE owner\_id

s: Recieved: echoed

c: 200 OK

to\_c: Balance for user first\_name last\_name: usd\_balance

SHUTDOWN: (WORKS!)

c: SHUTDOWN

s: Received: echoed

c: 200 OK

~~QUIT: (WORKS!)~~

~~c: QUIT~~

~~c: 200 OK~~

Error for input-> "400 invalid command"

"403 message format error"

+ what the error is (what incorrect input)

-Not enough USD balance

User user\_id doesn't exist (for selling to wrong user)

->Have to edit but done. Right now I have it outputting everything for format and invalid commands as the same to just have something initially

After BUY and SELL the Pokemon Cards table and Users table should be updated

~~• Both the server and client should be able to run on any UMD login servers.~~

• Make sure that you do sufficient error handling such that a user can't crash your server.

For instance, what will you do if a user provides invalid input?

~~• The server IP address should be a command line parameter for the client program.~~

~~• The server should print out all messages received from clients on the screen.~~

• When the previous client exits, the server should allow the next client to connect.

• Include a README.docx file in your submission with all instructions about the development environment and how compile and run instructions in details.

• Record and submit a less than 10 minutes video showing that your project is running as expected in all cases above. This is like what you would demo in an on-campus class setup. Make sure the command line font is large enough to be read.