



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

Table of Contents	2
Implementation of RSA over TCP.....	4
Introduction.	4
Methodology.	4
Alice.....	4
Bob.....	5
The application codes	5
Results	9
Drawbacks.....	12
Conclusion and Recommendations.	12
Credit Card Vault Implementation.....	13
Phase 1.....	13
Background.....	13
Sensitive, confidential, and public information	13
User access levels	13
3NF database schema	13
Phase 2.....	14
Database in 3NF	14
Encryption & Decryption	16
Password hashing	45
Views	45
Authentication	47
GUI Implementation	48
Tkinter	48
Importing tkinter	48
Creating the GUI window	48
Loading the background of the GUI window	48
Adding widgets	49
Running the window	49
PIL Module.....	49
Importing PIL.....	49
Loading and displaying images	49
OS Module.....	50
Importing os	50

Creating Directories	50
Listing Files in a Directory	50
Conclusion.....	50
SSL Certificate.....	50
The Results	51

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:

```

(venv) murage@L:~/Documents/school/APT3090/assignments/group_pro
o j/apt3090-group-project/q1-rsa-implementation-over-tcp$ python3
server.py
Decrypted Session key: b'e683ab1b0963f85bd7305162aa1d3bd0'

Socket created
Waiting for connection...
□

```

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

```
(venv) murage@L:~/Documents/school/APT3090/assignment-over-tcp$ python3 server.py
Decrypted Session key: b'e683ab1b0963f85bd7305162aald3bd0'

Socket created
Waiting for connection...
Connected with ('127.0.0.1', 56798)
Bob public key: -----BEGIN PUBLIC KEY-----
MIGfMA0GCsGSIb3DQEBAQUAA4GNADCBiQKBgQCIbI+J25AzwdGiQqLLu0QQUH4A
HAsK8W96Zd7WaKe3bqZ6R1ohiKAa8IKTs/kJtE+N8XrXe5DLf/nCtv9b4K7Sp90P
HIiWsq9EHVktVTWkwZxTxnkutUZgu6b2SEX9lQQFb3nC6uW7ML/JJLR0FyA24MUu
gg6i6EkXGwOP9QoruwIDAQAB
-----END PUBLIC KEY-----

Encrypted session key: b'0a6f4f4ebab8f6d25d808702163872208848f3e4349cbb18771d961011cea989dd3e53362febe1e8a704231425b62fbfee0afe789aa9c1146769168480b97553ea175ad94de2da28e247f0e440274f9da0a3509cc2591354c9f6a46273f4bc42d9274419dbdfdd7f525ae0719d03b354441e308960a8b9807bcfd6a0ec2a223'

Encrypted message: b'664c1c73b08e4a29b03deed9a39647d3'

Bob: Hi Alice.
Alice -> [ ]

(venv) murage@L:~/Documents/school/APT3090/assignment-over-tcp$ python3 client.py
Bob public key: -----BEGIN PUBLIC KEY-----
MIGfMA0GCsGSIb3DQEBAQUAA4GNADCBiQKBgQCIbI+J25AzwdGiQqLLu0QQUH4A
HAsK8W96Zd7WaKe3bqZ6R1ohiKAa8IKTs/kJtE+N8XrXe5DLf/nCtv9b4K7Sp90P
HIiWsq9EHVktVTWkwZxTxnkutUZgu6b2SEX9lQQFb3nC6uW7ML/JJLR0FyA24MUu
gg6i6EkXGwOP9QoruwIDAQAB
-----END PUBLIC KEY-----

Bob private key: -----BEGIN RSA PRIVATE KEY-----
MIICXgIBAAKBgQCIbI+J25AzwdGiQqLLu0QQUH4AHAsK8W96Zd7WaKe3bqZ6R1ohiKAa8IKTs/kJtE+N8XrXe5DLf/nCtv9b4K7Sp90PHIiWsq9EHVktVTWkwZxTxnkutUZgu6b2SEX9lQQFb3nC6uW7ML/JJLR0FyA24MUuagg6i6EkXGwOP9QoruwIDAQAB
AoGAPEX0Uudc+ejvL+0F3kHh92MFYt006r4iBiaz5qUQsARsCN3CCq/JbfN00jq
JaXyofEmxy4SEY06rRLBljuZsNYQANKAXZz0V0nQA4IBen0jd4N0GHVuXjWqLK0C
5PV7nuoTPcmijoGHwz9N/LVtbMorzG6Jx5ZMaAnPosFqp60CQQC6gzY32jCtFVLH
ue2/QcSseoVDgksZnZuRxuNxRiczc6Kyo0Fci0SmmMYRgFxA5eCsRyz/eZr7PBUAT
0WUHHbUFAKEAu0AeYEFq7jPsZAax9P1WDG4ChyQbI2bL2zR8F2HwJhpHI01230LF
gHBQ0lu2Ieu0vaVx0ju4nZ8m2Tunch0ZvwJBAIgbz0tBbRhCYWCjjj5wDLWe17WG
Vf609nKR09U8DsvQ1n8EboHpGT3REPuhTs6BpVyc6IVedtx4xmQwAPxVJUkCQCg
9LanqtPNudhDCvZBTtp+iLmnjM0/JIeq/2yDd05G8m0f1uCLjexTQThY1gazFZZ3
MhUoJ1nEM5/jL0H4kbBlAKEAtSSGS+pq7jVZY03w7JYTEh/hL7ZdX/K/0FtQPowY
TbsmUFIh62tjn786pWftr1zB4VYJuZCmXEKJ2Lx502rIA==
-----END RSA PRIVATE KEY-----

Encrypted session key: b'0a6f4f4ebab8f6d25d808702163872208848f3e4349cbb18771d961011cea989dd3e53362febe1e8a704231425b62fbfee0afe789aa9c1146769168480b97553ea175ad94de2da28e247f0e440274f9da0a3509cc2591354c9f6a46273f4bc42d9274419dbdfdd7f525ae0719d03b354441e308960a8b9807bcfd6a0ec2a223'

Decrypted Session key: b'e683ab1b0963f85bd7305162aald3bd0'

Bob -> Hi Alice.
Encrypted message: b'664c1c73b08e4a29b03deed9a39647d3'
```

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'e683ab1b0963f85bd7305162a
ald3bd0'

Socket created
Waiting for connection...
Connected with ('127.0.0.1', 56798)
Bob public key: -----BEGIN PUBLIC KEY-----
MIGfMA0GCsGSIb3DQEBAQUAA4GNADCBiQKBgQCIbI+J25Azw
dGiQqLLu0QQUH4A
HAsK8W96Zd7WaKe3bqZ6R1ohiKAa8IKTs/kJtE+N8XrXe5DlF/
nCtv9b4K7Sp90P
HIiWsq9EHVktVTWkwZxTxnkuUzgu6b2SEX9lQQFb3nC6uW7ML
/JJLR0FyA24MUu
gg6i6EkXGwOP9QoruwIDAQAB
-----END PUBLIC KEY-----

Encrypted session key: b'0a6f4f4ebab8f6d25d8087021
63872208848f3e4349cbb18771d961011cea989dd3e53362fe
be1e8a704231425b62fbfee0afe789aa9c1146769168480b97
553ea175ad94de2da28e247f0e440274f9da0a3509cc259135
4c9f6a46273f4bc42d9274419dbdfdd7f525ae0719d03b3544
441e308960a8b9807bcfd6a0ec2a223'

Encrypted message: b'664c1c73b08e4a29b03deed9a3964
7d3'

Bob: Hi Alice.
Alice -> Hi Bob!, I hope you are well?
Encrypted response: b'49cbe06fd14dd212fc95d6640b5a
fc6da0676d949d7c9ed8c064150a9f75275a'

Encrypted message: b'fbd9f7087f64582298e6030357169
b5db786011371e902f9e5684b2e8a61d8f6e583f09194e236d
8a2cefa024056b215935bc7a4d5589ab65f5323a6aa7d6f30'

Bob: I am well, I was just checking in, I can't st
ay long though...
Alice -> That's okay, enjoy the rest of your day.
Bye.
Encrypted response: b'5da02dfffbd1c0079ce0eb45c247
bcdff0bf11e48829e5b55fae5654350bafcb9af15bcd8b58be
ce2b57eabf5f2ea357'

(venv) murage@L:~/Documents/school/APT3090/assignm
ents/group_proj/apt3090-group-project/q1-rsa-imple
mentation-over-tcp$
```

```
-
MIICXgIBAAKBgQCIbI+J25Azw dGiQqLLu0QQUH4AHAsK8W96
Zd7WaKe3bqZ6R1oh
iKAa8IKTs/kJtE+N8XrXe5DlF/nCtv9b4K7Sp90PHIiWsq9E
HVkTVTWkwZxTxnku
tUZgu6b2SEX9lQQFb3nC6uW7ML/JJLR0FyA24MUugg6i6EkX
GwOP9QoruwIDAQAB
AoGAPEX0Uudc+ejvL+OF3kHh92MFYt006r4iBiaz5qUQsAR
sCN3CCq/Jbfn00jq
JaXyofEmxy4SEY06rRLB1juZsNYQAnKAXZ0V0nQA4IBen0j
d4NOGHVuxjWqLK0C
5PV7nuoTPcmijoGHwz9N/LVtbMorzG6Jx5ZMaANposFqp60C
QQC6gzY32jCtFVLH
ue2/QcSSeoVDgksZNRXuNxrRic6Kyo0Fci0SmmMYRgFA5
eCsRyz/eZr7PBUAT
0WUHHBUFAkEau0AeYEFq7jPsZAax9P1WDG4ChyQbI2bL2zR8
F2HwJhpHI0123QLF
gHB001u2Ieu0vaVx0ju4nZ8m2TuncH0ZvWJBAIgb0tBbRhC
YWCjjJ5wDlWe17WG
Vf609nKR09U8DsvQ1n8EboHpGT3REPuhTs6BpVyc6IVedtx4
xmQwAPxVJUkCQQCg
9LanqtPNudhCvZBTtp+iLmnmjM0/JIEq/2yDd05G8m0fluCL
jexTQThY1gazFZZ3
MhUoJ1nEM5/jL0H4kbBLAkeAtSSGS+pq7jVZY03w7JYTEh/h
L7ZdX/K/0FtQPowY
TbsmUFIh62tjn786pWftr1zB4VYJuzCmXEKJ2lxF502rIA==
-----END RSA PRIVATE KEY-----

Encrypted session key: b'0a6f4f4ebab8f6d25d80870
2163872208848f3e4349cbb18771d961011cea989dd3e533
62febe1e8a704231425b62fbfee0afe789aa9c1146769168
480b97553ea175ad94de2da28e247f0e440274f9da0a3509
cc2591354c9f6a46273f4bc42d9274419dbdfdd7f525ae07
19d03b3544441e308960a8b9807bcfd6a0ec2a223'

Decrypted Session key: b'e683ab1b0963f85bd730516
2aald3bd0'

Bob -> Hi Alice.
Encrypted message: b'664c1c73b08e4a29b03deed9a39
647d3'

Encrypted response: b'49cbe06fd14dd212fc95d6640b
5afc6da0676d949d7c9ed8c064150a9f75275a'

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'fbd9f7087f64582298e60303571
69b5db786011371e902f9e5684b2e8a61d8f6e583f09194e
236d8a2cefa024056b215935bc7a4d5589ab65f5323a6aa7
d6f30'

Encrypted response: b'5da02dfffbd1c0079ce0eb45c2
47bcdff0bf11e48829e5b55fae5654350bafcb9af15bcd8b
58bece2b57eabf5f2ea357'

Alice: That's okay, enjoy the rest of your day.
Bye.
Bob -> bye
(venv) murage@L:~/Documents/school/APT3090/assig
nments/group_proj/apt3090-group-project/q1-rsa-i
plementation-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

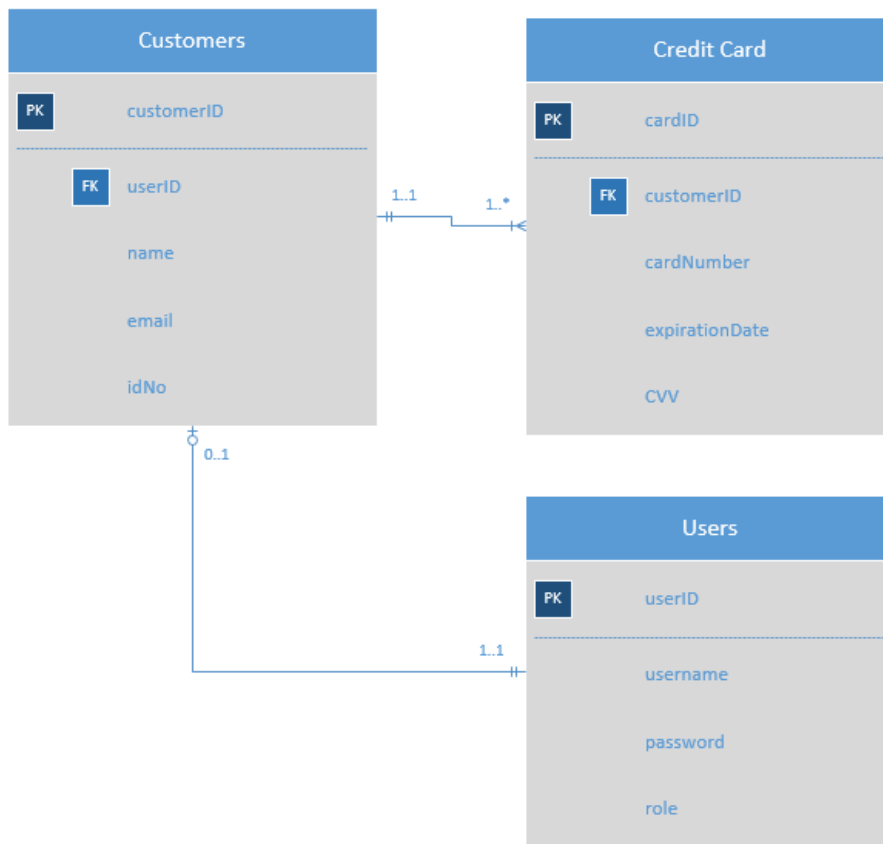
User access levels

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.

```

61 # Creation of database
62 def createDatabase():
63     conn = sqlite3.connect('creditcard_vault.db')
64     c = conn.cursor()
65
66     try:
67         c.execute("""
68             CREATE TABLE IF NOT EXISTS CUSTOMERS (
69                 customerID INTEGER PRIMARY KEY AUTOINCREMENT,
70                 name TEXT NOT NULL,
71                 email TEXT UNIQUE,
72                 idNo INTEGER NOT NULL,
73                 userID INTEGER,
74                 FOREIGN KEY (userID) REFERENCES USERS(userID)
75             )
76         """)
77     except sqlite3.OperationalError as e:
78         print(f"Error creating table: {e}")
79
80     try:
81         c.execute("""
82             CREATE TABLE IF NOT EXISTS CREDITCARDS (
83                 cardID INTEGER PRIMARY KEY AUTOINCREMENT,
84                 customerID INTEGER NOT NULL,
85                 cardNumber INTEGER NOT NULL,
86                 expirationDate TEXT NOT NULL,
87                 CVV INTEGER NOT NULL,
88                 FOREIGN KEY (customerID) REFERENCES CUSTOMERS(customerID)
89             )
90         """)
91     except sqlite3.OperationalError as e:
92         print(f"Error creating table: {e}")
93
94     try:
95         c.execute("""
96             CREATE TABLE IF NOT EXISTS USERS (
97                 userID INTEGER PRIMARY KEY AUTOINCREMENT,
98                 username TEXT NOT NULL,
99                 password TEXT NOT NULL,
100                 role TEXT NOT NULL
101             )
102         """)
103     except sqlite3.OperationalError as e:
104         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.


```

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

18 entry_widgets = None      You, 4 days ago • make it easier to navigate
19
20 # Function to encrypt credit card details
21 def encrypt_sensitive_information(plain_text, password):
22     # Generate a random salt
23     salt = get_random_bytes(AES.block_size)
24
25     # Use the Script KDF to get a private key from the password
26     private_key = hashlib.scrypt(
27         password.encode(), salt=salt, n=2 ** 14, r=8, p=1, dklen=32)
28
29     # Create cipher config
30     cipher_config = AES.new(private_key, AES.MODE_GCM)
31
32     # Encrypt the plain text
33     cipher_text, _ = cipher_config.encrypt_and_digest(bytes(plain_text, 'utf-8'))
34
35     # Return encrypted data as a single string (including salt and nonce)
36     return b64encode(salt + cipher_config.nonce + cipher_text).decode('utf-8')
37
38 # Function to decrypt credit card details
39 def decrypt_sensitive_information(encrypted_data, password):
40     # Decode base64 encoded data
41     encrypted_text = b64decode(encrypted_data)
42
43     # Extract salt, nonce, and cipher text
44     salt = encrypted_text[:AES.block_size]
45     nonce = encrypted_text[AES.block_size:AES.block_size * 2]
46     cipher_text = encrypted_text[AES.block_size * 2:]
47
48     # Use the Script KDF to derive the private key from the password and salt
49     private_key = hashlib.scrypt(
50         password.encode(), salt=salt, n=2 ** 14, r=8, p=1, dklen=32)
51
52     # Create cipher config
53     cipher_config = AES.new(private_key, AES.MODE_GCM, nonce=nonce)
54
55     # Decrypt the cipher text
56     plain_text = cipher_config.decrypt(cipher_text)
57
58     # Return decrypted plain text
59     return plain_text.decode('utf-8')

```

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 Script 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.

```

ccv.py x
q2-credit-card-vault > ccv.py > addCustomerRecord
178 def get_user_id(username):
186     else:
187         return None
188
189 def addCustomerRecord():
190     global entry_widgets
191     try:
192         username = entry_widgets["Username"].get()
193         customer_name = entry_widgets["Full Name"].get()
194         customer_email = entry_widgets["Email Address"].get()
195         customer_id_no = entry_widgets["National Identification Number"].get()
196         credit_card_number = entry_widgets["Credit Card Number"].get()
197         expiration_date = entry_widgets["Credit Card Expiration Date"].get()
198         cvv = entry_widgets["Credit Card CVV"].get()
199
200         # Retrieve userID based on username
201         user_id = get_user_id(username)
202
203         if user_id is None:
204             raise ValueError("User not found")
205
206         # Encrypt credit card details
207         encrypted_id_number = encrypt_sensitive_information(customer_id_no, "secretKey")
208         encrypted_cc_number = encrypt_sensitive_information(credit_card_number, "secretKey")
209         encrypted_exp_date = encrypt_sensitive_information(expiration_date, "secretKey")
210         encrypted_cvv = encrypt_sensitive_information(cvv, "secretKey")
211
212         conn = sqlite3.connect('creditcard_vault.db')
213         c = conn.cursor()
214
215         c.execute("INSERT INTO CUSTOMERS (name, email, idNo, userID) VALUES (?, ?, ?, ?)",
216                 | (customer_name, customer_email, encrypted_id_number, user_id))
217         conn.commit()
218
219         customer_id = c.lastrowid
220
221         c.execute("INSERT INTO CREDITCARDS (customerID, cardNumber, expirationDate, CVV) VALUES (?, ?, ?, ?)",
222                 | (customer_id, encrypted_cc_number, encrypted_exp_date, encrypted_cvv))
223         conn.commit()
224
225         conn.close()
226
227         for entry_widget in entry_widgets.values():
228             entry_widget.delete(0, END)
229
230         messagebox.showinfo("Success", "Customer record added successfully!")
231     except sqlite3.Error as e:

```

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.

```

ccv.py
q2-credit-card-vault > ccv.py > showRecordsPage
437
438 def showRecordsPage():
439     global current_page
440     if current_page:
441         current_page.pack_forget()
442
443     current_page = Frame(root)
444     current_page.pack()
445
446     conn = sqlite3.connect('creditcard_vault.db')
447     c = conn.cursor()
448
449     c.execute("""
450     SELECT c.customerID, c.name, c.email, c.idNo, cc.cardNumber, cc.expirationDate, cc.CW
451     FROM CUSTOMERS c
452     INNER JOIN CREDITCARDS cc ON c.customerID = cc.customerID
453     """)
454     records = c.fetchall()
455
456     record_listbox = Listbox(current_page, width=200)
457     record_listbox.pack(padx=10, pady=10)
458
459     for record in records:
460         # Decrypt credit card details if they are encrypted
461         decrypted_id_number = record[3]
462         decrypted_cc_number = record[4]
463         decrypted_exp_date = record[5]
464         decrypted_cvv = record[6]
465
466         try:
467             decrypted_id_number = decrypt_sensitive_information(record[3], "secretKey")
468             decrypted_cc_number = decrypt_sensitive_information(record[4], "secretKey")
469             decrypted_exp_date = decrypt_sensitive_information(record[5], "secretKey")
470             decrypted_cvv = decrypt_sensitive_information(record[6], "secretKey")
471         except Exception as e:
472             print(f"Error decrypting record: {e}")
473
474         # Format the record
475         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}"
476
477         # Insert the formatted record into the Listbox
478         record_listbox.insert(END, formatted_record)
479
480     conn.close()
481     hideShowAllCustomerDetailsNavBtn()
482     hideAddCustomerDetailsButton()
483     showAddCustomerNavigationBtn()

```

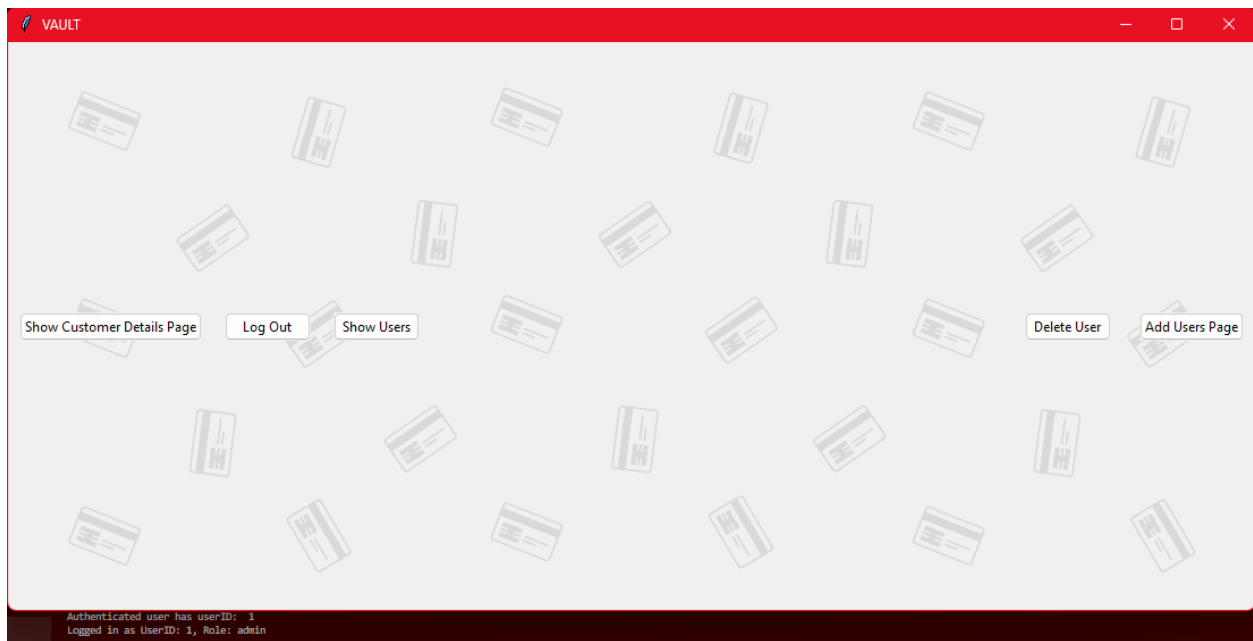
Encrypted customer details in the database.

Reset Filters Records: 1						
	customerID 🔍 # ⇅	name 🗨 ⇅	email 🗨 ⇅	idNo # ⇅	userID 🔍 # ⇅	
	<input data-bbox="310 1184 467 1213" type="text" value="Search column..."/>	<input data-bbox="505 1184 662 1213" type="text" value="Search column..."/>	<input data-bbox="699 1184 857 1213" type="text" value="Search column..."/>	<input data-bbox="894 1184 1052 1213" type="text" value="Search column..."/>	<input data-bbox="1089 1184 1247 1213" type="text" value="Search column..."/>	
1	1	Alice Doe	adoc@usiu.acke	SJhB9dWfKo17V0eEj...	2	

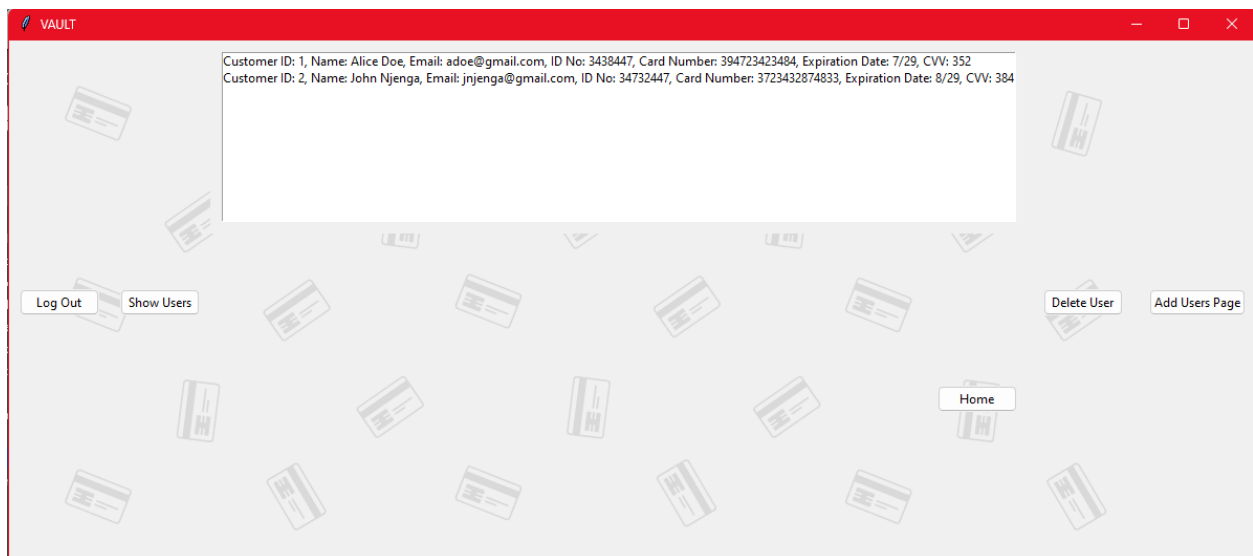
Encrypted credit card details in the database

Reset Filters Records: 1					
	cardID 🔍 # ⇅	customerID 🔍 # ⇅	cardNumber # ⇅	expirationDate 🗨 ⇅	CVV # ⇅
	<input data-bbox="310 1413 467 1442" type="text" value="Search column..."/>	<input data-bbox="505 1413 662 1442" type="text" value="Search column..."/>	<input data-bbox="699 1413 857 1442" type="text" value="Search column..."/>	<input data-bbox="894 1413 1052 1442" type="text" value="Search column..."/>	<input data-bbox="1089 1413 1247 1442" type="text" value="Search column..."/>
1	1	1	BvkY5mvzsaK9n0Mmk...	wx7Mmaxq2wvXES...	L3q1EGNoCRVn6TvAS...

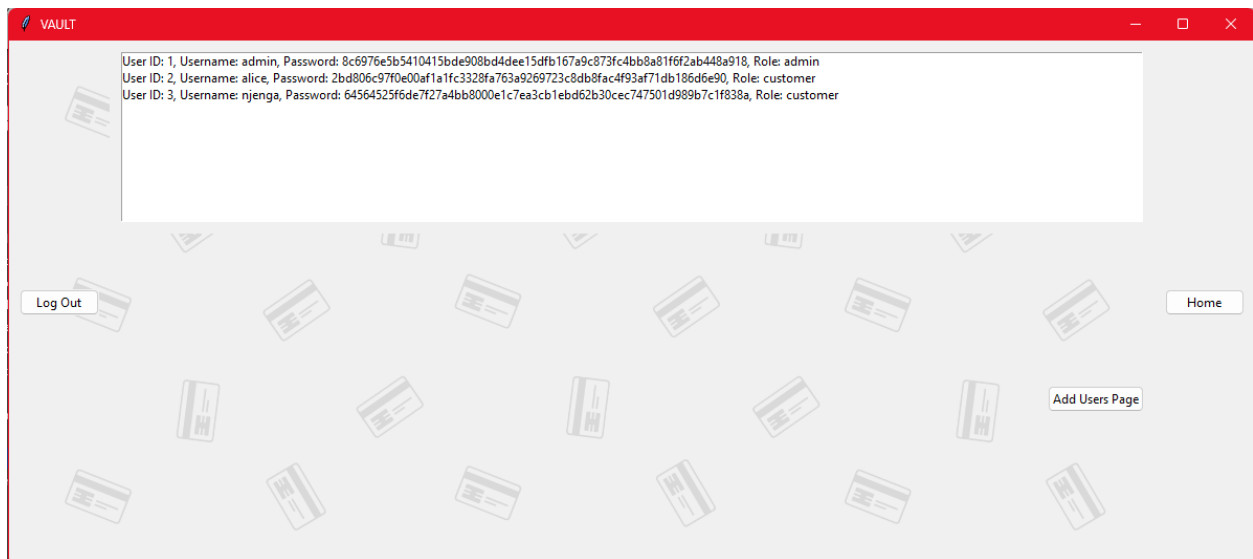
Only the admin can see decrypted customer sensitive information.



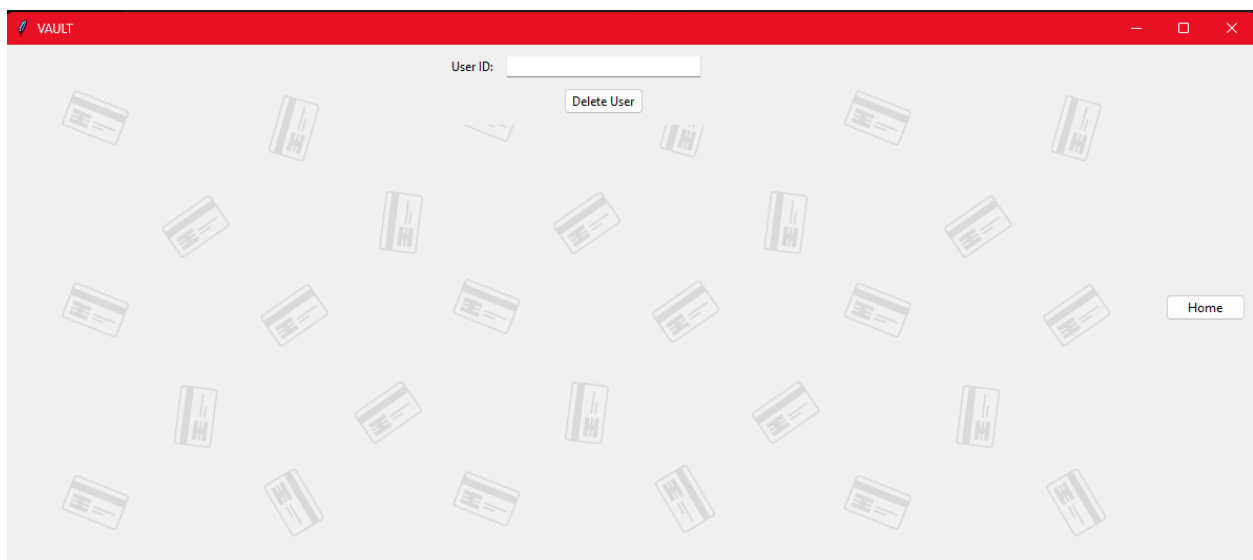
Decrypted customer details.



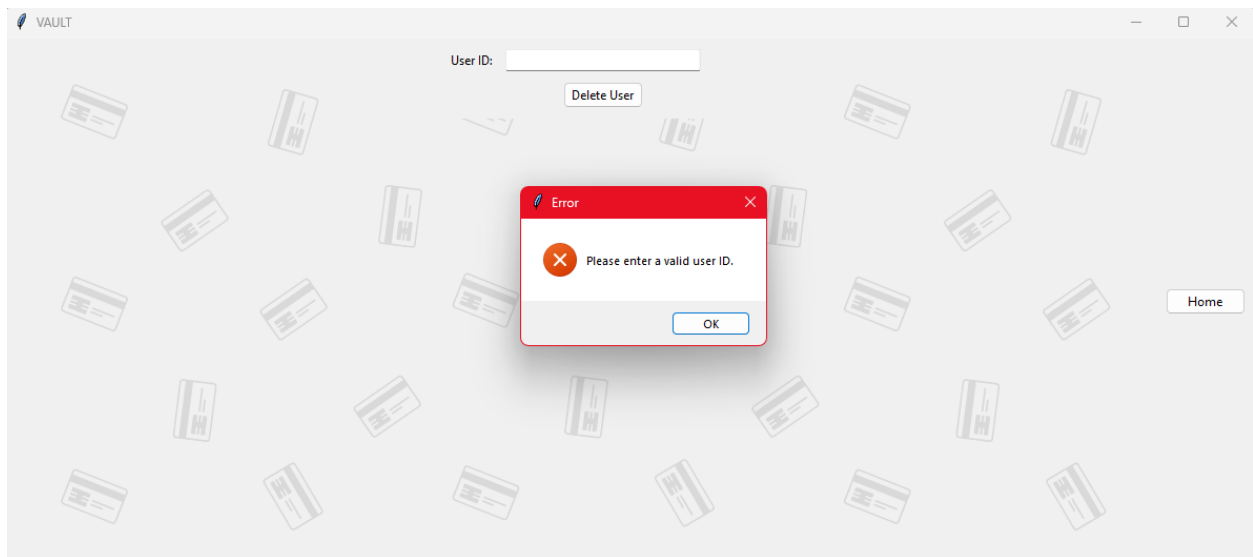
Only the admin can see the current users in the system when they click the “Show Users” button. They cannot see the users’ passwords though.



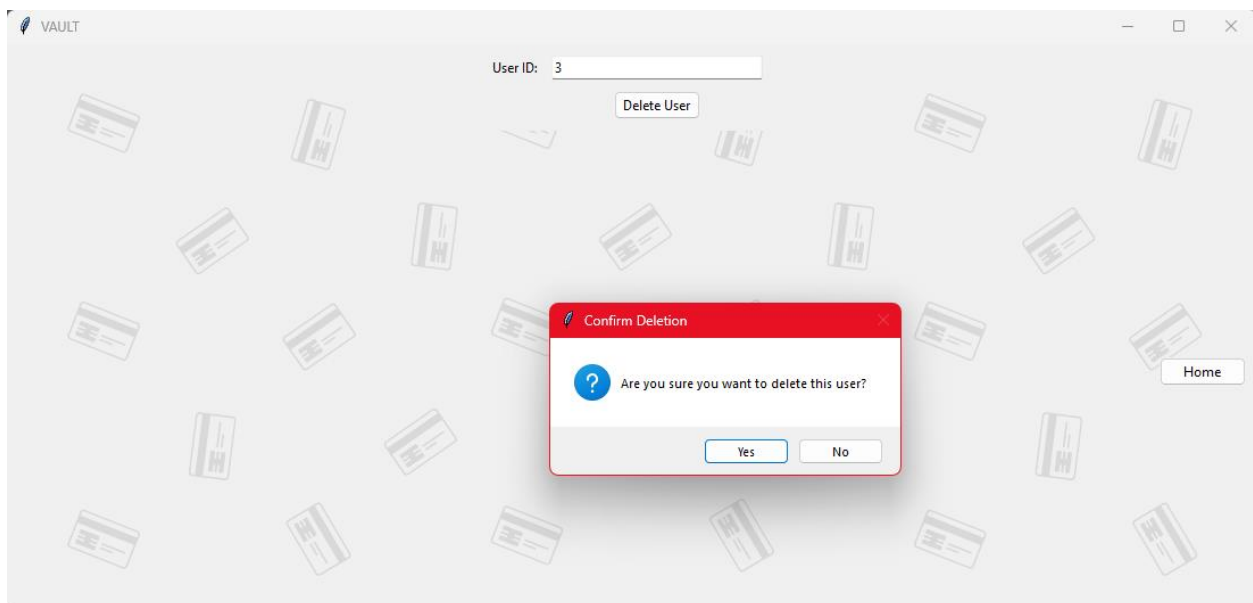
They can delete a user in the delete users page by keying in the user id they want to delete.



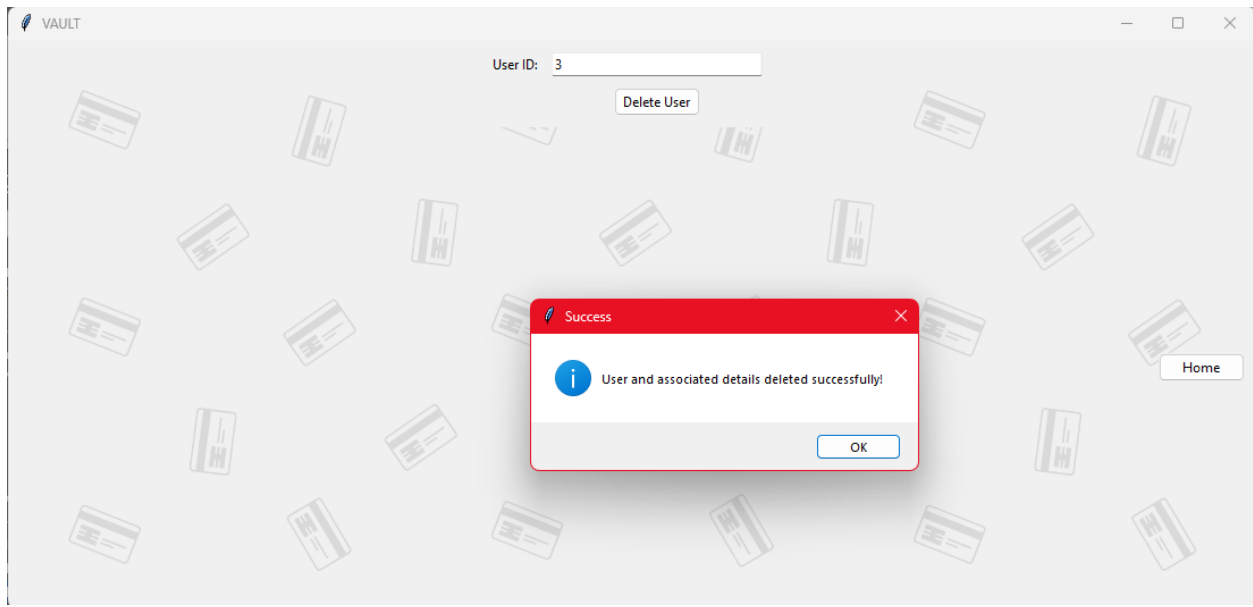
Form validation put in place



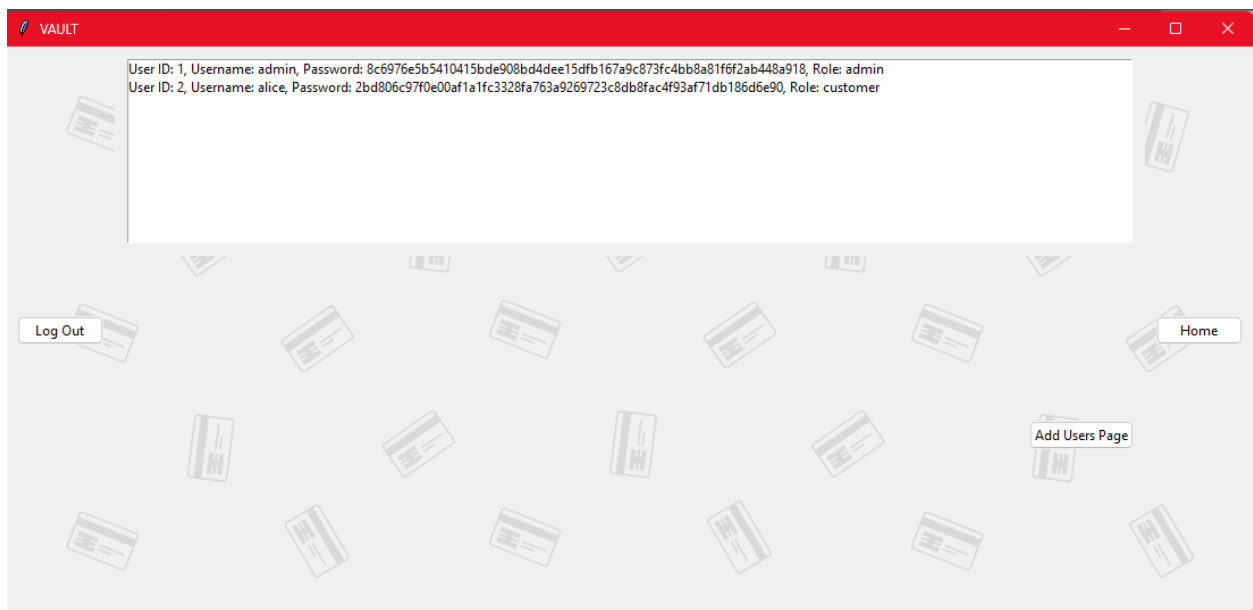
Say the merchant wants to delete user id 3, they simply key in 3 and hit the “Delete User” button. They are presented with a confirmation dialog box as show below in which when they click “Yes”, all the details about that user are deleted across the entire database.



A feedback message is shown to the merchant



Confirmation that user id 3 has been deleted



Delete user page function


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

744 def delete_user_and_details(user_id):
745     conn = sqlite3.connect('creditcard_vault.db')
746     c = conn.cursor()
747
748     try:
749         # Get customer ID associated with the user
750         c.execute("SELECT customerID FROM CUSTOMERS WHERE userID = ?", (user_id,))
751         customer_id = c.fetchone()[0]
752
753         # Delete customer's credit card details
754         c.execute("DELETE FROM CREDITCARDS WHERE customerID = ?", (customer_id,))
755
756         # Delete customer details
757         c.execute("DELETE FROM CUSTOMERS WHERE customerID = ?", (customer_id,))
758
759         # Delete the user
760         c.execute("DELETE FROM USERS WHERE userID = ?", (user_id,))
761
762         conn.commit()
763         conn.close()
764         messagebox.showinfo("Success", "User and associated details deleted successfully!")
765
766         # Redirect to add customer info page after successful deletion
767         addCustomerInfoPage()
768     except sqlite3.Error as e:
769         conn.rollback()
770         conn.close()
771         messagebox.showerror("Error", f"Error occurred: {e}")
772
```

```

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

773 def deleteUserPage():
774     def deleteUser():
775         try:
776             user_id_str = userIdEntry.get().strip()
777             if not user_id_str:
778                 raise ValueError("User ID is required")
779
780             user_id = int(user_id_str)
781
782             # Check if the user ID exists before attempting deletion
783             conn = sqlite3.connect('creditcard_vault.db')
784             c = conn.cursor()
785             c.execute("SELECT * FROM USERS WHERE userID = ?", (user_id,))
786             user = c.fetchone()
787             conn.close()
788
789             if user:
790                 # Confirm deletion
791                 confirm = messagebox.askyesno("Confirm Deletion", "Are you sure you want to delete this user?")
792                 if confirm:
793                     delete_user_and_details(user_id)
794                     # Clear input field after deletion
795                     userIdEntry.delete(0, END)
796                 else:
797                     # Display error message if user ID does not exist
798                     messagebox.showerror("Error", "User ID does not exist.")
799             except ValueError:
800                 messagebox.showerror("Error", "Please enter a valid user ID.")
801
802     global current_page
803     if current_page:
804         current_page.pack_forget() CG, 5 days ago • Added delete user by ID functionality
805
806     current_page = Frame(root)
807     current_page.pack()
808
809     hideAddCustomerDetailsButton()
810     hideLogoutBtn()
811     hideShowAllCustomerDetailsNavBtn()
812     hideAddUsersNavigationBtn()
813     hideShowAllUsersButton()
814     hide_update_password_button()
815     hideDeleteUserButton()
816     showAddCustomerNavigationBtn()
817
818     userIdLabel = Label(current_page, text="User ID:")
819     userIdLabel.grid(row=0, column=0, pady=(10, 0))

```

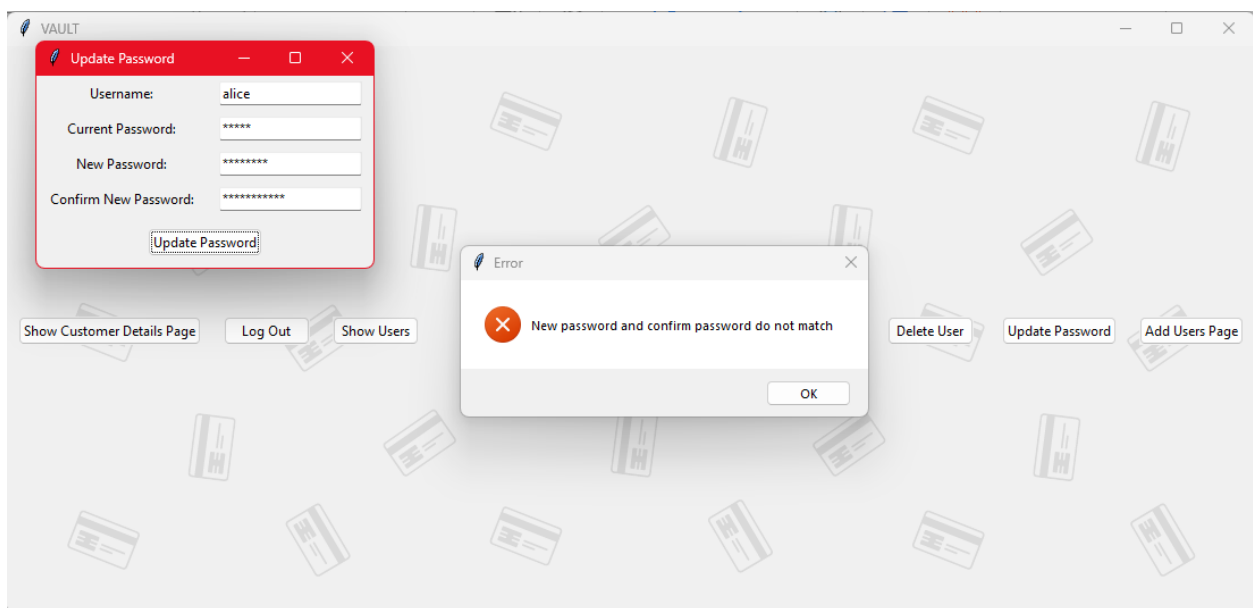
```

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()

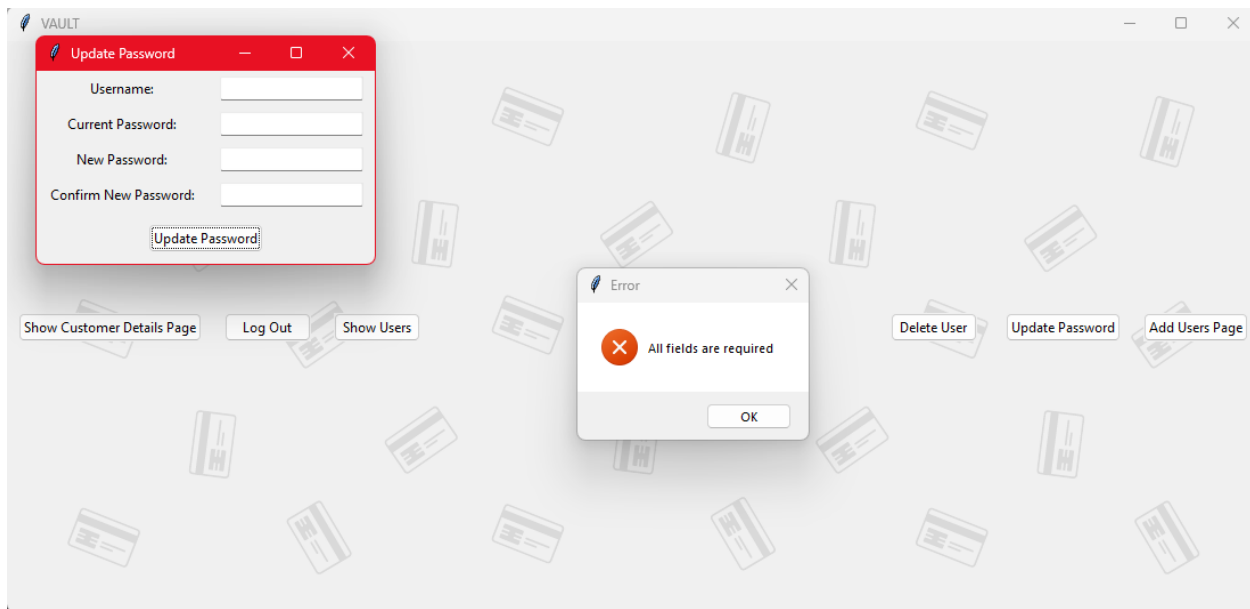
```

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

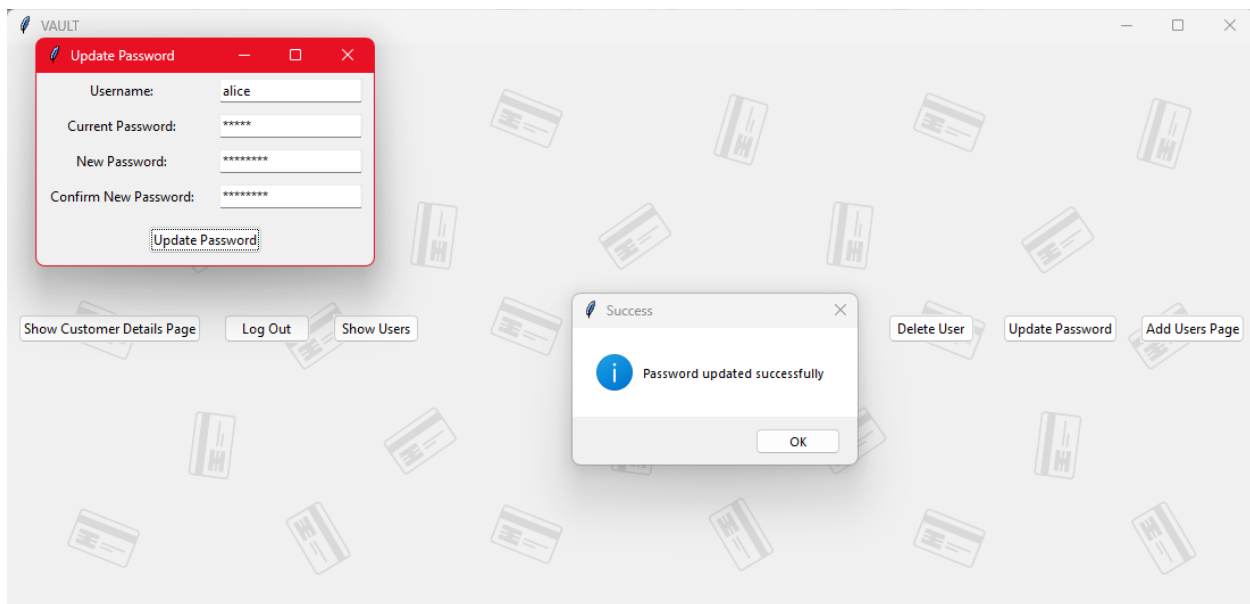
Error handling



Form validation



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



Update password function

```

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

646 # Function to update user's password in the database CG, 5 days ago * Admin can update user password
647 def update_password(username, current_password, new_password):
648     conn = sqlite3.connect('creditcard_vault.db')
649     c = conn.cursor()
650
651     # Hash the current password for comparison
652     hashed_current_password = hash_password(current_password)
653
654     # Check if the username and current password match
655     c.execute("SELECT * FROM USERS WHERE username = ? AND password = ?", (username, hashed_current_password))
656     user = c.fetchone()
657
658     if user:
659         # Hash the new password before updating
660         hashed_new_password = hash_password(new_password)
661
662         # Update the password in the database
663         c.execute("UPDATE USERS SET password = ? WHERE username = ?", (hashed_new_password, username))
664         conn.commit()
665         conn.close()
666         return True
667     else:
668         conn.close()
669         return False
670
671 # UI for password update form
672 def password_update_page():
673     def update_password_action():
674         # Get values from the input fields
675         username = username_entry.get().strip()
676         current_password = current_password_entry.get().strip()
677         new_password = new_password_entry.get().strip()
678         confirm_new_password = confirm_new_password_entry.get().strip()
679
680         # Validate input fields
681         if not all([username, current_password, new_password, confirm_new_password]):
682             messagebox.showerror("Error", "All fields are required")
683             return
684
685         if new_password != confirm_new_password:
686             messagebox.showerror("Error", "New password and confirm password do not match")
687             return
688
689         # Check if the username and current password match before updating
690         if update_password(username, current_password, new_password):
691             messagebox.showinfo("Success", "Password updated successfully")
692             # Clear input fields

```

```

693         username_entry.delete(0, END)
694         current_password_entry.delete(0, END)
695         new_password_entry.delete(0, END)
696         confirm_new_password_entry.delete(0, END)
697     else:
698         messagebox.showerror("Error", "Incorrect username or current password")
699
700     # Create the password update form
701     password_update_window = Toplevel(root)
702     password_update_window.title("Update Password")
703
704     # Username field
705     username_label = Label(password_update_window, text="Username:")
706     username_label.grid(row=0, column=0, padx=10, pady=5)
707     username_entry = Entry(password_update_window)
708     username_entry.grid(row=0, column=1, padx=10, pady=5)
709
710     # Current password field
711     current_password_label = Label(password_update_window, text="Current Password:")
712     current_password_label.grid(row=1, column=0, padx=10, pady=5)
713     current_password_entry = Entry(password_update_window, show="*")
714     current_password_entry.grid(row=1, column=1, padx=10, pady=5)
715
716     # New password field
717     new_password_label = Label(password_update_window, text="New Password:")
718     new_password_label.grid(row=2, column=0, padx=10, pady=5)
719     new_password_entry = Entry(password_update_window, show="*")
720     new_password_entry.grid(row=2, column=1, padx=10, pady=5)
721
722     # Confirm new password field
723     confirm_new_password_label = Label(password_update_window, text="Confirm New Password:")
724     confirm_new_password_label.grid(row=3, column=0, padx=10, pady=5)
725     confirm_new_password_entry = Entry(password_update_window, show="*")
726     confirm_new_password_entry.grid(row=3, column=1, padx=10, pady=5)
727
728     # Button to update password
729     update_button = Button(password_update_window, text="Update Password", command=update_password_action)
730     update_button.grid(row=4, columnspan=2, padx=10, pady=10)
731
732     # Function to show password update page
733     def show_update_password_button():
734         password_update_button.config(command=password_update_page)
735         password_update_button.pack(side=RIGHT, padx=10, pady=15)
736
737     # Create a button to show the password update page
738     password_update_button = Button(root, text="Update Password", command=show_update_password_button)
739
740     # Function to hide the update password button
741     def hide_update_password_button():
742         password_update_button.pack_forget()

```

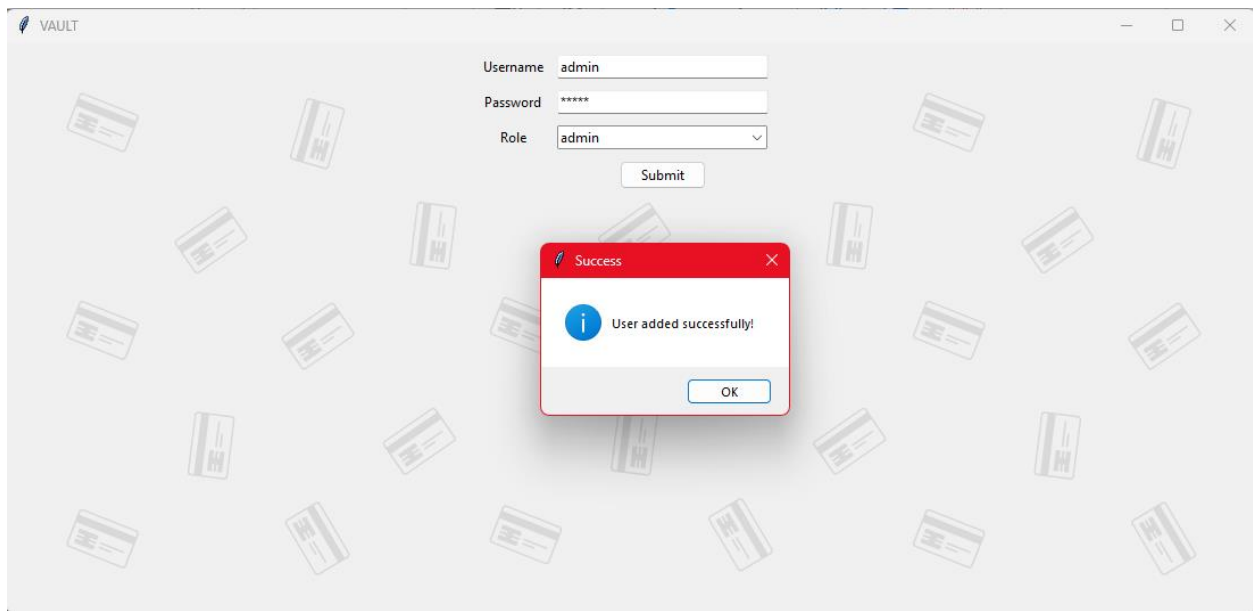
A merchant signs up as an admin by filling out their details in the sign up form

A screenshot of a web application window titled "VAULT". The window has a red header bar with the "VAULT" logo on the left and standard window controls (minimize, maximize, close) on the right. The main content area has a light gray background with a repeating pattern of small, faint card icons. In the center, there is a login form with three input fields: "Username" containing the text "admin", "Password" containing six asterisks "*****", and "Role" with a dropdown menu showing "admin". Below these fields is a "Submit" button.

Form validation put in place

A screenshot of the same "VAULT" login form, but with an error message dialog box displayed in the center. The dialog box has a red header bar with the word "Error" and a close button. It contains an orange circular icon with a white "X" and the text "All fields are required". At the bottom of the dialog is an "OK" button. In the background, the login form is visible, but the "Password" field is now empty, and the "Role" dropdown still shows "admin".

Feedback message put in place



Add User Page function


```

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

498 def addUser(username, password, role):
499     try:
500         # Hash the password before storing
501         hashed_password = hash_password(password)
502
503         conn = sqlite3.connect('creditcard_vault.db')
504         c = conn.cursor()
505         c.execute("INSERT INTO USERS (username, password, role) VALUES (?, ?, ?)", (username, hashed_password, role))
506         conn.commit()
507         conn.close()
508         messagebox.showinfo("Success", "User added successfully!")
509     except sqlite3.Error as e:
510         messagebox.showerror("Error", f"Error occurred: {e}")
511
512 def hash_password(password):
513     # Hash the password using SHA-256
514     hashed_password = hashlib.sha256(password.encode()).hexdigest()
515     return hashed_password
516
517 def addUserPage():
518     def submitUser():
519         username = getUsername.get().strip()
520         password = getPassword.get().strip()
521         role = getRole.get().strip()
522         if not (username and password and role):
523             messagebox.showerror("Error", "All fields are required")
524             return
525         addUser(username, password, role)
526         loginPage()
527
528     global current_page
529     if current_page:
530         current_page.pack_forget()
531
532     current_page = Frame(root)
533     current_page.pack()
534
535     hideAddCustomerDetailsButton()
536     hideLogoutBtn()
537     hideShowAllCustomerDetailsNavigationButton()
538     hideShowAllCustomerDetailsNavigationButton()
539     hideAddCustomersNavigationButton()
540     hideShowAllUsersButton()
541     hideShowAllCustomerDetailsNavBtn()
542     hide_update_password_button()
543     hideDeleteUserButton()
544

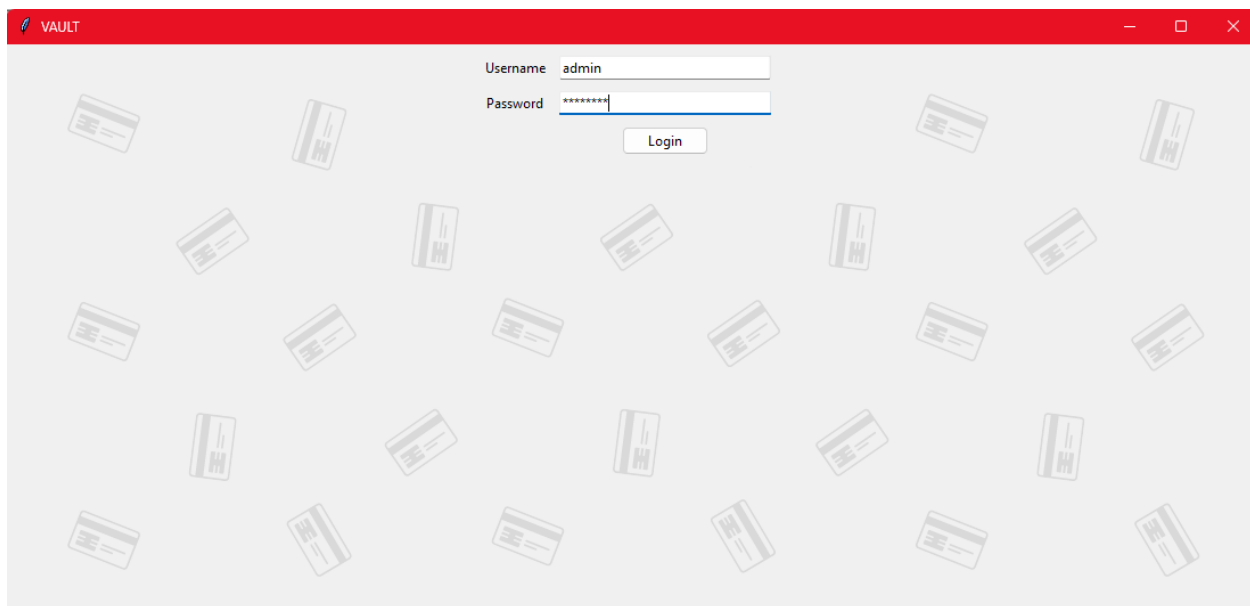
```

```

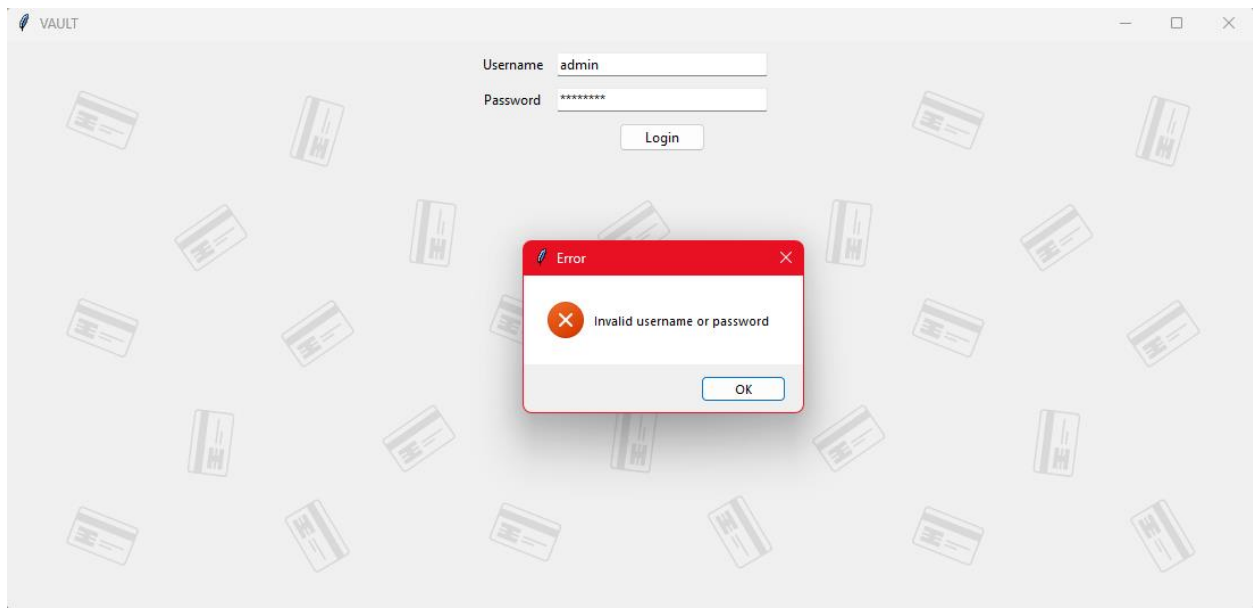
545 getUsernameLabel = Label(current_page, text="Username")
546 getUsernameLabel.grid(row=0, column=0, pady=(10, 0))
547 getPasswordLabel = Label(current_page, text="Password")
548 getPasswordLabel.grid(row=1, column=0, pady=(10, 0))
549 getRoleLabel = Label(current_page, text="Role")
550 getRoleLabel.grid(row=2, column=0, pady=(10, 0))
551
552 getUsername = Entry(current_page, width=30)
553 getUsername.grid(row=0, column=1, padx=10, pady=(10, 0))
554 getPassword = Entry(current_page, width=30, show="*")
555 getPassword.grid(row=1, column=1, padx=10, pady=(10, 0))
556 # Combobox for role selection
557 role_options = ["admin", "customer"]
558 getRole = ttk.Combobox(current_page, values=role_options, width=27, state="readonly") # Set state to "readonly"
559 getRole.grid(row=2, column=1, padx=10, pady=(10, 0))
560
561
562 submitBtn = Button(current_page, text="Submit", command=submitUser)
563 submitBtn.grid(row=3, column=1, pady=10)
564 # hideShowAllUsersButton()
565

```

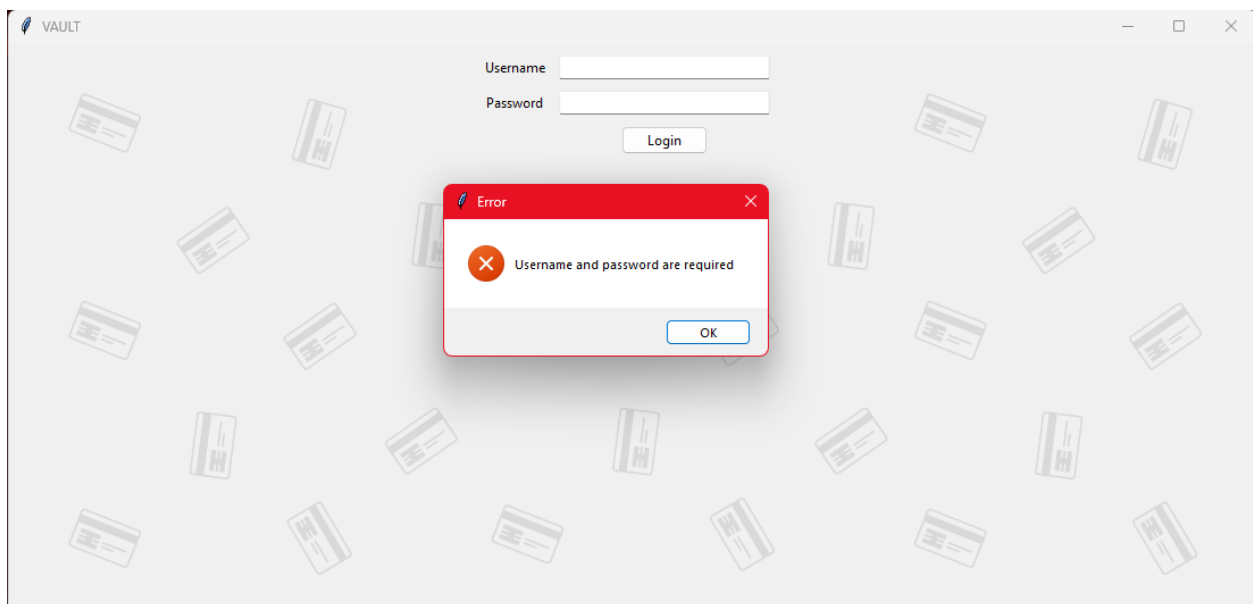
They are then taken to the login page to get authenticated



Error handling put in place



Form validation put in place



Login Page function

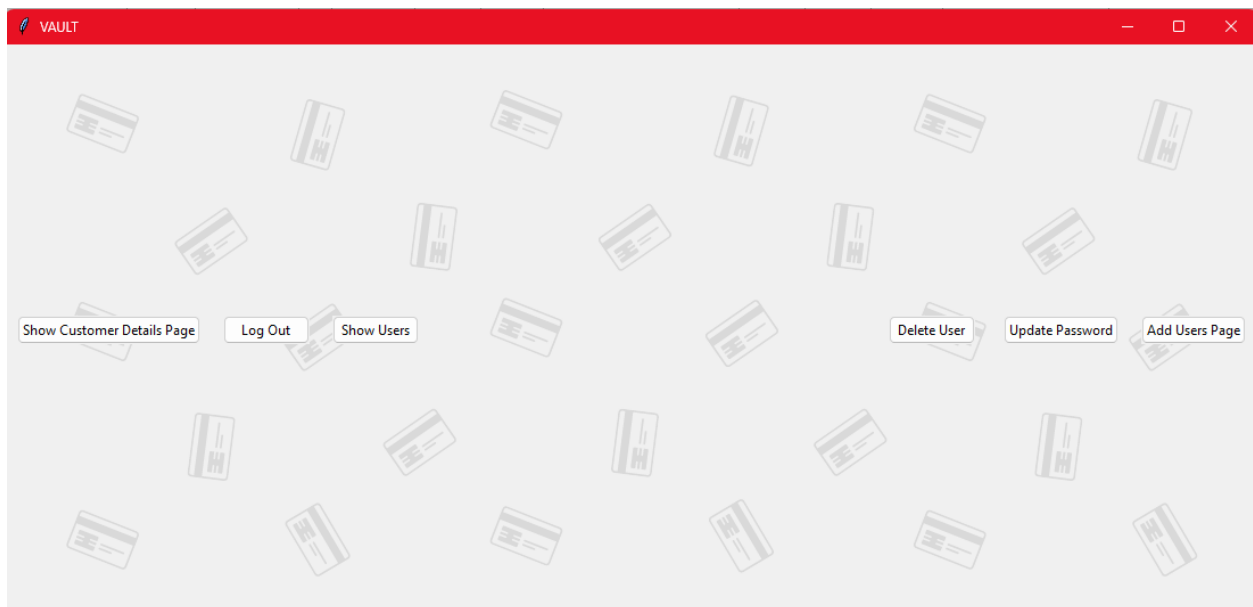
```
ccv.py M X
q2-credit-card-vault > ccv.py > loginPage
566 def loginPage(): You, 2 weeks ago • add auth, SELECT and INSERT
567     def authenticateUser():
568         global authenticated_user, customerUserID
569         username = getUsername.get().strip()
570         password = getPassword.get().strip()
571
572         if not (username and password):
573             messagebox.showerror("Error", "Username and password are required")
574             return
575
576         # Hash the provided password
577         hashed_password = hash_password(password)
578
579         try:
580             conn = sqlite3.connect('creditcard_vault.db')
581             c = conn.cursor()
582
583             c.execute("SELECT * FROM USERS WHERE username = ?", (username,))
584             user = c.fetchone()
585
586             if user:
587                 # Retrieve the hashed password stored in the database
588                 stored_password = user[2] # Assuming password is stored at index 2
589                 # Compare the hashed password from the database with the provided hashed password
590                 if hashed_password == stored_password:
591                     authenticated_user = user[0] # Set authenticated_user to userID
592                     print("Authenticated user has userID: ", authenticated_user)
593
594                     # Retrieve user role
595                     user_role = user[3]
596
597                     # Print out who is logged in based on their role and user ID
598                     print(f"Logged in as UserID: {authenticated_user}, Role: {user_role}")
599
600                     # Check if the authenticated user has already added their customer details
601                     c.execute("SELECT COUNT(*) FROM CUSTOMERS WHERE userID = ?", (authenticated_user,))
602                     customer_details_count = c.fetchone()[0]
603
604                     if customer_details_count > 0:
605                         # If customer details exist, show the customer details page
606                         showCustomerDetailsPage()
607                     else:
608                         # If no customer details exist, show the add customer details page
609                         addCustomerInfoPage()
610                     return
611             messagebox.showerror("Error", "Invalid username or password")
612         except sqlite3.Error as e:
```

```

613         messagebox.showerror("Error", f"Database error: {e}")
614     finally:
615         conn.close()
616
617     global current_page
618     if current_page:
619         current_page.pack_forget()
620
621     current_page = Frame(root)
622     current_page.pack()
623
624     hideAddCustomersNavigationButton()
625     hideAddCustomerDetailsButton()
626     hideLogoutBtn()
627     hideShowAllCustomerDetailsNavBtn()
628     hideAddUsersNavigationBtn()
629     hideShowAllUsersButton()
630     hide_update_password_button()
631     hideDeleteUserButton()
632
633     getUsernameLabel = Label(current_page, text="Username")
634     getUsernameLabel.grid(row=0, column=0, pady=(10, 0))
635     getPasswordLabel = Label(current_page, text="Password")
636     getPasswordLabel.grid(row=1, column=0, pady=(10, 0))
637
638     getUsername = Entry(current_page, width=30)
639     getUsername.grid(row=0, column=1, padx=10, pady=(10, 0))
640     getPassword = Entry(current_page, width=30, show="*")
641     getPassword.grid(row=1, column=1, padx=10, pady=(10, 0))
642
643     LoginBtn = Button(current_page, text="Login", command=authenticateUser)
644     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

A screenshot of a web application window titled "VAULT". The window has a red header bar with the title and standard window controls (minimize, maximize, close). The background is a light gray with a pattern of faint, tilted credit card icons. In the center, there is a login form with three fields: "Username" containing the text "alice", "Password" containing six asterisks "*****", and "Role" with a dropdown menu showing "customer". Below these fields is a "Submit" button.

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

A screenshot of the same "VAULT" web application window. The login form now has a "Login" button instead of "Submit". The "Username" field still contains "alice", and the "Password" field contains "*****". The "Role" dropdown menu is no longer visible. The background pattern of credit card icons remains the same.

They are prompted to add their own details.

The screenshot shows a web application window titled "VAULT". The background is a light gray with a pattern of faint credit card icons. In the center, there is a form with the following fields and values:

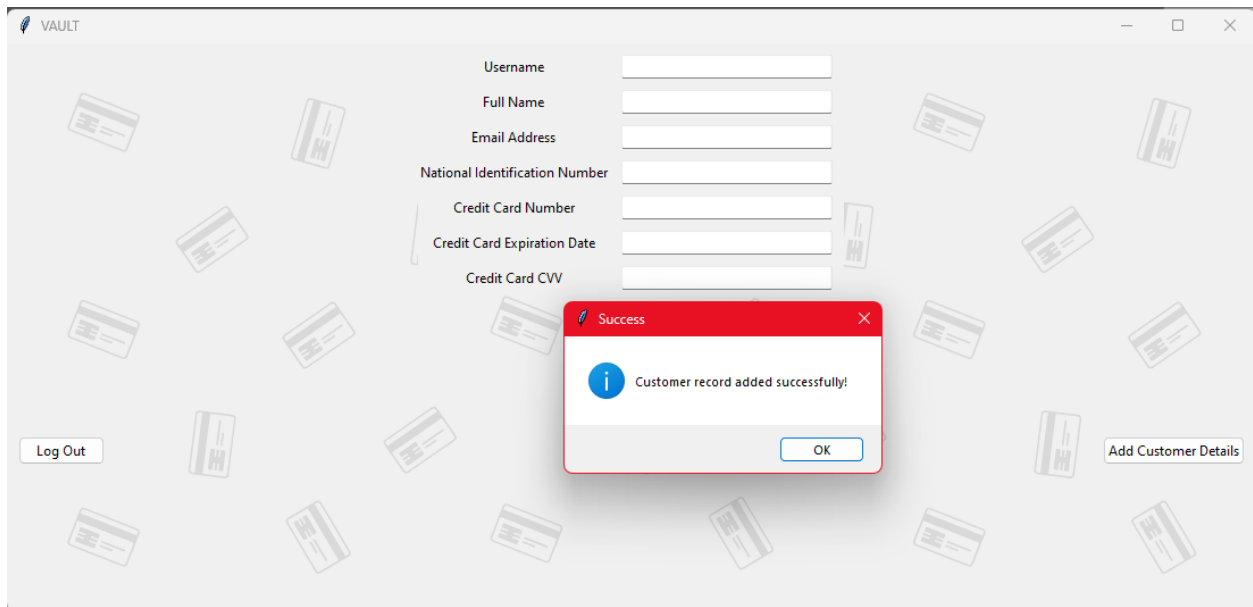
Field	Value
Username	alice
Full Name	Alice Doe
Email Address	adoe@gmail.com
National Identification Number	282384743
Credit Card Number	397424928374238
Credit Card Expiration Date	20/28
Credit Card CVV	325

At the bottom left, there is a "Log Out" button. At the bottom right, there are two buttons: "Add Customer Details" and "Update Password".

Form validation put in place

This screenshot shows the same VAULT application window, but the form fields are now empty. An error dialog box is displayed in the center of the screen. The dialog box has a red header with the word "Error" and a close button (X). The main content area of the dialog box contains a red circle with a white 'X' icon and the text "Error occurred: All fields are required". At the bottom right of the dialog box is an "OK" button.

Feedback message put in place




```

177 def define_widgets(root):
178     entry_widgets = {}
179     labels = [
180         "Username", "Full Name", "Email Address", "National Identification Number",
181         "Credit Card Number", "Credit Card Expiration Date", "Credit Card CVV"
182     ]
183     for i, label in enumerate(labels):
184         entry = Entry(root, width=30)
185         entry.grid(row=i, column=1, padx=10, pady=(10, 0))
186         entry_widgets[label] = entry
187
188         label_widget = Label(root, text=label)
189         label_widget.grid(row=i, column=0, pady=(10, 0))
190
191     return entry_widgets
192

```

ccv.py M X

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

```

340 def addCustomerInfoPage():    You, last week + make it such that if authenticated
341     global current_page, entry_widgets
342     if current_page:
343         current_page.pack_forget()
344
345     current_page = Frame(root)
346     current_page.pack()
347
348     #entry_widgets = define_widgets(current_page)
349     #showAddCustomerDetailsBtn()
350     showAllCustomerDetailsButton()
351     showAddUsersNavigationBtn()
352     showLogoutBtn()
353     show_update_password_button()
354     showDeleteUserButton()
355
356     # Check if the authenticated user is a customer
357     global authenticated_user
358     if authenticated_user:
359         conn = sqlite3.connect('creditcard_vault.db')
360         c = conn.cursor()
361         c.execute("SELECT role FROM USERS WHERE userID = ?", (authenticated_user,))
362         role = c.fetchone()[0]
363         conn.close()
364
365     # If the role is 'customer', hide the "Add Users Page" button
366     if role == 'customer':
367         entry_widgets = define_widgets(current_page)
368         showAddCustomerDetailsBtn()
369         hideDeleteUserButton()
370         hideAddUsersNavigationBtn()
371         hideShowAllUsersButton()
372         hideShowAllCustomerDetailsNavBtn()
373     else:
374         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

```

```

257 def showCustomerDetailsPage():
258     global authenticated_user, current_page
259     conn = sqlite3.connect('creditcard_vault.db')
260     c = conn.cursor()
261
262     try:
263         print("Authenticated user ID:", authenticated_user)
264
265         # Retrieve customer ID based on user ID
266         c.execute("SELECT customerID FROM CUSTOMERS WHERE userID = ?", (authenticated_user,))
267         customer_id = c.fetchone()[0]
268
269         # Query customer details based on the retrieved customer ID
270         c.execute("SELECT * FROM CustomerCreditCardView WHERE customerID = ?", (customer_id,))
271         customer_details = c.fetchone()
272
273         # Clear current page if it exists
274         if current_page:
275             current_page.pack_forget()
276
277         # Display customer details
278         if customer_details:
279             current_page = Frame(root)
280             current_page.pack()
281
282             customer_id = customer_details[0]
283             name = customer_details[1]
284             email = customer_details[2]
285             id_no = customer_details[3]
286             card_number = customer_details[4]
287             expiration_date = customer_details[5]
288             cvv = customer_details[6]
289
290             # Decrypt sensitive information
291             try:
292                 id_no = decrypt_sensitive_information(id_no, "secretKey")
293                 card_number = decrypt_sensitive_information(card_number, "secretKey")
294                 expiration_date = decrypt_sensitive_information(expiration_date, "secretKey")
295                 cvv = decrypt_sensitive_information(cvv, "secretKey")
296             except Exception as e:
297                 print("Error decrypting customer details:", e)
298
299             # Labels to display customer details
300             Label(current_page, text="Customer Details", font=("Helvetica", 16, "bold")).grid(row=0, columnspan=2)
301             Label(current_page, text=f"Customer ID: {customer_id}").grid(row=1, column=0, sticky="w")
302             Label(current_page, text=f"Name: {name}").grid(row=2, column=0, sticky="w")
303             Label(current_page, text=f"Email: {email}").grid(row=3, column=0, sticky="w")

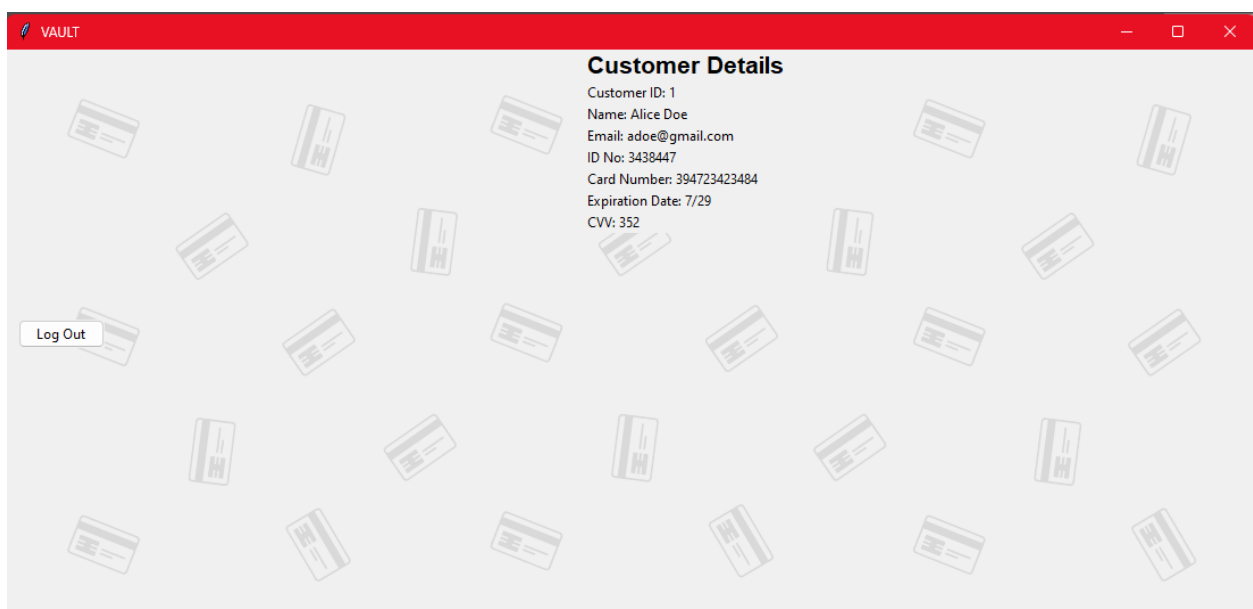
```

```

304 Label(current_page, text=f"ID No: {id_no}").grid(row=4, column=0, sticky="w")
305 Label(current_page, text=f"Card Number: {card_number}").grid(row=5, column=0, sticky="w")
306 Label(current_page, text=f"Expiration Date: {expiration_date}").grid(row=6, column=0, sticky="w")
307 Label(current_page, text=f"CVV: {cvv}").grid(row=7, column=0, sticky="w")
308
309 # Hide the form for adding customer details
310 hideAddCustomerDetailsButton()
311
312 # Hide all other buttons except "Logout"
313 hideAddUsersNavigationBtn()
314 hideAddCustomersNavigationButton()
315 hideShowAllCustomerDetailsNavBtn()
316 hideShowAllUsersButton()
317 hideDeleteUserButton()
318 showLogOutBtn()
319 hide_update_password_button()
320
321 else:
322     print("No customer details found for the authenticated user.")
323
324 # Clear current page if it exists
325 if current_page:
326     current_page.pack_forget()
327
328 # Display message
329 current_page = Frame(root)
330 current_page.pack()
331
332 # Display message
333 Label(current_page, text="No customer details found.").grid(row=0, columnspan=2)
334
335 except sqlite3.OperationalError as e:
336     print(f"Error querying customer details: {e}")
337
338 conn.close()
339

```

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

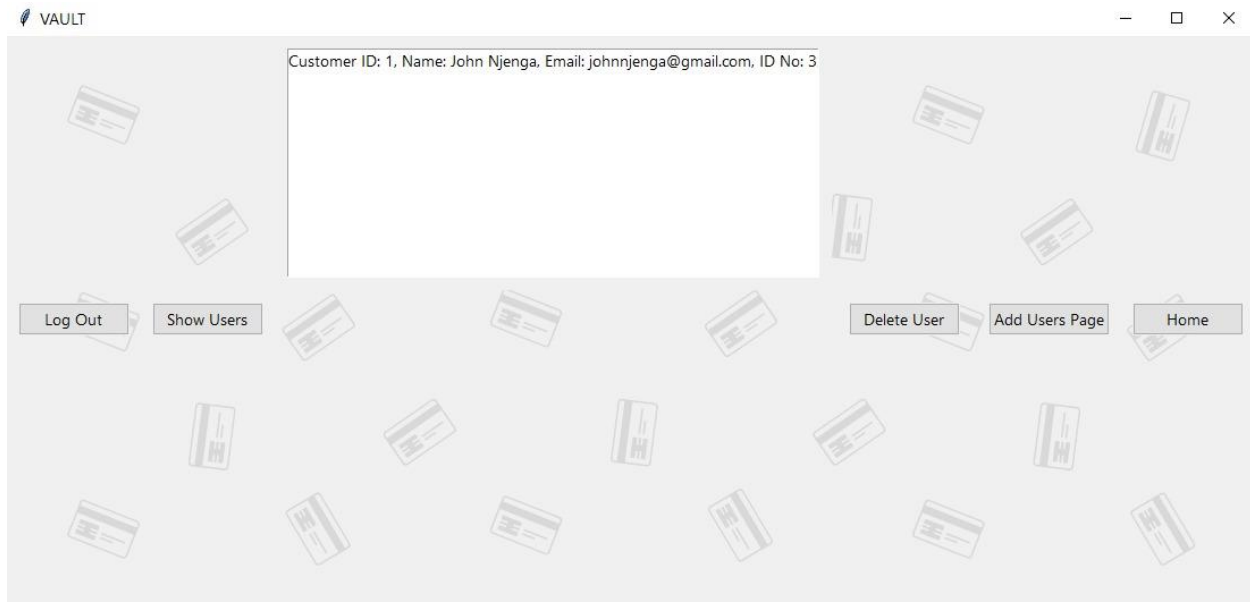
The application can be entered using a username and password for different type of users. The password should be hashed using SHA-2.

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.



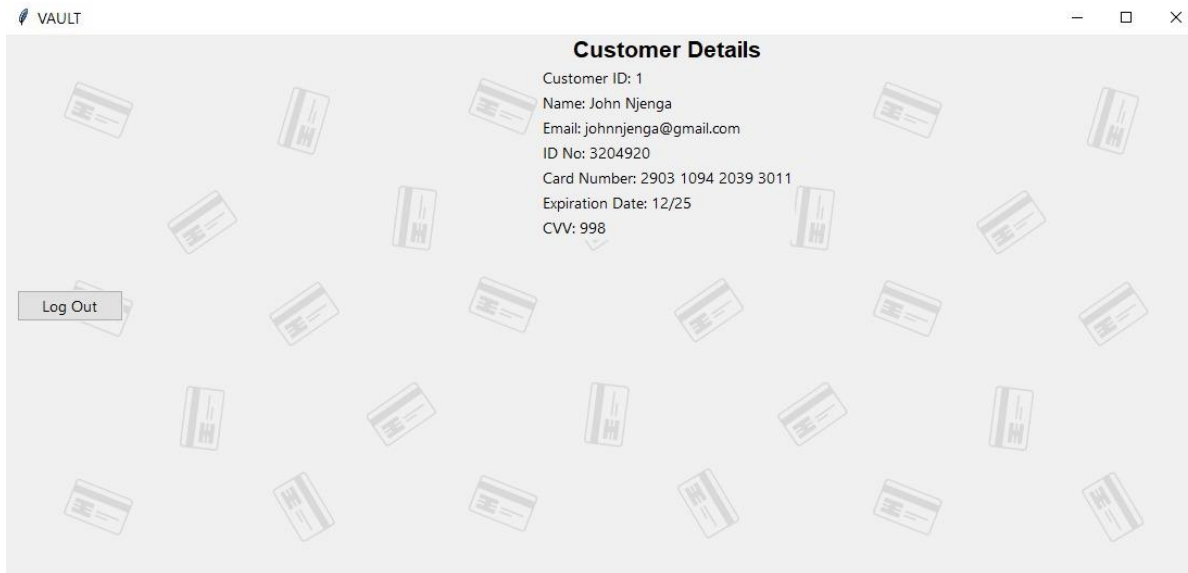
From the image above, only one customer has registered their customer details.

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:

```

ssl_activity > private-key.key
You, last week | 1 author (You)
1  -----BEGIN PRIVATE KEY-----
2  MIIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKggggSkAgEAAoIBAQCtm26e8IqYtLgK
3  e91m96Cp/YiAkHg2GZmAc/ZwFBlzLrSK/6DZJoe5QtRqA30ceshaJnpXXsZXaAl
4  ZH2z6mh+4mM06rrjDU81YXXzXnutEcvb5uSqzh9XqZuhPffMKet+WdsxxGV/FC5l
5  iv19sJYS6wQqLcuQ/FzUJkbARAYmkwyr2nwTT03h9kdCzyWGaQsdyBZJcPqf0wV0
6  ovH1uUkmetx7S042wkSFPHE4syrfP6eom/IJwvLQxvXncrixldFtKdXlEV4uvPNn
7  UfHPvyU3JGP17Qxv50G09MBpP6hS21BFaJK4a32rs2DruafuxLkRl9T1ZTw3+FBY
8  tgtteWhfAgMBAACGgEAMgUHHdc7b3+fIUIn++tcCK5qXnSBlkDR8nLSLDCbWYFf
9  0H9pg00C90TUw04bmZpJPSkqaNbhClJN8HlLrEafjopMBDr+d6hhhiHSA36l
10 qHHBqDEJWawEZqbo2B2w+xa2d92y0MFLS+bMtQmoUPCeu6f60G1U1DF8M6IoSMi
11 k6jMQK5bYv+THK2o9GImzBsCK07C3r8yHqs3DB7o4RbdsKk6AmV/G0cNd2zVu+I
12 fwP55aL3pZRYXJ6dU/PqjEPmvRYQhd9oIdTnJ3uE9qREQJL20vARcAbzWycMbLwQ
13 m18dNp/F6uQznv/30CJizLwp6rR7CweStewV4vAc9QKBgQDdLwRVWxzmgFEwQZR
14 efpmp6LmY05dtZlCtF9l6QpMrTRY+c2B337eUihUs1FHxpukBw+VGpiI17defaA
15 FpHeVYP4BsGkDbZGSt3i2vK97/vPV40p0e7phBojA+tnVPW95vYywh/jXF0q+I+
16 cKwlvUdp8q+1CB3mMJNNYwbXxQKBgQDI7zyUX8xqy0s+Yl1gaMnKkQa6Iw8KLQd
17 ACQFbpItxpLGc0g1o1WsqGdymthbvE8f6VDAc6UX2MML43uAMDgvy0yGIoG9Q0ut
18 yaTtkmMx+sZPD26ghiIGuCd3lonr0JUQgPMC9xnQubGKhbIF4JoY6dcrbXSm5dkM
19 g+2nbNdd0wKBgQDLESSQW495vnxpRdHp0exTva8iDnljRmFhWZkgR0kooyZ1sCX
20 sSKgaJuAkBvASgDpCldFSTfmzMN/bNa9lGYAZs/pqxXuwMS6qF3yn68ZT0x8m3
21 9iyP0nqsEXM/nUcLALXDxwzwpJynVCbewjel3Dh0np1ksAqS4tvY2QWmB5QKBgBlE
22 88QUGBXPdy1SNsKf1Yj97h7AsdrySL6whGZg4WaJCBau2GqbP/vB5jV6mmy3j7G
23 Q/b44yieqwo0QCLQc9e9RUQuiTkVRF0+L7/7kuDVRkBTVTkuNkmWX1tL7v9wpt
24 bcG7F5xeRgeal/7b/DkXFZ8bI/aK4Rw0cGyKdqsXa0GBAK4PbXk6SU4kk9s200oZ
25 BVXy/fYxklDbTFcmLJY3a8A9iXeISKA3I3XmBmX2Yo2bfe27suLNRwLqceX0NL7Q
26 71QYPwj9gW95mPDmXdd9ckEuV5JzVD9Nl04Vgpub8MCthN0ehasxdlM1+RjYTJm
27 UkzhxqKrgtKeoJhLiznMGqSl
28 ✨-----END PRIVATE KEY-----
29

```

The certificate request:

```

ssl_activity > csr.txt
You, last week | 1 author (You)
1  -----BEGIN CERTIFICATE REQUEST-----
2  MIIC6DCCAdACAQAwGAIxGAJBgNVBAYTAktFMRAwDgYDVQIDAdOYwlyb2JpMRAw
3  DgYDVQOHDAAdOYwlyb2JpMRYwFAYDVQQKDA1BZHJpYW4gTXVvYyYwLmRYwFAYDVQQL
4  DA1BZHJpYW4gTXVvYyYwLmRYwFAYDVQQDDA1BZHJpYW4gTXVvYyYwLmScwJQYJKoZI
5  hvCNAAkBFhHhHJpYW5tdXJhZ2UyMUBnbWVpbC5jb20wggeiMA0GCSqGSIb3DQEB
6  AQUAA4IBDwAwggEKAoIBAQCtm26e8IqYtLgKe91m96Cp/YiAkHg2GZmAc/ZwFBl
7  zLrSK/6DZJoe5QtRqA30ceshaJnpXXsZXaAlZH2z6mh+4mM06rrjDU81YXXzXnut
8  Ecvb5uSqzh9XqZuhPffMKet+WdsxxGV/FC5liv19sJYS6wQqLcuQ/FzUJkbARAYm
9  kwyr2nwTT03h9kdCzyWGaQsdyBZJcPqf0wV0ovH1uUkmetx7S042wkSFPHE4syrf
10 P6eom/IJwvLQxvXncrixldFtKdXlEV4uvPNnUfHPvyU3JGP17Qxv50G09MBpP6hS
11 z1BFaJK4a32rs2DruafuxLkRl9T1ZTw3+FBYtgtteWhfAgMBAAGGADANBgkqhkiG
12  9w0BAQsFAA0CAQEAOQTHIHQIgSq+6FHd21RgmkuNmDxyFEf1U30mLj52fhSuGgZz
13  cu/Vvy9IVliC7TrGFZ327Ed2vPTpswoWpcg0yn8gpFG+0f5xKh/X45QhBgMyrkJI
14  MbRQX+yZ9XNVTCe9reDBUTfmMjPTacCUHmXML4UMZ+70m88z7tAoLTnq+7NSisaw
15  8QXGQmxyQDlNSftvurLouLkUEZr7go/LLhMzHQcYd0StwQJ6eoQxh04MsLQ6mYw/
16  ZWH4lB00ZnHwHjAcjrlEXcJmn8CXbnxpttji3aUr9Wkg1B5WwYubzWCD+08G4rLX
17  hJv72lcMGt81KrmVRMTIdD5YitbHyZpl0rtA3A==
18  ✨-----END CERTIFICATE REQUEST-----
19

```

The SSL certificate:

```

ssl_activity > ≡ crt.txt
    You, last week | 1 author (You)
1  -----BEGIN CERTIFICATE-----      You, last week * generate SSL ce
2  MIIDzTCCArUCFDqs7HKa7NGpq5NSnluFN52bGNdYMA0GCSqGSIb3DQEBGwUAMIGi
3  MQswCQYDVQQGEwJLRTEQMA4GA1UECAwHTmFpcm9iaTEQMA4GA1UEBwwHTmFpcm9i
4  aTEWMBQGA1UECgwNQWRyaWFuIE11cmFnZTEWMBQGA1UECwwNQWRyaWFuIE11cmFn
5  ZTEWMBQGA1UEAwwNQWRyaWFuIE11cmFnZTENMCUGCSqGSIb3DQEJARYYYWRyaWFu
6  bXVyYWdlMjFAZ21haWwY29tMB4XDTI0MDMzMTE4MDgwOVoXDTI1MDMzMTE4MDgw
7  OVowgaIxZzAjbGVBAYTAktFMRAwDgYDVQQIDAd0YWlyb2JpMRAwDgYDVQQHDA0
8  YWlyb2JpMRYwFAYDVQQKDA1BZHJpYW4gTXVvYWdlMRYwFAYDVQQDA1BZHJpYW4g
9  TXVvYWdlMRYwFAYDVQQDDA1BZHJpYW4gTXVvYWdlMScwJQYJKoZIhvcNAQkBFhhh
10 ZHJpYW50dXJhZ2UyMUBnbWFPbC5jb20wgGEiMA0GCSqGSIb3DQEBQUAA4IBDwAw
11 ggEKAoIBAQCtm26e8IqYtLgKe91m96Cp/YiAkHg2GZzmAc/ZwFBlzLrSK/6DZJoe
12 5QtRqA30ceshaJnpxxsZXaALZH2z6mh+4mM06rrjDU81YXXzXnutEcvb5uSQzh9X
13 qZuhPffMKEt+WdsxxGV/FC5liv19sJYS6wQqLcuQ/FzUJkbARAYmkwyr2nwTT03h
14 9kdCzyWGaGsdYBZJcPqf0wV0ovH1uUkmetx7SQ42wkSFPHE4syrfP6eom/IJwvLQ
15 xvXncrixldFtKdXlEV4uvPNnUfHPvyU3JGP17Qxv50G09MBpP6hSz1BFaJK4a32r
16 s2DruafuxLkRl9T1ZTw3+FBYtgtteWhfAgMBAAEwDQYJKoZIhvcNAQELBQADggEB
17 AE4M0/iotI1DRfmkcf0ZVh90Qoq0eSkj8ZoaEbFprRKC5j5H5FJpJsiREeRo7zmf
18 vZgEC9sS9xlXBabRr6c/BvRgy5dShWoJ1xKKUbwR0AKztRsyM9TR7lTVvpEjC1MQ
19 D0jQmh7c/y/LR1veQYlI24aoaPJf8oN3dhWRZYbahMm/s8tq/yazZK4lcvpEpJkH
20 PYRHiv5uU2qVllwzNsBRchxcVP0LLqFUR9v0cwaXi+7N10tWahHNYwjK7mIXKeS6
21 fNn4iUmFLUt3qvR81XA3sVsWLo/EB2ASILZ/oNAmiCSqfe0Mf0p210nNvz8PFzR4
22 9zGDlaNv5iJWwJ5ab/93PFU=
23 -----END CERTIFICATE-----
24

```