PROJECT REPORT

Oliver Njeru - 663565
Celestine Gitau – 664474
Adrian Murage – 662292
Martha Murage - 665036

School of Sciences and Technology, United States International University-Africa
APT3090B: Cryptography & Network Security
Dr. Stanley Githinji
April 2, 2024

Table of Contents

Implementation of RSA over TCP.

Introduction.

RSA (Rivest-Shamir-Adleman) is a widely used public-key cryptosystem known for its security and versatility in ensuring secure network communication. The integration of RSA with TCP (Transmission Control Protocol) works to secure data transmitted over the internet, allowing encrypted messages to be exchanged between parties while minimizing the risk of eavesdropping and tampering with the messages. This report explores the implementation of RSA over TCP, pointing out the key concepts, design considerations, and technical details involved in achieving encryption, decryption, and transmission of data for secure communication.

Methodology.

The language of choice for this project was Python. The implementation of the RSA algorithm over TCP involved two main components: the Alice and Bob applications. These applications serve as the endpoints for secure communication, utilizing TCP sockets to exchange encrypted messages.

Alice

Alice's application acts like a server that uses sockets for communication and RSA for key exchange. The server is set up to listen for connections on localhost (the machine where the server is running) on port 9999.

We start by generating a session key using the generateSessionKey() function. This session key is used for symmetric encryption of the actual data sent over the connection. We then create a socket and bind it to the specified host and port. Once that is set up, we can start listening for incoming connections.

When a client connects to the server, we accept the connection and return a new socket object and the address of the client that made the connection. We then receive the client's public RSA key and use it to encrypt the session key. The encrypted session key is sent back to the client.

The server then enters a loop where it continuously receives encrypted data from the client, decrypts it, sanitizes it, and prints it. The server also takes input, encrypts it, and sends it back to the client. This loop continues until no data is received from the client.

The sanitizeRequest() function is used to sanitize or clean up the incoming requests. It removes any characters from the request that are not alphanumeric (letters and numbers), whitespace, or punctuation. This is a common practice in web development to prevent malicious attacks such as SQL injection or Cross-Site Scripting (XSS). The function uses Python's built-in string module to get sets of all ASCII letters, digits, and punctuation. It then creates a regular expression pattern that matches any character not in these sets. The re.sub() function is used to replace all characters matching this pattern with an empty string, effectively removing them from the request. The sanitized request is then returned.

Bob

Bob's application acts like a client in a client-server architecture, using sockets for communication and RSA for encryption. We start by generating a pair of RSA keys - a private key and a public key. The generateRSAKey() function uses the RSA algorithm to generate a 1024-bit private key and then derives the corresponding public key.

Next, a socket is created and connected to the server running on 'localhost' at port 9999. Bob's public key is then sent to the server. This public key will be used by the server to encrypt a session key and send it back to Bob. Bob receives this encrypted session key and uses his private key to decrypt it. This session key is used for further communication between Bob and the server.

Bob's application then enters a loop where it waits for user input, encrypts it using the session key, and sends it to the server. Bob's application then waits for a response from the server, decrypts it using the session key, and prints it. This loop continues until Bob types 'bye'.

The application codes

The following code is for Alice. For Bob and Alice to communicate, Alice first must run her application as it serves as the server that Bob, the client, connects to.

Alice code

```
import socket
from utils import *
from Crypto.PublicKey import RSA
import re
import string
import binascii


def sanitizeRequest(request):
# Allow alphanumeric characters, spaces, and punctuation
```

```
    allowed_chars = string.ascii_letters + string.digits + string.punctuation + ' '
    pattern = f'[^{re.escape(allowed_chars)}]'
    sanitized_request = re.sub(pattern, '', request.decode())
    return sanitized_request


    def Alice():

    session_key = generateSessionKey()
    print(f"Decrypted Session key: {binascii.hexlify(session_key)}\n")

    s = socket.socket()
    print('Socket created')
    s.bind(('localhost', 9999))

    s.listen(1)
    print('Waiting for connection...')

    c, addr = s.accept()
    print('Connected with ', addr)

    # to share session key using RSA
    c_public_key = RSA.import_key(c.recv(1024))
    print(f"Bob public key: {c_public_key.export_key().decode()}\n")
    enc_session_key = encryptSessionKey(session_key, c_public_key)
    print(f"Encrypted session key: {binascii.hexlify(enc_session_key)}\n")
    c.send(bytes(enc_session_key))

    while True:
    # message from Bob to Alice
    encrypted_request = b""
    while True:
    chunk = c.recv(4096)
    encrypted_request += chunk
    if len(chunk) < 1024:
    break
    if not encrypted_request:
    break
    print(f"Encrypted message: {binascii.hexlify(encrypted_request)}\n")
    request = decryptData(encrypted_request, session_key)
    sanitized_request = sanitizeRequest(request)
    print("Bob:", sanitized_request)

    # message from Alice to Bob
    response = bytes(input("Alice -> ").encode())
```

```
        encrypted_response = encryptData(response, session_key)
        print(f"Encrypted response: {binascii.hexlify(encrypted_response)}\n")
        c.send(encrypted_response)

        c.close()


        if __name__ == "__main__":
        Alice()
```

Once the server (Alice Application) is running, the client (Bob Application) can then run establish a connection to enable communication back and forth between them.

Bob code

```
        import socket
        from utils import *
        import binascii


        def Bob():
        private_key, public_key = generateRSAKey()
        print(
        f"Bob public key: {public_key.export_key().decode()}\n Bob private key:
{private_key.export_key().decode()}\n")

        c = socket.socket()
        c.connect(('localhost', 9999))

        # send Bob's public key to receive an encrypted session key from Alice
        c.send(public_key.export_key())
        enc_session_key = c.recv(1024)
        print(f"Encrypted session key: {binascii.hexlify(enc_session_key)}\n")

        # only Bob's private key can decrypt the encrypted session key
        session_key = decryptSessionKey(enc_session_key, private_key)
        print(f"Decrypted Session key: {binascii.hexlify(session_key)}\n")

        # message from Bob to Alice
        request = bytes(input('Bob -> ').encode())
        while request.lower().strip() != b"bye":
        encrypted_request = encryptData(request, session_key)
        print(f"Encrypted message: {binascii.hexlify(encrypted_request)}\n")
        c.send(encrypted_request)
```

```
# message from Alice to Bob
encrypted_response = c.recv(1024)
print(f"Encrypted response: {binascii.hexlify(encrypted_response)}\n")
response = decryptData(encrypted_response, session_key)
print("Alice:", response.decode())

request = bytes(input('Bob -> ').encode())


if __name__ == "__main__":
Bob()
```

Then there are utility methods used by Bob and Alice present in a separate utilities file.

#utilities file

```
from Crypto.Cipher import AES, PKCS1_OAEP
from Crypto.Random import get_random_bytes
from Crypto.PublicKey import RSA
from Crypto import Random


def generateRSAKey():
private_key = RSA.generate(1024)
public_key = private_key.publickey()
return private_key, public_key


def generateSessionKey():
# generating an aes key or a session key
session_key = Random.new().read(AES.block_size)
return session_key


def encryptSessionKey(session_key, c_public_key):
rsa_encrypt = PKCS1_OAEP.new(c_public_key)
enc_session_key = rsa_encrypt.encrypt(session_key)
return enc_session_key


def decryptSessionKey(enc_session_key, c_private_key):
rsa_decrypt = PKCS1_OAEP.new(c_private_key)
session_key = rsa_decrypt.decrypt(enc_session_key)
```

```
    return session_key


    def encryptData(data, session_key):
    pad = 16 - (len(data) % 16)
    data += bytes([pad]) * pad
    cipher_aes = AES.new(session_key, AES.MODE_CBC, session_key)
    ciphertext = cipher_aes.encrypt(data)
    return ciphertext


    def decryptData(ciphertext, session_key):
    cipher_aes = AES.new(session_key, AES.MODE_CBC, session_key)
    plaintext = cipher_aes.decrypt(ciphertext)
    plaintext = plaintext[:-plaintext[-1]]
    return plaintext
```

There is also a python requirement needed for the application to work.

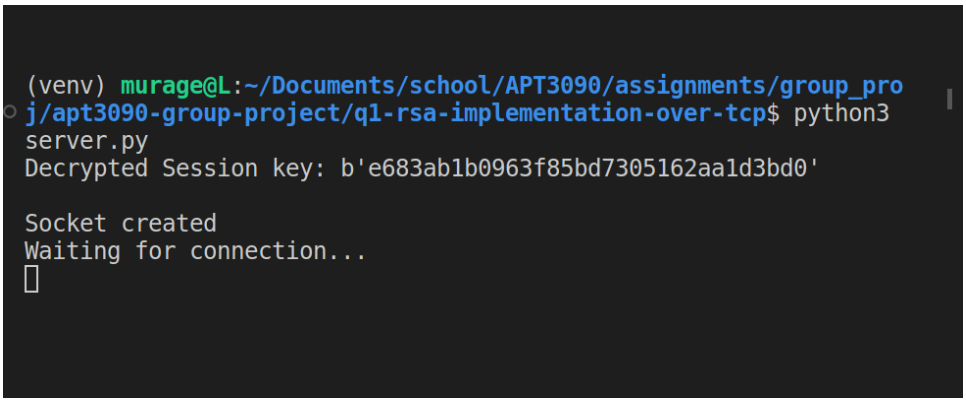
requirements

```
    pycryptodome==3.20.0
```

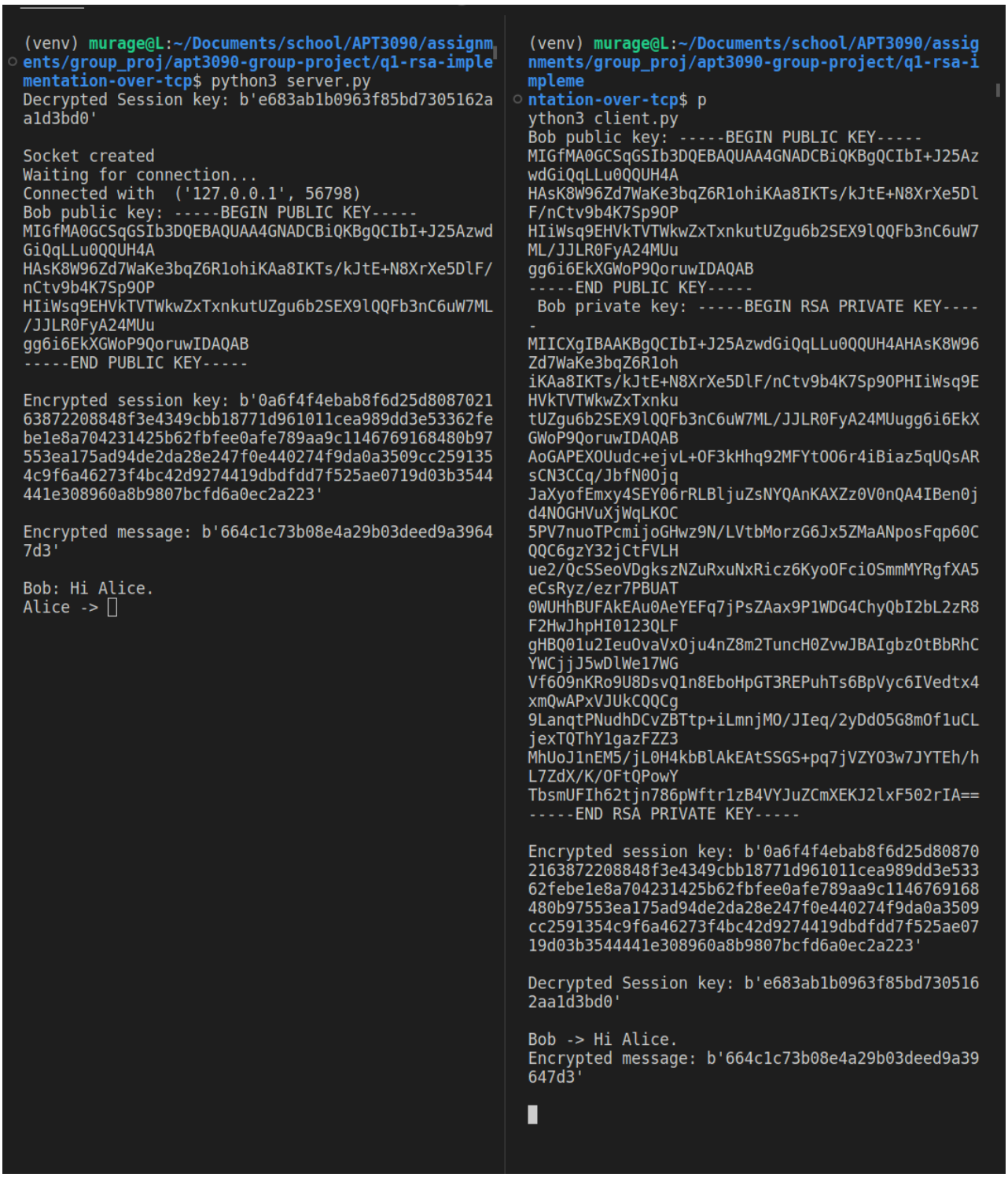
Results

When you run the provided code, the following are the results.

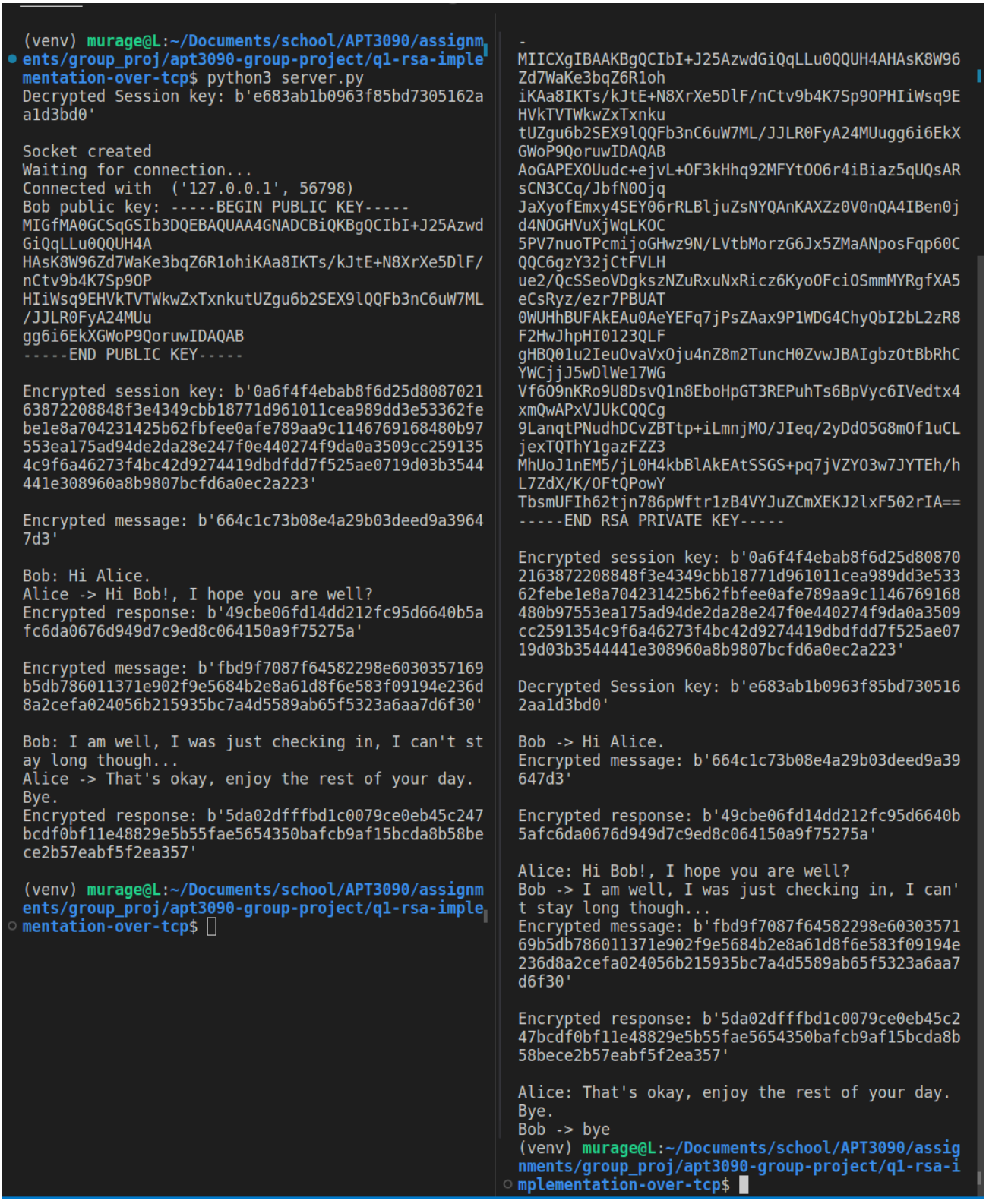
Alice:



Then we run Bob's code and send the initial message to Alice:



Then Alice and Bob can send a few messages to each other. Then finally the connection is terminated when Bob says 'bye':


(venv) murage@L:~/Documents/school/APT3090/assignments/group_proj/apt3090-group-project/q1-rsa-implementation-over-tcp$ python3 server.py
Decrypted Session key: b'e683ab1b0963f85bd7305162aa1d3bd0'
Socket created
Waiting for connection...
Connected with ('127.0.0.1', 56798)
Bob public key: -----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCIbI+J25AzwdGiQqLLu0QQUH4A
HAsK8W96Zd7WaKe3bqZ6R1ohiKAa8IKTs/kJtE+N8XrXe5DlF/nCtv9b4K7Sp9OP
HIiWsq9EHVkTVTWkwZxTxnkutUZgu6b2SEX9lQQFb3nC6uW7ML/JJLR0FyA24MUu
gg6i6EkXGWoP9QoruwIDAQAB
-----END PUBLIC KEY-----
Encrypted session key: b'0a6f4f4ebab8f6d25d8087021638722088848f3e4349cbb18771d961011cea989dd3e53362febe1e8a704231425b62fbfee0afe789aa9c1146769168480b97553ea175ad94de2da28e247f0e440274f9da0a3509cc2591354c9f6a46273f4bc42d9274419dbdfdd7f525ae0719d03b3544441e308960a8b9807bcfd6a0ec2a223'
Encrypted message: b'664c1c73b08e4a29b03deed9a39647d3'
Bob: Hi Alice.
Alice -> Hi Bob!, I hope you are well?
Encrypted response: b'49cbe06fd14dd212fc95d6640b5afc6da0676d949d7c9ed8c064150a9f75275a'
Encrypted message: b'fbd9f7087f64582298e6030357169b5db786011371e902f9e5684b2e8a61d8f6e583f09194e236d8a2cefa024056b215935bc7a4d5589ab65f5323a6aa7d6f30'
Bob: I am well, I was just checking in, I can't stay long though...
Alice -> That's okay, enjoy the rest of your day.
Bye.
Encrypted response: b'5da02dfffbd1c0079ce0eb45c247bcdf0bf11e48829e5b55fae5654350bafcb9af15bcda8b58bece2b57eabf5f2ea357'
(venv) murage@L:~/Documents/school/APT3090/assignments/group_proj/apt3090-group-project/q1-rsa-implementation-over-tcp$
-
MIICXgIBAAKBgQCIbI+J25AzwdGiQqLLu0QQUH4AHAsK8W96Zd7WaKe3bqZ6R1oh
iKAa8IKTs/kJtE+N8XrXe5DlF/nCtv9b4K7Sp9OPHIiWsq9EHVkTVTWkwZxTxnku
tUZgu6b2SEX9lQQFb3nC6uW7ML/JJLR0FyA24MUugg6i6EkXGWoP9QoruwIDAQAB
AoGAPEXOUudc+ejvL+OF3kHhq92MFYtOO6r4iBiaz5qUQsARsCN3CCq/JbfN0Ojq
JaXyofEmxy4SEY06rRLBljuZsNYQAnKAXZz0V0nQA4IBen0jd4NOGHVuXjWqLKOC
5PV7nuoTPcmijoGHwz9N/LVtbMorzG6Jx5ZMaANposFqp60CQQC6gzY32jCtFVLH
ue2/QcSSeoVDgkszNZuRxuNxRicz6KyoOFciOSmmMYRgfXA5eCsRyz/ezr7PBUAT
0WUHhBUFAkEAu0AeYEFq7jPsZAax9P1WDG4ChyQbI2bL2zR8F2HwJhpHI0123QLF
gHBQ01u2IeuOvaVxOju4nZ8m2TuncH0ZvwJBAIgbzOtBbRhCYWCjjJ5wDlWe17WG
Vf6O9nKRo9U8DsvQ1n8EboHpGT3REPuhTs6BpVyc6IVedtx4xmQwAPxVJUkCQQCg
9LanqtPNudhDCvZBTtp+iLmnjMO/JIeq/2yDdO5G8mOf1uCLjexTQThY1gazFZZ3
MhUoJ1nEM5/jL0H4kbBlAkEAtSSGS+pq7jVZYO3w7JYTEh/hL7ZdX/K/OFtQPowY
TbsmUFIh62tjn786pWftr1zB4VYJuZCmXEKJ2lxF502rIA==
-----END RSA PRIVATE KEY-----
Encrypted session key: b'0a6f4f4ebab8f6d25d8087021638722088848f3e4349cbb18771d961011cea989dd3e53362febe1e8a704231425b62fbfee0afe789aa9c1146769168480b97553ea175ad94de2da28e247f0e440274f9da0a3509cc2591354c9f6a46273f4bc42d9274419dbdfdd7f525ae0719d03b3544441e308960a8b9807bcfd6a0ec2a223'
Decrypted Session key: b'e683ab1b0963f85bd7305162aa1d3bd0'
Bob -> Hi Alice.
Encrypted message: b'664c1c73b08e4a29b03deed9a39647d3'
Encrypted response: b'49cbe06fd14dd212fc95d6640b5afc6da0676d949d7c9ed8c064150a9f75275a'
Alice: Hi Bob!, I hope you are well?
Bob -> I am well, I was just checking in, I can't stay long though...
Encrypted message: b'fbd9f7087f64582298e60303571769b5db786011371e902f9e5684b2e8a61d8f6e583f09194e236d8a2cefa024056b215935bc7a4d5589ab65f5323a6aa7d6f30'
Encrypted response: b'5da02dfffbd1c0079ce0eb45c247bcdf0bf11e48829e5b55fae5654350bafcb9af15bcda8b58bece2b57eabf5f2ea357'
Alice: That's okay, enjoy the rest of your day.
Bye.
Bob -> bye
(venv) murage@L:~/Documents/school/APT3090/assignments/group_proj/apt3090-group-project/q1-rsa-implementation-over-tcp$

Drawbacks

While our application works well to illustrate the concept of using RSA to secure communication over TCP, it has a few shortcomings. The first is that communication must be initiated and terminated by only the client (Bob). Ideally, once a connection is established, both the client and the server should be able to send the first message. The second one is that only one can communicate at a time and this communication is limited to a single message at a time. Ideally, the communication should be free form and both should be able to send as many messages as they like in any order.

Conclusion and Recommendations.

The project is a good way to learn about TCP, sockets and RSA. It does a good job of combining the various topics we have covered in the class in a practical and hands-on manner. The only recommendation is that maybe the group could work on making this a more user-friendly application by implementing a GUI for it. This would enable non-technical users to try out their project and give them feedback on how they find using the project.

Credit Card Vault Implementation

Phase 1

Background

Credit Card Vault Project: A merchant needs to store customer's credit card details. You only need to collect credit card information from a customer once, then store the card details in the vault. The next time you want to invoice a customer, you can use the stored card information.

Sensitive, confidential, and public information

Identify sensitive, confidential, and public information.

a) Sensitive: Credit card number, expiration date, CVV code (all encrypted in the database)
b) Confidential: Customer ID number (encrypted in the database)
c) Public: Customer name, email address

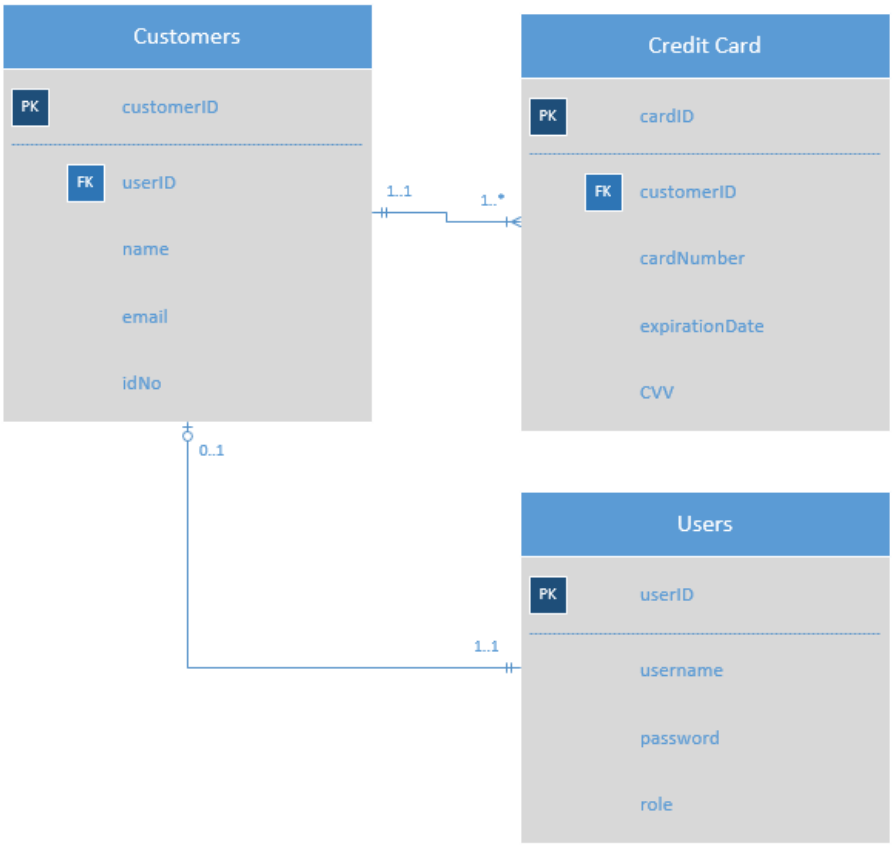
User access levels

Identify users access levels (Who should Select, Insert, Delete, Update)

User Role	Select	Insert	Delete	Update
Customer	✓ (Their own information)	✓ (Their own information)	✗	✗
Admin	✓ (All data)	✓ (Users)	✓(Users)	✓ (Users)

3NF database schema

Provide a schema showing the relationship of tables in 3NF.



Phase 2

Database in 3NF

Create your database as per your 3NF. Here is a snippet of the function that does this.

```python
# Creation of database
def createDatabase():
    conn = sqlite3.connect('creditcard_vault.db')
    c = conn.cursor()

    try:
        c.execute("""
            CREATE TABLE IF NOT EXISTS CUSTOMERS (
                customerID INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT NOT NULL,
                email TEXT UNIQUE,
                idNo INTEGER NOT NULL,
                userID INTEGER,
                FOREIGN KEY (userID) REFERENCES USERS(userID)
            )
        """)
    except sqlite3.OperationalError as e:
        print(f"Error creating table: {e}")

    try:
        c.execute("""
            CREATE TABLE IF NOT EXISTS CREDITCARDS (
                cardID INTEGER PRIMARY KEY AUTOINCREMENT,
                customerID INTEGER NOT NULL,
                cardNumber INTEGER NOT NULL,
                expirationDate TEXT NOT NULL,
                CVV INTEGER NOT NULL,
                FOREIGN KEY (customerID) REFERENCES CUSTOMERS(customerID)
            )
        """)
    except sqlite3.OperationalError as e:
        print(f"Error creating table: {e}")

    try:
        c.execute("""
            CREATE TABLE IF NOT EXISTS USERS (
                userID INTEGER PRIMARY KEY AUTOINCREMENT,
                username TEXT NOT NULL,
                password TEXT NOT NULL,
                role TEXT NOT NULL
            )
        """)
    except sqlite3.OperationalError as e:
        print(f"Error creating table: {e}")
```

The code creates three tables: CUSTOMERS, CREDITCARDS, and USERS. Here's an analysis of their relationships:

CUSTOMERS:

customerID (primary key)

name (not null)

email (text unique)

idNo (not null, encrypted)

userID (foreign key referencing USERS(userID))

CREDITCARDS:

cardID (primary key)

customerID (foreign key referencing CUSTOMERS(customerID))

cardNumber (not null, encrypted)

expirationDate (text not null)

CVV (not null, encrypted)

USERS:

userID (primary key)

username (text not null)

password (text not null, hashed)

role (text not null)

This database structure adheres to 3NF principles as there are no transitive dependencies between attributes. Each table represents a distinct entity, and foreign keys are used to establish relationships.

Encryption & Decryption

Write scripts to insert and retrieve (sensitive, confidential and public information) using AES_ENCRYPT and AES_DECRYPT FUNCTIONS.

These scripts define two functions for encrypting and decrypting sensitive information.

i.encrypt_sensitive_information(plain_text, password):

a. Generate Random Salt. This function first generates a random string called a salt. This adds an extra layer of security by making the encryption unique for each process.
b. Derive Private Key. It uses a password-based key derivation function (KDF) called Scrypt to create a private key from the user's password and the random salt. This KDF makes it computationally expensive to guess the password from the encrypted data.
c. Create Cipher Configuration. The script uses the derived private key and a specific encryption mode (AES-GCM) to set up the encryption process.
d. Encrypt Plain Text. The actual variable (plain text) is converted to bytes and then encrypted using the configured cipher.

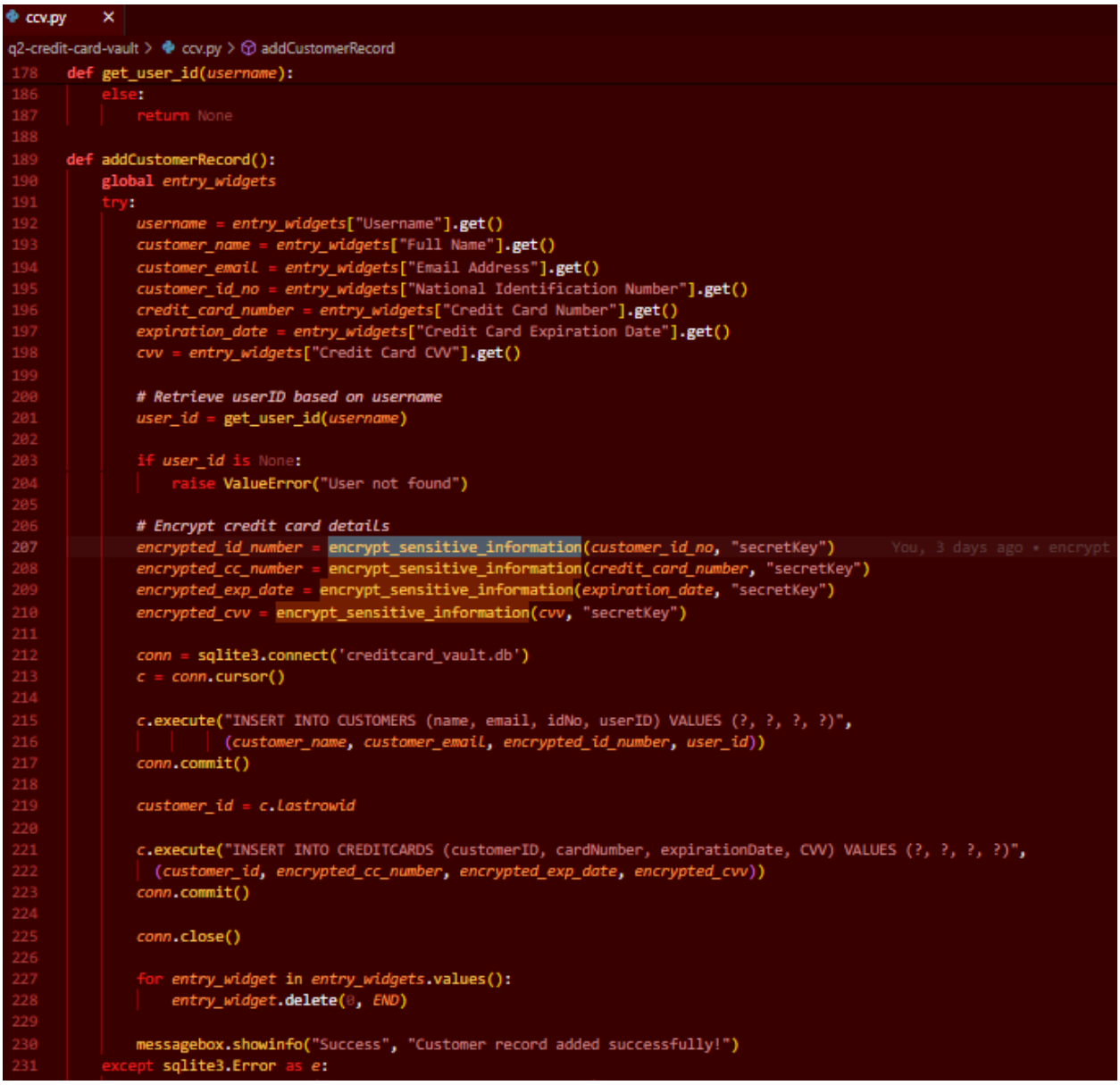
e. Combine and Return. The encrypted data is combined with the salt and a random nonce (used in the encryption process) and encoded into a single string for storage.

ii.decrypt_sensitive_information(encrypted_data, password):

a. Decode Encrypted Data. This function first decodes the received string back into its separate parts: salt, nonce, and encrypted text.
b. Derive Private Key. Similar to the encryption process, it uses the Scrypt KDF with the password and the extracted salt to derive the same private key used for encryption.
c. Create Cipher Configuration. The private key, encryption mode (AES-GCM), and the extracted nonce are used to set up the decryption process.
d. Decrypt Cipher Text. The encrypted text is decrypted using the configured cipher.
e. Return Decrypted Text. Finally, the decrypted data is converted back into a human-readable format (string) and returned.
f. Important points to remember:

These scripts provide a secure way to encrypt and decrypt sensitive information. They use a key which is secret and is imported, not hard coded.

Encrypt function is used to encrypt sensitive information such as customer identification number, credit card number, credit card expiration date and credit card cvv before storing the plain text as cipher text in the database. as shown below.



Decrypt function is used to decrypt sensitive information such as customer identification number, credit card number, credit card expiration date and credit card cvv before displaying them to authenticated users with the right access level as show below.

```python
def showRecordsPage():
    global current_page
    if current_page:
        current_page.pack_forget()

    current_page = Frame(root)
    current_page.pack()

    conn = sqlite3.connect('creditcard_vault.db')
    c = conn.cursor()

    c.execute("""
        SELECT c.customerID, c.name, c.email, c.idNo, cc.cardNumber, cc.expirationDate, cc.CVV
        FROM CUSTOMERS c
        INNER JOIN CREDITCARDS cc ON c.customerID = cc.customerID
    """)
    records = c.fetchall()

    record_listbox = Listbox(current_page, width=200)
    record_listbox.pack(padx=10, pady=10)

    for record in records:
        # Decrypt credit card details if they are encrypted
        decrypted_id_number = record[3]
        decrypted_cc_number = record[4]
        decrypted_exp_date = record[5]
        decrypted_cvv = record[6]

        try:
            decrypted_id_number = decrypt_sensitive_information(record[3], "secretKey")
            decrypted_cc_number = decrypt_sensitive_information(record[4], "secretKey")
            decrypted_exp_date = decrypt_sensitive_information(record[5], "secretKey")
            decrypted_cvv = decrypt_sensitive_information(record[6], "secretKey")
        except Exception as e:
            print(f"Error decrypting record: {e}")

        # Format the record
        formatted_record = f"Customer ID: {record[0]}, Name: {record[1]}, Email: {record[2]}, ID No: {decrypted_id_number}, Card Number: {decrypted_cc_number}, Expiration Date:
        {decrypted_exp_date}, CVV: {decrypted_cvv}"

        # Insert the formatted record into the listbox
        record_listbox.insert(END, formatted_record)

    conn.close()
    hideShowAllCustomerDetailsNavBtn()
    hideAddCustomerDetailsButton()
    showAddCustomerNavigationBtn()
```

Encrypted customer details in the database.

Reset Filters **Records: 1**

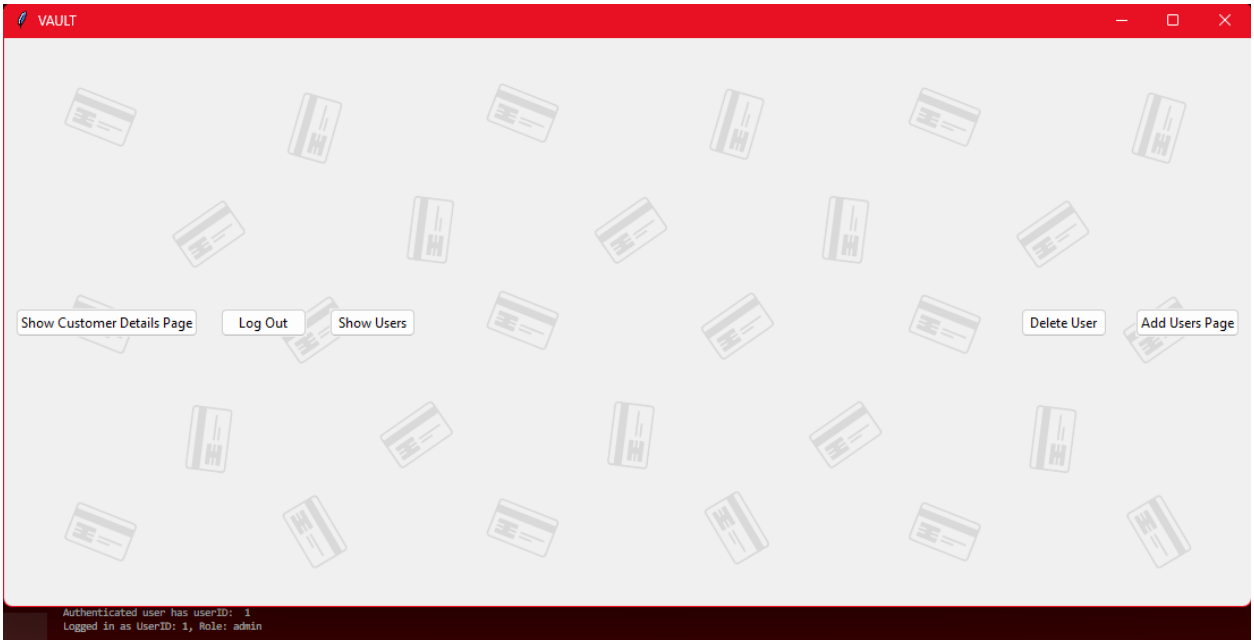
	customerID	name	email	idNo	userID
	Search column...	Search column...	Search column...	Search column...	Search column...
1	1	Alice Doe	adoe@usiu.ac.ke	SJhB9dWfKo17VOeEj...	2

Encrypted credit card details in the database

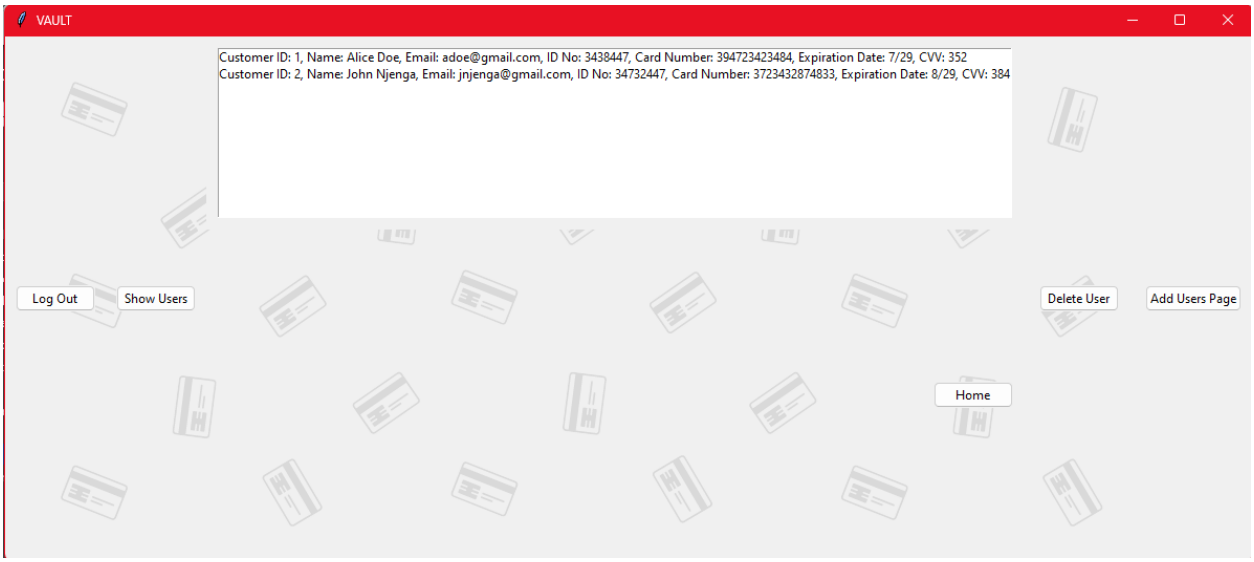
Reset Filters **Records: 1**

	cardID	customerID	cardNumber	expirationDate	CVV
	Search column...	Search column...	Search column...	Search column...	Search column...
1	1	1	BvkvSmvzsaK9n0Mwk...	wx7Mmaxq2wavXE5...	L3qiEGNoCRVN6TvAS...

Only the admin can see decrypted customer sensitive information.

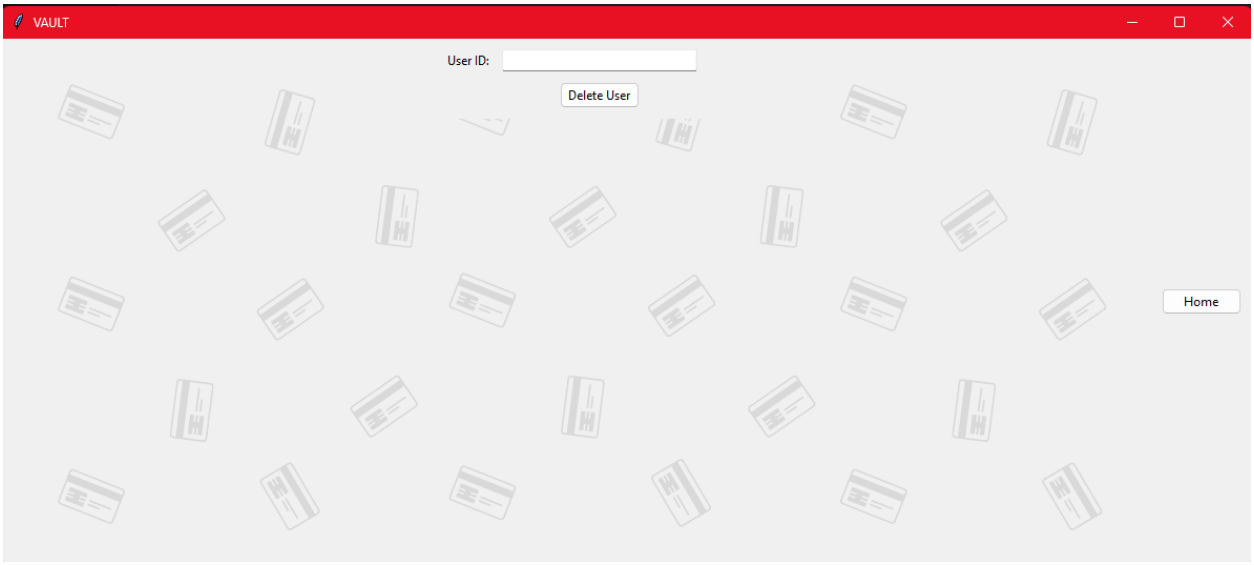


Decrypted customer details.

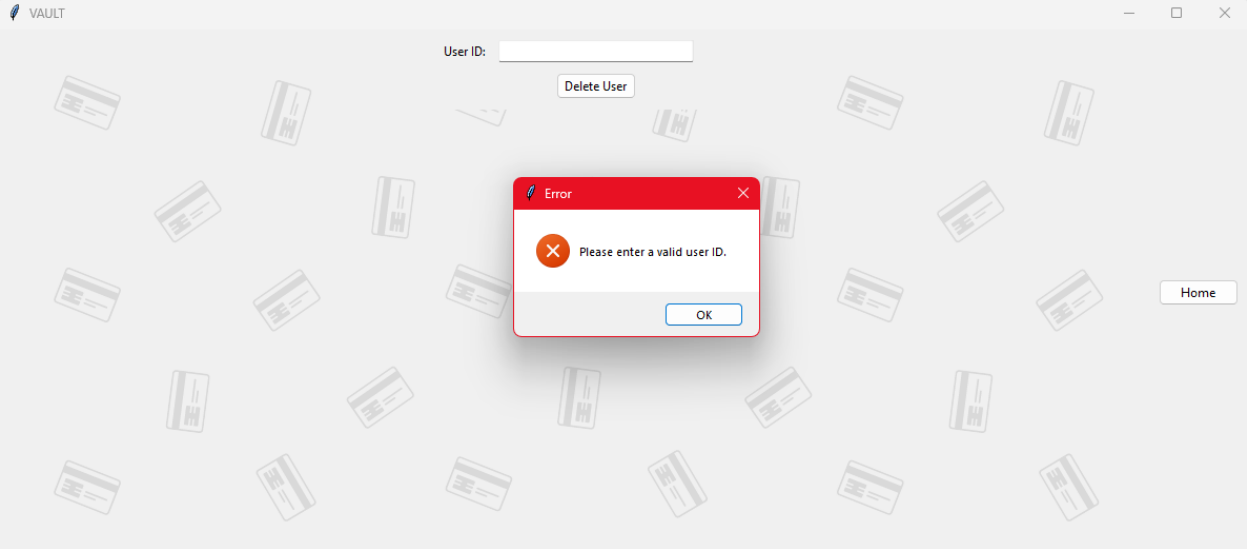


Only the admin can see the current users in the system when they click the “Show Users” button. They cannot see the users’ passwords though.

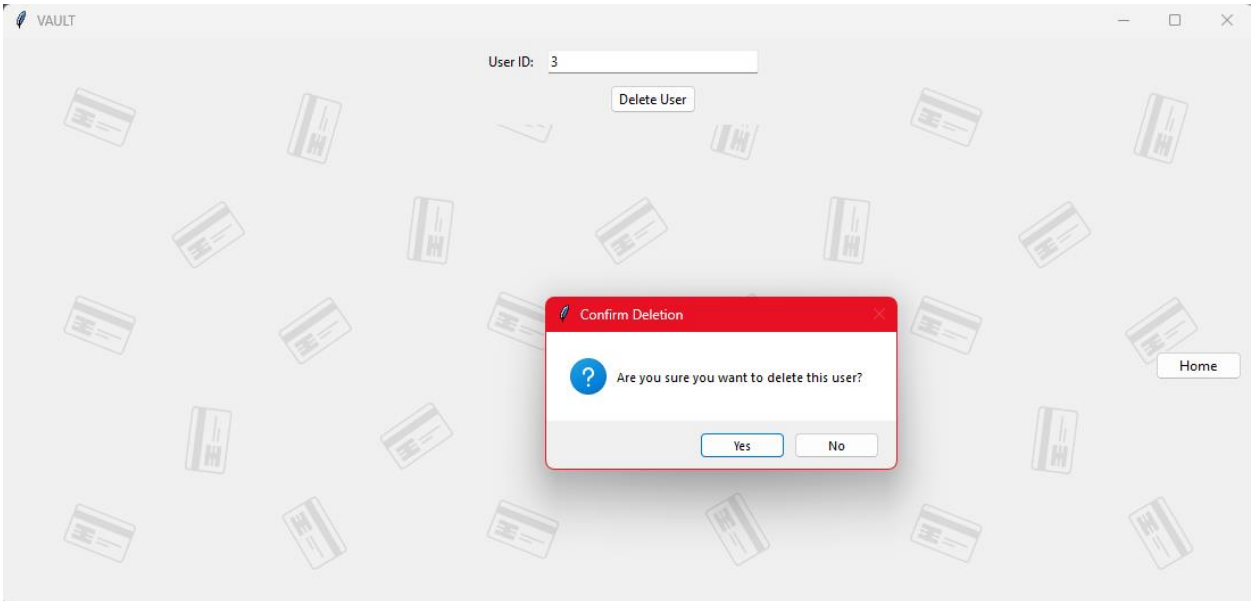
They can delete a user in the delete users page by keying in the user id they want to delete.



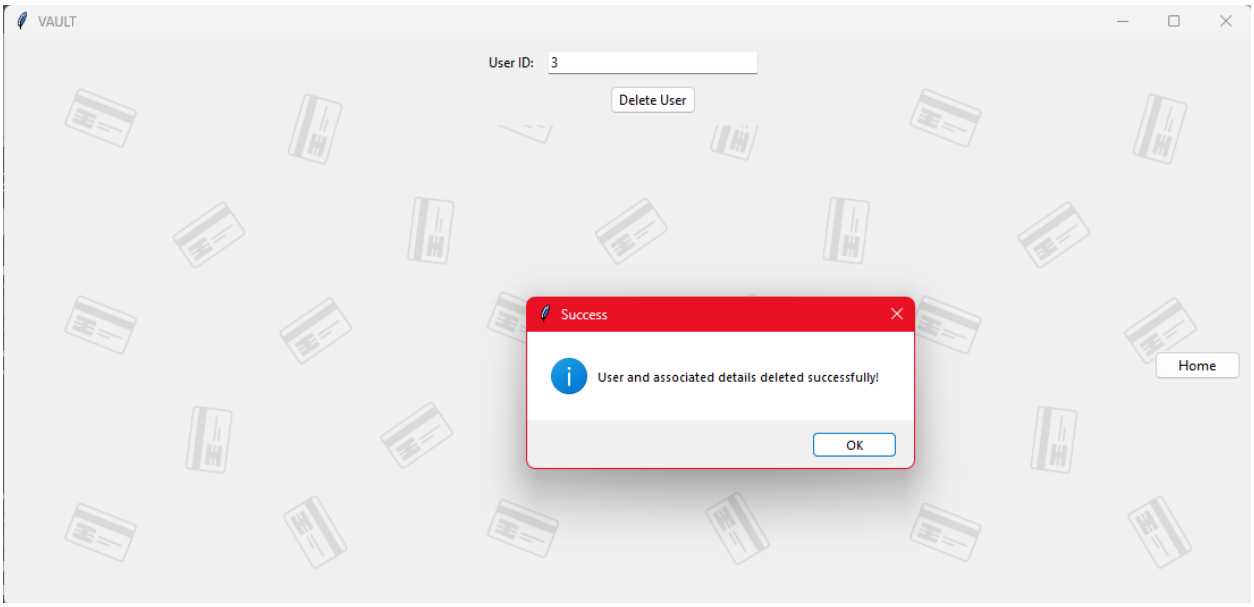
Form validation put in place



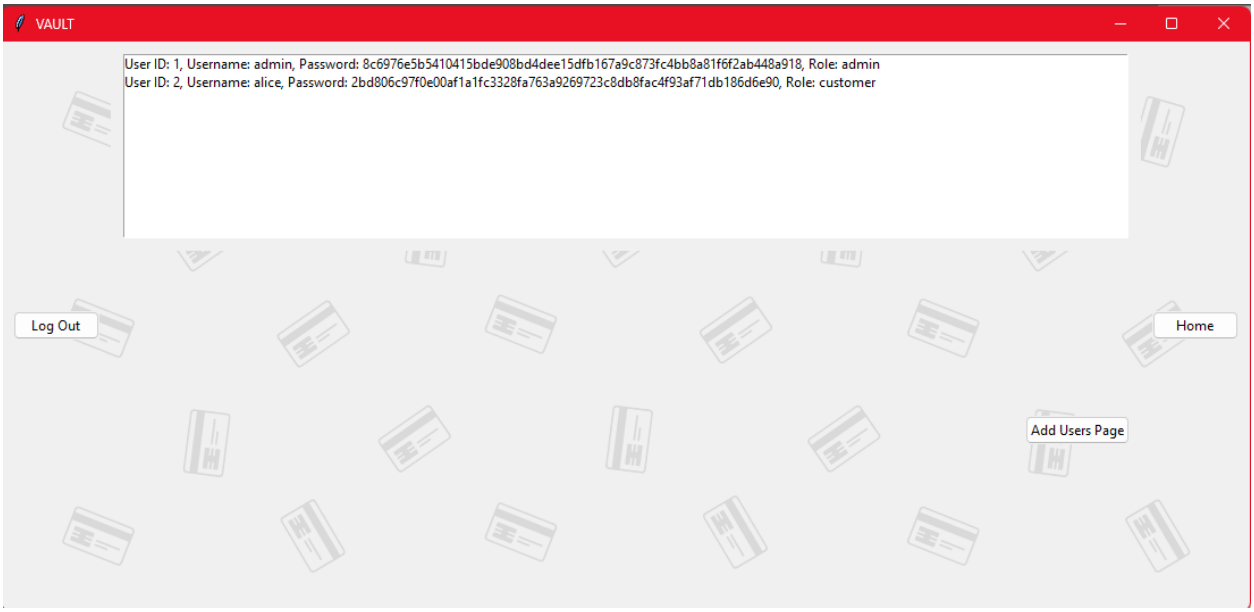
Say the merchant wants to delete user id 3, they simply key in 3 and hit the “Delete User” button. They are presented with a confirmation dialog box as show below in which when they click “Yes”, all the details about that user are deleted across the entire database.



A feedback message is shown to the merchant



Confirmation that user id 3 has been deleted



Delete user page function

ccv.py

q2-credit-card-vault > ccv.py > ...

```python
def delete_user_and_details(user_id):
    conn = sqlite3.connect('creditcard_vault.db')
    c = conn.cursor()

    try:
        # Get customer ID associated with the user
        c.execute("SELECT customerID FROM CUSTOMERS WHERE userID = ?", (user_id,))
        customer_id = c.fetchone()[0]

        # Delete customer's credit card details
        c.execute("DELETE FROM CREDITCARDS WHERE customerID = ?", (customer_id,))

        # Delete customer details
        c.execute("DELETE FROM CUSTOMERS WHERE customerID = ?", (customer_id,))

        # Delete the user
        c.execute("DELETE FROM USERS WHERE userID = ?", (user_id,))

        conn.commit()
        conn.close()
        messagebox.showinfo("Success", "User and associated details deleted successfully!")

        # Redirect to add customer info page after successful deletion
        addCustomerInfoPage()
    except sqlite3.Error as e:
        conn.rollback()
        conn.close()
        messagebox.showerror("Error", f"Error occurred: {e}")

```

ccv.py

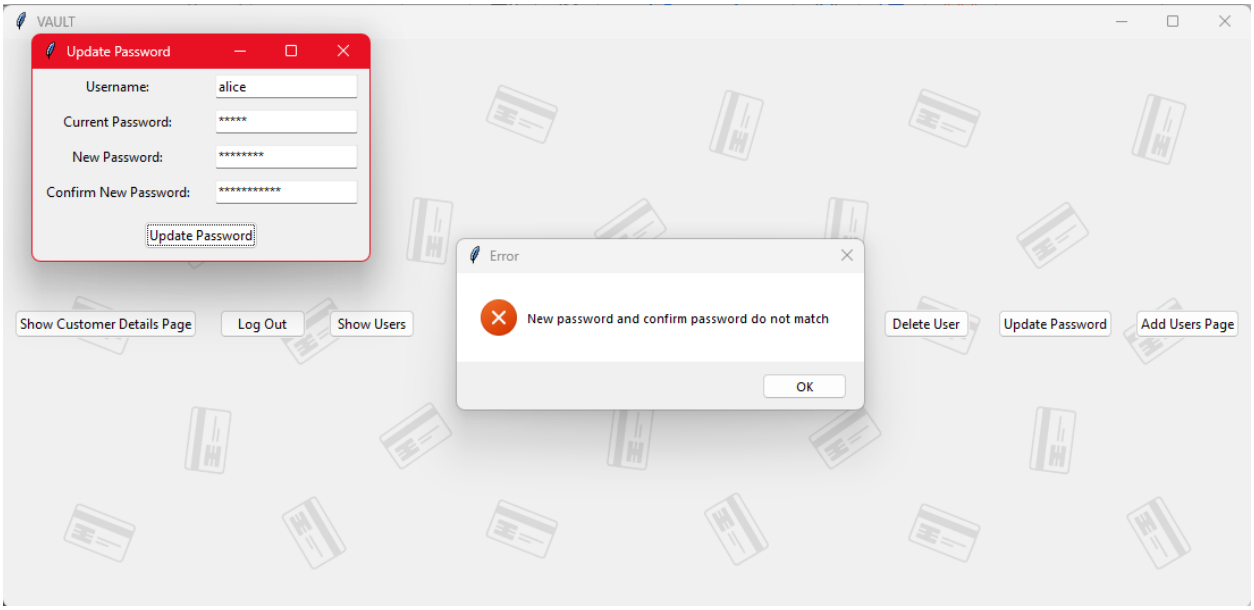
q2-credit-card-vault > ccv.py > deleteUserPage

```python
def deleteUserPage():
    def deleteUser():
        try:
            user_id_str = userIdEntry.get().strip()
            if not user_id_str:
                raise ValueError("User ID is required")

            user_id = int(user_id_str)

            # Check if the user ID exists before attempting deletion
            conn = sqlite3.connect('creditcard_vault.db')
            c = conn.cursor()
            c.execute("SELECT * FROM USERS WHERE userID = ?", (user_id,))
            user = c.fetchone()
            conn.close()

            if user:
                # Confirm deletion
                confirm = messagebox.askyesno("Confirm Deletion", "Are you sure you want to delete this user?")
                if confirm:
                    delete_user_and_details(user_id)
                    # Clear input field after deletion
                    userIdEntry.delete(0, END)
            else:
                # Display error message if user ID does not exist
                messagebox.showerror("Error", "User ID does not exist.")
        except ValueError:
            messagebox.showerror("Error", "Please enter a valid user ID.")

    global current_page
    if current_page:
        current_page.pack_forget()      CG, 5 days ago • Added delete user by ID functionality

    current_page = Frame(root)
    current_page.pack()

    hideAddCustomerDetailsButton()
    hideLogOutBtn()
    hideShowAllCustomerDetailsNavBtn()
    hideAddUsersNavigationBtn()
    hideShowAllUsersButton()
    hide_update_password_button()
    hideDeleteUserButton()
    showAddCustomerNavigationBtn()

    userIdLabel = Label(current_page, text="User ID:")
    userIdLabel.grid(row=0, column=0, pady=(10, 0))
```

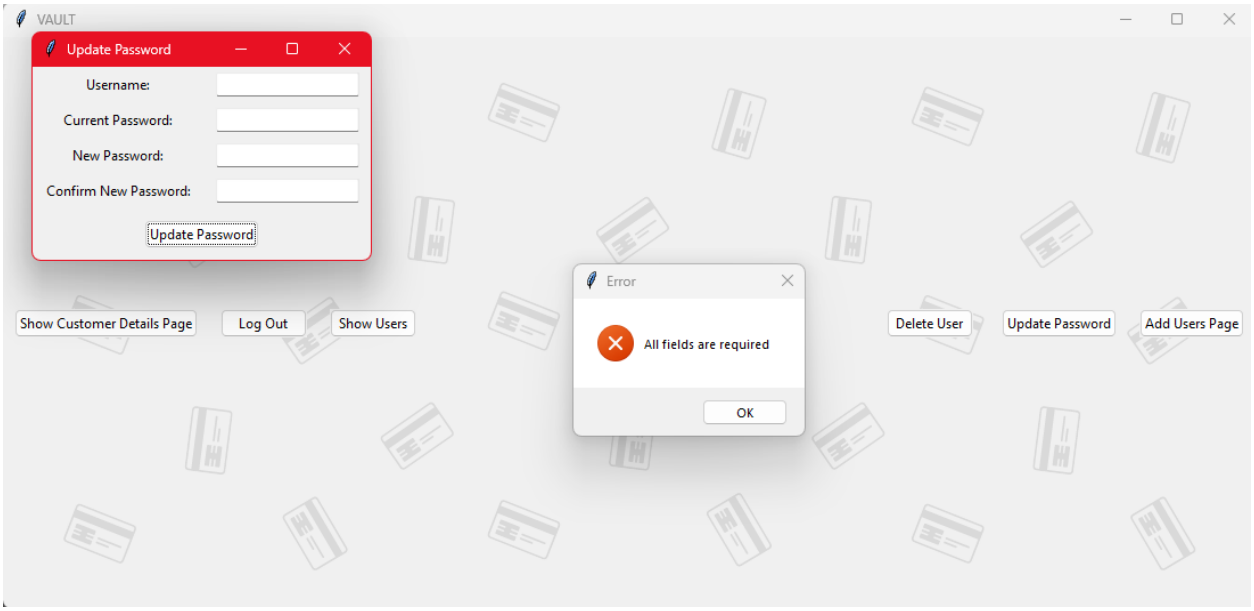
```
820
821     userIdEntry = Entry(current_page, width=30)
822     userIdEntry.grid(row=0, column=1, padx=10, pady=(10, 0))
823
824     deleteBtn = Button(current_page, text="Delete User", command=deleteUser)
825     deleteBtn.grid(row=1, column=1, pady=10)
826
827
828 def showDeleteUserButton():
829     deleteUserButton.pack(side=RIGHT, padx=15, pady=15)
830
831 # Create a button to navigate to the delete user page
832 deleteUserButton = Button(root, text="Delete User", command=deleteUserPage)
833 deleteUserButton.pack(side=RIGHT, padx=15, pady=15)
834
835 # Add a function to hide the delete user page
836 def hideDeleteUserButton():
837     deleteUserButton.pack_forget()
```

The merchant can also update the password of a user by clicking the “Update Password” button in the dashboard.

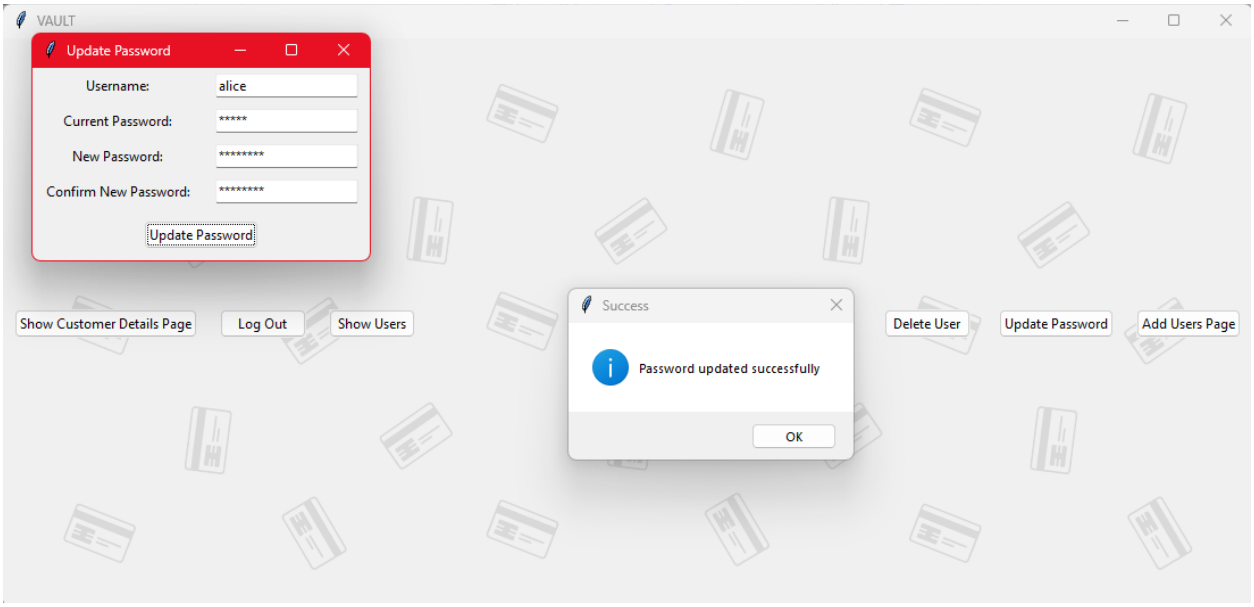
Error handling



Form validation



Feeback message. Alice now has a new password which is different from the old one.



Update password function

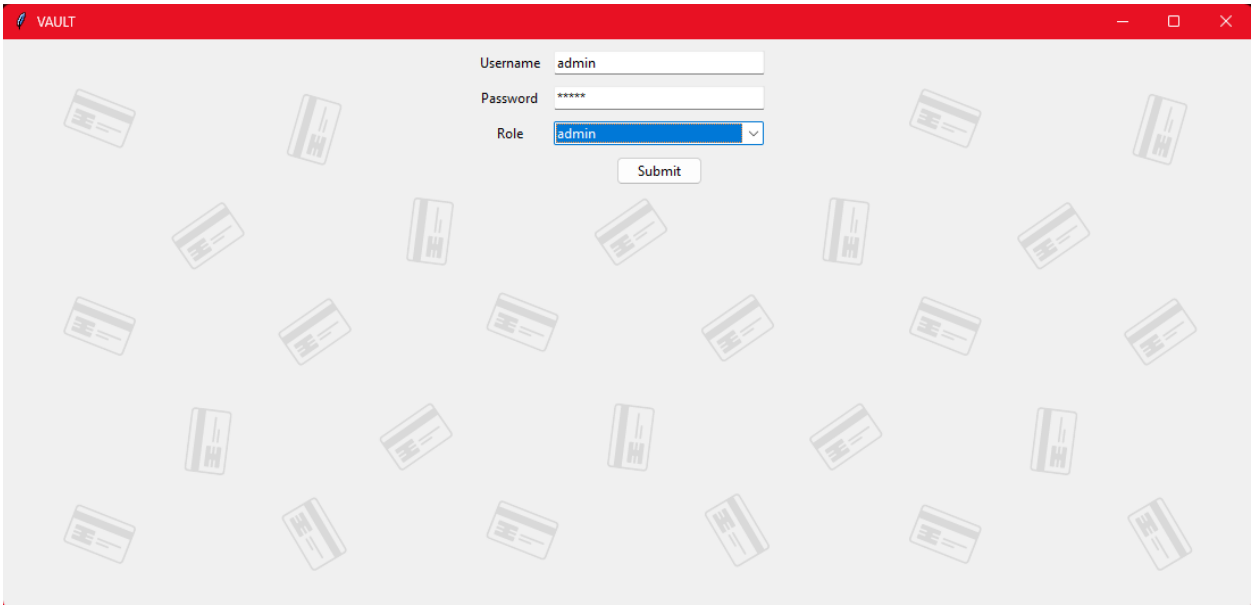
ccv.py

q2-credit-card-vault > ccv.py > ...

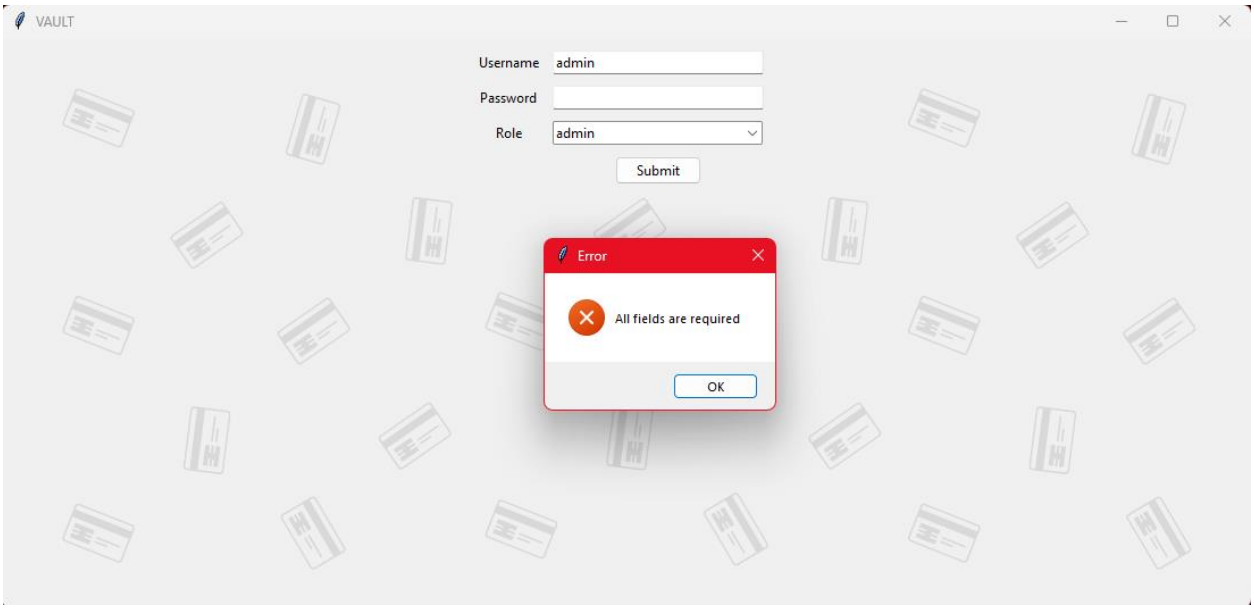
```python
# Function to update user's password in the database        CG, 5 days ago • Admin can update user password
def update_password(username, current_password, new_password):
    conn = sqlite3.connect('creditcard_vault.db')
    c = conn.cursor()

    # Hash the current password for comparison
    hashed_current_password = hash_password(current_password)

    # Check if the username and current password match
    c.execute("SELECT * FROM USERS WHERE username = ? AND password = ?", (username, hashed_current_password))
    user = c.fetchone()

    if user:
        # Hash the new password before updating
        hashed_new_password = hash_password(new_password)

        # Update the password in the database
        c.execute("UPDATE USERS SET password = ? WHERE username = ?", (hashed_new_password, username))
        conn.commit()
        conn.close()
        return True
    else:
        conn.close()
        return False

# UI for password update form
def password_update_page():
    def update_password_action():
        # Get values from the input fields
        username = username_entry.get().strip()
        current_password = current_password_entry.get().strip()
        new_password = new_password_entry.get().strip()
        confirm_new_password = confirm_new_password_entry.get().strip()

        # Validate input fields
        if not all([username, current_password, new_password, confirm_new_password]):
            messagebox.showerror("Error", "All fields are required")
            return

        if new_password != confirm_new_password:
            messagebox.showerror("Error", "New password and confirm password do not match")
            return

        # Check if the username and current password match before updating
        if update_password(username, current_password, new_password):
            messagebox.showinfo("Success", "Password updated successfully")
            # Clear input fields
```

```
            username_entry.delete(0, END)
            current_password_entry.delete(0, END)
            new_password_entry.delete(0, END)
            confirm_new_password_entry.delete(0, END)
        else:
            messagebox.showerror("Error", "Incorrect username or current password")

    # Create the password update form
    password_update_window = Toplevel(root)
    password_update_window.title("Update Password")

    # Username field
    username_label = Label(password_update_window, text="Username:")
    username_label.grid(row=0, column=0, padx=10, pady=5)
    username_entry = Entry(password_update_window)
    username_entry.grid(row=0, column=1, padx=10, pady=5)

    # Current password field
    current_password_label = Label(password_update_window, text="Current Password:")
    current_password_label.grid(row=1, column=0, padx=10, pady=5)
    current_password_entry = Entry(password_update_window, show="*")
    current_password_entry.grid(row=1, column=1, padx=10, pady=5)

    # New password field
    new_password_label = Label(password_update_window, text="New Password:")
    new_password_label.grid(row=2, column=0, padx=10, pady=5)
    new_password_entry = Entry(password_update_window, show="*")
    new_password_entry.grid(row=2, column=1, padx=10, pady=5)

    # Confirm new password field
    confirm_new_password_label = Label(password_update_window, text="Confirm New Password:")
    confirm_new_password_label.grid(row=3, column=0, padx=10, pady=5)
    confirm_new_password_entry = Entry(password_update_window, show="*")
    confirm_new_password_entry.grid(row=3, column=1, padx=10, pady=5)

    # Button to update password
    update_button = Button(password_update_window, text="Update Password", command=update_password_action)
    update_button.grid(row=4, columnspan=2, padx=10, pady=10)

# Function to show password update page
def show_update_password_button():
    password_update_button.config(command=password_update_page)
    password_update_button.pack(side=RIGHT, padx=10, pady=15)

# Create a button to show the password update page
password_update_button = Button(root, text="Update Password", command=show_update_password_button)

# Function to hide the update password button
def hide_update_password_button():
    password_update_button.pack_forget()
```

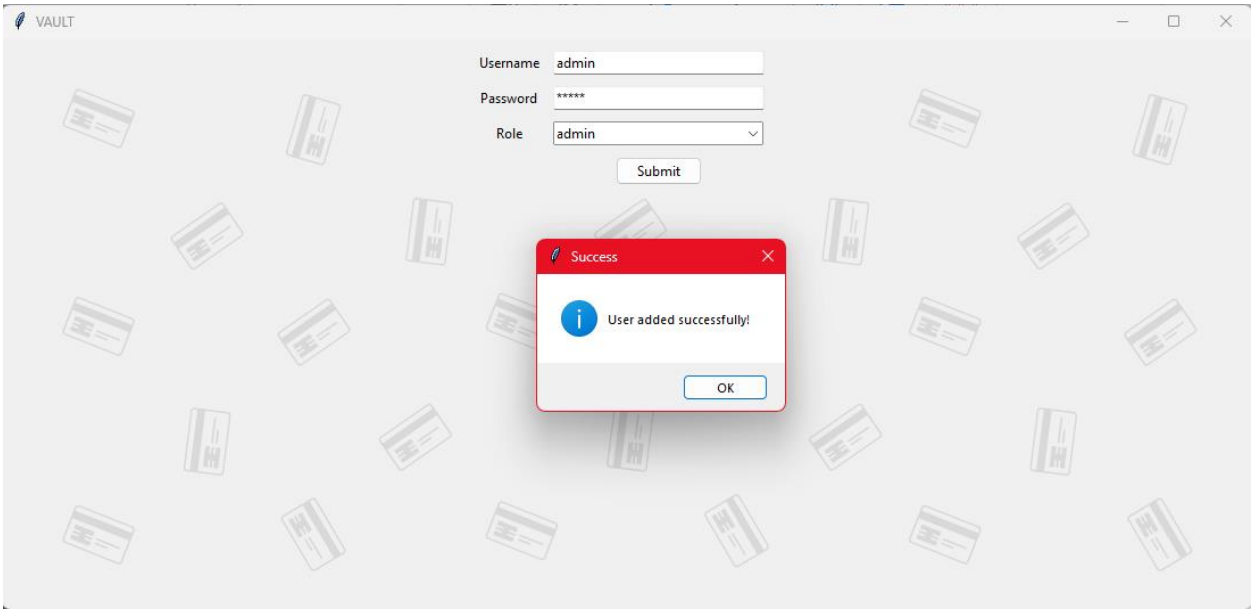
A merchant signs up as an admin by filling out their details in the sign up form



Form validation put in place



Feedback message put in place



Add User Page function

ccv.py

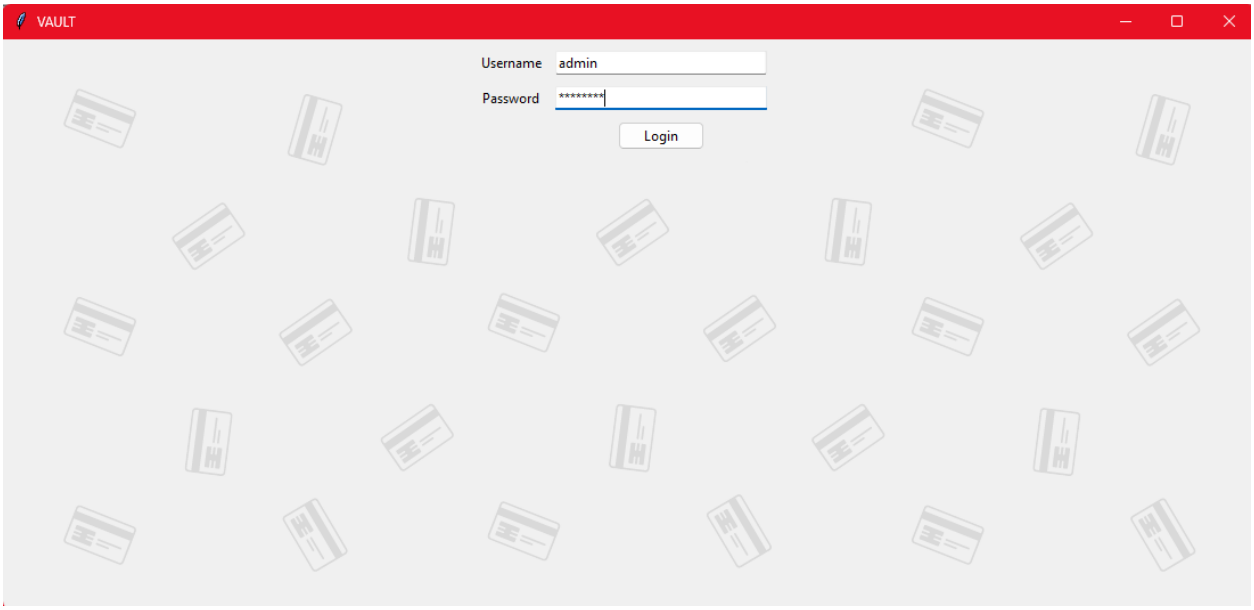
q2-credit-card-vault > ccv.py > ...

```python
def addUser(username, password, role):
    try:
        # Hash the password before storing
        hashed_password = hash_password(password)

        conn = sqlite3.connect('creditcard_vault.db')
        c = conn.cursor()
        c.execute("INSERT INTO USERS (username, password, role) VALUES (?, ?, ?)", (username, hashed_password, role))
        conn.commit()
        conn.close()
        messagebox.showinfo("Success", "User added successfully!")
    except sqlite3.Error as e:
        messagebox.showerror("Error", f"Error occurred: {e}")

def hash_password(password):
    # Hash the password using SHA-256
    hashed_password = hashlib.sha256(password.encode()).hexdigest()
    return hashed_password

def addUserPage():
    def submitUser():
        username = getUsername.get().strip()
        password = getPassword.get().strip()
        role = getRole.get().strip()
        if not (username and password and role):
            messagebox.showerror("Error", "All fields are required")
            return
        addUser(username, password, role)
        loginPage()

    global current_page
    if current_page:
        current_page.pack_forget()

    current_page = Frame(root)
    current_page.pack()

    hideAddCustomerDetailsButton()
    hideLogOutBtn()
    hideShowAllCustomerDetailsNavigationButton()
    hideShowAllCustomerDetailsNavigationButton()
    hideAddCustomersNavigationButton()
    hideShowAllUsersButton()
    hideShowAllCustomerDetailsNavBtn()
    hide_update_password_button()
    hideDeleteUserButton()

```

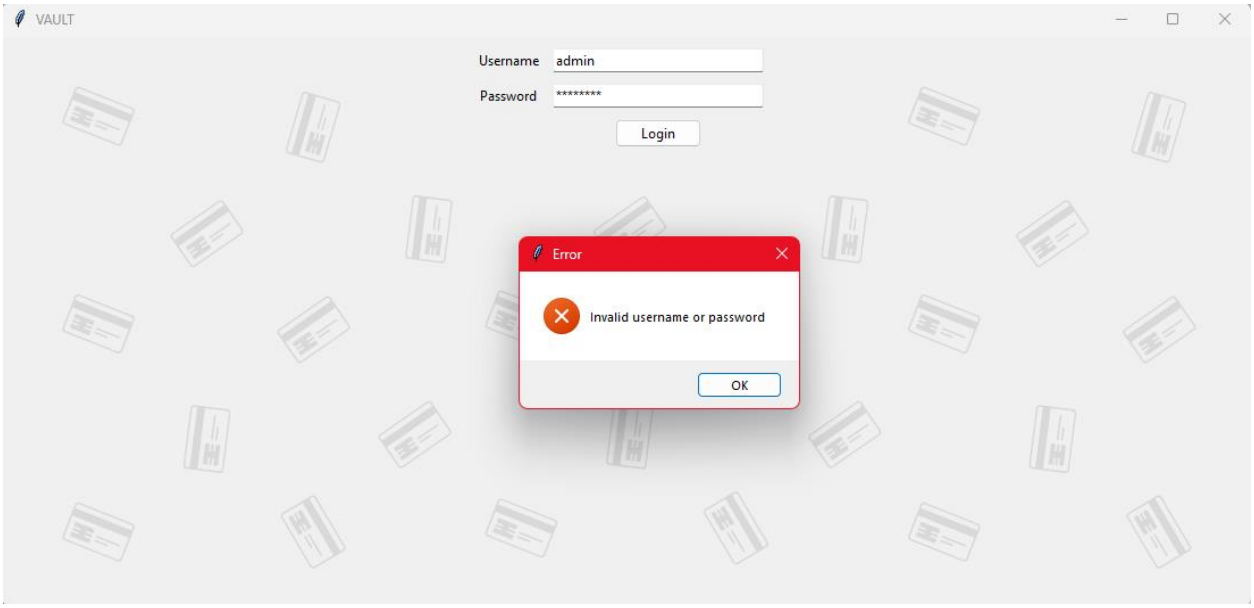
\

```
    getUsernameLabel = Label(current_page, text="Username")
    getUsernameLabel.grid(row=0, column=0, pady=(10, 0))
    getPasswordLabel = Label(current_page, text="Password")
    getPasswordLabel.grid(row=1, column=0, pady=(10, 0))
    getRoleLabel = Label(current_page, text="Role")
    getRoleLabel.grid(row=2, column=0, pady=(10, 0))

    getUsername = Entry(current_page, width=30)
    getUsername.grid(row=0, column=1, padx=10, pady=(10, 0))
    getPassword = Entry(current_page, width=30, show="*")
    getPassword.grid(row=1, column=1, padx=10, pady=(10, 0))
    # Combobox for role selection
    role_options = ["admin", "customer"]
    getRole = ttk.Combobox(current_page, values=role_options, width=27, state="readonly")  # Set state to "readonly"
    getRole.grid(row=2, column=1, padx=10, pady=(10, 0))


    submitBtn = Button(current_page, text="Submit", command=submitUser)
    submitBtn.grid(row=3, column=1, pady=10)
    # hideShowAllUsersButton()
```

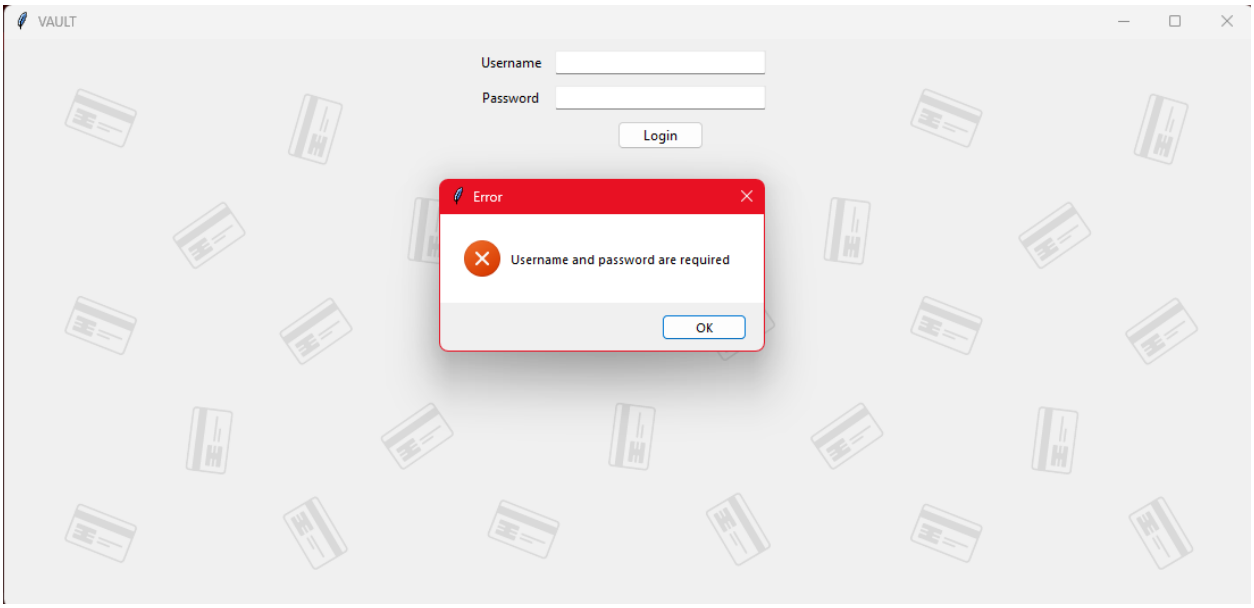
They are then taken to the login page to get authenticated



Error handling put in place



Form validation put in place



Login Page function

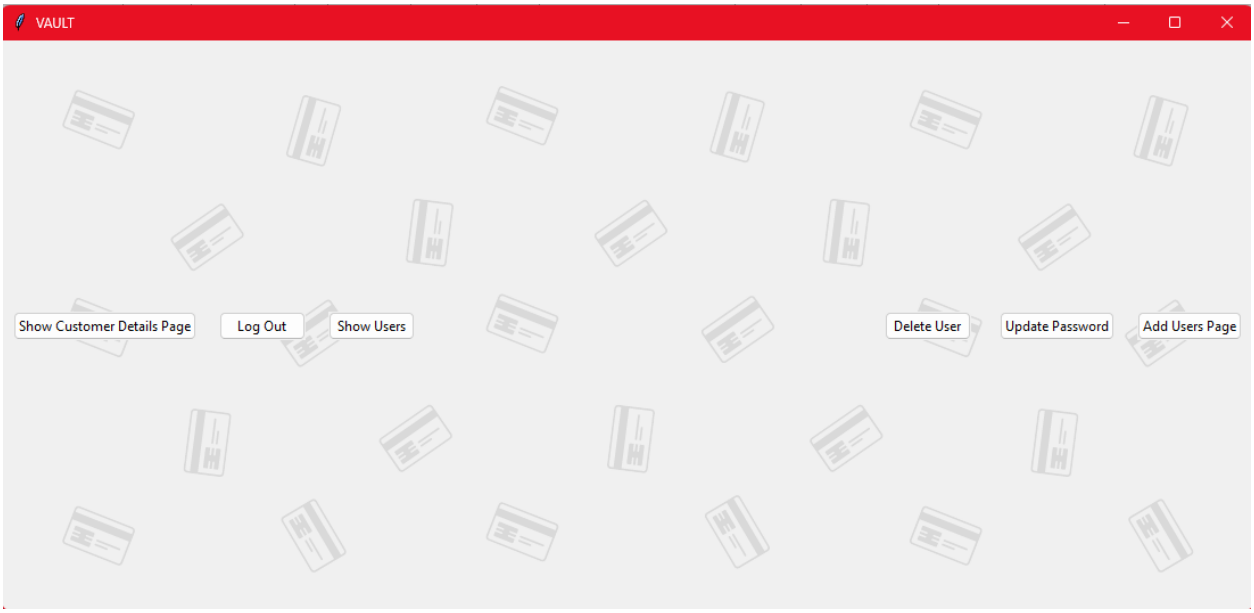
ccv.py M

q2-credit-card-vault > ccv.py > loginPage

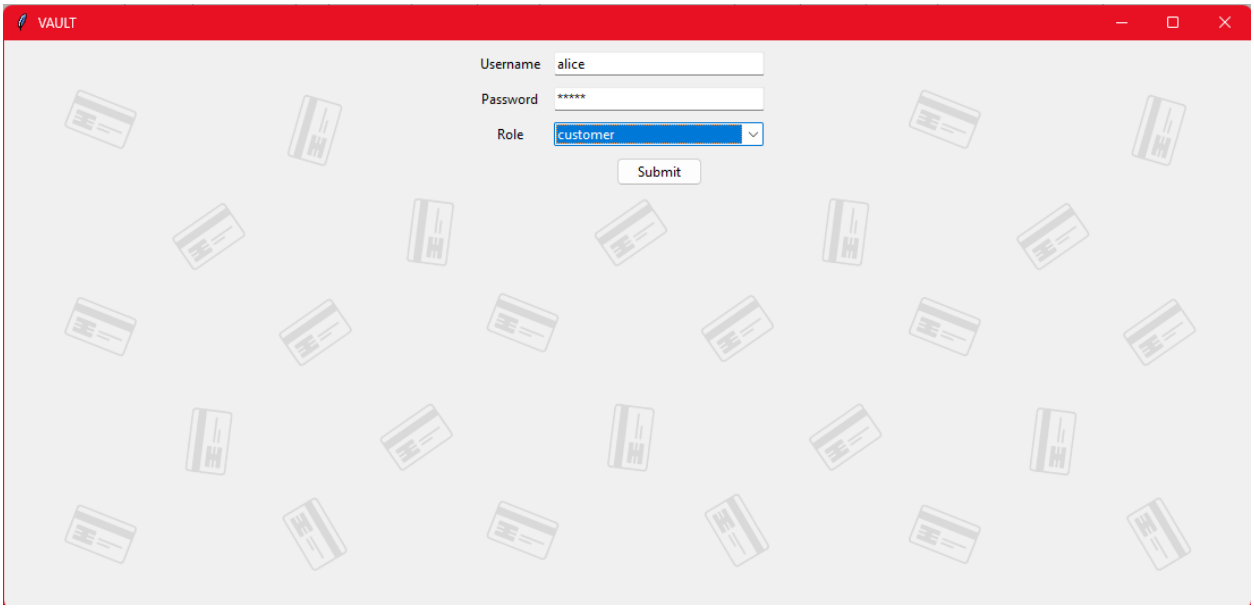
```python
def loginPage():      You, 2 weeks ago • add auth, SELECT and INSERT
    def authenticateUser():
        global authenticated_user, customerUserID
        username = getUsername.get().strip()
        password = getPassword.get().strip()

        if not (username and password):
            messagebox.showerror("Error", "Username and password are required")
            return

        # Hash the provided password
        hashed_password = hash_password(password)

        try:
            conn = sqlite3.connect('creditcard_vault.db')
            c = conn.cursor()

            c.execute("SELECT * FROM USERS WHERE username = ?", (username,))
            user = c.fetchone()

            if user:
                # Retrieve the hashed password stored in the database
                stored_password = user[2]  # Assuming password is stored at index 2
                # Compare the hashed password from the database with the provided hashed password
                if hashed_password == stored_password:
                    authenticated_user = user[0]  # Set authenticated_user to userID
                    print("Authenticated user has userID: ", authenticated_user)

                    # Retrieve user role
                    user_role = user[3]

                    # Print out who is logged in based on their role and user ID
                    print(f"Logged in as UserID: {authenticated_user}, Role: {user_role}")

                    # Check if the authenticated user has already added their customer details
                    c.execute("SELECT COUNT(*) FROM CUSTOMERS WHERE userID = ?", (authenticated_user,))
                    customer_details_count = c.fetchone()[0]

                    if customer_details_count > 0:
                        # If customer details exist, show the customer details page
                        showCustomerDetailsPage()
                    else:
                        # If no customer details exist, show the add customer details page
                        addCustomerInfoPage()
                    return
            messagebox.showerror("Error", "Invalid username or password")
        except sqlite3.Error as e:
```

```
            messagebox.showerror("Error", f"Database error: {e}")
        finally:
            conn.close()

    global current_page
    if current_page:
        current_page.pack_forget()

    current_page = Frame(root)
    current_page.pack()

    hideAddCustomersNavigationButton()
    hideAddCustomerDetailsButton()
    hideLogOutBtn()
    hideShowAllCustomerDetailsNavBtn()
    hideAddUsersNavigationBtn()
    hideShowAllUsersButton()
    hide_update_password_button()
    hideDeleteUserButton()

    getUsernameLabel = Label(current_page, text="Username")
    getUsernameLabel.grid(row=0, column=0, pady=(10, 0))
    getPasswordLabel = Label(current_page, text="Password")
    getPasswordLabel.grid(row=1, column=0, pady=(10, 0))

    getUsername = Entry(current_page, width=30)
    getUsername.grid(row=0, column=1, padx=10, pady=(10, 0))
    getPassword = Entry(current_page, width=30, show="*")
    getPassword.grid(row=1, column=1, padx=10, pady=(10, 0))

    LoginBtn = Button(current_page, text="Login", command=authenticateUser)
    LoginBtn.grid(row=2, column=1, pady=10)
```

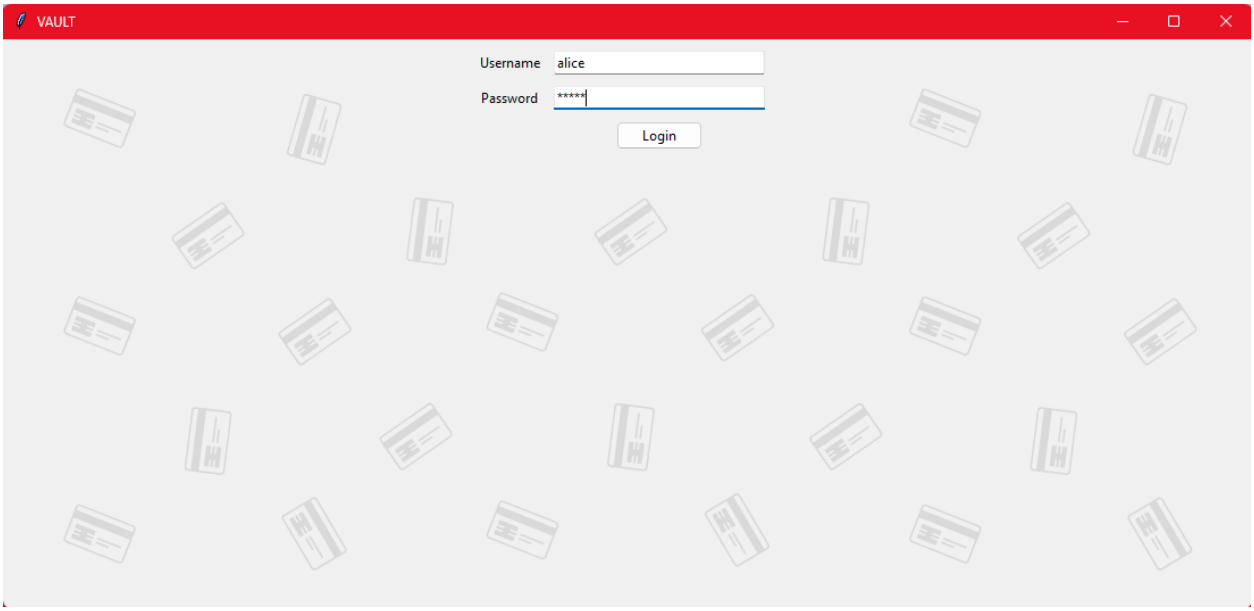
After the merchant logs in, they are taken to their dashboard



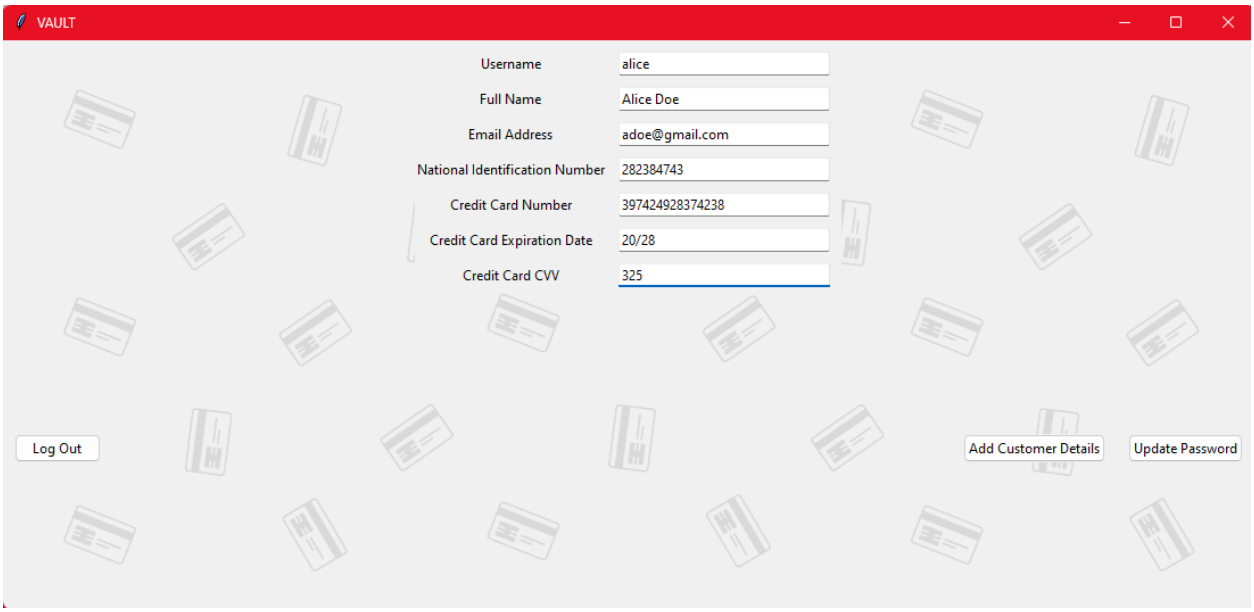
They can add both admin users and customer users. Below is them adding a customer user



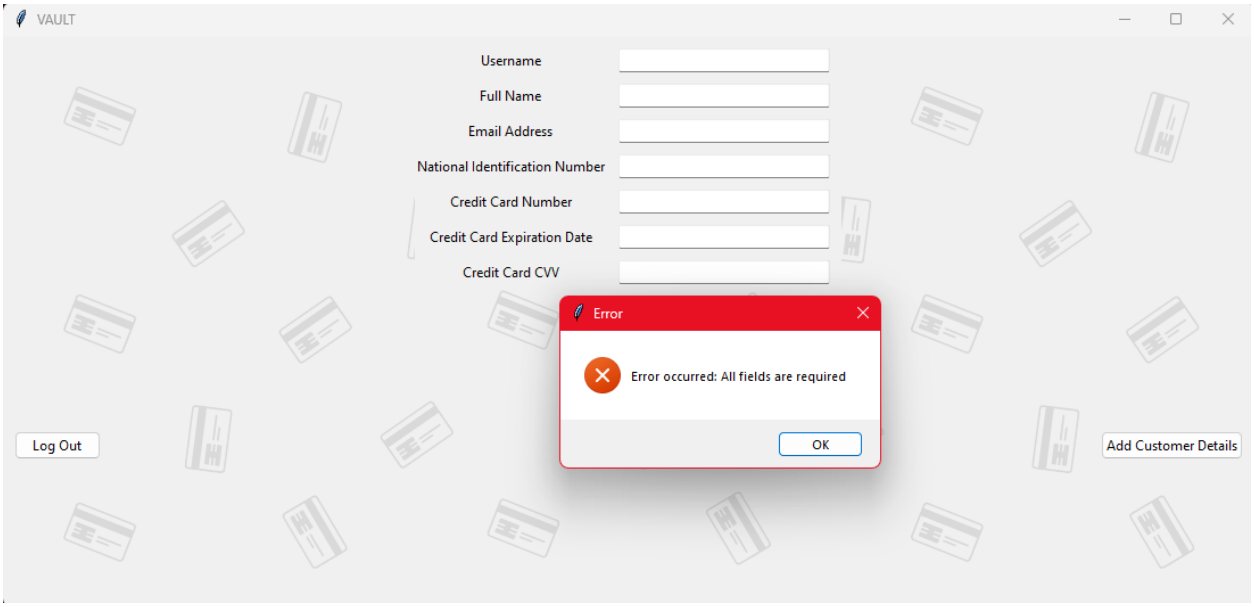
The customer log in with the credentials the merchant created and assigned to them.



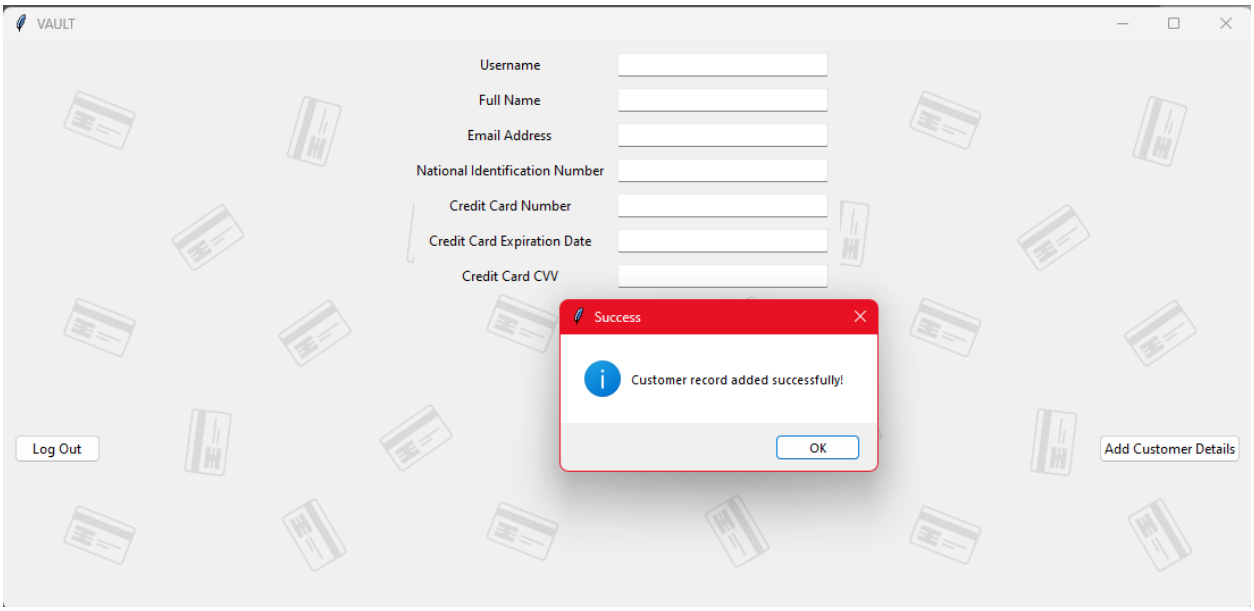
They are prompted to add their own details.



Form validation put in place



Feedback message put in place


VAULT
Username
Full Name
Email Address
National Identification Number
Credit Card Number
Credit Card Expiration Date
Credit Card CVV
Success
Customer record added successfully!
OK
Log Out
Add Customer Details

```python
def define_widgets(root):
    entry_widgets = {}
    labels = [
        "Username", "Full Name", "Email Address", "National Identification Number",
        "Credit Card Number", "Credit Card Expiration Date", "Credit Card CVV"
    ]
    for i, label in enumerate(labels):
        entry = Entry(root, width=30)
        entry.grid(row=i, column=1, padx=10, pady=(10, 0))
        entry_widgets[label] = entry

        label_widget = Label(root, text=label)
        label_widget.grid(row=i, column=0, pady=(10, 0))

    return entry_widgets

```

ccv.py M

q2-credit-card-vault > ccv.py > addCustomerInfoPage

```python
def addCustomerInfoPage():      You, last week • make it such that if authenticated
    global current_page, entry_widgets
    if current_page:
        current_page.pack_forget()

    current_page = Frame(root)
    current_page.pack()

    #entry_widgets = define_widgets(current_page)
    #showAddCustomerDetailsBtn()
    showAllCustomerDetailsButton()
    showAddUsersNavigationBtn()
    showLogOutBtn()
    show_update_password_button()
    showDeleteUserButton()

    # Check if the authenticated user is a customer
    global authenticated_user
    if authenticated_user:
        conn = sqlite3.connect('creditcard_vault.db')
        c = conn.cursor()
        c.execute("SELECT role FROM USERS WHERE userID = ?", (authenticated_user,))
        role = c.fetchone()[0]
        conn.close()

        # If the role is 'customer', hide the "Add Users Page" button
        if role == 'customer':
            entry_widgets = define_widgets(current_page)
            showAddCustomerDetailsBtn()
            hideDeleteUserButton()
            hideAddUsersNavigationBtn()
            hideShowAllUsersButton()
            hideShowAllCustomerDetailsNavBtn()
        else:
            showAllUsersButton()
```

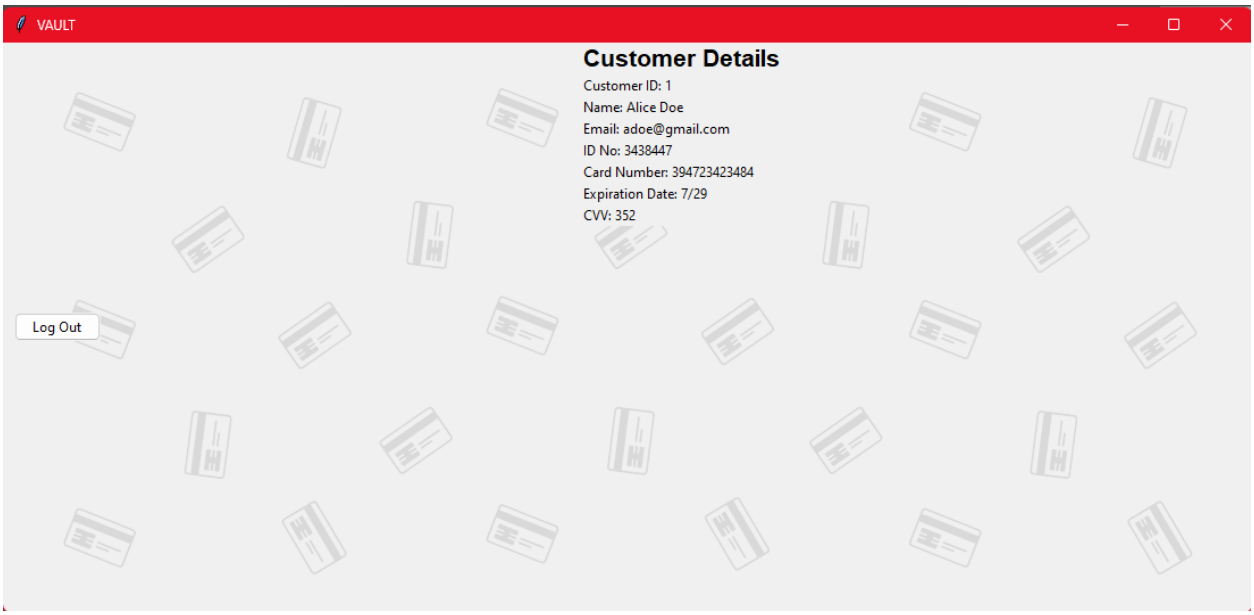
```
204  def addCustomerRecord():
205      global entry_widgets
206      try:
207          username = entry_widgets["Username"].get().strip()
208          customer_name = entry_widgets["Full Name"].get().strip()
209          customer_email = entry_widgets["Email Address"].get().strip()
210          customer_id_no = entry_widgets["National Identification Number"].get().strip()
211          credit_card_number = entry_widgets["Credit Card Number"].get().strip()
212          expiration_date = entry_widgets["Credit Card Expiration Date"].get().strip()
213          cvv = entry_widgets["Credit Card CVV"].get().strip()
214
215          # Form validation
216          if not all([username, customer_name, customer_email, customer_id_no, credit_card_number, expiration_date, cvv]):
217              raise ValueError("All fields are required")
218
219          # Retrieve userID based on username
220          user_id = get_user_id(username)
221
222          if user_id is None:
223              raise ValueError("User not found")
224
225          # Encrypt credit card details
226          encrypted_id_number = encrypt_sensitive_information(customer_id_no, "secretKey")
227          encrypted_cc_number = encrypt_sensitive_information(credit_card_number, "secretKey")
228          encrypted_exp_date = encrypt_sensitive_information(expiration_date, "secretKey")
229          encrypted_cvv = encrypt_sensitive_information(cvv, "secretKey")
230
231          conn = sqlite3.connect('creditcard_vault.db')
232          c = conn.cursor()
233
234          c.execute("INSERT INTO CUSTOMERS (name, email, idNo, userID) VALUES (?, ?, ?, ?)",
235                    (customer_name, customer_email, encrypted_id_number, user_id))
236          conn.commit()
237
238          customer_id = c.lastrowid
239
240          c.execute("INSERT INTO CREDITCARDS (customerID, cardNumber, expirationDate, CVV) VALUES (?, ?, ?, ?)",
241            (customer_id, encrypted_cc_number, encrypted_exp_date, encrypted_cvv))
242          conn.commit()
243
244          conn.close()
245
246          for entry_widget in entry_widgets.values():
247              entry_widget.delete(0, END)
248
249          messagebox.showinfo("Success", "Customer record added successfully!")
250
```

```
251          # Redirect to show customer details page after adding customer details
252          showCustomerDetailsPage()
253
254      except (sqlite3.Error, ValueError) as e:
255          messagebox.showerror("Error", f"Error occurred: {e}")
256
```

```python
def showCustomerDetailsPage():
    global authenticated_user, current_page
    conn = sqlite3.connect('creditcard_vault.db')
    c = conn.cursor()

    try:
        print("Authenticated user ID:", authenticated_user)

        # Retrieve customer ID based on user ID
        c.execute("SELECT customerID FROM CUSTOMERS WHERE userID = ?", (authenticated_user,))
        customer_id = c.fetchone()[0]

        # Query customer details based on the retrieved customer ID
        c.execute("SELECT * FROM CustomerCreditCardView WHERE customerID = ?", (customer_id,))
        customer_details = c.fetchone()

        # Clear current page if it exists
        if current_page:
            current_page.pack_forget()

        # Display customer details
        if customer_details:
            current_page = Frame(root)
            current_page.pack()

            customer_id = customer_details[0]
            name = customer_details[1]
            email = customer_details[2]
            id_no = customer_details[3]
            card_number = customer_details[4]
            expiration_date = customer_details[5]
            cvv = customer_details[6]

            # Decrypt sensitive information
            try:
                id_no = decrypt_sensitive_information(id_no, "secretKey")
                card_number = decrypt_sensitive_information(card_number, "secretKey")
                expiration_date = decrypt_sensitive_information(expiration_date, "secretKey")
                cvv = decrypt_sensitive_information(cvv, "secretKey")
            except Exception as e:
                print("Error decrypting customer details:", e)

            # Labels to display customer details
            Label(current_page, text="Customer Details", font=("Helvetica", 16, "bold")).grid(row=0, columnspan=2)
            Label(current_page, text=f"Customer ID: {customer_id}").grid(row=1, column=0, sticky="w")
            Label(current_page, text=f"Name: {name}").grid(row=2, column=0, sticky="w")
            Label(current_page, text=f"Email: {email}").grid(row=3, column=0, sticky="w")
```

```
            Label(current_page, text=f"ID No: {id_no}").grid(row=4, column=0, sticky="w")
            Label(current_page, text=f"Card Number: {card_number}").grid(row=5, column=0, sticky="w")
            Label(current_page, text=f"Expiration Date: {expiration_date}").grid(row=6, column=0, sticky="w")
            Label(current_page, text=f"CVV: {cvv}").grid(row=7, column=0, sticky="w")

            # Hide the form for adding customer details
            hideAddCustomerDetailsButton()

            # Hide all other buttons except "Logout"
            hideAddUsersNavigationBtn()
            hideAddCustomersNavigationButton()
            hideShowAllCustomerDetailsNavBtn()
            hideShowAllUsersButton()
            hideDeleteUserButton()
            showLogOutBtn()
            hide_update_password_button()

        else:
            print("No customer details found for the authenticated user.")

            # Clear current page if it exists
            if current_page:
                current_page.pack_forget()

            # Display message
            current_page = Frame(root)
            current_page.pack()

            # Display message
            Label(current_page, text="No customer details found.").grid(row=0, columnspan=2)

    except sqlite3.OperationalError as e:
        print(f"Error querying customer details: {e}")

    conn.close()
```

The customer is then taken to a page that only shows their own details and not anyone else's.



When they log out and log back in, they can only see their details.

Password hashing

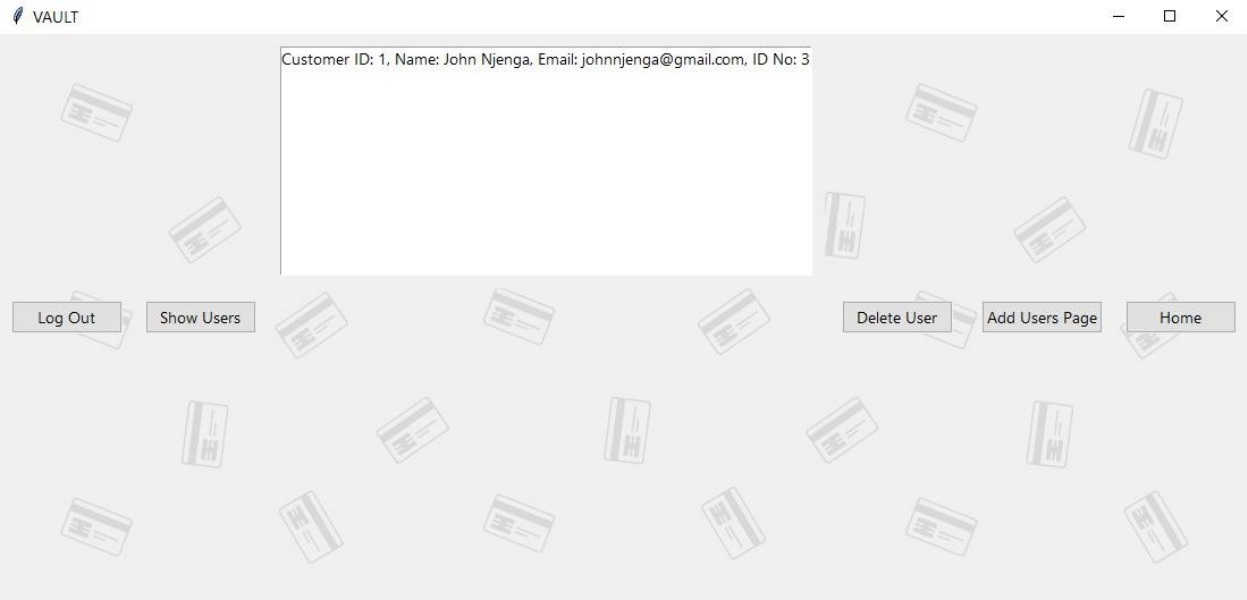
The application can be entered using a username and password for different type of users. The password should be hashed using SHA-2.

It allows for managing customer records, including credit card details, and user authentication with different roles. To enter the application using a username and password for different types of users, the following functions are called:

1. User Authentication. Users would enter their username and password on the login page.
2. Upon clicking the "Login" button, the authenticateUser() function is called. Inside this function, the provided password is hashed using the SHA-256 algorithm (hash_password() function). The hashed password is then compared with the hashed password stored in the database for the corresponding username. If the passwords match, the user is authenticated, and their role is retrieved from the database.
3. User Roles. Based on the user's role retrieved from the database, different functionalities are made available. There are two roles defined, "admin" and "customer". If the user role is "admin", they have access to add customer details, view all customer details, add users, and view all users. If the user role is "customer", they have access to add customer details and view their own customer details.
4. Password Hashing. When a user creates an account (using the "Add Users Page"), their password is hashed using the SHA-256 algorithm before being stored in the database. This is done in the addUser() function.
5. User Registration. To register a new user, an admin user would access the "Add Users Page" and enter the username, password, and role for the new user. Upon submission, the user's details are added to the database.
6. User Interface. The GUI allows users to navigate between different pages based on their role and perform actions such as adding customer details, viewing customer details, adding users, and viewing all users.
7. Overall, the application provides secure authentication using hashed passwords and role-based access control to ensure that users can only access functionalities appropriate for their role.

Views

There were three views that were created. One view was for showing all Customer records. Only the admin has the privilege of using this view to keep tabs on the current user information in the system.

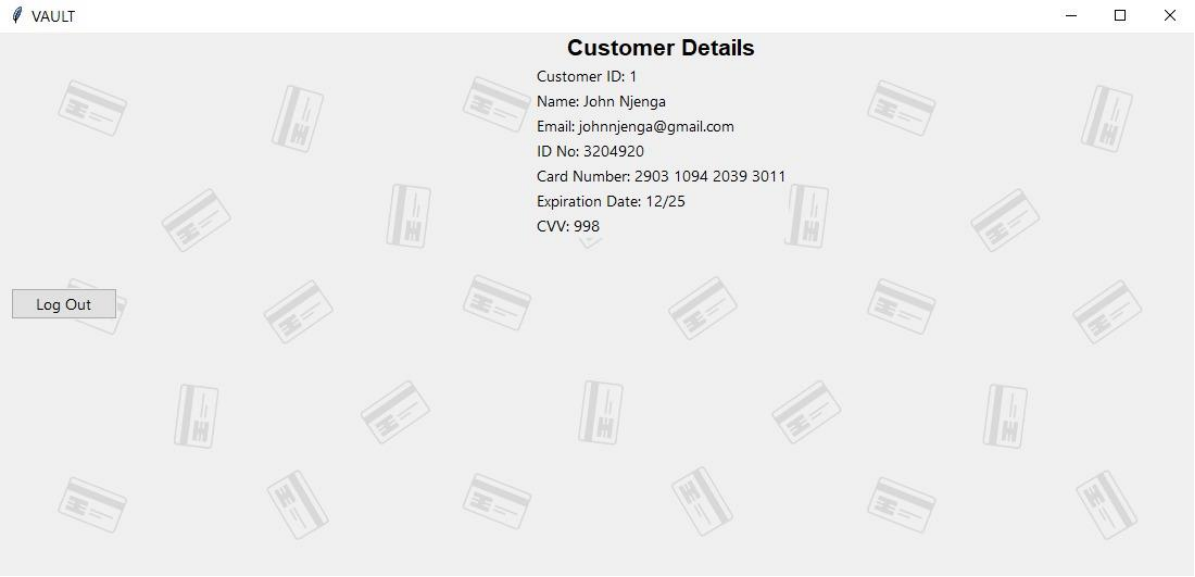


From the image above, only one customer has registered their customer details.

The other view was for viewing all users registered in the system, which was also a task that only the admin could do.

The password is hashed because it is sensitive information that only the user should know.

And finally, the last view was to show an individual customer's details. Upon log in by a customer, they are redirected to a page that shows only their specific customer and credit card details. In this case, John logged in and was directed to this page



Authentication

The application should allow different types of authenticated users to access the data base based on their level of privileges.

1. User Roles. The application defines two user roles: "admin" and "customer". These roles are stored in the database along with user information. The role of each user determines their level of privileges within the application.
2. User Authentication. When a user logs in, the application authenticates them by checking their username and hashed password against the records stored in the database. If the authentication is successful, the application retrieves the user's role from the database.
3. Role-Based Access Control (RBAC). Based on the user's role, the application enables or disables certain functionalities. For example, an "admin" user has access to functionalities such as adding customer details, viewing all customer details, adding users, and viewing all users. A "customer" user, on the other hand, has access to functionalities limited to adding their own customer details and viewing their own customer details. Implementation, the application includes logic to conditionally display or hide buttons and features based on the user's role. For example, the "Add Users Page" button is displayed only for "admin" users. The "Show Customer Details Page" button is displayed for both "admin" and "customer" users, but it provides different functionalities based on the user's role. Similarly, certain buttons for administrative tasks are hidden for "customer" users to prevent unauthorized access.
4. Database Queries. When interacting with the database, the application executes queries that are tailored to the user's role and permissions. For example, when fetching customer records, the application ensures that "customer" users can only retrieve their own records, while "admin" users can retrieve all records. Similarly, when adding new records or users, the application validates the user's role and permissions to prevent unauthorized actions.

5. Overall, by implementing role-based access control, the application ensures that each authenticated user can only access functionalities and data that are appropriate for their role, thereby enhancing security and data protection.

GUI Implementation

The following documentation will help one understand three essential modules used — tkinter for the graphical user interface, PIL for image processing, and os for file operations.

Tkinter

The tkinter module helps design the application's front-end for a better graphical user interface for the Card Vault. It includes a wide variety of user-centric widgets like buttons, labels, and entry fields to interact with the user.

Importing tkinter

```
import tkinter.messagebox as messagebox
from tkinter import *
from tkinter import ttk
from tkinter.ttk import *
```

Creating the GUI window

```
root = Tk()
root.title("VAULT")
root.geometry("1100x500")
```

Loading the background of the GUI window

```
# Load background image
current_dir = os.path.dirname(os.path.realpath(__file__))  # Get the
current directory
image_path = os.path.join(current_dir, "images", "vaultBg.png")
background_image = Image.open(image_path)
background_image_tk = ImageTk.PhotoImage(background_image)

# Create a label to hold the background image
background_label = Label(root, image=background_image_tk)
background_label.place(x=0, y=0, relwidth=1, relheight=1)
```

Adding widgets

```
      # UI
      def define_widgets(root):
          entry_widgets = {}
          labels = [
              "Username", "Full Name", "Email Address", "National Identification
Number",
              "Credit Card Number", "Credit Card Expiration Date", "Credit Card
CVV"
          ]
          for i, label in enumerate(labels):
              entry = Entry(root, width=30)
              entry.grid(row=i, column=1, padx=10, pady=(10, 0))
              entry_widgets[label] = entry

              label_widget = Label(root, text=label)
              label_widget.grid(row=i, column=0, pady=(10, 0))

          return entry_widgets
```

Running the window

```
      root.mainloop()
```

PIL Module

This module is employed to process images using the Card Vault application. It allows you to perform operations on images and manipulate them by re-sizing and saving.

Importing PIL

```
      from PIL import Image, ImageTk
```

Loading and displaying images

```
      # Load background image
      current_dir = os.path.dirname(os.path.realpath(__file__))  # Get the
current directory
      image_path = os.path.join(current_dir, "images", "vaultBg.png")
      background_image = Image.open(image_path)
      background_image_tk = ImageTk.PhotoImage(background_image)
```

OS Module

This module is employed to perform file and directory operations. This module allows you to navigate the operating system, create directories, and perform file operations.

Importing os

```
import os
```

Creating Directories

```
# Load background image
current_dir = os.path.dirname(os.path.realpath(__file__))  # Get the current directory
image_path = os.path.join(current_dir, "images", "vaultBg.png")
```

Listing Files in a Directory

```
background_image = Image.open(image_path)
```

Conclusion

In conclusion, the primary functionalities of tkinter, PIL, and os modules in Python scripts are to facilitate the creation of a user interface, image processing, and file management.

Including a background image in your tkinter GUI is a great way to enhance the application's looks and appeal professionals to the users. With the combination of the PIL module's vast image management capabilities and tkinter's strong control over widget placemats, one can comfortably place images on the GUI. You can also adjust the widgets' behavior on the canvas to place them on the image as per your requirements, making the UI more dynamic and interactive.

SSL Certificate

Since our application is a desktop application, we did not have a SSL certificate. However, we understood the concept of how to generate a SSL certificate and below is our implementation for it.

1. For the SSL certificate, the first step was to ensure that OpenSSL was installed on the system. For Ubuntu, we installed OpenSSL with the command below:

   sudo apt-get install openssl

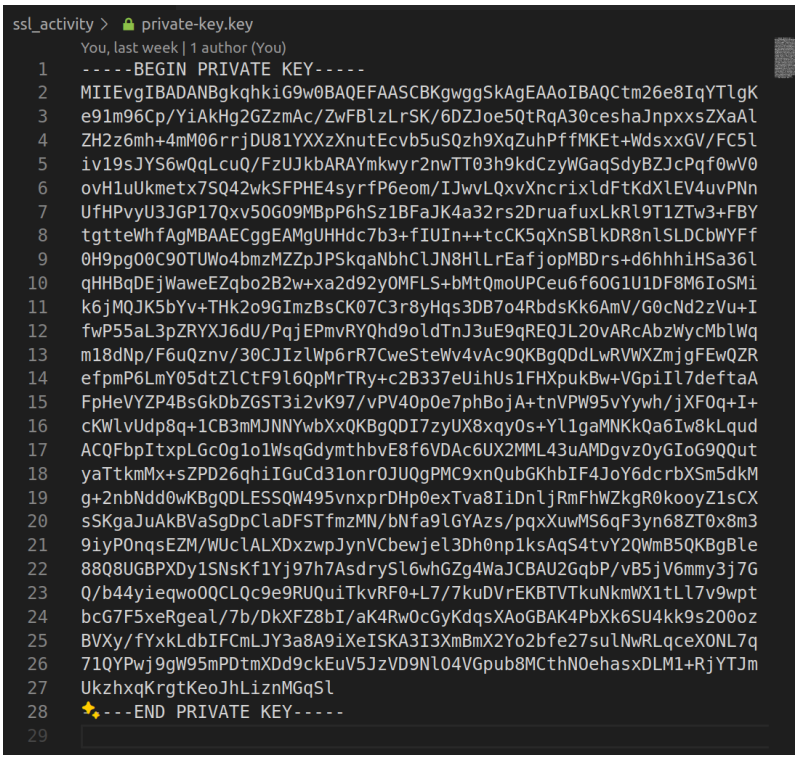
2. Then we generated a private key that we used to secure the CSR and later the SSL certificate. We generated a 2048-bit RSA private key with the following command:

   openssl genpkey -algorithm RSA -out private.key -pkeyopt rsa_keygen_bits:2048
3. Once we had the private key, we were able to generate the CSR as shown:
   openssl req -new -key private.key -out csr.pem
4. We were then prompted to provide information about our organization and the domain for which we are requesting the SSL certificate:

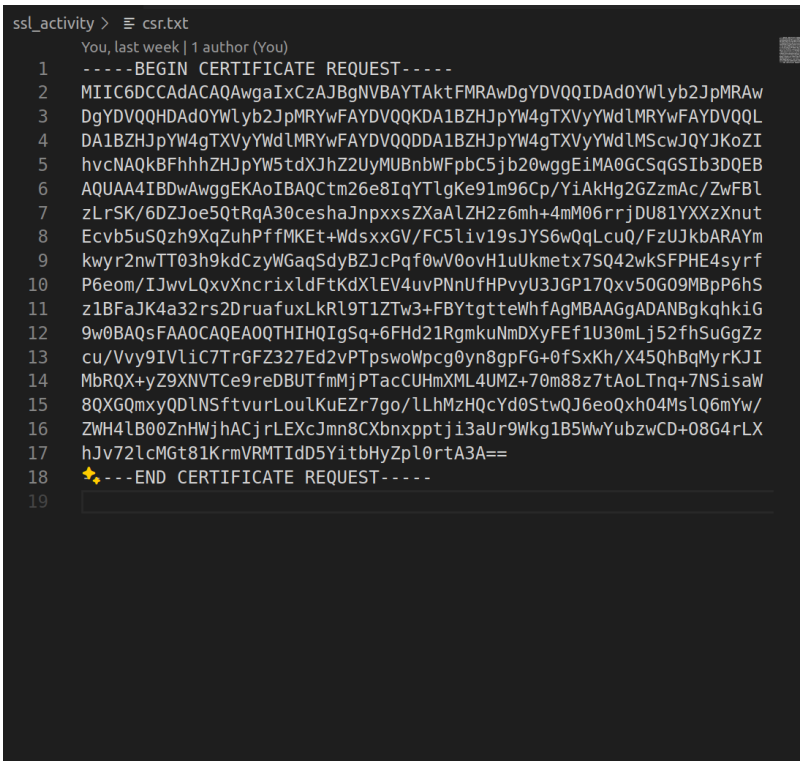
   Country Name (2 letter code) [AU]:KE
   State or Province Name (full name) [Some-State]:Nairobi
   Locality Name (eg, city) []:Nairobi
   Organization Name (eg, company) [Internet Widgits Pty Ltd]:USIU (United States International University - Africa)
   Organizational Unit Name (eg, section) []:
   Common Name (e.g. server FQDN or YOUR name) []:yourdomain.com
   Email Address []:
5. Finally, we generated a self-signed SSL certificate as shown:
   openssl x509 -req -days 365 -in csr.pem -signkey private.key -out self_signed_certificate.crt

The Results

The private key:



The certificate request:



The SSL certificate:


ssl_activity > crt.txt
You, last week | 1 author (You)
-----BEGIN CERTIFICATE----- You, last week • generate SSL ce
MIIDzTCCArUCFDqs7HKa7NGpq5NSnluFN52bGNdYMA0GCSqGSIb3DQEBCwUAMIGi
MQswCQYDVQQGEwJLRTEQMA4GA1UECAwHTmFpcm9iaTEQMA4GA1UEBwwHTmFpcm9i
aTEWMBQGA1UECgwNQWRyaWFuIE11cmFnZTEWMBQGA1UECwwNQWRyaWFuIE11cmFn
ZTEWMBQGA1UEAwwNQWRyaWFuIE11cmFnZTEnMCUGCSqGSIb3DQEJARYYYWRyaWFu
bXVyYWdlMjFAZ21haWwuY29tMB4XDTI0MDMzMTE4MDgwOVoXDTI1MDMzMTE4MDgw
OVowgaIxCzAJBgNVBAYTAktFMRAwDgYDVQQIDAdOYWlyb2JpMRAwDgYDVQQHDAdO
YWlyb2JpMRYwFAYDVQQKDA1BZHJpYW4gTXVyYWdlMRYwFAYDVQQLDA1BZHJpYW4g
TXVyYWdlMRYwFAYDVQQDDA1BZHJpYW4gTXVyYWdlMScwJQYJKoZIhvcNAQkBFhhh
ZHJpYW5tdXJhZ2UyMUBnbWFpbC5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAw
ggEKAoIBAQCtm26e8IqYTlgKe91m96Cp/YiAkHg2GZzmAc/ZwFBlzLrSK/6DZJoe
5QtRqA30ceshaJnpxxsZXaAlZH2z6mh+4mM06rrjDU81YXXzXnutEcvb5uSQzh9X
qZuhPffMKEt+WdsxxGV/FC5liv19sJYS6wQqLcuQ/FzUJkbARAYmkwyr2nwTT03h
9kdCzyWGaqSdyBZJcPqf0wV0ovH1uUkmetx7SQ42wkSFPHE4syrfP6eom/IJwvLQ
xvXncrixldFtKdXlEV4uvPNnUfHPvyU3JGP17Qxv5OGO9MBpP6hSz1BFaJK4a32r
s2DruafuxLkRl9T1ZTw3+FBYtgtteWhfAgMBAAEwDQYJKoZIhvcNAQELBQADggEB
AE4MO/ioTI1DRfmkcf0ZVh90QoqOeSkj8ZoaEbFprRKC5j5H5FJpJsiREeRo7zmf
vZgEC9sS9xlXBabRr6c/BvRgy5dShWoJ1xKKUbWR0AKztRsyM9TR7lTVwpEjC1MQ
DOjQmh7c/y/LR1veQYlI24aoaPJf8oN3dhWRZYbahMm/s8tq/yazZK4lcvpEpJkH
PYRHiV5uU2qVllwzNsBRchxcVP0lLqFUR9v0cwaXi+7N10tWahHNyWjK7mIXKeS6
fNn4iUmFLUt3qvR81XA3sVsWLo/EB2ASILZ/oNAmiCSqfeOMf0p21OnNvz8PFzR4
9zGDlaNv5iJWwJ5ab/93PFU=
-----END CERTIFICATE-----