

**SUMMER 2024**

**APT3090 CRYPTOGRAPHY AND NETWORK SECURITY PROJECT**

**Group Members**

**Collins Otiende | 662427**

**Joy Chemutai | 664830**

**Francis Matheka | 665206**

**Lawrence Magak | 664055**

**Macrine Akinyi | 667569**

**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)**

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'

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)

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.

1. **Connection Establishment:** Bob connects to Alice using Alice's IP address and port number. Alice listens for incoming connections.
2. **Public Key Exchange:** Once connected, Alice and Bob exchange their public keys. This allows them to encrypt messages intended for each other.
3. **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.
4. **Termination:** The connection can be terminated by sending a message with the content "quit" from either Alice or Bob. This gracefully closes the connection.
5. 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.

### 

**Bob Connects to Alice:**

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

### 

**Public Key Exchange:**

* Alice and Bob exchange public keys.

Alice's Output:

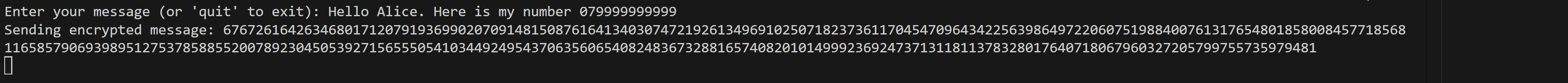
### 

### Bob’s output:

### 

**Message Exchange:**

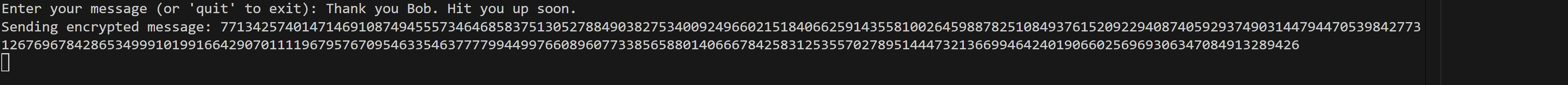
* **Bob sends a message:**

****

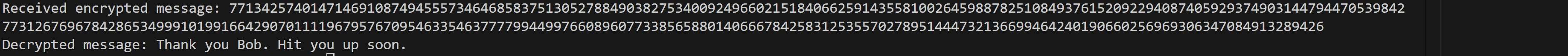
Alice receives and decrypts:

****

Alice sends a response:

****

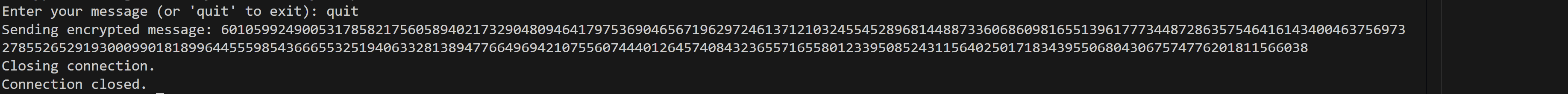
Bob receives and decrypts:

****

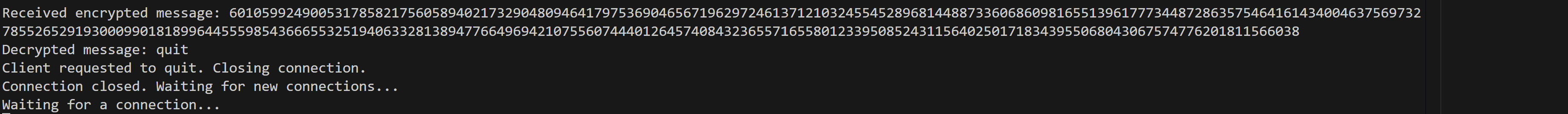
**Termination:**

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

Bob's Output when sending "quit":

****

Alice's Output upon receiving "quit":

****

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.

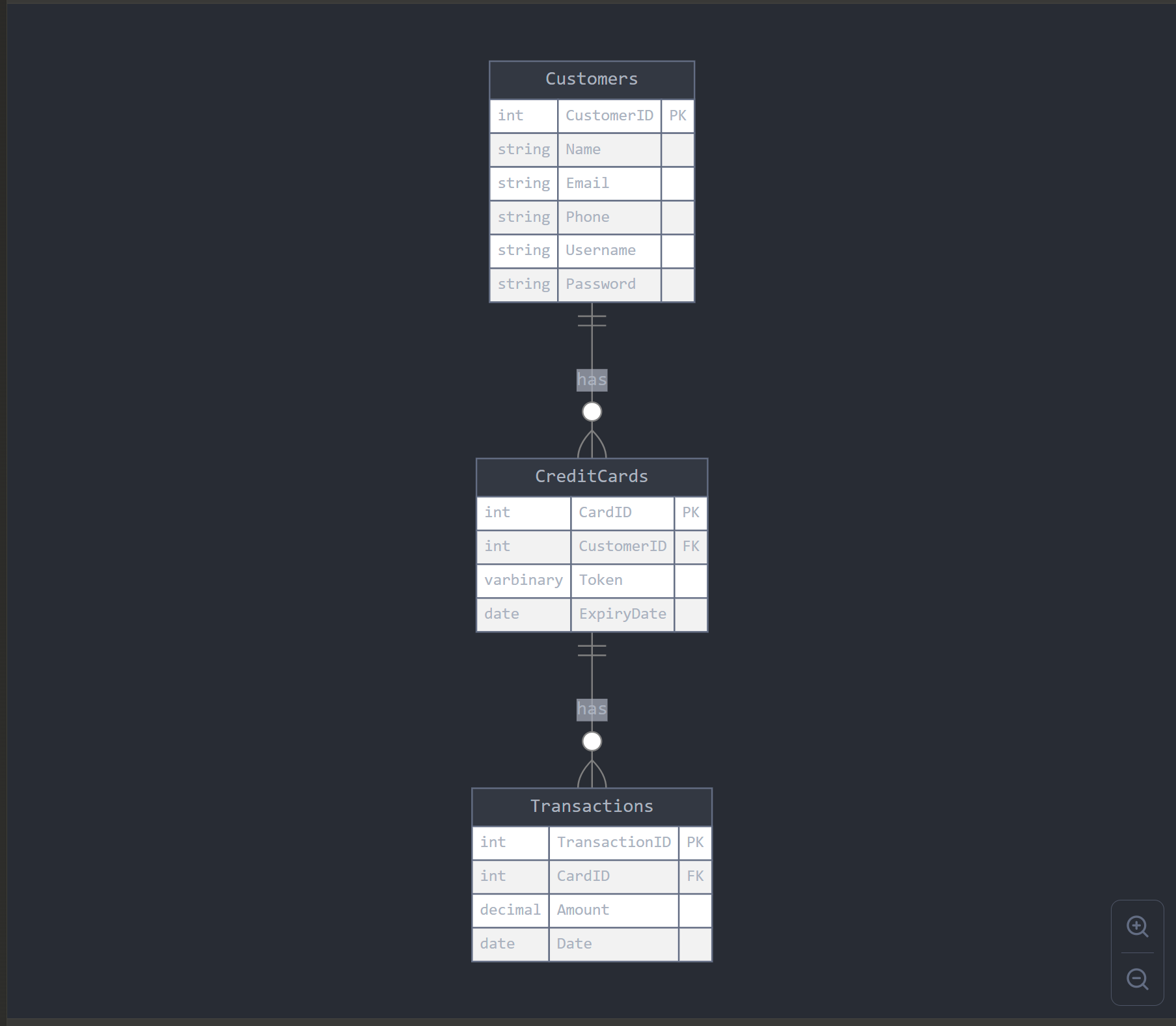
**Key Features:**

* 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.

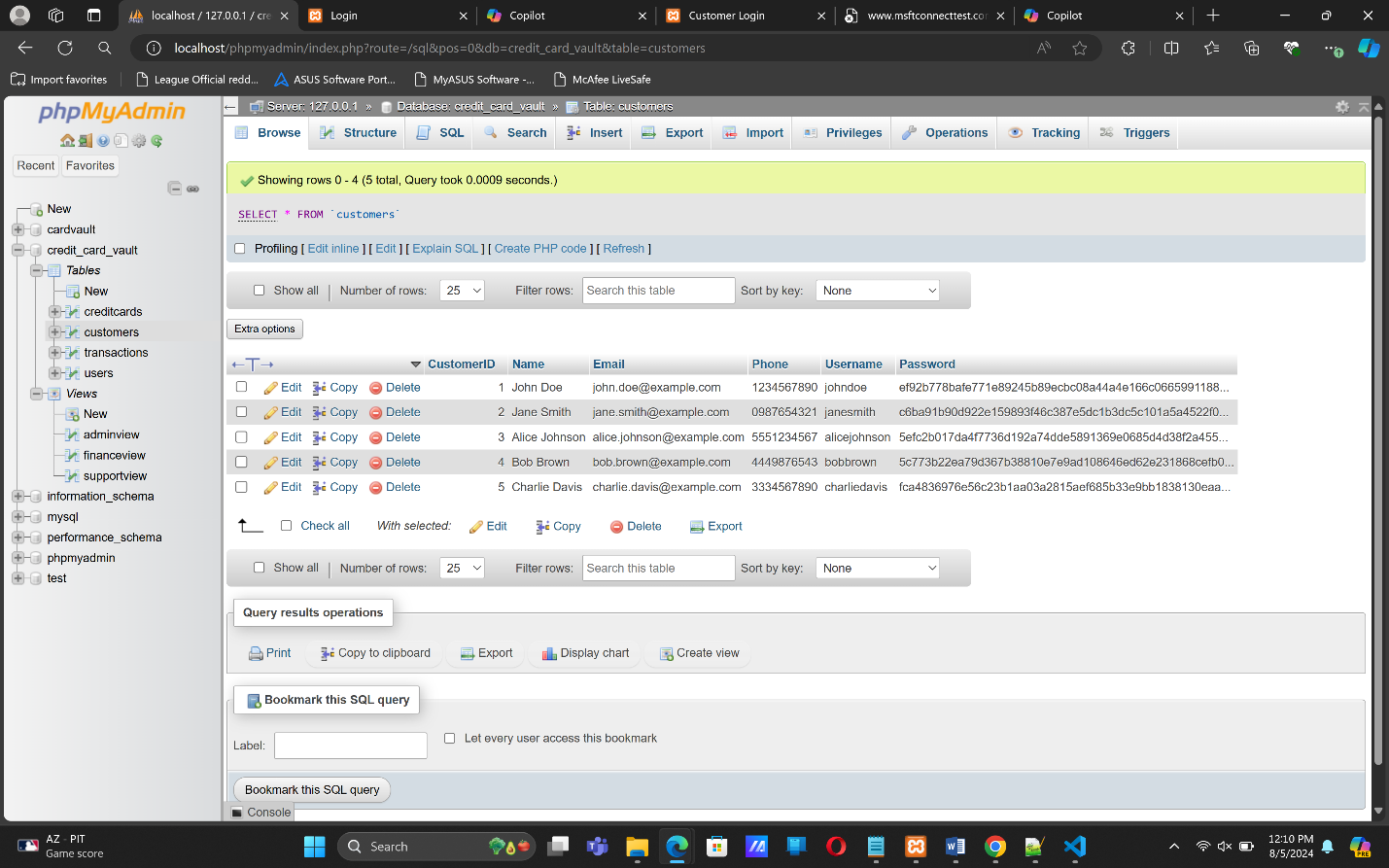
Credit Card Vault Entity Relationship Diagram:  
  


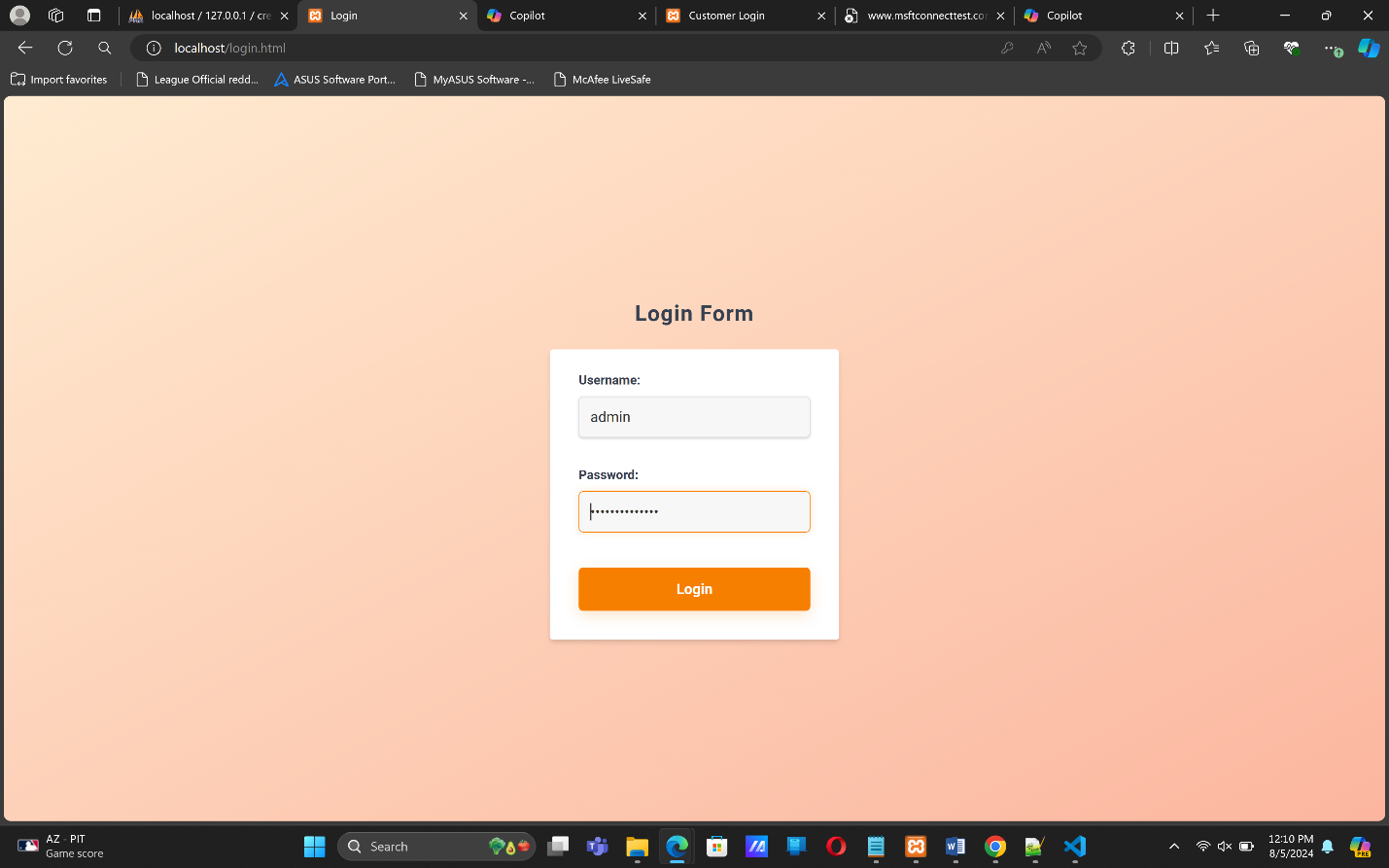
This diagram provides a clearer picture of how the data is structured and related in the Credit Card Vault system. It shows that:

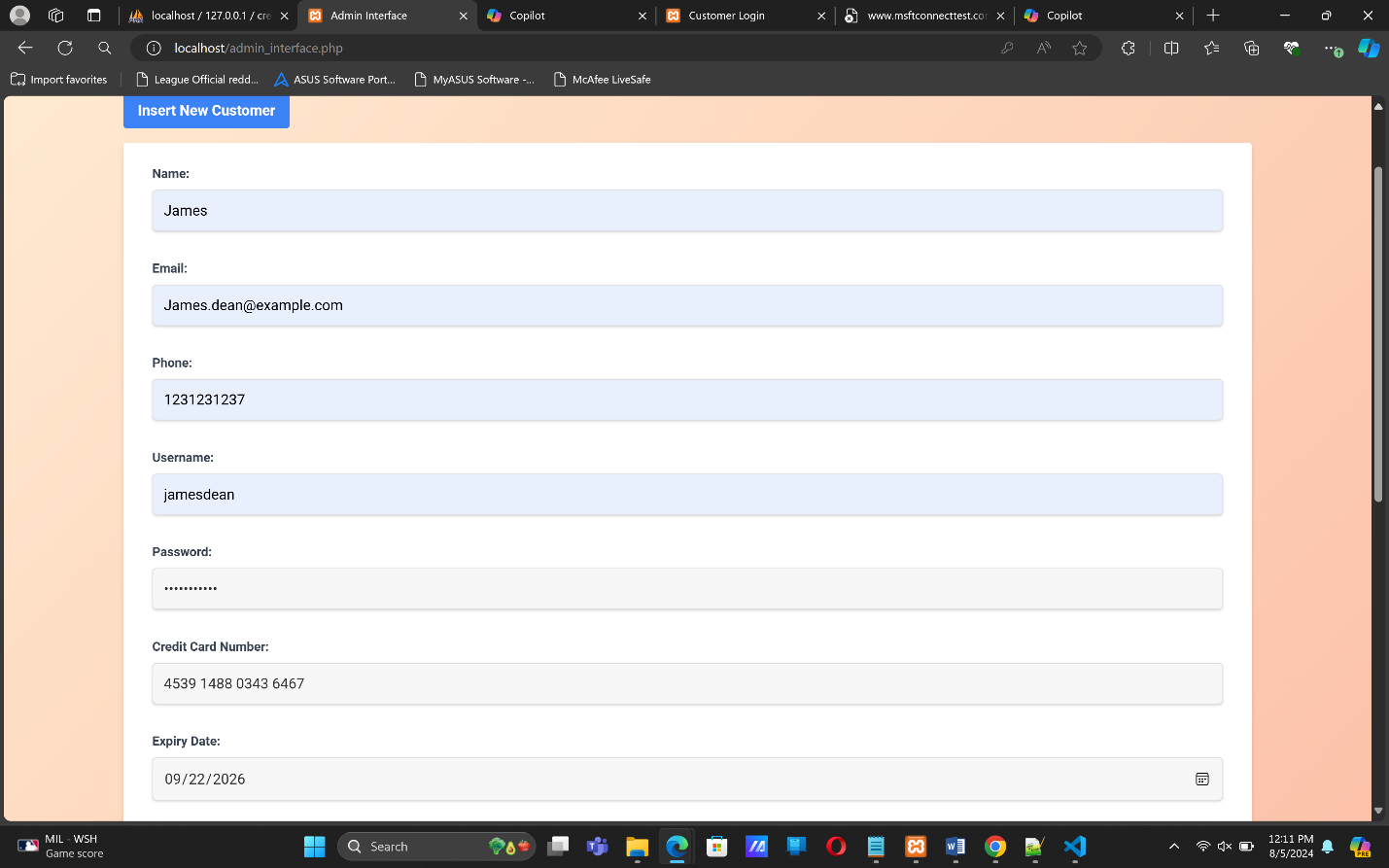
* Each customer can have multiple credit cards
* Each credit card is associated with exactly one customer
* Each credit card can have multiple transactions
* Each transaction is associated with exactly one credit card

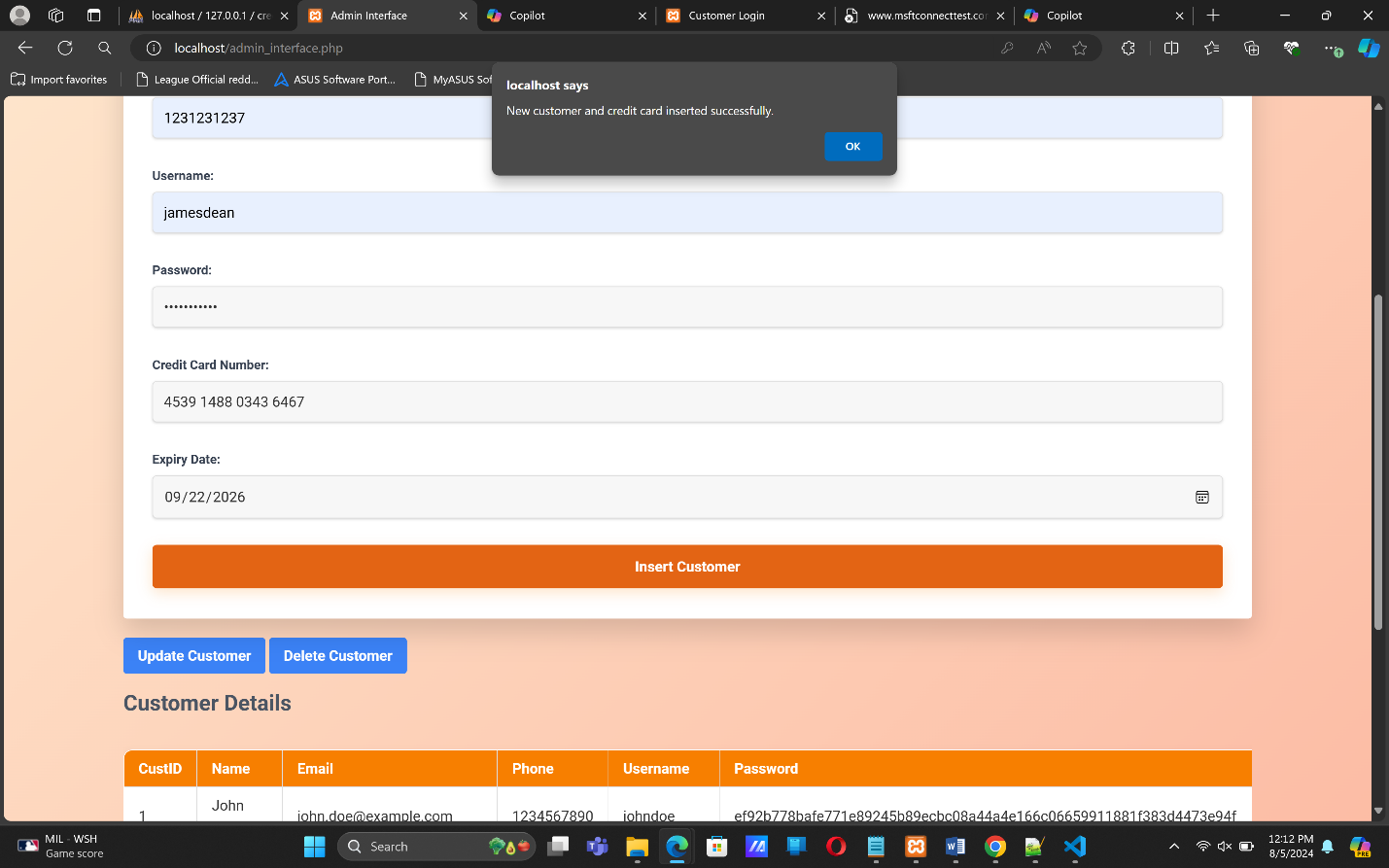
This structure is in the Third Normal Form (3NF) clearly illustrating the relationships between all entities in the system.

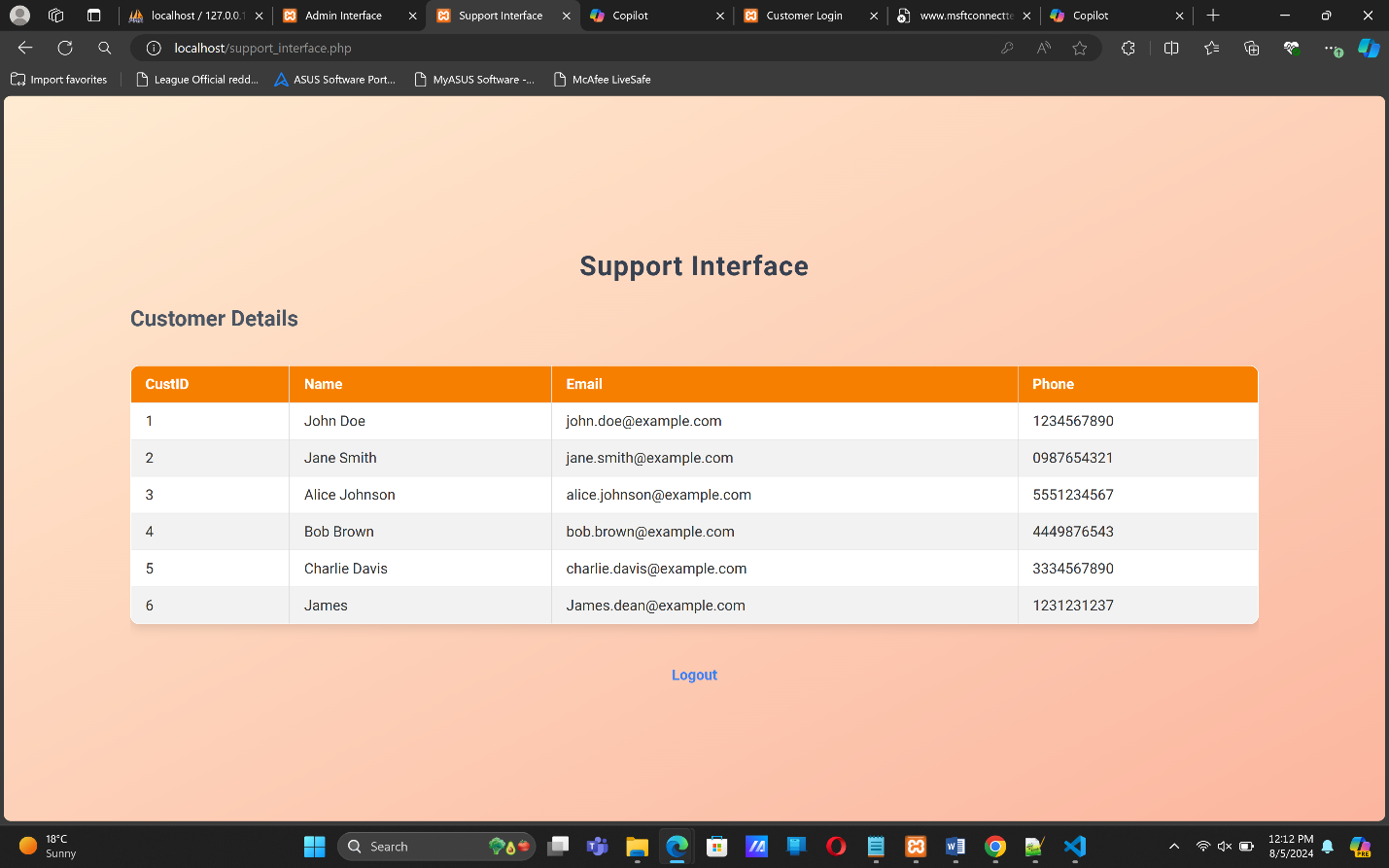
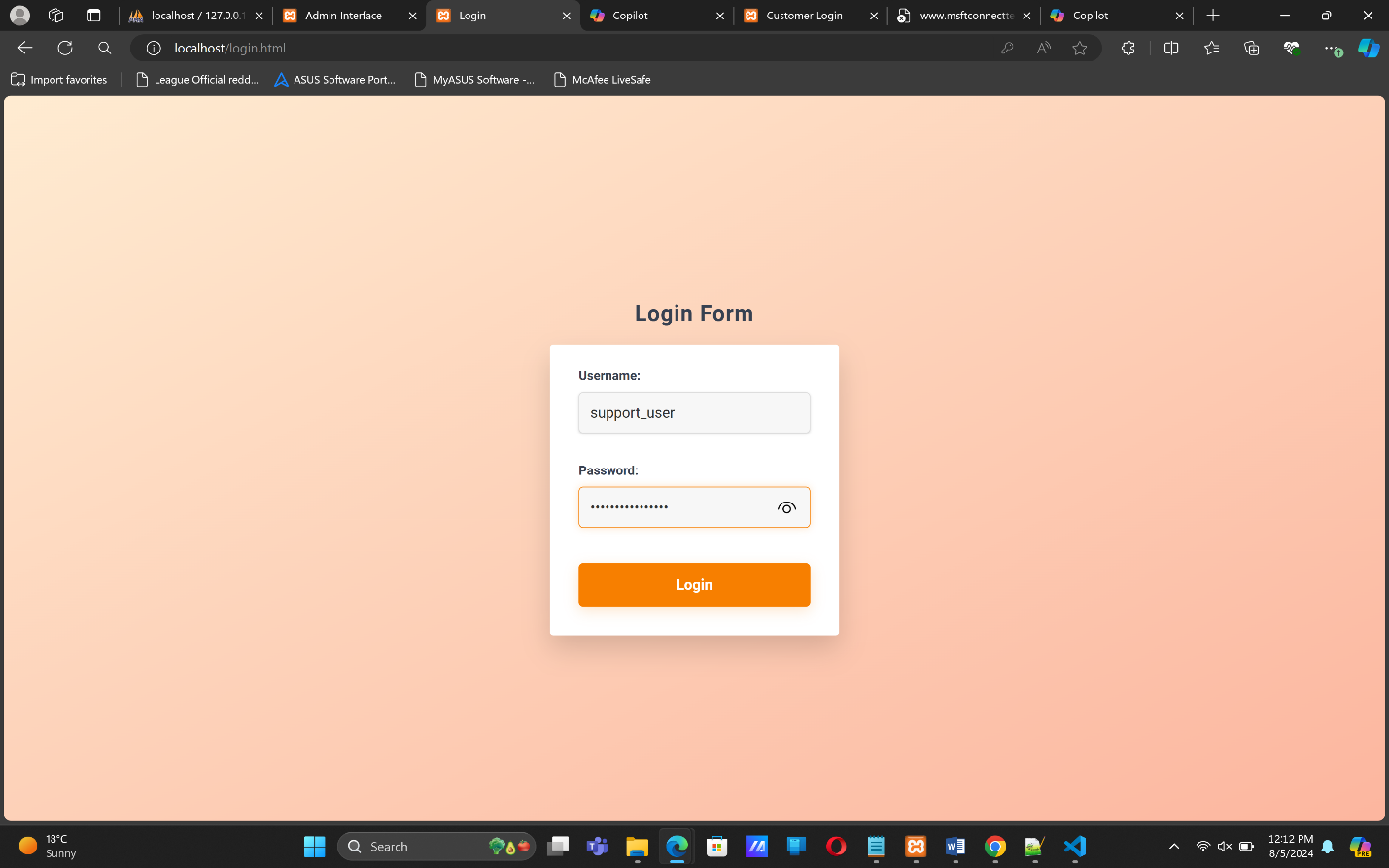
In the following sections, we'll take you on a journey through the Credit Card Vault system. We'll explore its database structure, examine how different user roles interact with the system, and demonstrate the security measures in place to protect valuable financial data.

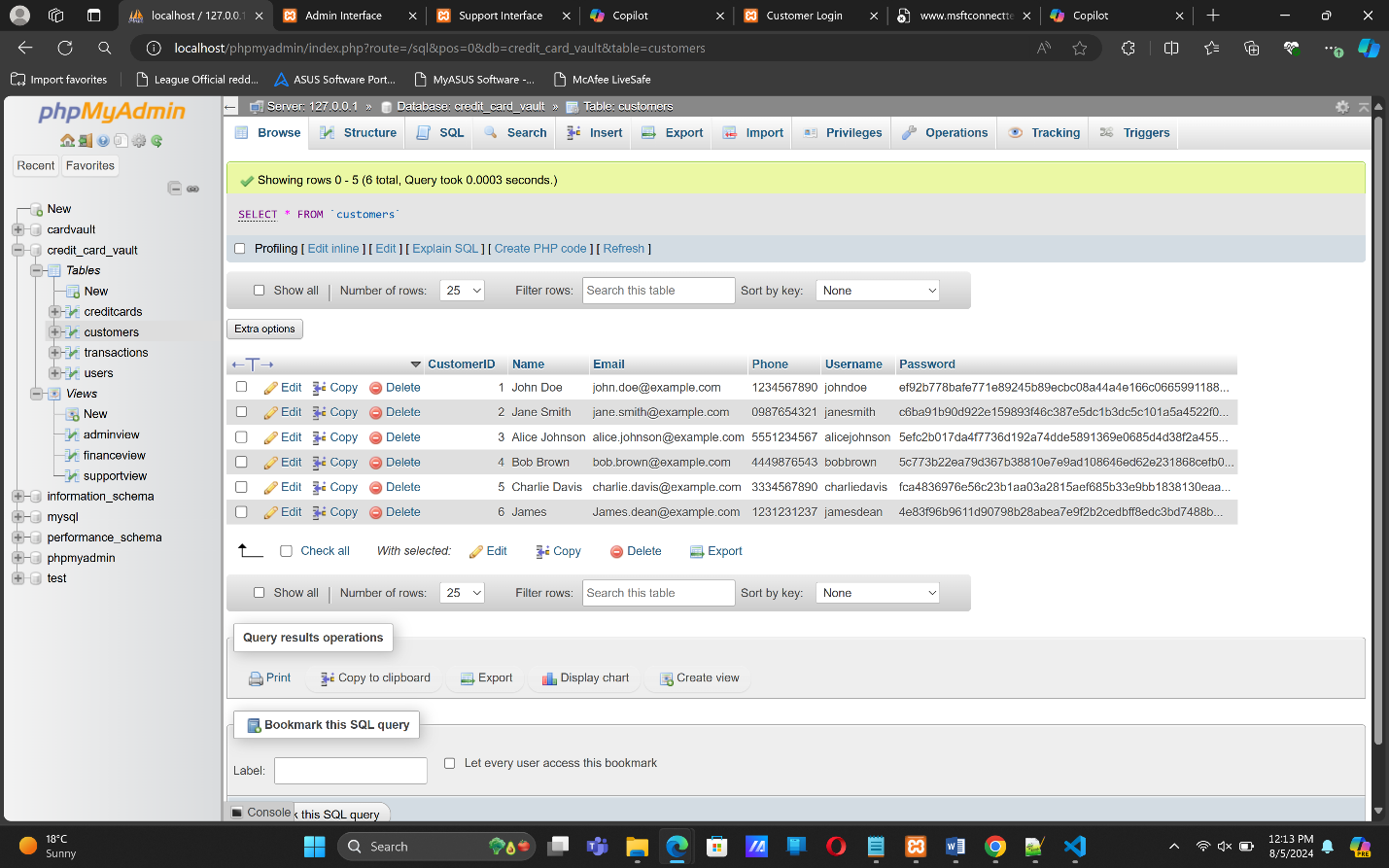


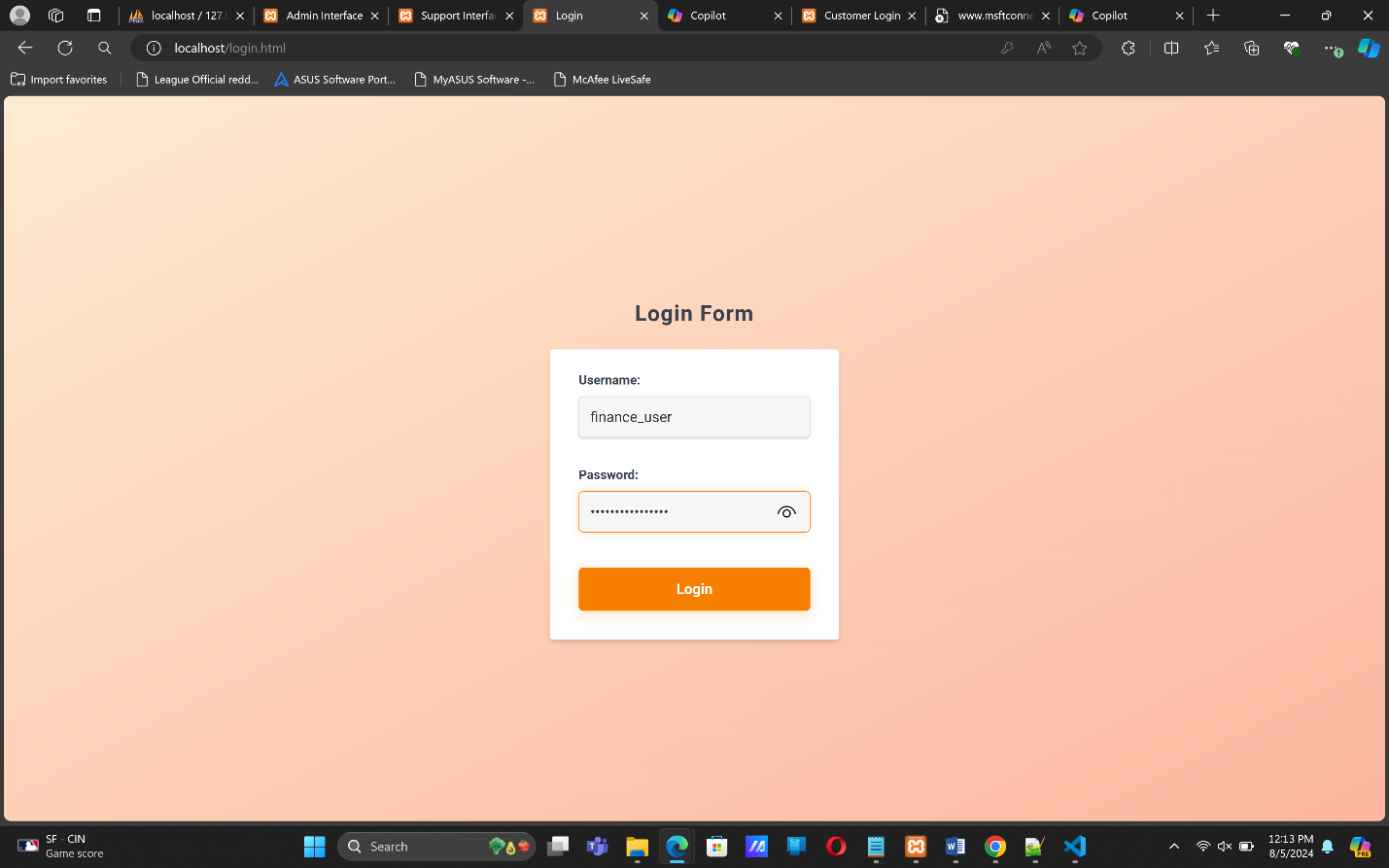
This is how the database looks like before we add a new customer. We had inserted these entries using SQL commands to act as a base to see if any effect would be there when trying to interact through the front end.

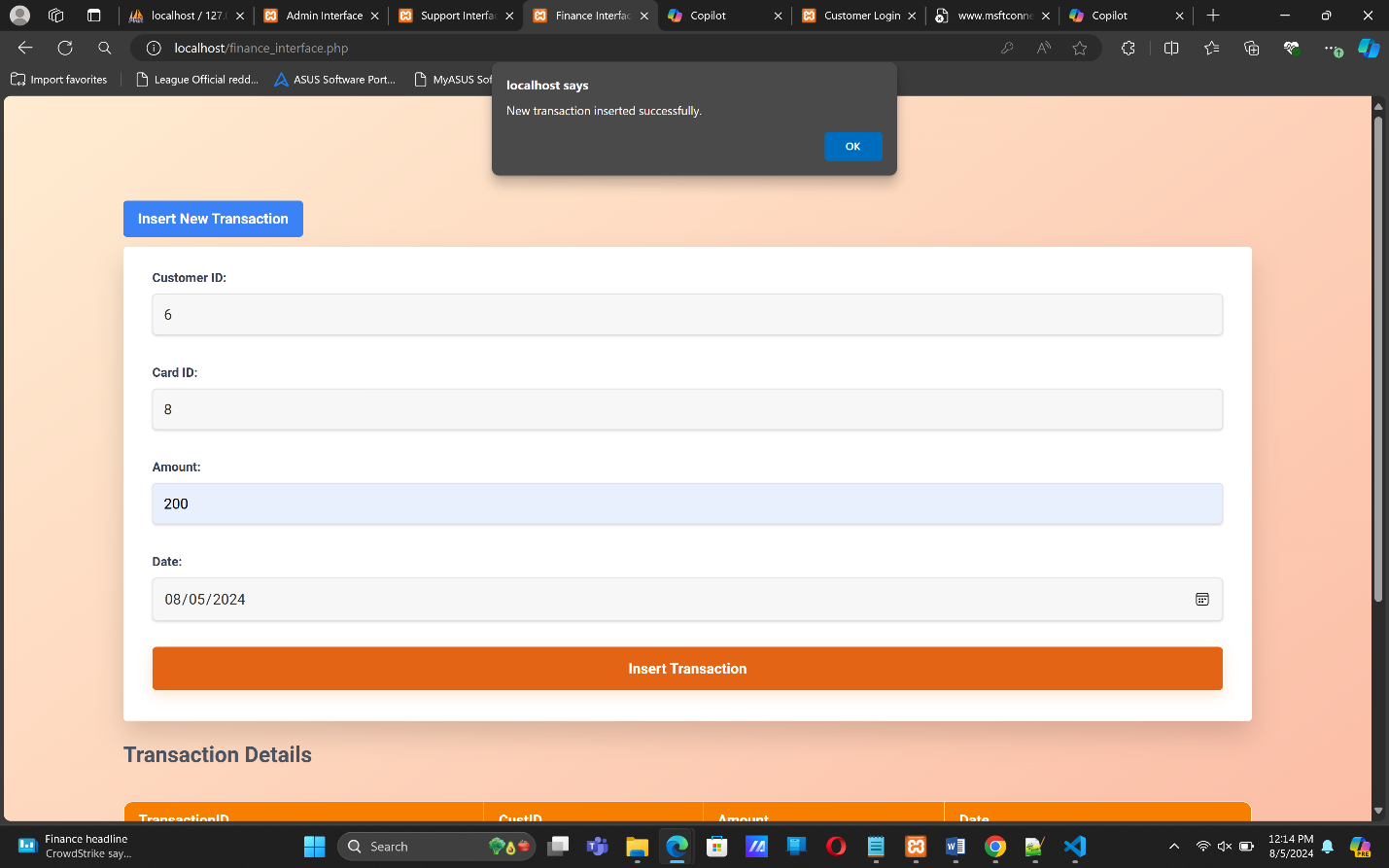
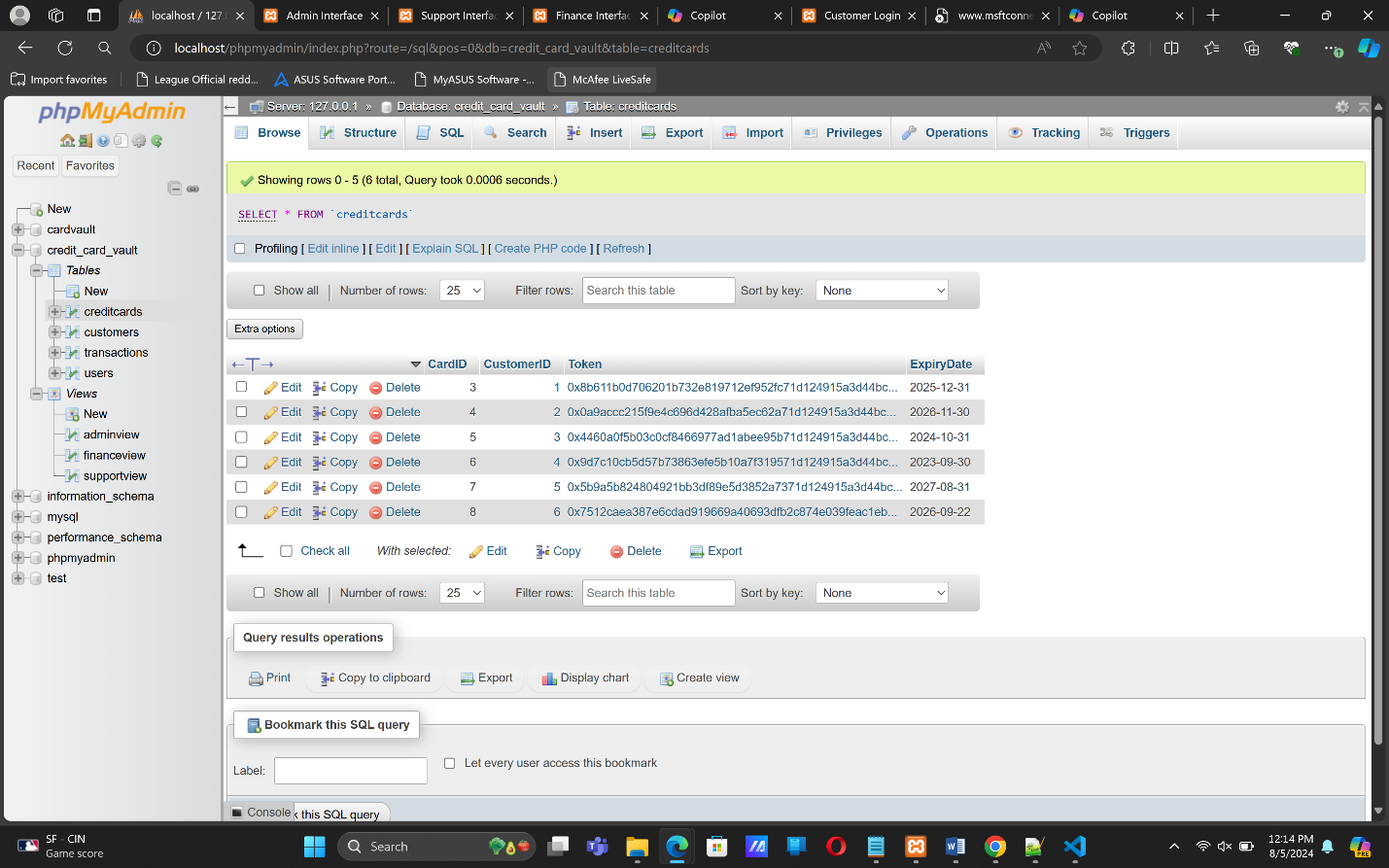
Here we have the ‘user role’ interface where the admin, finance\_user, and support user can logic to see their specific view which are determined by the privileges each have. The admin has permission to perform all CRUD operations, while the finance\_user can only enter transactions. The support user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

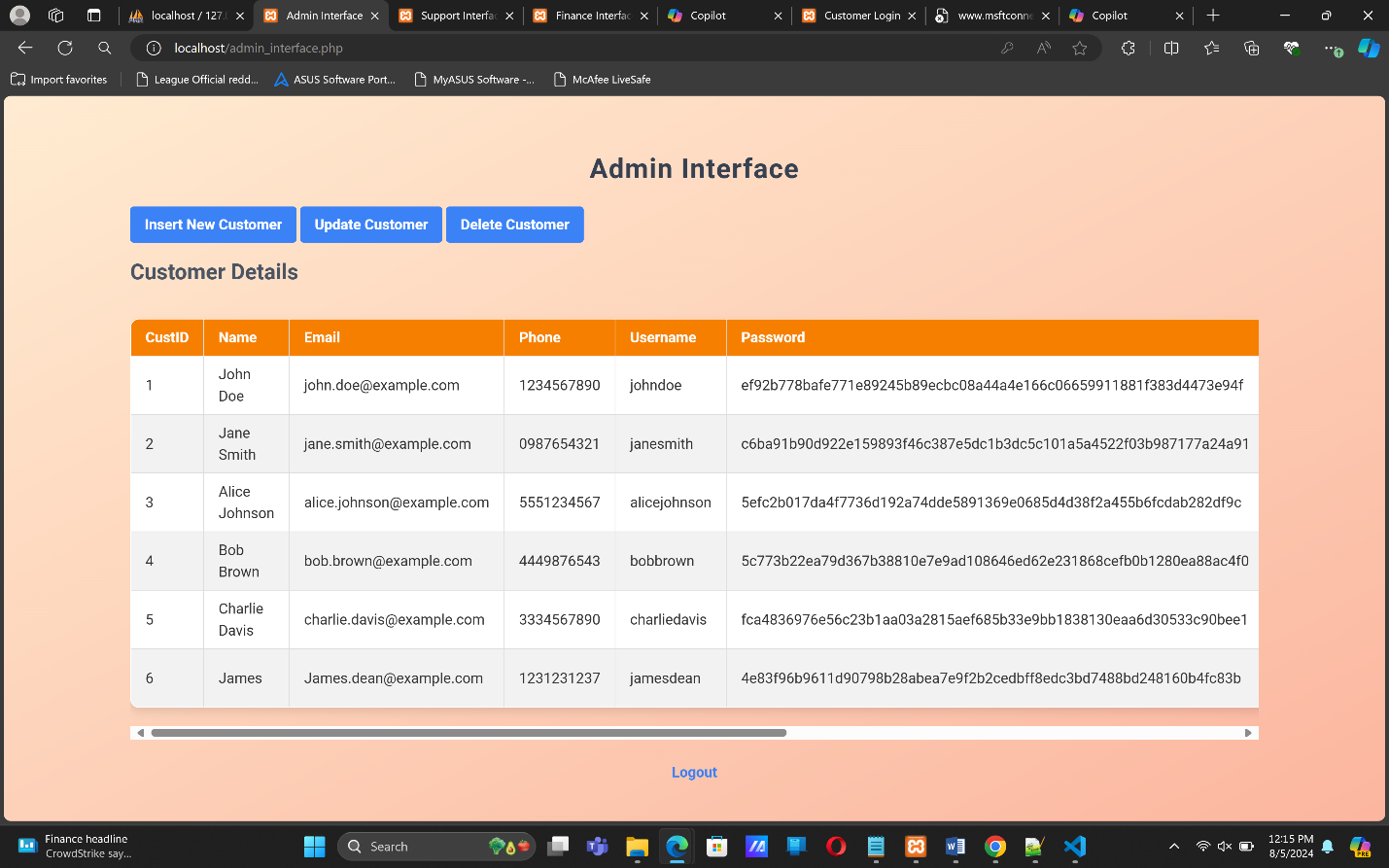
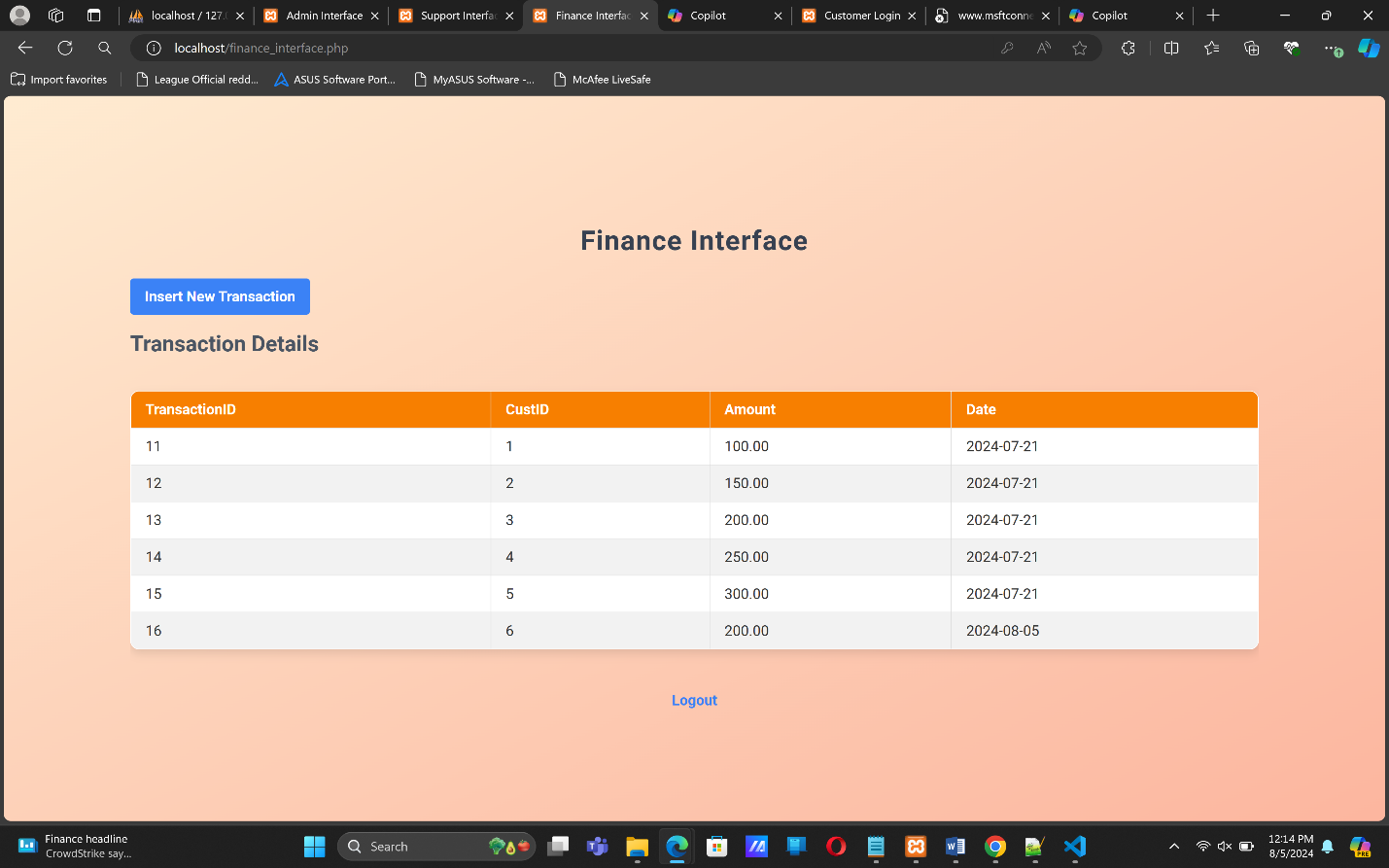
In this section we are adding a new customer through the admin. Validation checks for the credit card number are there to ensure only the correct amount and sequence of digits is allowed. On insertion we get a confirmation message for a successful entry. 

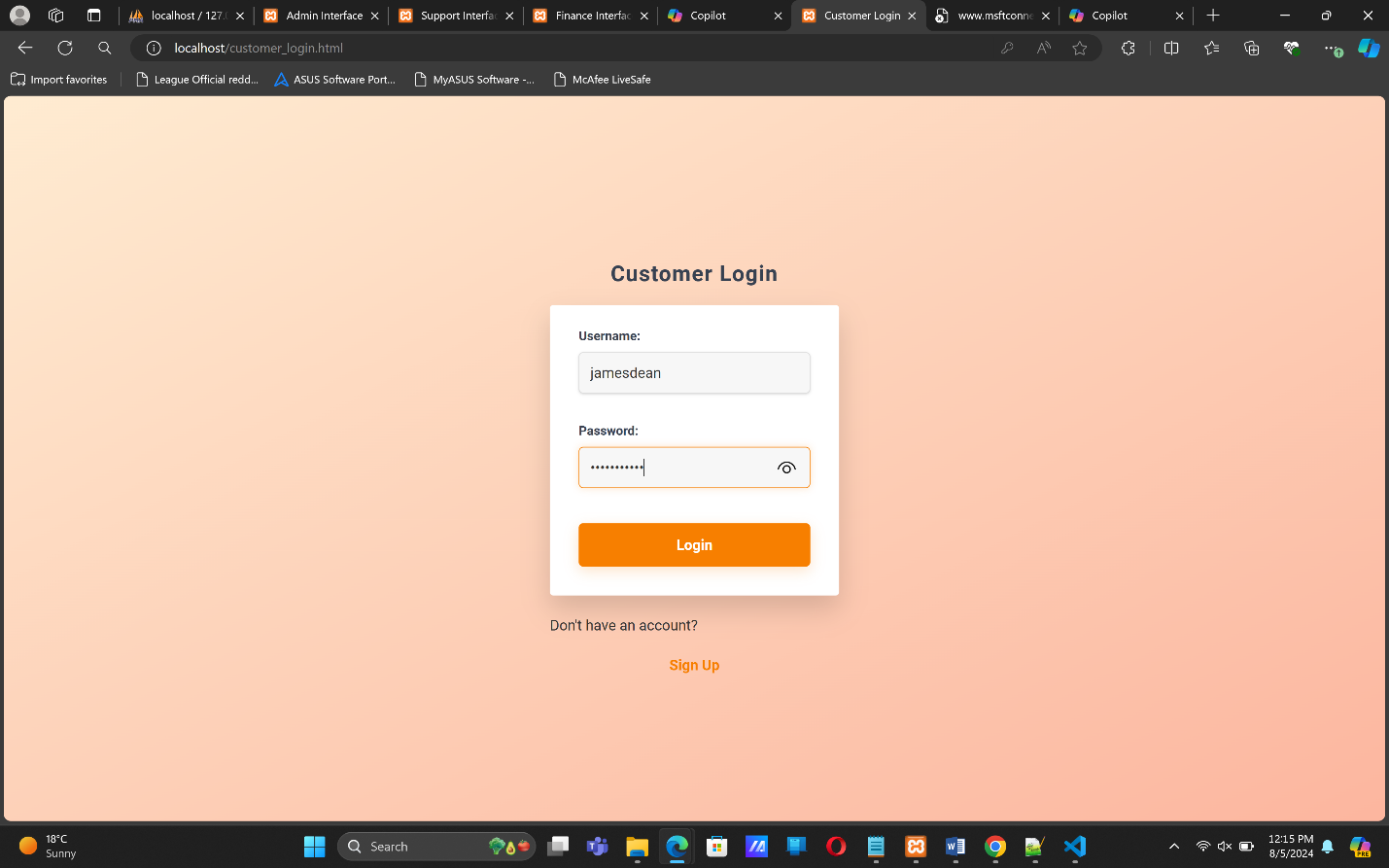
Now we login as a support user to see whether the add member can be seen on the front end.

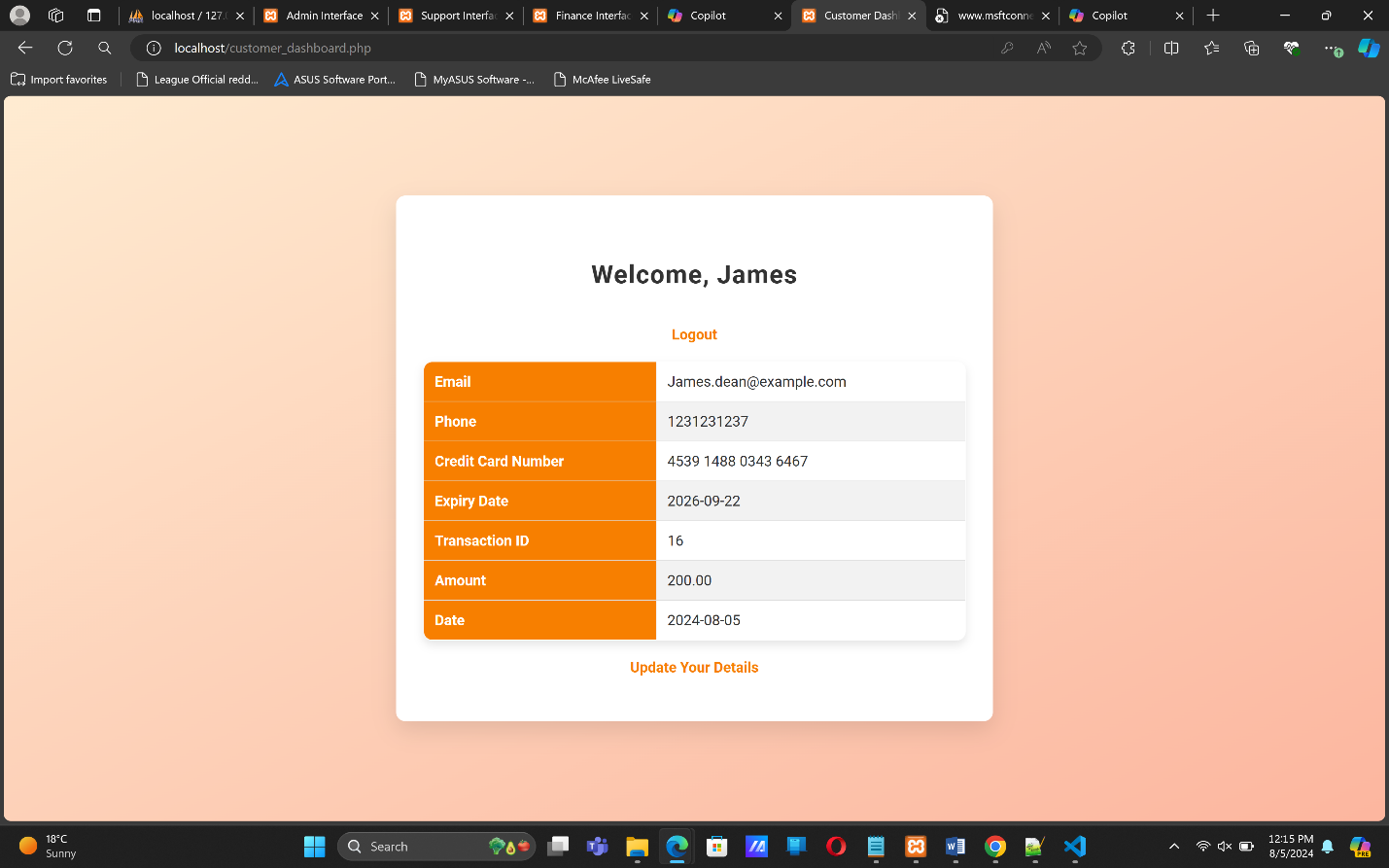
Hurray! We actually see that he has been added, and the support user has access to only afew of the new customer’s details.

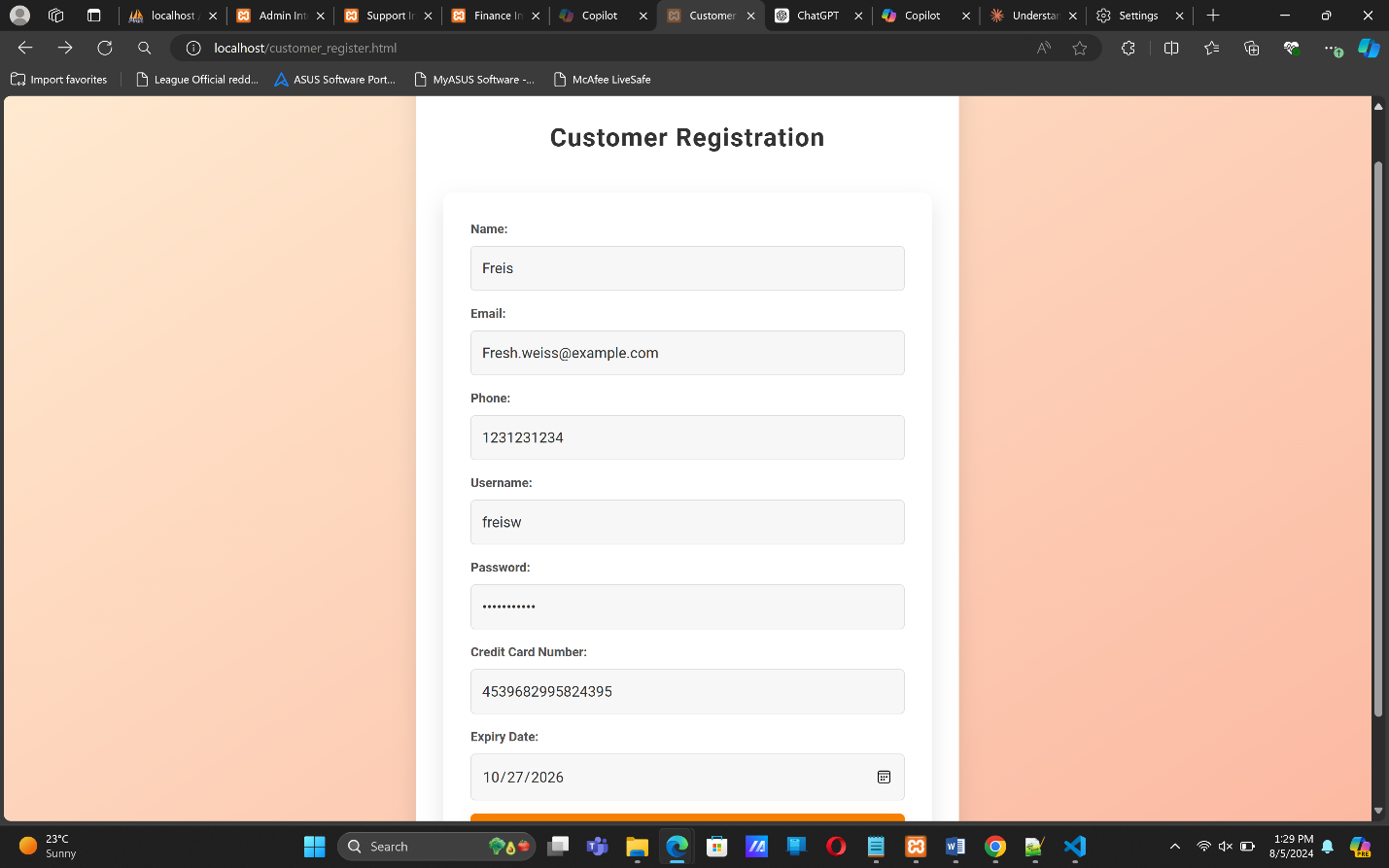
We can confirm the addition of the new customer when we take a look at the database and see that the customer has been added, and password hashed using SHA-2.

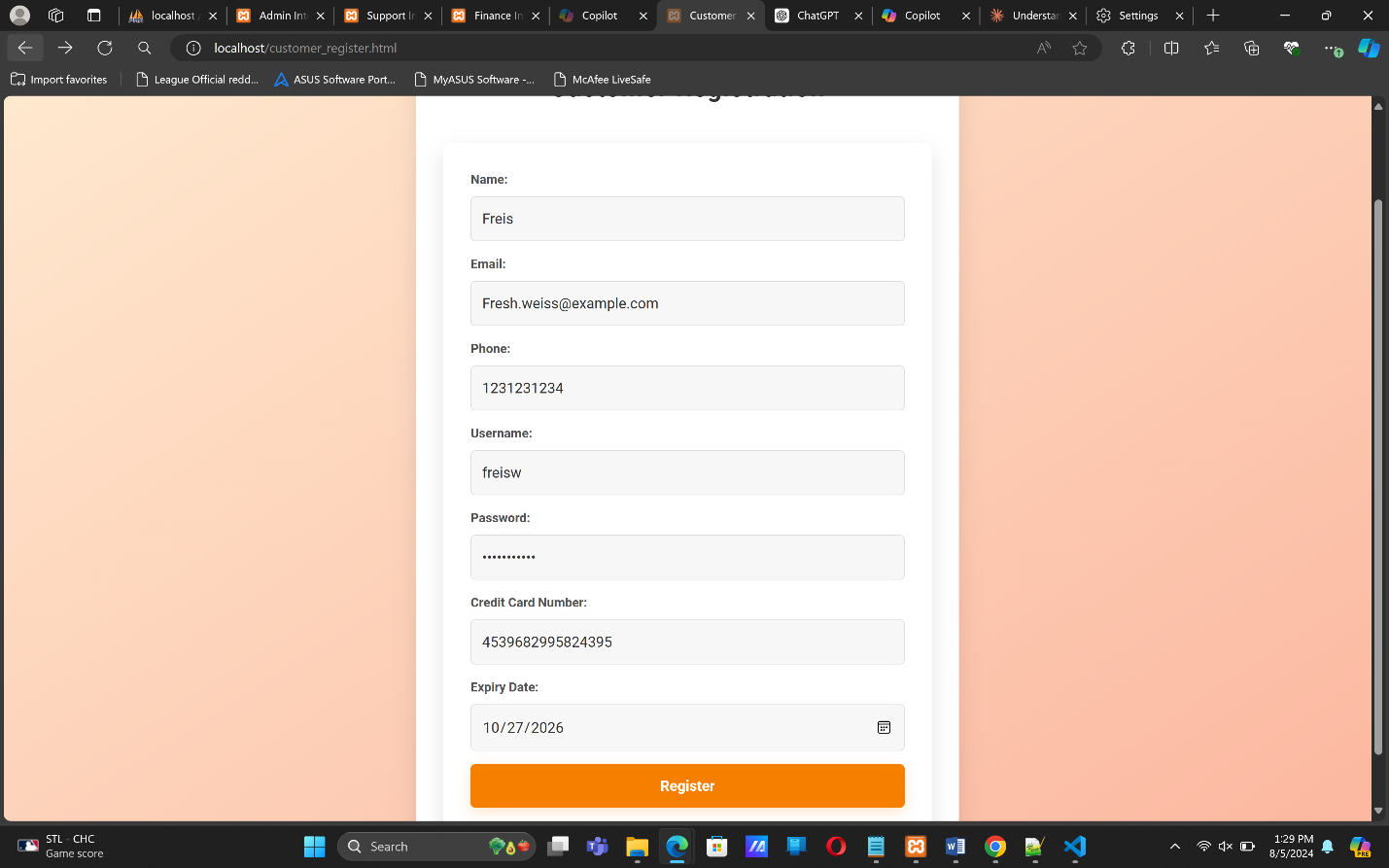
The finance user can add a new transaction and it updates the table in his view. This new transaction also updates the admin’s table and he can see the entirety of the new customer’s details. When you peek at the creditcards table, the new customer has been assigned one, details encrypted using AES.

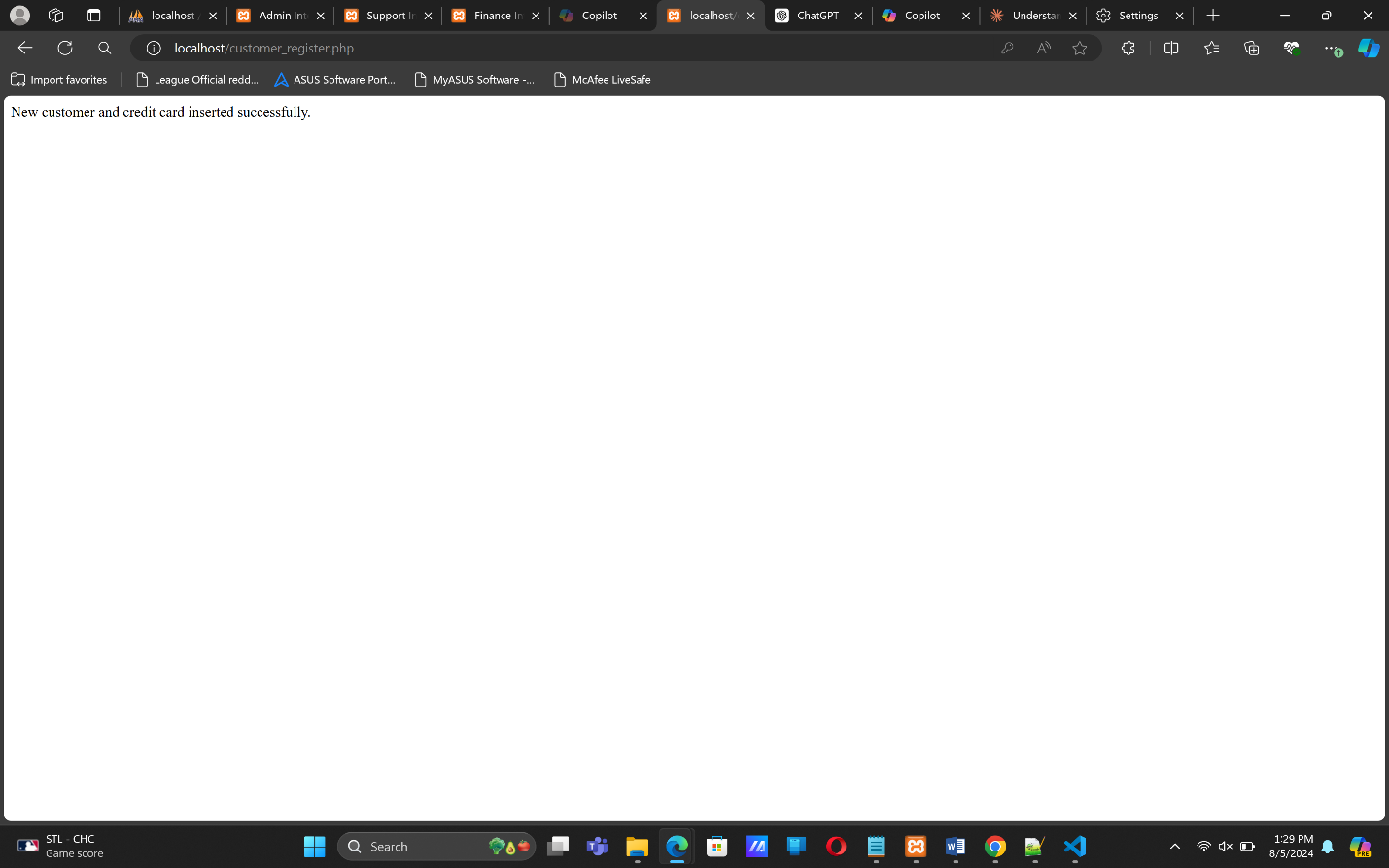
Upon adding a new transaction, we get a confirmation for a successful entry, and the transaction details are updated.

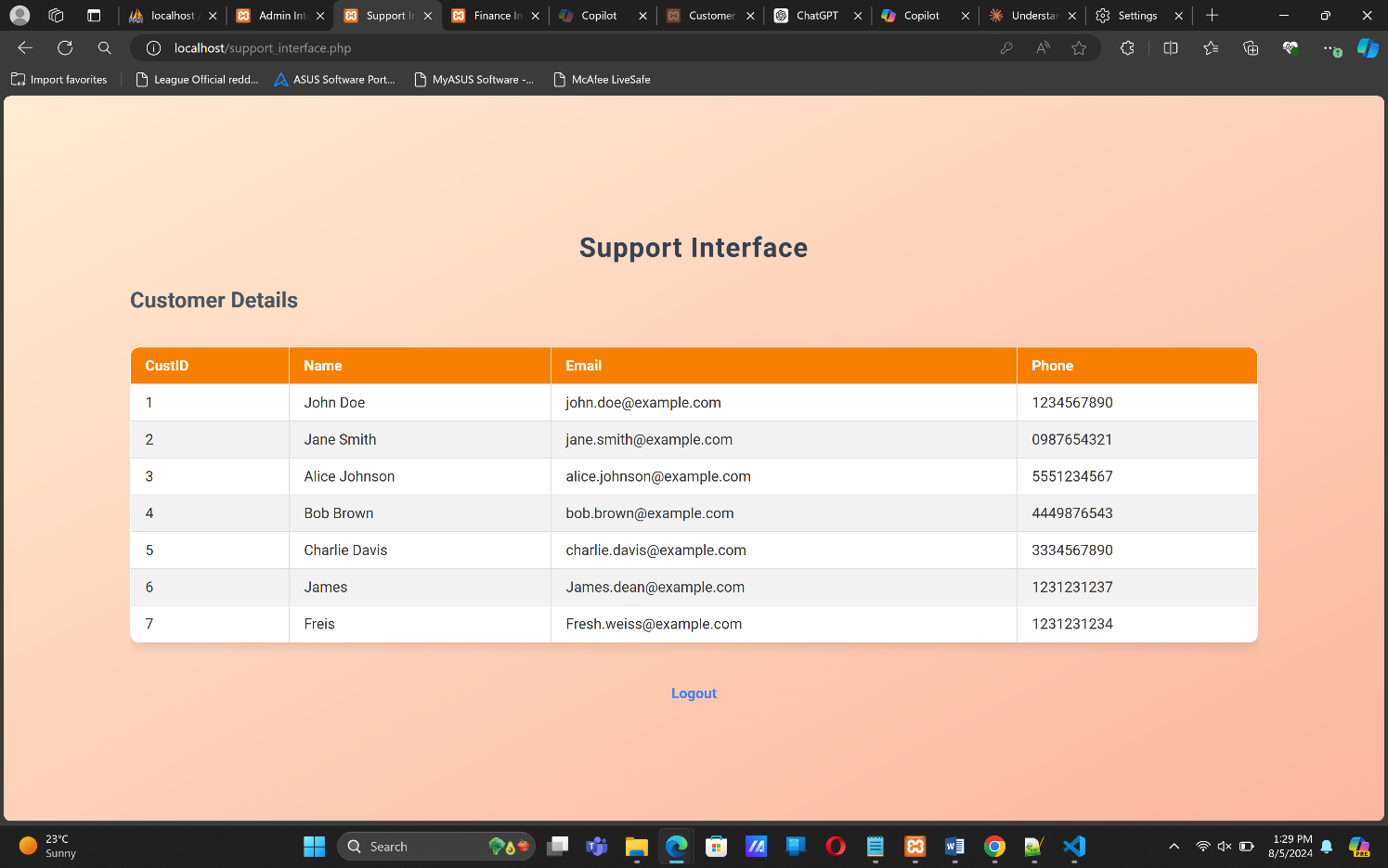
This is reflected on the admin interface as well as the database.

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.







The codes for various important functionality:

**Admin Interface**

Inserting 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);

}

// 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.min.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.