United States International University-Africa

SUMMER 2024

APT3090 CRYPTOGRAPHY AND NETWORK SECURITY PROJECT

Group Members

Report for RSA Encrypted Client-Server Communication Programs

Overview

These Python programs implement a client-server communication system using RSA encryption. The server (named **Alice**) and client (named **Bob**) exchange messages securely by encrypting and decrypting them with RSA public and private keys. This ensures confidentiality, as only the intended recipient can decrypt the messages.

Alice (Server) Program

Alice's program performs the following tasks:

1. Generates RSA keys (public and private).
2. Listens for incoming client connections.
3. Exchanges public keys with Bob.
4. Receives encrypted messages from Bob, decrypts them, and responds with encrypted messages.

Alice (Server)

```python
import socket
import signal
from Crypto.Util.number import getPrime, inverse, bytes_to_long, long_to_bytes

# RSA Key generation
def generate_rsa_keys(key_size=1024):
    e = 65537
    p = getPrime(key_size // 2)
    q = getPrime(key_size // 2)
    n = p * q
    phi_n = (p - 1) * (q - 1)
    d = inverse(e, phi_n)
    return (e, d, n, p, q, phi_n), (e, n)

# Encryption and decryption functions
def encrypt(m, e, n):
    return pow(bytes_to_long(m), e, n)

def decrypt(c, d, n):
    return long_to_bytes(pow(c, d, n))

HOST = '127.0.0.1'
```

```python
PORT = 65432

server_private_key, server_public_key = generate_rsa_keys()
(e, d, n, p, q, phi_n) = server_private_key
(e, n) = server_public_key

print(f"Server's Public key: (e={e}, n={n})")

def handle_interrupt(sig, frame):
    print("Server shutting down...")
    server_socket.close()
    exit(0)

signal.signal(signal.SIGINT, handle_interrupt)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
    server_address = (HOST, PORT)
    print('Server listening on %s port %s' % server_address)
    server_socket.bind(server_address)
    server_socket.listen()

    while True:
        print("Waiting for a connection...")
        conn, addr = server_socket.accept()
        with conn:
            print('Connected by', addr)

            # Exchange public keys
            client_public_key = conn.recv(1024).decode()
            conn.sendall(f"{e},{n}".encode())

            client_e, client_n = map(int, client_public_key.split(','))
            print(f"Client's Public key: (e={client_e}, n={client_n})")

            while True:
                # Receive encrypted data from client
                encrypted_data = conn.recv(1024).decode()
                if not encrypted_data:
                    print("Client disconnected.")
                    break

                print(f"Received encrypted message: {encrypted_data}")
                decrypted_data = decrypt(int(encrypted_data), d, n)
                message = decrypted_data.decode()
                print(f"Decrypted message: {message}")
```

```
                if message.lower() == 'quit':
                    print("Client requested to quit. Closing connection.")
                    break

                # Send a response
                response = input("Enter your message (or 'quit' to exit): ")
                encrypted_response = encrypt(response.encode(), client_e,
client_n)
                print(f"Sending encrypted message: {encrypted_response}")
                conn.sendall(str(encrypted_response).encode())

                if response.lower() == 'quit':
                    print("Closing connection.")
                    break

        print("Connection closed. Waiting for new connections...")
```

Bob (Client)

Bob's program performs the following tasks:

1. Generates RSA keys (public and private).
2. Connects to Alice.
3. Exchanges public keys with Alice.
4. Sends encrypted messages to Alice and receives encrypted responses.

```
import socket
from Crypto.Util.number import getPrime, inverse, bytes_to_long, long_to_bytes

# RSA Key generation
def generate_rsa_keys(key_size=1024):
    e = 65537
    p = getPrime(key_size // 2)
    q = getPrime(key_size // 2)
    n = p * q
    phi_n = (p - 1) * (q - 1)
    d = inverse(e, phi_n)
    return (e, d, n, p, q, phi_n), (e, n)

# Encryption and decryption functions
def encrypt(m, e, n):
    return pow(bytes_to_long(m), e, n)
```

```python
def decrypt(c, d, n):
    return long_to_bytes(pow(c, d, n))

HOST = '127.0.0.1'
PORT = 65432

client_private_key, client_public_key = generate_rsa_keys()
(e, d, n, p, q, phi_n) = client_private_key
(e, n) = client_public_key

print(f"Client's Public key: (e={e}, n={n})")

try:
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        server_address = (HOST, PORT)
        print('Connecting to %s port %s' % server_address)
        s.connect(server_address)

        # Exchange public keys
        s.sendall(f"{e},{n}".encode())
        server_public_key = s.recv(1024).decode()
        server_e, server_n = map(int, server_public_key.split(','))

        print(f"Server's Public key: (e={server_e}, n={server_n})")

        while True:
            # Send message to server
            message = input("Enter your message (or 'quit' to exit): ")
            encrypted_message = encrypt(message.encode(), server_e, server_n)
            print(f"Sending encrypted message: {encrypted_message}")
            s.sendall(str(encrypted_message).encode())

            if message.lower() == 'quit':
                print("Closing connection.")
                break

            # Receive response from server
            encrypted_response = s.recv(1024).decode()
            if not encrypted_response:
                print("Server disconnected.")
                break

            print(f"Received encrypted message: {encrypted_response}")
            decrypted_response = decrypt(int(encrypted_response), d, n)
            print(f"Decrypted message: {decrypted_response.decode()}")
```

```
            if decrypted_response.decode().lower() == 'quit':
                print("Server requested to quit. Closing connection.")
                break

except ConnectionError:
    print('Failed to connect to the server.')
except KeyboardInterrupt:
    print('Client interrupted.')

print("Connection closed.")
```

Interaction between Alice and Bob

1. **RSA Key Generation:** Both Alice and Bob generate their own RSA keys upon startup. The public keys are exchanged when a connection is established.

- **Parameters:**
  - `key_size`: Size of the RSA key in bits (default is 1024).
- **Returns:**
  - A tuple containing the private key `(e, d, n, p, q, phi_n)` and public key `(e, n)`.

Explanation:

- `e` is the public exponent, typically set to 65537.
- `p` and `q` are large prime numbers.
- `n` is the modulus for encryption and decryption.
- `phi_n` is the totient of `n`.
- `d` is the private exponent, calculated as the modular inverse of `e` modulo `phi_n`.

Encryption and Decryption Functions

Encryption Function:

```
def encrypt(m, e, n):
    return pow(bytes_to_long(m), e, n)
```

- **Parameters:**
  - `m`: Message to be encrypted (as bytes).
  - `e`: Public exponent.
  - `n`: Modulus.
- **Returns:**
  - Encrypted message (integer).

Decryption Function:

```
def decrypt(c, d, n):
    return long_to_bytes(pow(c, d, n))
```

- **Parameters:**
  - `c`: Encrypted message (integer).
  - `d`: Private exponent.
  - `n`: Modulus.
- **Returns:**
  - Decrypted message (as bytes).

Explanation:

- `encrypt()` converts the message to an integer, encrypts it using the public key, and returns the encrypted integer.
- `decrypt()` reverses this process by decrypting the integer message using the private key and returning the original bytes.

2. **Connection Establishment:** Bob connects to Alice using Alice's IP address and port number. Alice listens for incoming connections.
3. **Public Key Exchange:** Once connected, Alice and Bob exchange their public keys. This allows them to encrypt messages intended for each other.
4. **Message Exchange:**
   - Bob sends an encrypted message to Alice.
   - Alice decrypts the message using her private key and reads it.
   - Alice then sends an encrypted response back to Bob.
   - Bob decrypts the response using his private key and reads it.
5. **Termination:** The connection can be terminated by sending a message with the content "quit" from either Alice or Bob. This gracefully closes the connection.
6. This setup ensures secure communication by leveraging RSA encryption for the message payloads, providing confidentiality and integrity.

Example Interaction

Alice Sets Up the Server:

- Alice starts her program, generating her RSA keys and setting up the server to listen on `127.0.0.1:65432`.

```
PS C:\Users\frank> python -u "c:\Users\frank\OneDrive\Desktop\Client-Server\RSAServer.py"
Server's Public key: (e=65537, n=67914571179758922468726013215956851393545796957269832820198025011663344186743356710225308047381739238884400395840400175605712422728738759171418511162140502
919353941823856959624906016366977907348908792335708985783403962391771492123869317801357972074234827884977482273530491014920178498945247494502407083471753)
Server listening on 127.0.0.1 port 65432
Waiting for a connection...
```

Bob Connects to Alice:

- Bob starts his program, generating his RSA keys and connecting to Alice.

```
PS C:\Users\frank> python -u "c:\Users\frank\OneDrive\Desktop\Client-Server\RSAClient.py"
Client's Public key: (e=65537, n=894541728757810582038770263537420747717160541588863321956248410625271332009053571984703819957785806893770553479427098882507492160861834464642722768485602766658277632449245155225538074097161459274980943153989398495439878201023220080418159579476705272261136235961666365841973239780899564169530283678303786522 71)
Connecting to 127.0.0.1 port 65432
Server's Public key: (e=65537, n=679145711797589224687260132159568513935457969572698328201980250116633441867433567102253080473817392388844003958404001756057124227287387591714185111621405029193539418238569596249060163669779073489087923357089857834039623917714921238693178013579720742348278849774822735304910149201784989452474945024070834717 53)
Enter your message (or 'quit' to exit):
```

Public Key Exchange:

- Alice and Bob exchange public keys.

Alice's Output:

```
Waiting for a connection...
Connected by ('127.0.0.1', 52904)
Client's Public key: (e=65537, n=894541728757810582038770263537420747717160541588863321956248410625271332009053571984703819957785806893770553479427098882507492160861834464642722768485602766658277632449245155225538074097161459274980943153989398495439878201023220080418159579476705272261136235961666365841973239780899564169530283678303786522 71)
```

Bob’s output:

```
Connecting to 127.0.0.1 port 65432
Server's Public key: (e=65537, n=679145711797589224687260132159568513935457969572698328201980250116633441867433567102253080473817392388844003958404001756057124227287387591714185111621405029193539418238569596249060163669779073489087923357089857834039623917714921238693178013579720742348278849774822735304910149201784989452474945024070834717 53)
Enter your message (or 'quit' to exit):
```

Message Exchange:

- **Bob sends a message:**

```
Enter your message (or 'quit' to exit): Hello Alice. Here is my number 079999999999
Sending encrypted message: 676726164263468017120791936990207091481508761641340307472192613496910250718237361170454709643422563986497220607519884007613176548018580084577185681165857906939895127537858855200789230450539271565550541034492495437063560654082483673288165740820101499923692473713118113783280176407180679603272057997557359794 81
```

Alice receives and decrypts:

```
Received encrypted message: 676726164263468017120791936990207091481508761641340307472192613496910250718237361170454709643422563986497220607519884007613176548018580084577185681165857906939895127537858855200789230450539271565550541034492495437063560654082483673288165740820101499923692473713118113783280176407180679603272057997557359794 81
Decrypted message: Hello Alice. Here is my number 079999999999
```

Alice sends a response:

```
Enter your message (or 'quit' to exit): Thank you Bob. Hit you up soon.
Sending encrypted message: 771342574014714691087494555734646858375130527884903827534009249660215184066259143558100264598878251084937615209229408740592937490314479447053984277312676967842865349991019916642907011119679576709546335463777799449976608960773385658801406667842583125355702789514447321366994642401906602569693063470849132894 26
```

Bob receives and decrypts:

```
Received encrypted message: 771342574014714691087494555734646858375130527884903827534009249660215184066259143558100264598878251084937615209229408740592937490314479447053984277312676967842865349991019916642907011119679576709546335463777799449976608960773385658801406667842583125355702789514447321366994642401906602569693063470849132894 26
Decrypted message: Thank you Bob. Hit you up soon.
```

Termination:

- Either Alice or Bob can send the message "quit" to terminate the connection.

Bob's Output when sending "quit":

```
Enter your message (or 'quit' to exit): quit
Sending encrypted message: 601059924900531785821756058940217329048094641797536904656719629724613712103245545289681448873360686098165513961777344872863575464161434004637569732
78552652919300099018189964455598543666553251940633281389477664969421075560744401264574084323655716558012339508524311564025017183439550680430675747762018115660 38
Closing connection.
Connection closed.
```

Alice's Output upon receiving "quit":

```
Received encrypted message: 601059924900531785821756058940217329048094641797536904656719629724613712103245545289681448873360686098165513961777344872863575464161434004637569732
78552652919300099018189964455598543666553251940633281389477664969421075560744401264574084323655716558012339508524311564025017183439550680430675747762018115660 38
Decrypted message: quit
Client requested to quit. Closing connection.
Connection closed. Waiting for new connections...
Waiting for a connection...
```

By following the above interaction example, users can understand how Alice and Bob set up their respective sockets, exchange messages securely, and gracefully terminate the connection.

What is a Credit Card Vault?

A Credit Card Vault is a comprehensive solution that allows for the secure storage, retrieval, and management of customer information, credit card details, and financial transactions. Built with a focus on data security and user experience, it caters to multiple user roles, each with tailored access and functionality.

Key Features:

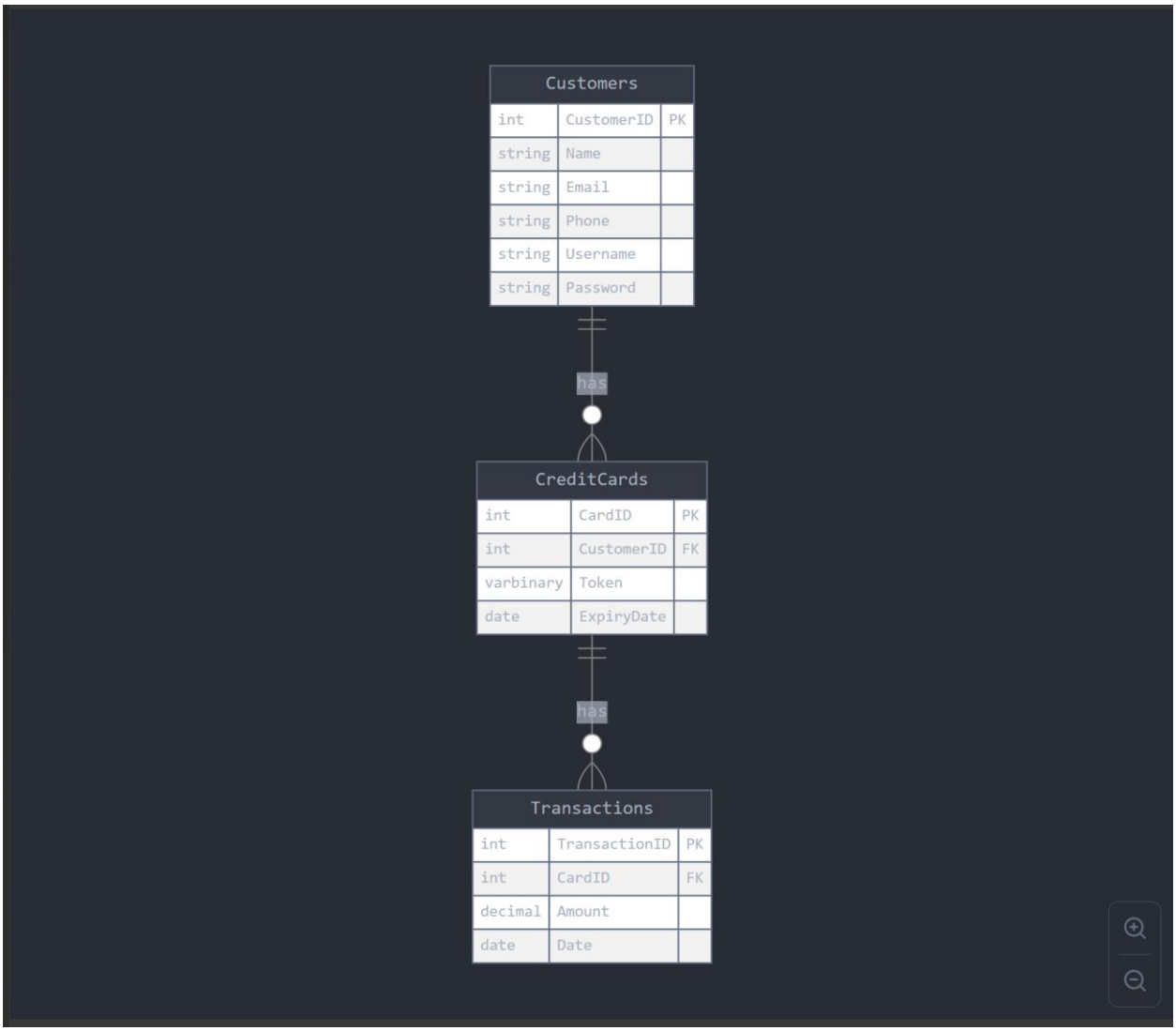
- Robust encryption for sensitive data
- Role-based access control
- Secure customer registration and management
- Real-time transaction processing and viewing
- User-friendly interfaces for all roles

Overview

This project is an implementation of a Credit Card Vault system, designed to securely manage customer information, credit card details, and financial transactions. The system implements role-based access control with different interfaces for administrators, finance users, support users, and customers. It uses encryption (AES) for sensitive data like credit card numbers and hashing (SHA-256) for passwords. The project includes features such as customer registration, transaction management, and a customer dashboard.

XAMPP was used to create and manage the database for this project. XAMPP provides a local Apache web server, MySQL database, and PHP environment, making it ideal for developing and testing web applications. The MySQL database included with XAMPP was used to create and manage the tables for the Credit Card Vault system.

Credit Card Vault Entity Relationship Diagram:

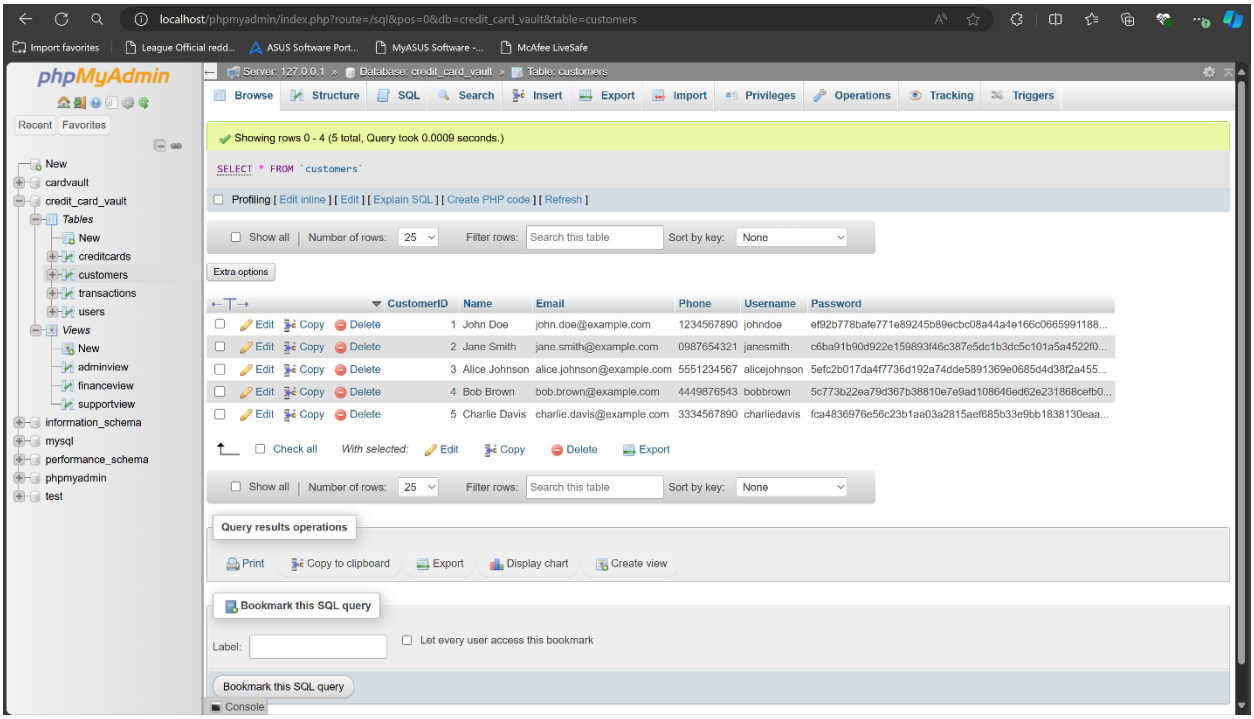


This diagram provides a clearer picture of how the data is structured and related in the Credit Card Vault system. It shows that:

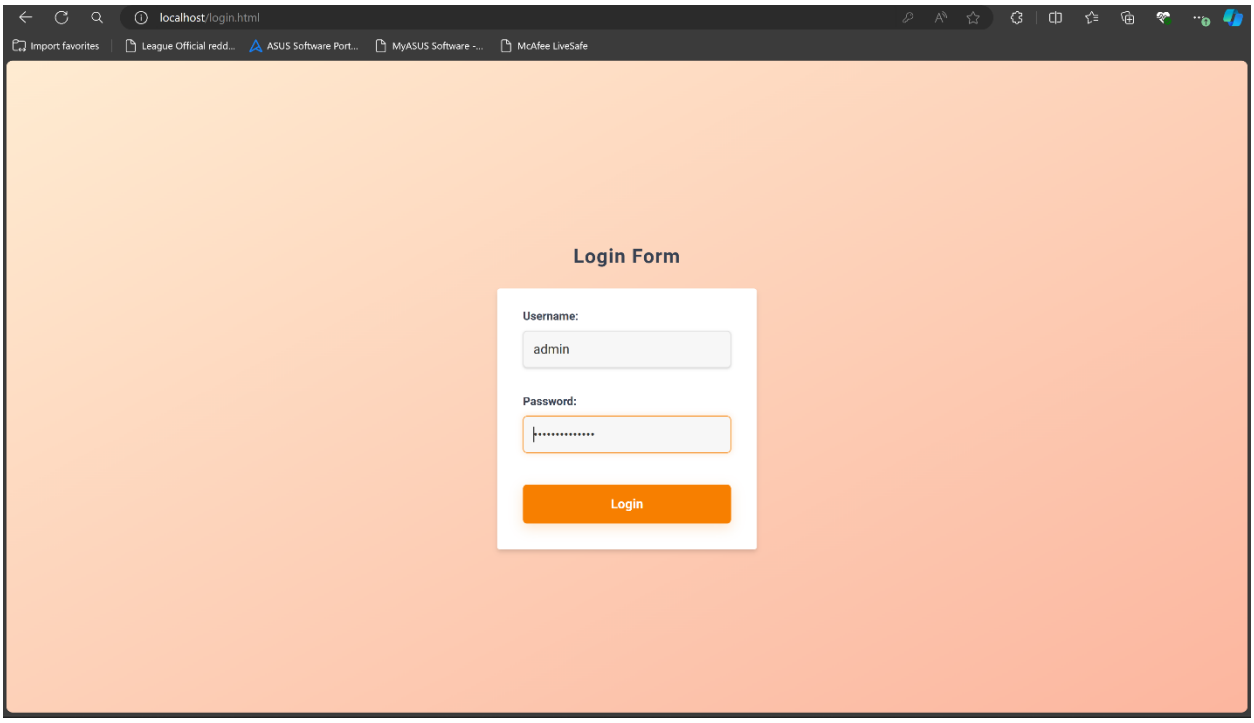
- Each customer can have multiple credit cards
- Each credit card is associated with exactly one customer
- Each credit card can have multiple transactions
- Each transaction is associated with exactly one credit card

This structure is in the Third Normal Form (3NF) clearly illustrating the relationships between all entities in the system.

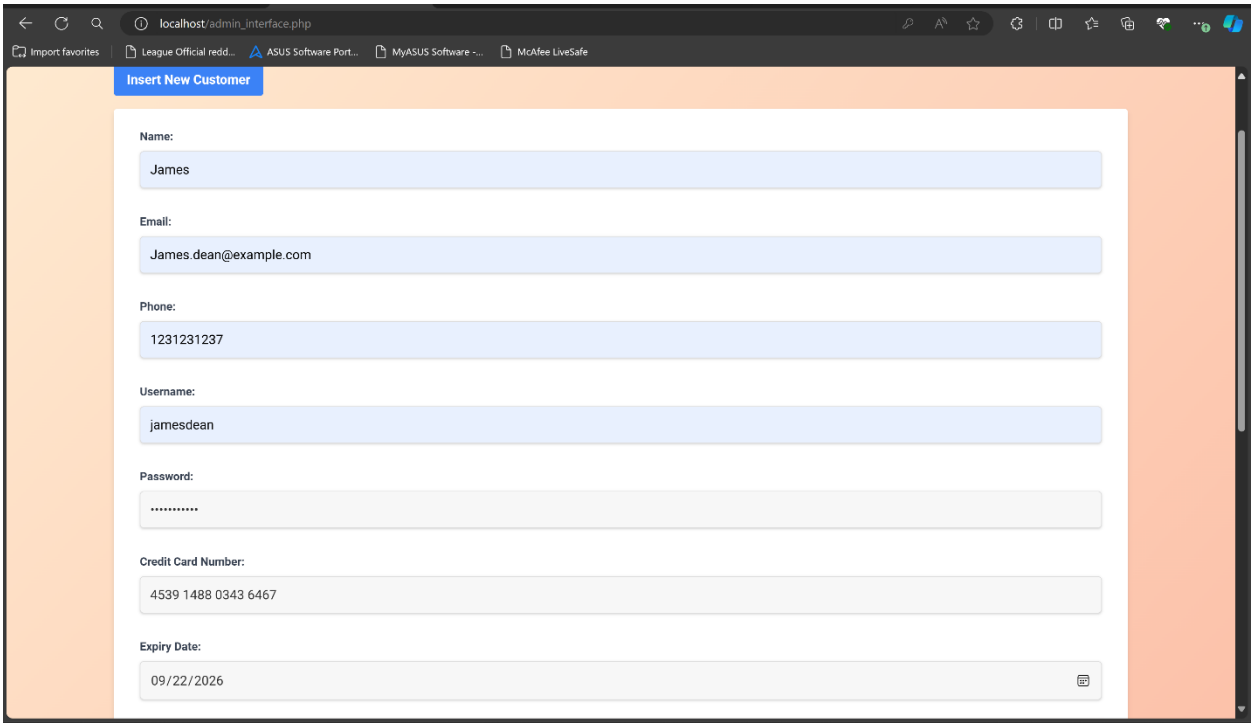
In the following sections, we'll take you on a journey through the Credit Card Vault system. We'll explore its database structure, examine how different user roles interact with the system, and demonstrate the security measures in place to protect valuable financial data.



This is how the database looks like before we add a new customer. We had inserted these entries using SQL commands to act as a base to see if any effect would be there when trying to interact through the front end.

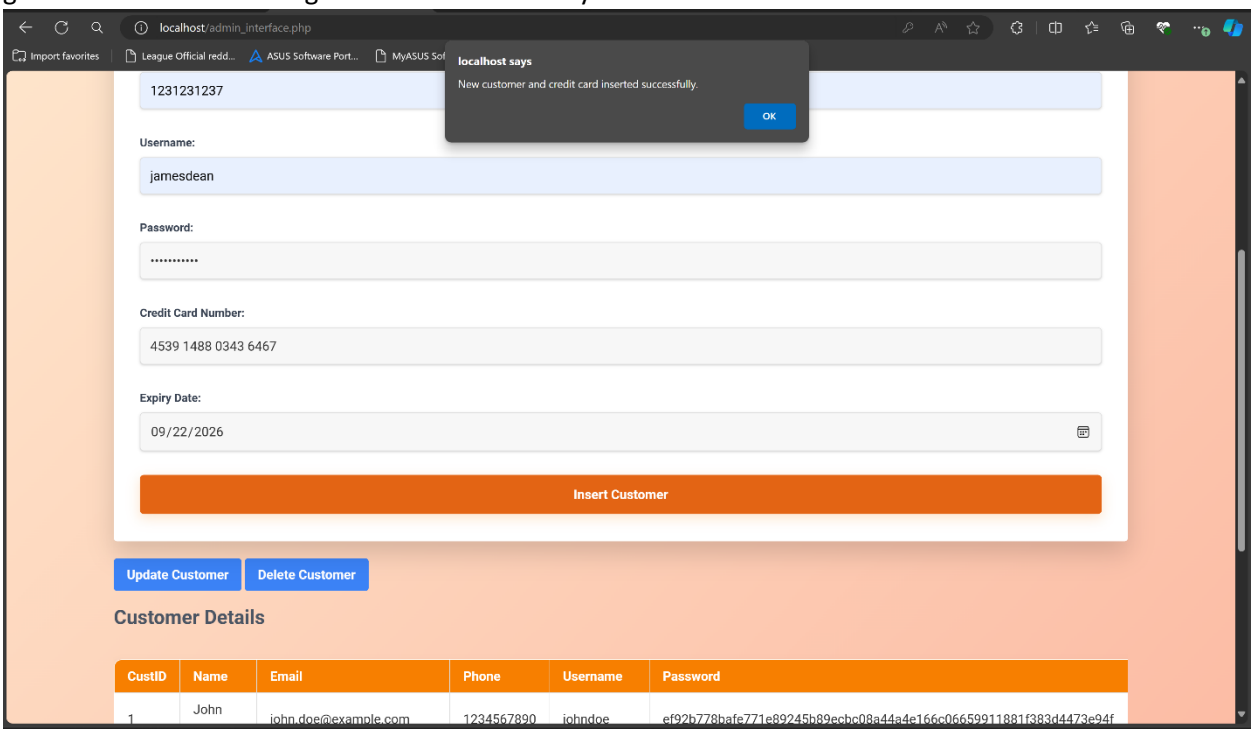


Here we have the ‘user role’ interface where the admin, finance_user, and support user can logic to see their specific view which are determined by the privileges each have. The admin has permission to perform all CRUD operations, while the finance_user can only enter transactions. The support user can only some of the customer details.

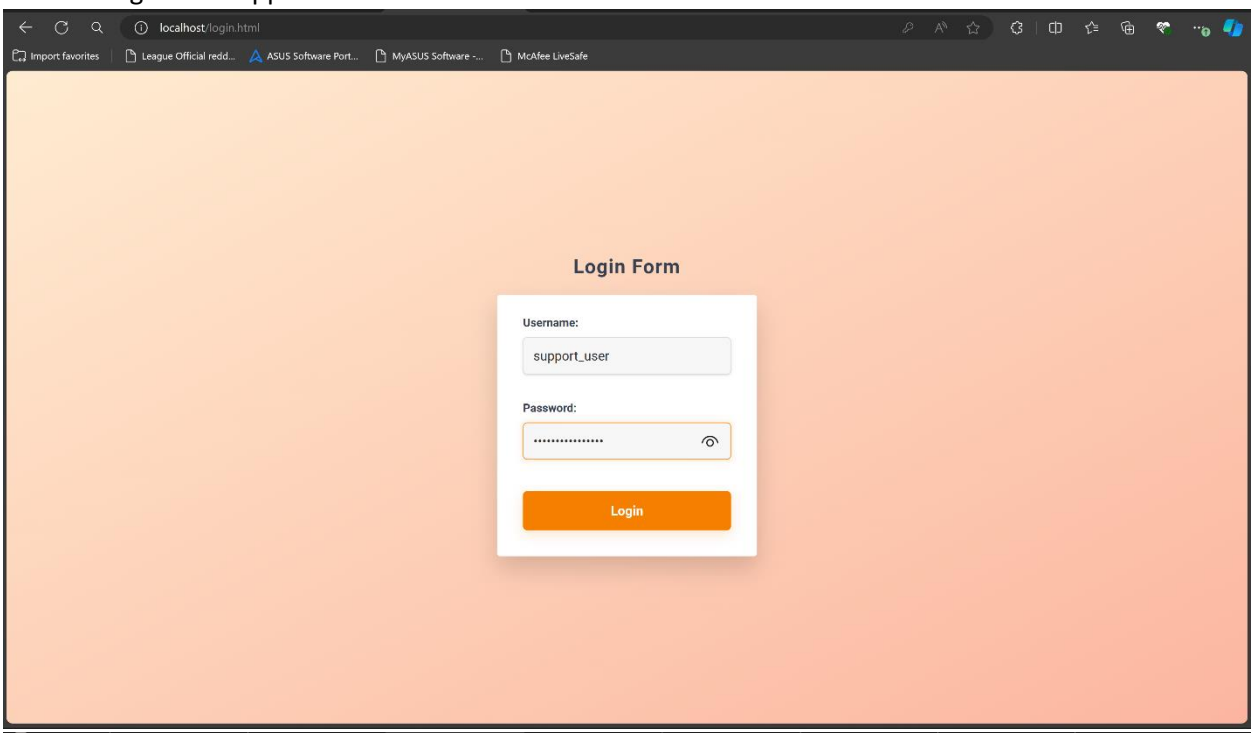


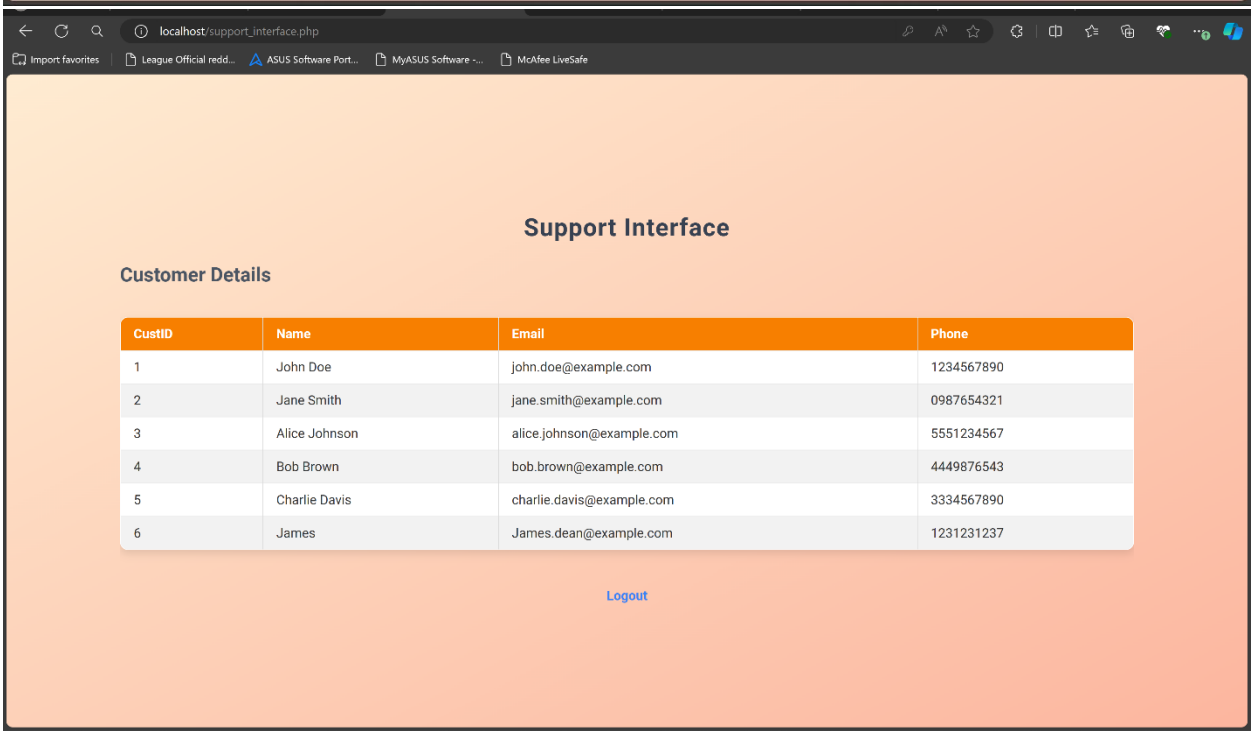
In this section we are adding a new customer through the admin. Validation checks for the credit card number are there to ensure only the correct amount and sequence of digits is allowed. On insertion we

get a confirmation message for a successful entry.



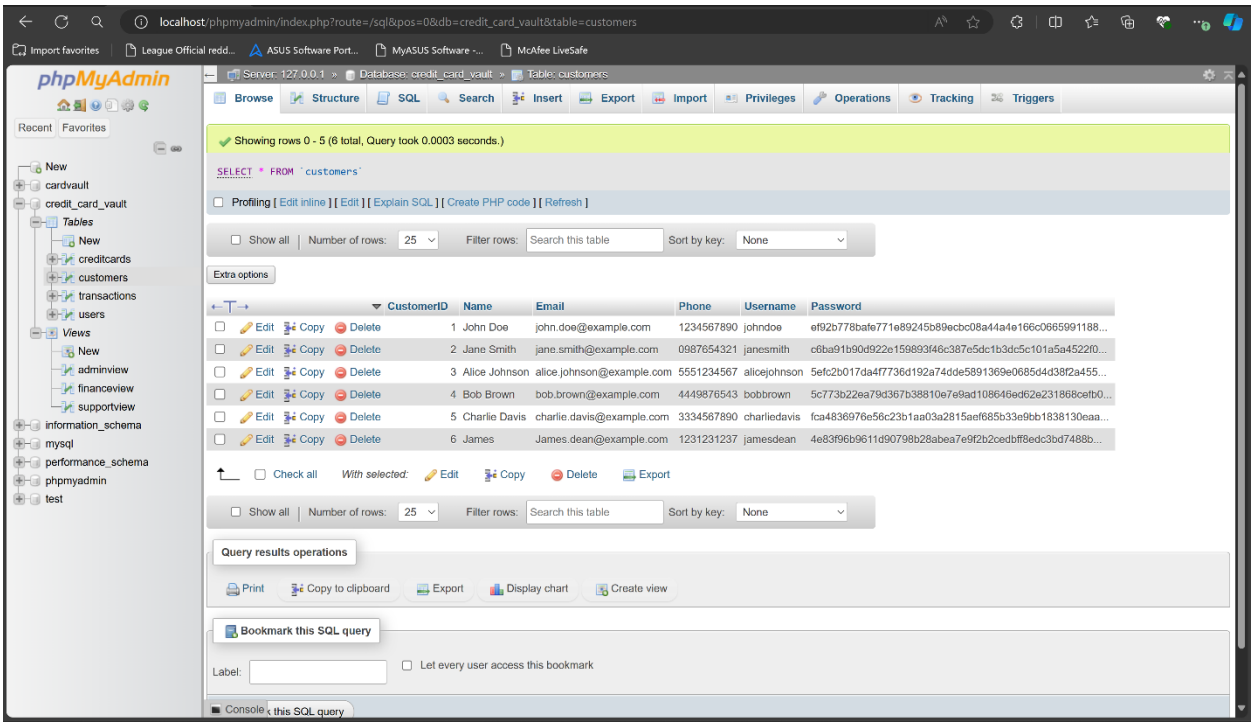
Now we login as a support user to see whether the add member can be seen on the front end.



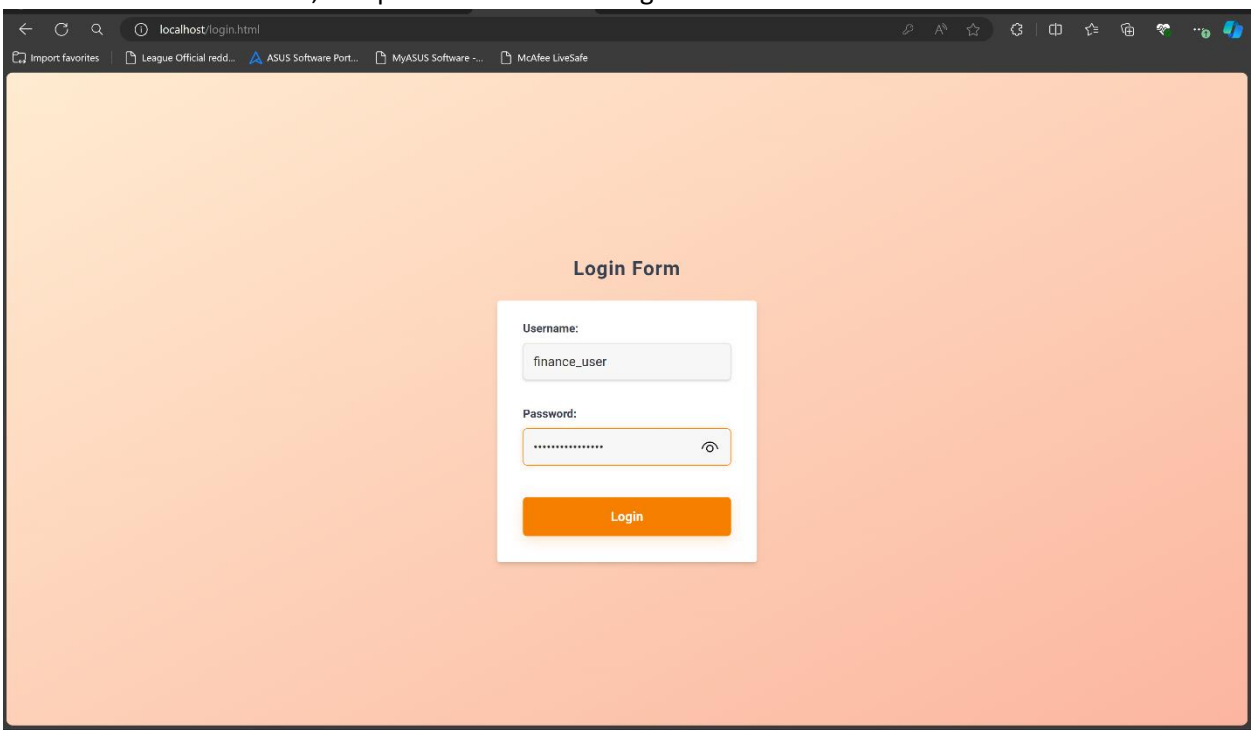


CustID	Name	Email	Phone
1	John Doe	john.doe@example.com	1234567890
2	Jane Smith	jane.smith@example.com	0987654321
3	Alice Johnson	alice.johnson@example.com	5551234567
4	Bob Brown	bob.brown@example.com	4449876543
5	Charlie Davis	charlie.davis@example.com	3334567890
6	James	James.dean@example.com	1231231237

Hurray! We actually see that he has been added, and the support user has access to only afew of the new customer's details.

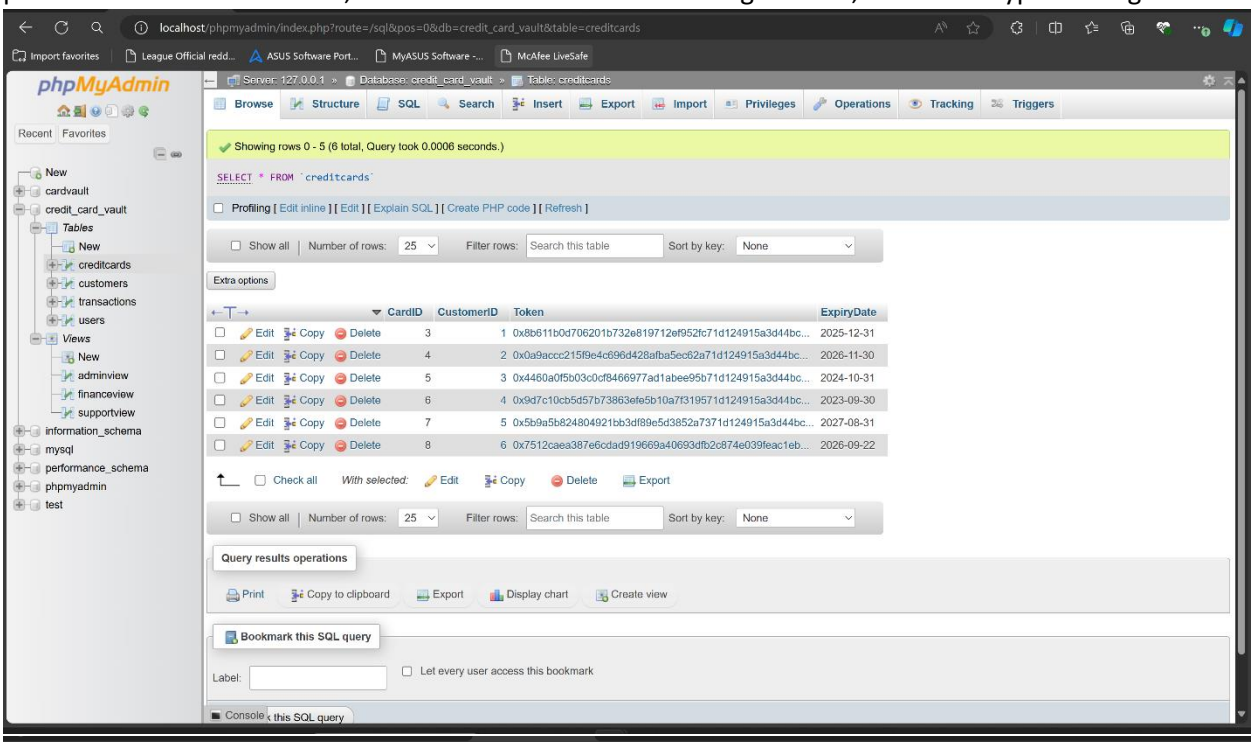


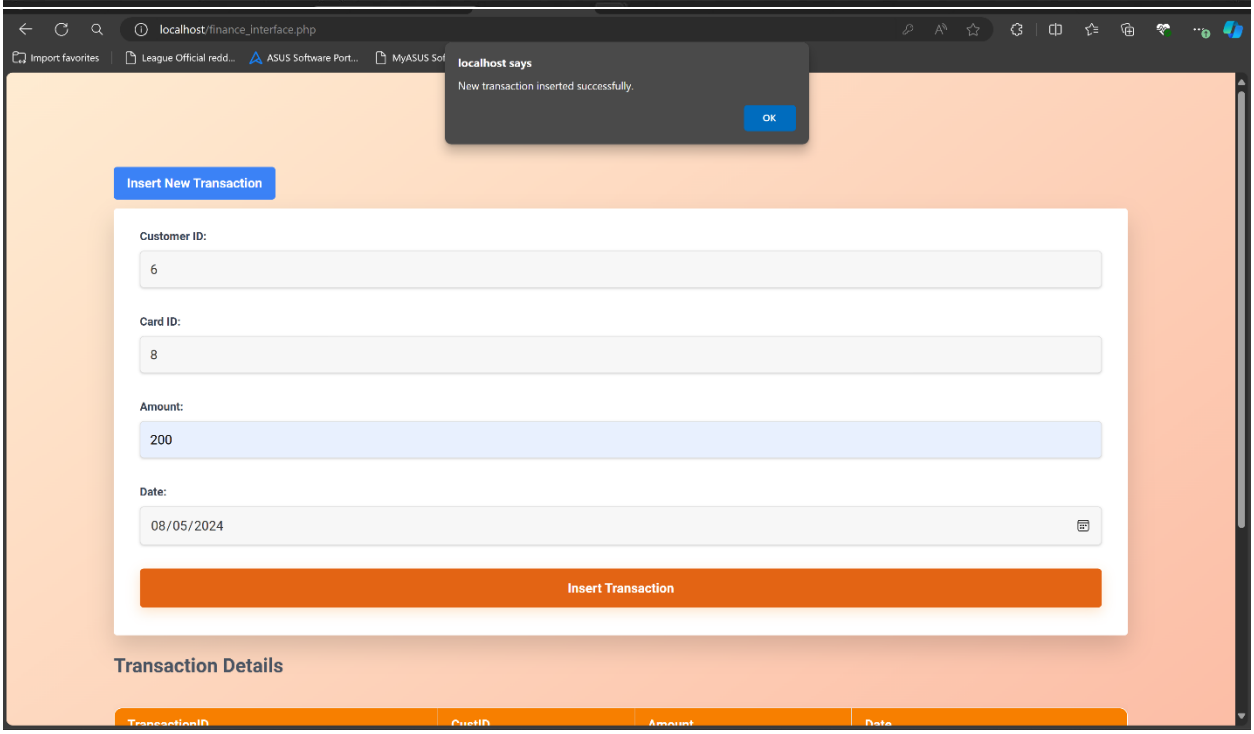
We can confirm the addition of the new customer when we take a look at the database and see that the customer has been added, and password hashed using SHA-2.



The finance user can add a new transaction and it updates the table in his view. This new transaction also updates the admin's table and he can see the entirety of the new customer's details. When you

peek at the creditcards table, the new customer has been assigned one, details encrypted using AES.





Upon adding a new transaction, we get a confirmation for a successful entry, and the transaction details are updated.

localhost/finance_interface.php

Import favorites | League Official redd... | ASUS Software Port... | MyASUS Software -... | McAfee LiveSafe

Finance Interface

Insert New Transaction

Transaction Details

TransactionID	CustID	Amount	Date
11	1	100.00	2024-07-21
12	2	150.00	2024-07-21
13	3	200.00	2024-07-21
14	4	250.00	2024-07-21
15	5	300.00	2024-07-21
16	6	200.00	2024-08-05

Logout

localhost/admin_interface.php

Import favorites | League Official redd... | ASUS Software Port... | MyASUS Software -... | McAfee LiveSafe

Admin Interface

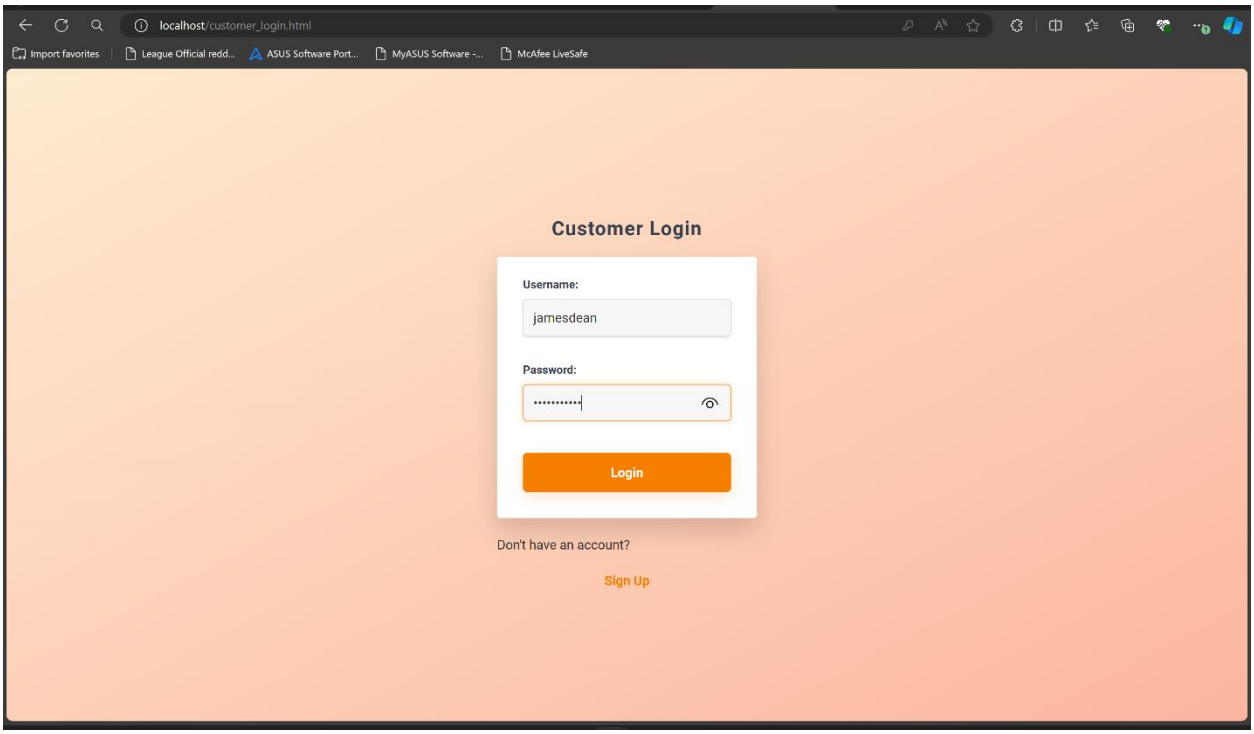
Insert New Customer | Update Customer | Delete Customer

Customer Details

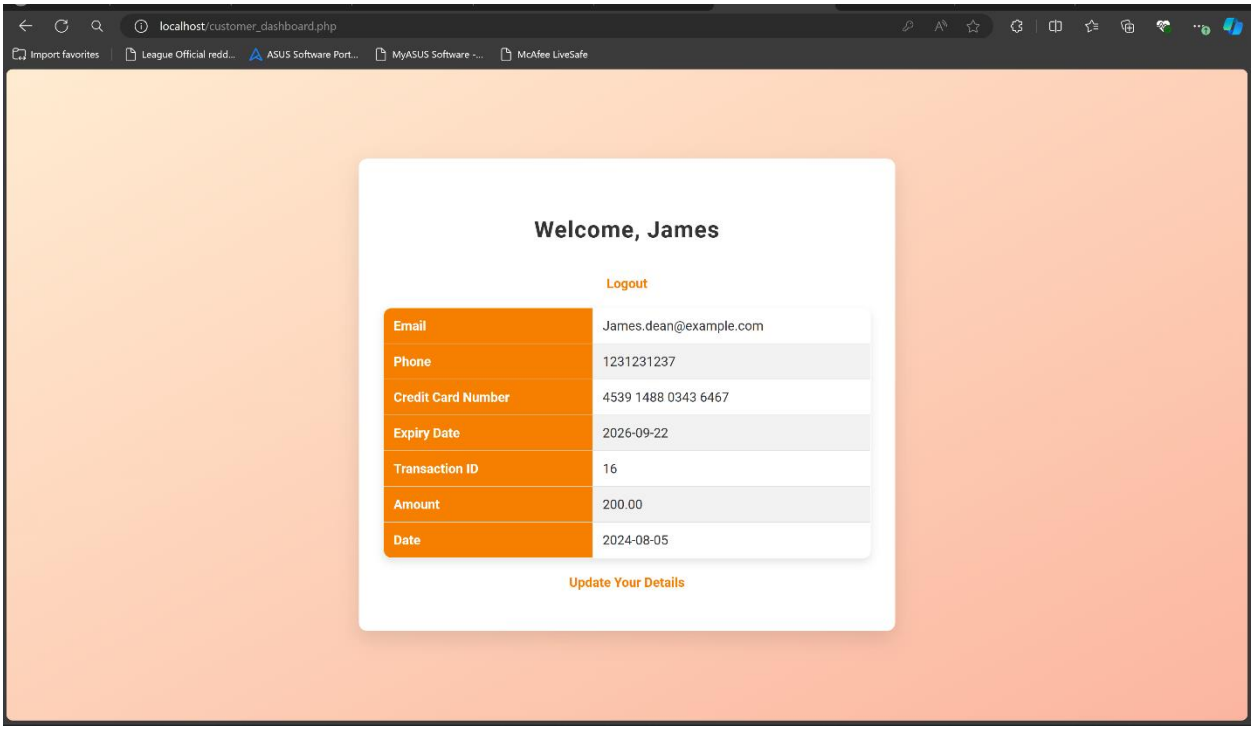
CustID	Name	Email	Phone	Username	Password
1	John Doe	john.doe@example.com	1234567890	johndoe	ef92b778bafe771e89245b89ecbc08a44a4e166c06659911881f383d4473e94f
2	Jane Smith	jane.smith@example.com	0987654321	janesmith	c6ba91b90d922e159893f46c387e5dc1b3dc5c101a5a4522f03b987177a24a91
3	Alice Johnson	alice.johnson@example.com	5551234567	alicejohnson	5efc2b017da4f7736d192a74dde5891369e0685d4d38f2a455b6fcdab282df9c
4	Bob Brown	bob.brown@example.com	4449876543	bobbrown	5c773b22ea79d367b38810e7e9ad108646ed62e231868cefb0b1280ea88ac4f0
5	Charlie Davis	charlie.davis@example.com	3334567890	charliedavis	fca4836976e56c23b1aa03a2815aef685b33e9bb1838130eaa6d30533c90bee1
6	James	James.dean@example.com	1231231237	jamesdean	4e83f96b9611d90798b28abea7e9f2b2cedbff8edc3bd7488bd248160b4fc83b

Logout

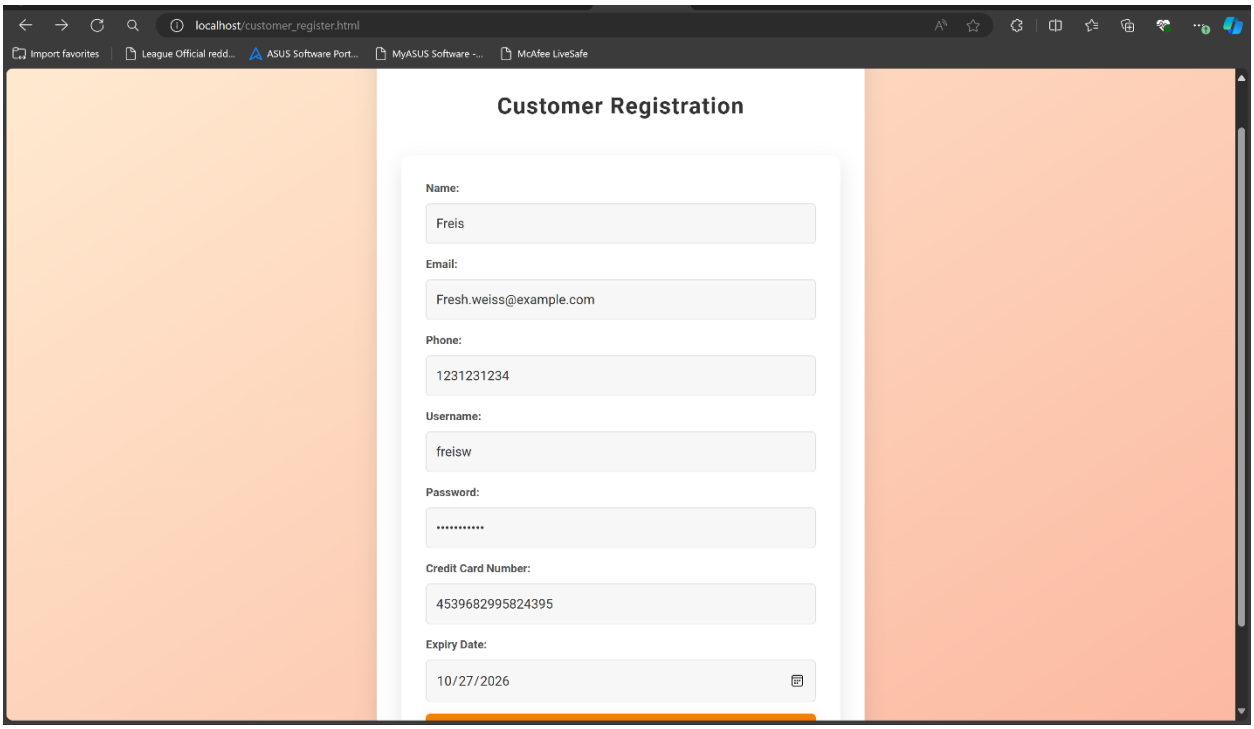
This is reflected on the admin interface as well as the database.

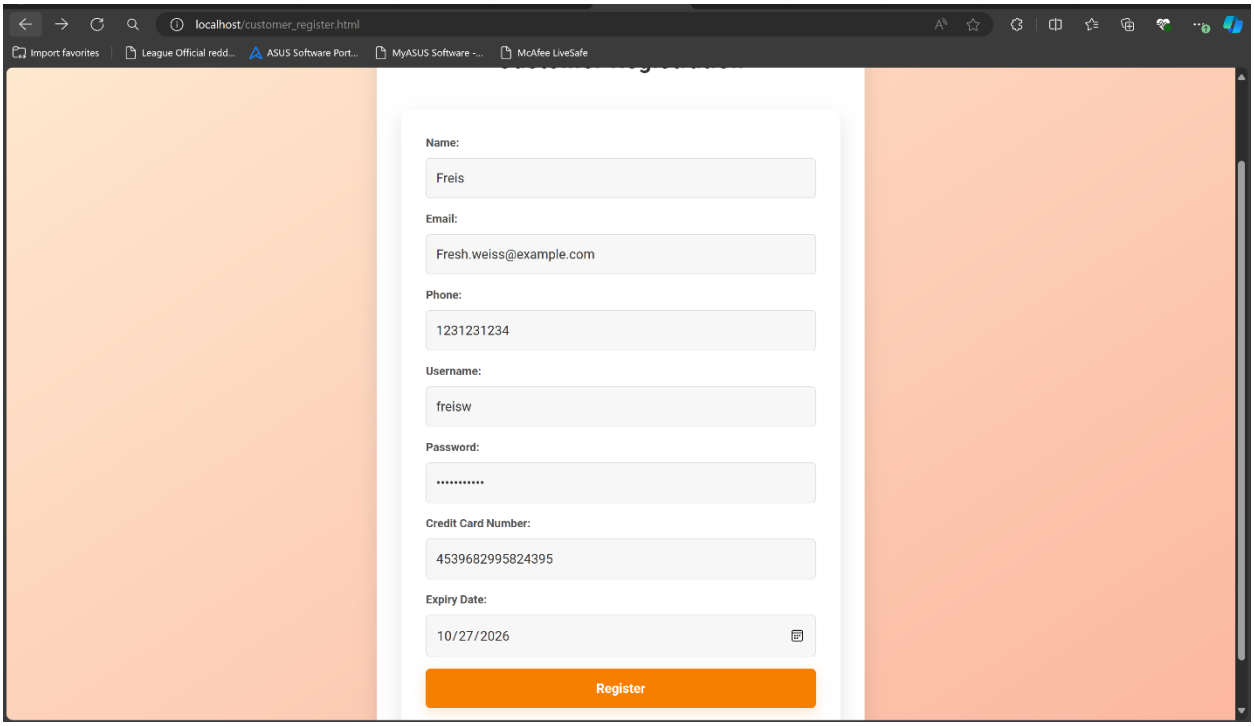


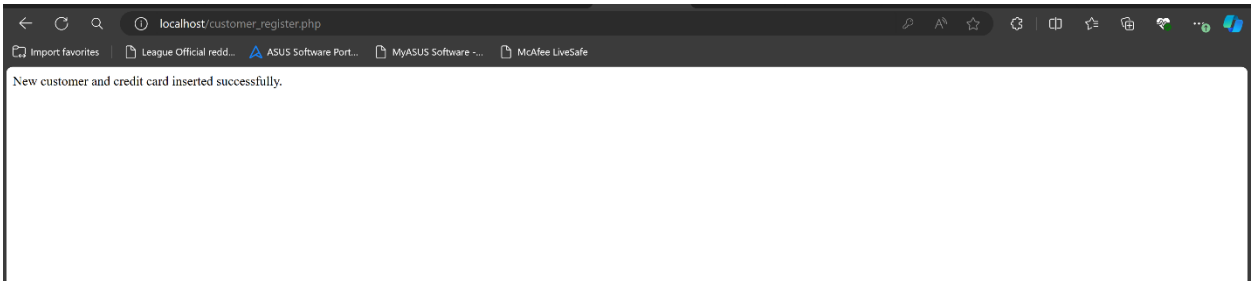
Now we log in as the customer to see whether the credit card details can be decrypted for the customer to view their details. We log in as the the new customer we have just added and see all his details presented to him, including the decrypted credit card number.

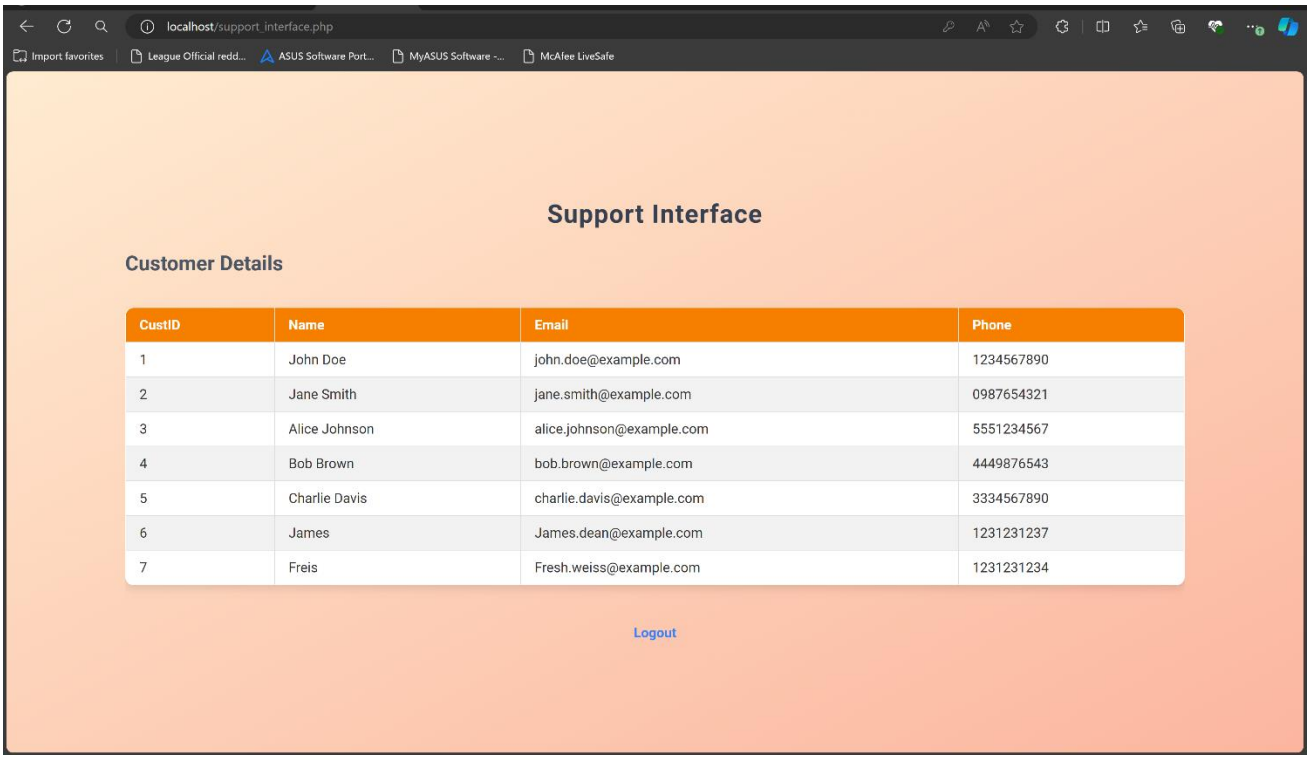


The customer has the option to update their details too. And if doesn't have an account, they can sign up and register, and see their details as well.









CustID	Name	Email	Phone
1	John Doe	john.doe@example.com	1234567890
2	Jane Smith	jane.smith@example.com	0987654321
3	Alice Johnson	alice.johnson@example.com	5551234567
4	Bob Brown	bob.brown@example.com	4449876543
5	Charlie Davis	charlie.davis@example.com	3334567890
6	James	James.dean@example.com	1231231237
7	Freis	Fresh.weiss@example.com	1231231234

The codes for various important functionality:

Admin Interface

<u>Inserting customer.</u>

```
<?php
include 'validateCreditCard.php';
include 'isCreditCardUnique.php';

// Database connection
$servername = "localhost";
$username = "admin";
```

```
$password = "admin_password";
$dbname = "credit_card_vault";
$encryption_key = "encryption_key"; // Use a secure key management
practice

$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Insert customer
$name = $_POST['name'];
$email = $_POST['email'];
$phone = $_POST['phone'];
$username = $_POST['username'];
$password = $_POST['password'];
$card_number = $_POST['card_number'];
$expiry_date = $_POST['expiry_date'];

// Hash the password using SHA-256
$hashed_password = hash('sha256', $password);

// Initialize an error message variable
$error_message = "";

// Validate credit card number
if (!validateCreditCard($card_number)) {
    $error_message = "Invalid credit card number.";
}
```

```
// Check if credit card number is unique
if ($error_message == "" && !isCreditCardUnique($card_number, $conn,
$encryption_key)) {
    $error_message = "Credit card number already exists.";
}


if ($error_message != "") {
    echo "<script>alert('$error_message');</script>";
} else {
    $conn->autocommit(FALSE); // Start transaction


    try {
        $sql = "INSERT INTO Customers (name, email, phone, username,
password) VALUES (?, ?, ?, ?, ?)";
        $stmt = $conn->prepare($sql);
        $stmt->bind_param("sssss", $name, $email, $phone, $username,
$hashed_password);


        if (!$stmt->execute()) {
            throw new Exception("Error: " . $stmt->error);
        }


        $customer_id = $stmt->insert_id; // Get the inserted customer
ID
        $stmt->close();


        $sql = "INSERT INTO CreditCards (CustomerID, Token,
ExpiryDate) VALUES (?, AES_ENCRYPT(?, ?), ?)";
        $stmt = $conn->prepare($sql);
        $stmt->bind_param("isss", $customer_id, $card_number,
$encryption_key, $expiry_date);
```

```
        if (!$stmt->execute()) {
            throw new Exception("Error: " . $stmt->error);
        }

        $conn->commit(); // Commit transaction
        echo "New customer and credit card inserted successfully.";
    } catch (Exception $e) {
        $conn->rollback(); // Rollback transaction on error
        echo $e->getMessage();
    }

    $stmt->close();
}

$conn->close();
?>
```

Updating customer

```
<?php
include 'validateCreditCard.php';
include 'isCreditCardUnique.php';

// Database connection
$servername = "localhost";
$username = "admin";
$password = "admin_password";
$dbname = "credit_card_vault";
$encryption_key = "encryption_key"; // Use a secure key management
practice

$conn = new mysqli($servername, $username, $password, $dbname);
```

```
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Update customer
$customer_id = $_POST['customer_id'];
$name = $_POST['name'];
$email = $_POST['email'];
$phone = $_POST['phone'];
$username = $_POST['username'];
$password = $_POST['password'];
$card_number = $_POST['card_number'];
$expiry_date = $_POST['expiry_date'];

// Hash the password using SHA-256
$hashed_password = hash('sha256', $password);

// Validate credit card number
if (!validateCreditCard($card_number)) {
    die("Invalid credit card number.");
}

// Check if credit card number is unique
if (!isCreditCardUnique($card_number, $conn, $encryption_key)) {
    die("Credit card number already exists.");
}

$conn->autocommit(FALSE); // Start transaction
```

```
try {
    $sql = "UPDATE Customers SET Name=?, Email=?, Phone=?, Username=?,
Password=? WHERE CustomerID=?";
    $stmt = $conn->prepare($sql);
    if ($stmt) {
        $stmt->bind_param("sssssi", $name, $email, $phone, $username,
$hashed_password, $customer_id);

        if (!$stmt->execute()) {
            throw new Exception("Error: " . $stmt->error);
        }
        $stmt->close();
    } else {
        throw new Exception("Error preparing statement: " . $conn-
>error);
    }

    $sql = "UPDATE CreditCards SET Token=AES_ENCRYPT(?, ?),
ExpiryDate=? WHERE CustomerID=?";
    $stmt = $conn->prepare($sql);
    if ($stmt) {
        $stmt->bind_param("sssi", $card_number, $encryption_key,
$expiry_date, $customer_id);

        if (!$stmt->execute()) {
            throw new Exception("Error: " . $stmt->error);
        }
        $stmt->close();
    } else {
        throw new Exception("Error preparing statement: " . $conn-
>error);
    }
```

```
    $conn->commit(); // Commit transaction
    echo "Customer and credit card updated successfully.";
} catch (Exception $e) {
    $conn->rollback(); // Rollback transaction on error
    echo $e->getMessage();
}

$conn->close();
?>
```

Delete Customer

```
<?php
// Database connection
$servername = "localhost";
$username = "admin";
$password = "admin_password";
$dbname = "credit_card_vault";

$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$customer_id = $_POST['customer_id'];

$conn->autocommit(FALSE); // Start transaction

try {
    // Delete from Transactions
```

```
    $sql = "DELETE FROM Transactions WHERE CardID IN (SELECT CardID
FROM CreditCards WHERE CustomerID=?)";
    $stmt = $conn->prepare($sql);
    $stmt->bind_param("i", $customer_id);

    if (!$stmt->execute()) {
        throw new Exception("Error: " . $stmt->error);
    }
    $stmt->close();

    // Delete from CreditCards
    $sql = "DELETE FROM CreditCards WHERE CustomerID=?";
    $stmt = $conn->prepare($sql);
    $stmt->bind_param("i", $customer_id);

    if (!$stmt->execute()) {
        throw new Exception("Error: " . $stmt->error);
    }
    $stmt->close();

    // Delete from Customers
    $sql = "DELETE FROM Customers WHERE CustomerID=?";
    $stmt = $conn->prepare($sql);
    $stmt->bind_param("i", $customer_id);

    if (!$stmt->execute()) {
        throw new Exception("Error: " . $stmt->error);
    }
    $stmt->close();

    $conn->commit(); // Commit transaction
```

```
    echo "Customer and related details deleted successfully.";
} catch (Exception $e) {
    $conn->rollback(); // Rollback transaction on error
    echo $e->getMessage();
}

$conn->close();
?>
```

Fetch Customer

```
<?php
// Database connection
$servername = "localhost";
$username = "admin";
$password = "admin_password";
$dbname = "credit_card_vault";

$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT * FROM AdminView";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
    echo '<div class="overflow-x-auto"><table class="min-w-full bg-
white border">';
    echo '<thead><tr>';
    while ($field = $result->fetch_field()) {
```

```
        echo '<th class="py-2 px-4 border">' .
htmlspecialchars($field->name) . '</th>';

    }

    echo '</tr></thead>';

    echo '<tbody>';

    while ($row = $result->fetch_assoc()) {

        echo '<tr>';

        foreach ($row as $data) {

            echo '<td class="py-2 px-4 border">' .
htmlspecialchars($data) . '</td>';

        }

        echo '</tr>';

    }

    echo '</tbody></table></div>';

} else {

    echo '<p class="text-center text-gray-600">No data
available.</p>';

}


$conn->close();

?>
```

Finance Interface

Finance_interface.php

```
<?php

session_start();

if ($_SESSION['role'] != 'Finance') {

    echo "Access denied.";

    exit;

}

?>

<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <title>Finance Interface</title>
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <link
href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.mi
n.css" rel="stylesheet">
    <link rel="stylesheet" href="style.css">
    <script src="https://code.jquery.com/jquery-
3.6.0.min.js"></script>
    <script>
        $(document).ready(function() {
            function fetchTransactions() {
                $.ajax({
                    url: 'fetch_transactions.php',
                    method: 'GET',
                    success: function(data) {
                        $('#transactionTable').html(data);
                    }
                });
            }

            fetchTransactions(); // Initial fetch

            $('#insertForm').on('submit', function(e) {
                e.preventDefault();
                $.ajax({
                    url: $(this).attr('action'),
                    method: $(this).attr('method'),
                    data: $(this).serialize(),
                    success: function(response) {
```

```
                        alert(response);
                        fetchTransactions(); // Fetch updated
transactions
                    }
                });
            });

            // Toggle form visibility
            $('.toggle-form').on('click', function() {
                var target = $(this).data('target');
                $('#' + target).toggleClass('hidden');
            });
        });
    </script>
</head>
<body class="bg-gray-100">
    <div class="container mx-auto p-4">
        <h2 class="text-center text-3xl font-bold text-gray-700 mb-
6">Finance Interface</h2>


        <button data-target="insertForm" class="toggle-form bg-blue-
500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded
focus:outline-none focus:shadow-outline mb-4">Insert New
Transaction</button>
        <form id="insertForm" action="insert_transaction.php"
method="post" class="bg-white shadow-md rounded px-8 pt-6 pb-8 mb-4
hidden">
            <div class="mb-4">
                <label for="customer_id" class="block text-gray-700
text-sm font-bold mb-2">Customer ID:</label>
                <input type="text" id="customer_id" name="customer_id"
required class="shadow appearance-none border rounded w-full py-2 px-3
text-gray-700 leading-tight focus:outline-none focus:shadow-outline">
            </div>
```

```
            <div class="mb-4">
                <label for="card_id" class="block text-gray-700 text-
sm font-bold mb-2">Card ID:</label>
                <input type="text" id="card_id" name="card_id"
required class="shadow appearance-none border rounded w-full py-2 px-3
text-gray-700 leading-tight focus:outline-none focus:shadow-outline">
            </div>
            <div class="mb-4">
                <label for="amount" class="block text-gray-700 text-sm
font-bold mb-2">Amount:</label>
                <input type="text" id="amount" name="amount" required
class="shadow appearance-none border rounded w-full py-2 px-3 text-
gray-700 leading-tight focus:outline-none focus:shadow-outline">
            </div>
            <div class="mb-4">
                <label for="date" class="block text-gray-700 text-sm
font-bold mb-2">Date:</label>
                <input type="date" id="date" name="date" required
class="shadow appearance-none border rounded w-full py-2 px-3 text-
gray-700 leading-tight focus:outline-none focus:shadow-outline">
            </div>
            <div class="flex items-center justify-between">
                <input type="submit" value="Insert Transaction"
class="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4
rounded focus:outline-none focus:shadow-outline">
            </div>
        </form>


        <h3 class="text-2xl font-semibold text-gray-600 mb-
4">Transaction Details</h3>
        <div id="transactionTable">
            <?php
            $servername = "localhost";
            $username = "finance_user";
            $password = "finance_password";
```

```
            $dbname = "credit_card_vault";

            $conn = new mysqli($servername, $username, $password,
$dbname);

            if ($conn->connect_error) {
                die("Connection failed: " . $conn->connect_error);
            }

            $sql = "SELECT * FROM FinanceView";
            $result = $conn->query($sql);

            if ($result->num_rows > 0) {
                echo '<div class="overflow-x-auto"><table class="min-
w-full bg-white border">';
                echo '<thead><tr>';
                while ($field = $result->fetch_field()) {
                    echo '<th class="py-2 px-4 border">' .
htmlspecialchars($field->name) . '</th>';
                }
                echo '</tr></thead>';
                echo '<tbody>';
                while ($row = $result->fetch_assoc()) {
                    echo '<tr>';
                    foreach ($row as $data) {
                        echo '<td class="py-2 px-4 border">' .
htmlspecialchars($data) . '</td>';
                    }
                    echo '</tr>';
                }
                echo '</tbody></table></div>';
            } else {
```

```
                echo '<p class="text-center text-gray-600">No data
available.</p>';
            }

            $conn->close();
            ?>
        </div>

        <a href="logout.php" class="block text-center mt-6 text-blue-
500 hover:underline">Logout</a>
    </div>
</body>
</html>
```

Fetch_transactions.php

```
<?php
// Database connection
$servername = "localhost";
$username = "admin";
$password = "admin_password";
$dbname = "credit_card_vault";

$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT * FROM FinanceView";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
```

```
    echo '<div class="overflow-x-auto"><table class="min-w-full bg-
white border">';

    echo '<thead><tr>';

    while ($field = $result->fetch_field()) {

        echo '<th class="py-2 px-4 border">' .
htmlspecialchars($field->name) . '</th>';

    }

    echo '</tr></thead>';

    echo '<tbody>';

    while ($row = $result->fetch_assoc()) {

        echo '<tr>';

        foreach ($row as $data) {

            echo '<td class="py-2 px-4 border">' .
htmlspecialchars($data) . '</td>';

        }

        echo '</tr>';

    }

    echo '</tbody></table></div>';

} else {

    echo '<p class="text-center text-gray-600">No data
available.</p>';

}


$conn->close();

?>
```

Insert_transactions.php

```
<?php

// Database connection

$servername = "localhost";

$username = "finance_user";

$password = "finance_password";

$dbname = "credit_card_vault";
```

```
$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Insert transaction
$customer_id = $_POST['customer_id'];
$card_id = $_POST['card_id'];
$amount = $_POST['amount'];
$date = $_POST['date'];

$sql = "INSERT INTO Transactions (CustomerID, CardID, Amount, Date)
VALUES (?, ?, ?, ?)";
$stmt = $conn->prepare($sql);
$stmt->bind_param("iids", $customer_id, $card_id, $amount, $date);

if ($stmt->execute()) {
    echo "New transaction inserted successfully.";
} else {
    echo "Error: " . $stmt->error;
}

$stmt->close();
$conn->close();
?>
```

Credit Card Validation and Uniqueness Check

Validation

```
<?php
```

```
function validateCreditCard($number) {
    // Remove any non-digit characters
    $number = preg_replace('/\D/', '', $number);

    // Check length
    if (strlen($number) < 13 || strlen($number) > 19) {
        return false;
    }

    // Luhn algorithm
    $sum = 0;
    $alt = false;
    for ($i = strlen($number) - 1; $i >= 0; $i--) {
        $n = $number[$i];
        if ($alt) {
            $n *= 2;
            if ($n > 9) {
                $n -= 9;
            }
        }
        $sum += $n;
        $alt = !$alt;
    }

    return ($sum % 10 == 0);
}
?>
```

Uniqueness Check

```
<?php
function isCreditCardUnique($number, $conn, $encryption_key) {
```

```
    $sql = "SELECT COUNT(*) FROM CreditCards WHERE AES_DECRYPT(Token,
?) = ?";

    $stmt = $conn->prepare($sql);

    $stmt->bind_param("ss", $encryption_key, $number);

    $stmt->execute();

    $stmt->bind_result($count);

    $stmt->fetch();

    $stmt->close();


    return $count == 0;

}

?>
```

Customer Interface

Customer dashboard.php

```
<?php

session_start();

if (!isset($_SESSION['customer_id'])) {

    header("Location: customer_login.html");

    exit();

}


// Database connection

$servername = "localhost";

$username = "root";

$password = "";

$dbname = "credit_card_vault";

$encryption_key = "encryption_key"; // Use the same key used during
encryption


$conn = new mysqli($servername, $username, $password, $dbname);
```

```
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Retrieve customer details
$customer_id = $_SESSION['customer_id'];
$sql = "SELECT Name, Email, Phone, AES_DECRYPT(Token, ?) AS
DecryptedCardNumber, ExpiryDate, TransactionID, Amount, Date FROM
AdminView WHERE CustID=?";
$stmt = $conn->prepare($sql);
$stmt->bind_param("si", $encryption_key, $customer_id);
$stmt->execute();
$result = $stmt->get_result();
$customer = $result->fetch_assoc();

$stmt->close();
$conn->close();
?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Customer Dashboard</title>
    <link rel="stylesheet" href="style.css"> <!-- Ensure this path is
correct -->
</head>
<body>
    <div class="dashboard-container">
```

```
        <h2>Welcome, <?php echo htmlspecialchars($customer['Name']);
?></h2>

     <a href="customer_logout.php">Logout</a>

        <table>

            <tr>

                <th>Email</th>

                <td><?php echo htmlspecialchars($customer['Email']);
?></td>

            </tr>

            <tr>

                <th>Phone</th>

                <td><?php echo htmlspecialchars($customer['Phone']);
?></td>

            </tr>

            <tr>

                <th>Credit Card Number</th>

                <td><?php echo
htmlspecialchars($customer['DecryptedCardNumber']); ?></td>

            </tr>

            <tr>

                <th>Expiry Date</th>

                <td><?php echo
htmlspecialchars($customer['ExpiryDate']); ?></td>

            </tr>

            <tr>

                <th>Transaction ID</th>

                <td><?php echo
htmlspecialchars($customer['TransactionID']); ?></td>

            </tr>

            <tr>

                <th>Amount</th>

                <td><?php echo htmlspecialchars($customer['Amount']);
?></td>
```

```
            </tr>
            <tr>
                <th>Date</th>
                <td><?php echo htmlspecialchars($customer['Date']);
?></td>
            </tr>
        </table>
      <a href="update_customer.html">Update Your Details</a>
    </div>
</body>
</html>
```

Summary

The Credit Card Vault system provides a comprehensive solution for managing financial data securely. Key features include:

1. Role-based access control (Admin, Finance, Support, Customer)
2. Secure storage of credit card information using AES encryption
3. Password hashing using SHA-256
4. Customer registration and management
5. Transaction recording and viewing
6. Credit card validation using the Luhn algorithm
7. Uniqueness check for credit card numbers
8. Customer dashboard for viewing personal information and transactions

What could be improved:

1. Balance Column: Implementing a balance column that gets updated with positive or negative transactions would provide a more comprehensive view of a customer's financial status. This would allow for real-time tracking of available funds and could be useful for both customers and finance users.
2. Two-Factor Authentication: Implementing 2FA would significantly enhance the security of the system, especially for sensitive operations like accessing credit card information or making transactions.

3. Audit Logging: Adding a comprehensive audit log to track all system activities, including login attempts, data modifications, and transaction entries, would improve security and aid in troubleshooting.
4. Pagination and Sorting: For tables displaying large amounts of data (e.g., transactions), implementing pagination and sorting features would improve usability and performance.
5. Password Complexity Requirements: Enforcing strong password policies (e.g., minimum length, special characters, numbers) would enhance overall system security.
6. Automated Notifications: Implementing an email or SMS notification system for important events like successful transactions, suspicious activities, or approaching credit card expiry dates would improve user experience and security.
7. Data Export Functionality: Adding the ability for users (especially admins and finance users) to export data in various formats (CSV, PDF) could be useful for reporting and analysis purposes.
8. Regular Security Scans: Implementing automated security scans and vulnerability assessments would help maintain the system's integrity over time.

These improvements would enhance the functionality, security, and user experience of the Credit Card Vault system.