A Project Report on

“Banking Web Application”

SUBMITTED BY

Batch B Group 16

MEMBERS

YESHWANTH BALAJI A P CB.EN.U4AIE22102

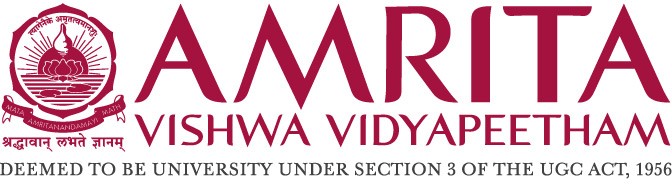
ABHISHEK S CB.EN.U4AIE22103

MADHAVA NARAYANAN KJ CB.EN.U4AIE22135

VIKASH J CB.EN.U4AIE22156

*As a part of the subject*

Introduction to Computer Networks (22AIE205)



Centre for Computational Engineering and Networking

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE - 641 112

December - 2023

DECLARATION

We hereby declare that our Project Report is a bonafide record of the project work which we have submitted to Amrita School of Computing, in partial fulfillment of the credit requirements for the degree of B.Tech in Artificial Intelligence is our authentic work. This project report has not been copied, duplicated, or plagiarized from any other paper, journal, document, or book.

This is an authentic piece of work and in case there is any query regarding the same, we shall be held responsible for answering any queries in this regard.

NAME ROLL NUMBER SIGNATURE

YESHWANTH BALAJI CB.EN.U4AIE22102

ABHISHEK CB.EN.U4AIE22103

MADHAVA NARAYANAN CB.EN.U4AIE22135

VIKASH J CB.EN.U4AIE22156

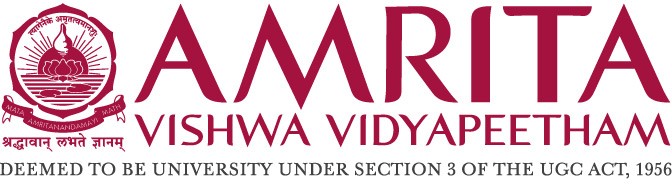
Place: Ettimadai

Date: 21-12-2023

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE - 641 112



BONAFIDE CERTIFICATE

This is to certify that the thesis entitled “Mobile Baking App” submitted by YeshwanthBalaji(CB.EN.UAIE22102),VikashJ(CB.EN.U4AIE22156),Abhishek(CB.EN.U4AIE22103),Madhava Narayanan(CB.EN.U4AIE22135), for the award of the Degree of Bachelor of Technology in the “CSE(AI) ” is a bonafide record of the work carried out by her under our guidance and supervision at Amrita School of Artificial Intelligence, Coimbatore.

Ms. Sruthi Guptha

Project Guide

Dr. K.P.Soman

Professor and Head CEN

Submitted for the university examination held on 22-12-2023

ACKNOWLEDGMENT

We would like to express our sincere gratitude to our project subject handler, Ms Sruthi Guptha, for her invaluable guidance, support, and encouragement throughout this project. Her expertise and knowledge in the field were instrumental in the success of this project.

We would like to express our profound gratitude to Dr. Soman K P of the CEN department for his contributions to the completion of our project.

This project would not have been possible without the help and guidance

of every member of our faculty.

**TABLE OF CONTENTS**

[LIST OF FIGURES 5](#_Toc154129379)

[ABSTRACT: 6](#_Toc154129380)

[INTRODUCTION: 7](#_Toc154129381)

[TCP CONNECTION: 8](#_Toc154129382)

[SOCKET PROGRAMMING: 8](#_Toc154129383)

[CODE: 9](#_Toc154129384)

[CODE EXPLANATION: 14](#_Toc154129385)

[OUTPUT: 19](#_Toc154129386)

[CONCLUSION: 20](#_Toc154129387)

# LIST OF FIGURES

[Figure 1 Open Account Form 19](#_Toc154129306)

[Figure 2 Balance checking form 19](#_Toc154129307)

[Figure 3 Transfer Amount form 20](#_Toc154129308)

BANKING WEB APPLICATION

# ABSTRACT:

This report presents the design and implementation of a simple client-server banking system using socket programming in Python. The system enables clients to perform various banking operations, such as opening an account, depositing and withdrawing funds, checking balances, displaying customer details, and closing accounts. The server-side implementation utilizes MySQL for database management, storing customer information and account balances. The client-side application interacts with the server by sending requests and receiving responses over a socket connection.

The client application provides a user-friendly interface for customers to interact with the banking system, while the server processes these requests, updates the database, and responds accordingly. The server incorporates essential functionalities, including opening and closing accounts, managing account balances, and providing detailed customer information.

The report details the client and server implementations, outlining the structure, functionalities, and communication protocols. Additionally, it discusses the MySQL database integration for persistent data storage. The system's modularity and extensibility allow for future enhancements and the incorporation of additional features.

This project serves as a foundational example of a distributed banking system, showcasing the integration of socket programming and database management to create a reliable and responsive financial services application. The report concludes with potential areas for improvement and future development, emphasizing the adaptability and scalability of the implemented solution .

# INTRODUCTION:

In the rapidly evolving landscape of financial technology, the integration of digital solutions has become paramount for enhancing accessibility and efficiency in banking services. This report introduces a comprehensive client-server banking system developed through socket programming in Python, coupled with MySQL for database management. The system aims to provide users with a seamless and secure interface to perform a variety of banking operations, fostering financial inclusion and streamlined customer experiences.

The modern banking sector faces the challenge of meeting the diverse needs of a tech-savvy clientele while ensuring robust security measures. This project addresses this challenge by presenting a dynamic client-server architecture, enabling clients to interact with the banking system in real-time. The server-side implementation incorporates MySQL, a widely-used relational database management system, to store and manage customer information, account details, and transaction records.

The client application offers a user-friendly interface, allowing customers to execute essential banking operations such as opening new accounts, depositing and withdrawing funds, checking balances, retrieving detailed customer information, and closing accounts. This client-server model facilitates efficient communication between end-users and the banking system, promoting a responsive and reliable financial service environment.

The subsequent sections of this report delve into the intricacies of the client and server implementations, highlighting the design principles, functionalities, and communication protocols. Additionally, the integration of MySQL for persistent data storage is explored, emphasizing the importance of maintaining accurate and secure financial records.

As the digital transformation in banking continues to evolve, this project serves as a testament to the adaptability and versatility of technology in meeting the demands of a contemporary banking landscape. The report concludes with insights into potential areas for further enhancement, acknowledging the

ongoing evolution of financial technology and the ever-expanding possibilities for innovative solutions in the banking sector.

***Protocols used:*** TCP (Transmission Control Protocol)

# TCP CONNECTION:

The Transmission Control Protocol (TCP) forms the backbone of reliable communication in computer networks, ensuring the orderly and error-free exchange of data between devices. In the context of the provided code, TCP is employed to establish a communication channel between the client and server components of the banking system. The client and server applications each create a socket, a communication endpoint, through which they connect to each other. The server, initialized on a specific host and port, listens for incoming connection requests from clients. When a client requests a connection, the server accepts the connection, forming a TCP socket connection. This connection is established as a reliable, bidirectional stream, enabling the seamless exchange of data between the client and server.

In the code, the socket module in Python is utilized to create and manage TCP socket connections. The server application binds to a specific host and port, listens for incoming connections, and responds to client requests. The client application, on the other hand, connects to the server using its host and port information. Once the connection is established, the client can send various commands, such as opening an account or checking balances, to the server. The server, in turn, processes these requests, performs the necessary database operations, and sends back the appropriate responses over the established TCP connection. This robust and reliable TCP connection ensures that data is transmitted accurately and efficiently between the client and server components, forming the basis for a responsive and secure banking system.

# SOCKET PROGRAMMING:

Socket programming is a fundamental aspect of network communication, allowing applications on different devices to establish connections and exchange data. At its core, a socket serves as an endpoint for communication, enabling processes on different devices to send and receive information. Socket programming involves the creation, configuration, and management of these communication endpoints, facilitating both the connection establishment and data transfer between client and server applications. In the context of the provided code, Python's socket module is leveraged for socket programming. This module provides a set of methods and classes that simplify the creation and management of sockets, making it accessible for developers to implement networked applications.

Sockets operate on the client-server model, where a server listens for incoming connection requests, and a client initiates a connection to the server. Once a connection is established, data can be exchanged bidirectionally. Socket programming supports various communication protocols, with Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) being the most common. TCP ensures reliable and ordered data transmission, making it suitable for applications where data integrity is critical, as demonstrated in the banking system code. On the other hand, UDP provides a lightweight and connectionless communication method, suitable for scenarios where real-time data exchange is prioritized over reliability. Socket programming thus forms the basis for a wide array of networked applications, enabling the development of scalable and responsive systems.

# CODE:

import mysql.connector

import socket

import threading as td

host = socket.gethostname()

port = 5000

server\_socket = socket.socket()

server\_socket.bind((host, port))

server\_socket.listen(1)

def handle\_open\_account(server\_socket):

    mydb = mysql.connector.connect(host='localhost', user='root', password='Vikash04', database='bank\_management')

    data = server\_socket.recv(1024).decode()

    n, ac, db, add, cn, ob, p = data.split(',')

    data1 = (n, ac, db, add, cn, ob, p)

    data2 = (n, ac, ob, p)

    sql1 = 'INSERT INTO account VALUES (%s, %s, %s, %s, %s, %s, %s)'

    sql2 = 'INSERT INTO amount VALUES (%s, %s, %s, %s)'

    x = mydb.cursor()

    try:

        x.execute(sql1, data1)

    except Exception as e :

        print(e)

        server\_socket.send("error pk".encode())

    server\_socket.send("Account created successfully".encode())

    x.execute(sql2, data2)

    mydb.commit()

    mydb.close()

    print("Data Entered successfully")

def handle\_deposit\_amount(server\_socket):

    mydb = mysql.connector.connect(host='localhost', user='root', password='Vikash04', database='bank\_management')

    amount\_info = server\_socket.recv(1024).decode()

    amount, ac, p = amount\_info.split(',')

    a = 'SELECT balance FROM amount WHERE AccNo = %s AND Pin = %s'

    data = (ac, p)

    x = mydb.cursor()

    x.execute(a, data)

    result = x.fetchone()

    updated\_balance = result[0] + int(amount)

    sql = 'UPDATE amount SET balance = %s WHERE AccNo = %s'

    d = (updated\_balance, ac)

    x.execute(sql, d)

    mydb.commit()

    server\_socket.send(str(updated\_balance).encode())

    mydb.close()

def handle\_withdraw\_amount(server\_socket):

    mydb = mysql.connector.connect(host='localhost', user='root', password='Vikash04', database='bank\_management')

    amount\_info = server\_socket.recv(1024).decode()

    amount, ac, p = amount\_info.split(',')

    a = 'SELECT balance FROM amount WHERE AccNo = %s AND Pin = %s'

    data = (ac,p)

    x = mydb.cursor()

    x.execute(a, data)

    result = x.fetchone()

    updated\_balance = result[0] - int(amount)

    sql = 'UPDATE amount SET balance = %s WHERE AccNo = %s'

    d = (updated\_balance, ac)

    x.execute(sql, d)

    mydb.commit()

    server\_socket.send(str(updated\_balance).encode())

    mydb.close()

def handle\_check\_balance(server\_socket):

    mydb = mysql.connector.connect(host='localhost', user='root', password='Vikash04', database='bank\_management')

    data = server\_socket.recv(1024).decode()

    ac,p = data.split(',')

    data = (ac,p)

    a = 'SELECT balance FROM amount WHERE AccNo = %s AND Pin = %s'

    x = mydb.cursor()

    x.execute(a, data)

    result = x.fetchone()

    if result:

        server\_socket.send(str(result[0]).encode())

    else:

        print("Invalid account number")

        server\_socket.send("Invalid account number or pin".encode())

    mydb.close()

def handle\_display\_details(server\_socket):

    mydb = mysql.connector.connect(host='localhost', user='root', password='Vikash04', database='bank\_management')

    data = server\_socket.recv(1024).decode()

    ac,p = data.split(',')

    data = (ac,p)

    a = 'SELECT Name,AccNo,Balance FROM amount WHERE AccNo = %s AND Pin = %s'

    x = mydb.cursor()

    x.execute(a, data)

    result = x.fetchone()

    print(result)

    if result:

        formatted\_details = f"Name: {result[0]}, Account No: {result[1]}, Balance: {result[2]}"

        print(formatted\_details)

        server\_socket.send(formatted\_details.encode())

    else:

        server\_socket.send("Invalid account number".encode())

    mydb.close()

def handle\_close\_account(server\_socket):

    mydb = mysql.connector.connect(host='localhost', user='root', password='Vikash04', database='bank\_management')

    data = server\_socket.recv(1024).decode()

    ac,p = data.split(',')

    data = (ac,p)

    a = 'SELECT Name FROM amount WHERE AccNo = %s AND Pin = %s'

    sql1 = 'DELETE FROM account WHERE AccNo = %s'

    sql2 = 'DELETE FROM amount WHERE AccNo = %s'

    x = mydb.cursor()

    x.execute(a, data)

    result = x.fetchone()

    print(result)

    if result:

        x.execute(sql1, (ac,))

        x.execute(sql2, (ac,))

        server\_socket.send("Account closed ".encode())

    else:

        server\_socket.send("Account No or pin incorrect".encode())

    mydb.commit()

    mydb.close()

def handle\_Transfer\_amount(server\_socket):

    mydb = mysql.connector.connect(host='localhost', user='root', password='Vikash04', database='bank\_management')

    data = server\_socket.recv(1024).decode()

    ac1, ac2, p, amount = data.split(',')

    print(data)

*# Update balance for account 1*

    data1 = (ac1, p)

    print(data1)

    query\_ac1 = 'SELECT balance FROM amount WHERE AccNo = %s AND Pin = %s'

    x = mydb.cursor()

    x.execute(query\_ac1, data1)

    result\_ac1 = x.fetchone()

    if result\_ac1:

        updated\_balance\_ac1 = result\_ac1[0] - int(amount)

        update\_query\_ac1 = 'UPDATE amount SET balance = %s WHERE AccNo = %s'

        data\_ac1 = (updated\_balance\_ac1, ac1)

        x.execute(update\_query\_ac1, data\_ac1)

        mydb.commit()

*# server\_socket.send(str(updated\_balance\_ac1).encode())*

    else:

        server\_socket.send("Account 1 details not found".encode())

        x.close()

        mydb.close()

*# Update balance for account 2*

    data2 = (ac2, )

    print(data2)

    query\_ac2 = 'SELECT balance FROM amount WHERE AccNo = %s'

    x.execute(query\_ac2, data2)

    result\_ac2 = x.fetchone()

    if result\_ac2:

        updated\_balance\_ac2 = result\_ac2[0] + int(amount)

        update\_query\_ac2 = 'UPDATE amount SET balance = %s WHERE AccNo = %s'

        data\_ac2 = (updated\_balance\_ac2, ac2)

        x.execute(update\_query\_ac2, data\_ac2)

        mydb.commit()

        res = "Amount transfered successfully. Current Balance ="+ str(updated\_balance\_ac1)

        server\_socket.send(res.encode())

    else:

        server\_socket.send("Account 2 details not found".encode())

    x.close()

    mydb.close()

def process\_request(data):

    if data.lower().strip() == '1':

        return "OpenAcc"

    elif data.lower().strip() == '2':

        return "DepositAmount"

    elif data.lower().strip() == '3':

        return "WithdrawAmount"

    elif data.lower().strip() == '4':

        return "Balcheck"

    elif data.lower().strip() == '5':

        return "Disdetails"

    elif data.lower().strip() == '6':

        return "Transfer Amount"

    elif data.lower().strip() == '7':

        return "CloseAcc"

    else:

        return "Invalid request"

def handle(conn):

    while True:

        data = conn.recv(1024).decode()

        if not data or data.lower().strip() == 'exit':

            break

        response = process\_request(data)

        if response == 'OpenAcc':

            handle\_open\_account(conn)

        elif response == 'DepositAmount':

            handle\_deposit\_amount(conn)

        elif response == 'WithdrawAmount':

            handle\_withdraw\_amount(conn)

        elif response == 'Balcheck':

            handle\_check\_balance(conn)

        elif response == 'Disdetails':

            handle\_display\_details(conn)

        elif response == 'CloseAcc':

            handle\_close\_account(conn)

        elif response == 'Transfer Amount':

            handle\_Transfer\_amount(conn)

        else:

            conn.send("Invalid request".encode())

def bank\_server():

    print("Bank server started...")

    while True:

        conn, address = server\_socket.accept()

        print("Connection from: " + str(address))

        thread = td.Thread(target=handle, args=(conn,))

        thread.start()

    conn.close()

if \_\_name\_\_ == '\_\_main\_\_':

    bank\_server()

# CODE EXPLANATION:

**Main Server Setup**

* **bank\_server():** This function is the main server loop that listens for incoming connections. When a client connects, it spawns a new thread to handle that client's requests**.**

**Operations on Bank Accounts**

1. **Open Account Operations**
   * **handle\_open\_account(server\_socket):** Handles the creation of a new bank account in the database. It receives account details from the client, inserts them into the 'account' and 'amount' tables in the database, and confirms the account creation status.
2. **Deposit Operations**
   * **handle\_deposit\_amount(server\_socket):** Accepts a deposit amount for a specified account, validates the account and PIN, updates the account balance in the database, and sends back the updated balance.
3. **Withdraw Operations**
   * **handle\_withdraw\_amount(server\_socket):** Processes a withdrawal from a specified account, validates the account and PIN, updates the account balance in the database, and sends back the updated balance.
4. **Check Balance Operations**
   * **handle\_check\_balance(server\_socket):** Retrieves the account balance for a specified account and PIN from the database and sends it back to the client.
5. **Display Account Details**
   * **handle\_display\_details(server\_socket**): Fetches and sends account details (Name, Account Number, Balance) for a specified account and PIN to the client.
6. **Close Account Operations**
   * **handle\_close\_account(server\_socket):** Closes a specified bank account, deletes its details from the database if the account and PIN provided by the client are valid.
7. **Transfer Amount between Accounts**
   * **handle\_Transfer\_amount(server\_socket):** Transfers an amount from one account to another, updating balancesaccordingly for both accounts in the database.

**Helper Functions and Request Processing**

* **process\_request(data):** Identifies and categorizes client requests based on the input data provided. It returns a string indicating the type of operation requested.
* **handle(conn):** Handles client requests based on the identified operation and calls the respective functions accordingly.

**Thread Management**

* The script uses threading (threading module) to handle multiple client connections simultaneously**.**

**Database Connectivity**

* MySQL Connector (mysql.connector) is used to establish connections to the local MySQL database.
* Multiple SQL queries are constructed to interact with the database for account operations (insertion, updates, deletions, and selections).

**Error Handling**

* The code includes some error handling, such as try-except blocks around database operations, but there may be potential improvements needed for robustness and handling various edge cases.**Top of Form**

**Bank Client Operations**

1. **Client Connection Setup**
   * **bank\_client():** Establishes a connection to the bank server using sockets, connects to the designated host and port.
2. **User Interaction Loop**
   * Prompts the user for various banking operations in a continuous loop until the user chooses to exit.
3. **User Input Processing**
   * Processes the user's choice and sends the appropriate request to the server.
4. **Operations Implementation**
   * **'1' - Open an Account**
     + Collects user input for account details, sends the data to the server for account creation, and displays the server's response.
   * **'2' - Deposit Amount**
     + Takes inputs for deposit amount, account number, and PIN, sends the deposit data to the server, and displays the updated balance received from the server.
   * **'3' - Withdraw Amount**
     + Similar to the deposit operation, but for withdrawals. Asks for withdrawal amount, account number, and PIN, then displays the updated balance received from the server.
   * **'4' - Check Balance**
     + Asks for the account number and PIN, sends the request to the server, and displays the current balance received.
   * **'5' - Display Customer Details**
     + Fetches and displays customer details based on the provided account number and PIN.
   * **'6' - Transfer Amount**
     + Facilitates the transfer of an amount between two accounts. Takes inputs for the sender's account, receiver's account, PIN, and amount to transfer. Displays the updated balances in both accounts.
   * **'7' - Close an Account**
     + Allows the user to close an account by providing the account number and PIN. Displays a success message or an error message based on the server's response.
   * **'8' - Exit**
     + Closes the connection with the server and terminates the client program.

**Socket Communication**

* Utilizes socket communication to send user requests and receive server responses.

**Input Validation**

* Collects user inputs for various banking operations and constructs data strings to send to the server.

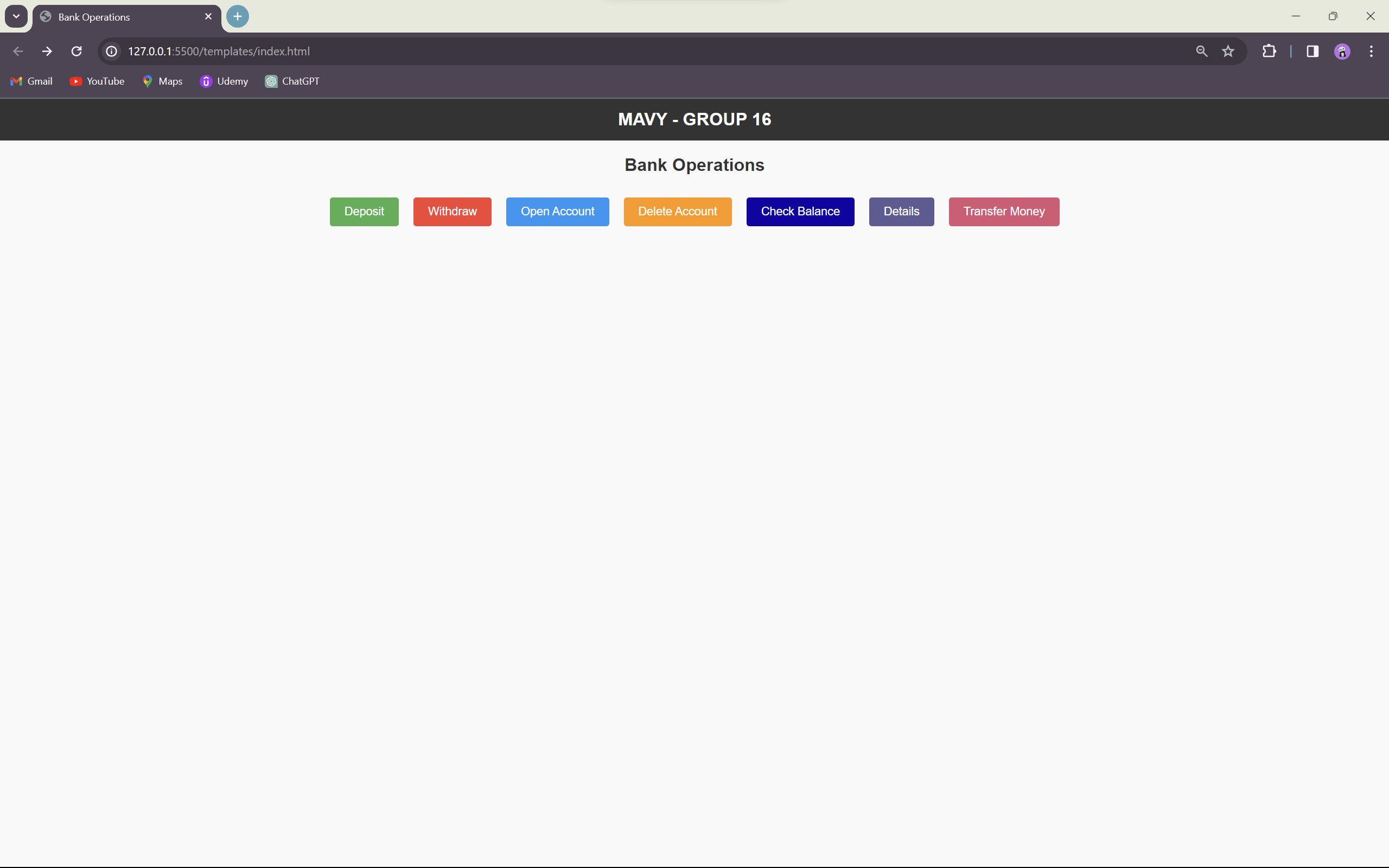
**Error Handling**

* Includes basic error handling to manage scenarios like attempting to create an account with an existing account number and displaying relevant messages based on the server's response.

**Closing Connection**

* Closes the client socket connection after the user chooses to exit the application.

# OUTPUT:

****

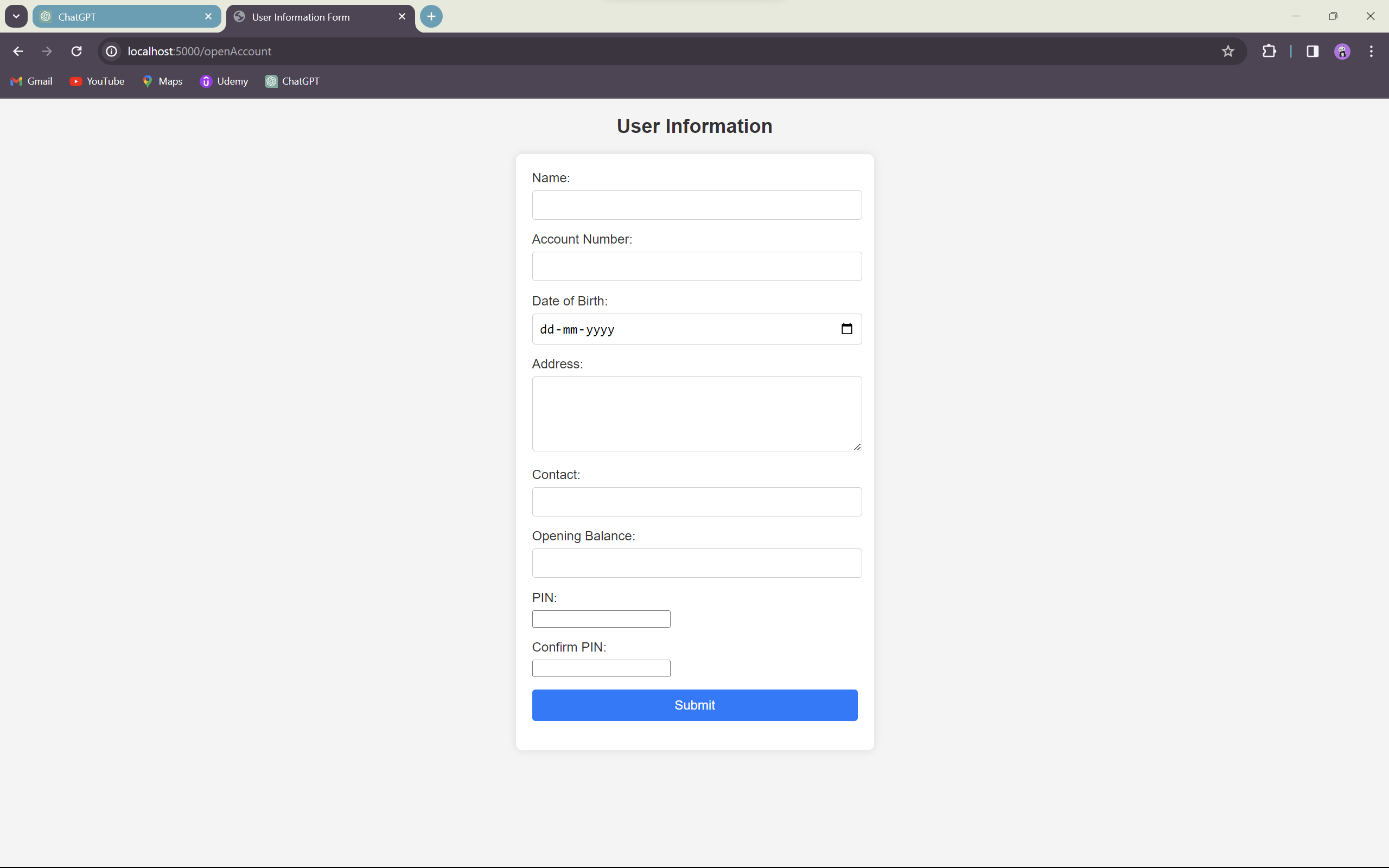
****

Figure 1 Open Account Form

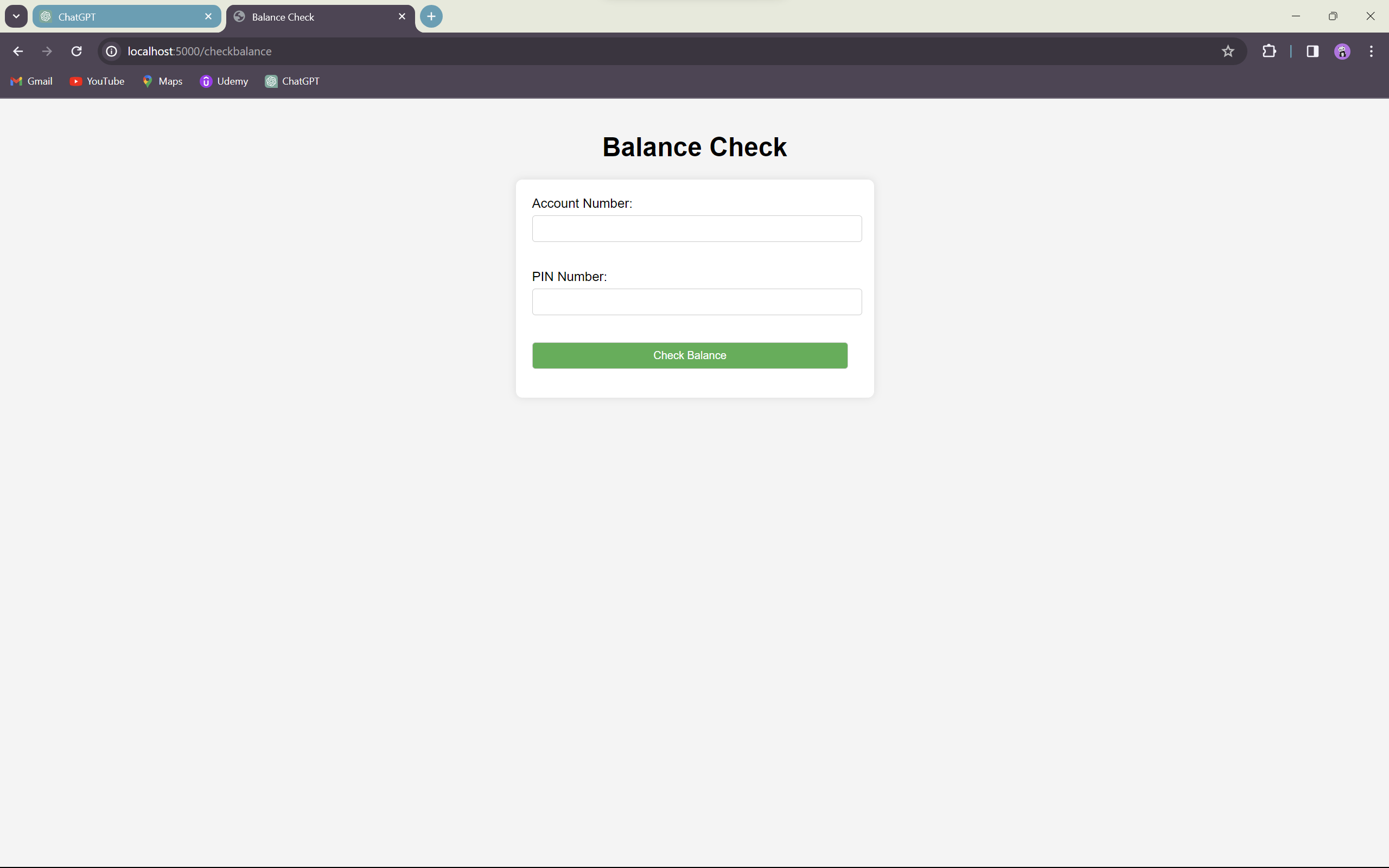
****

Figure 2 Balance checking form

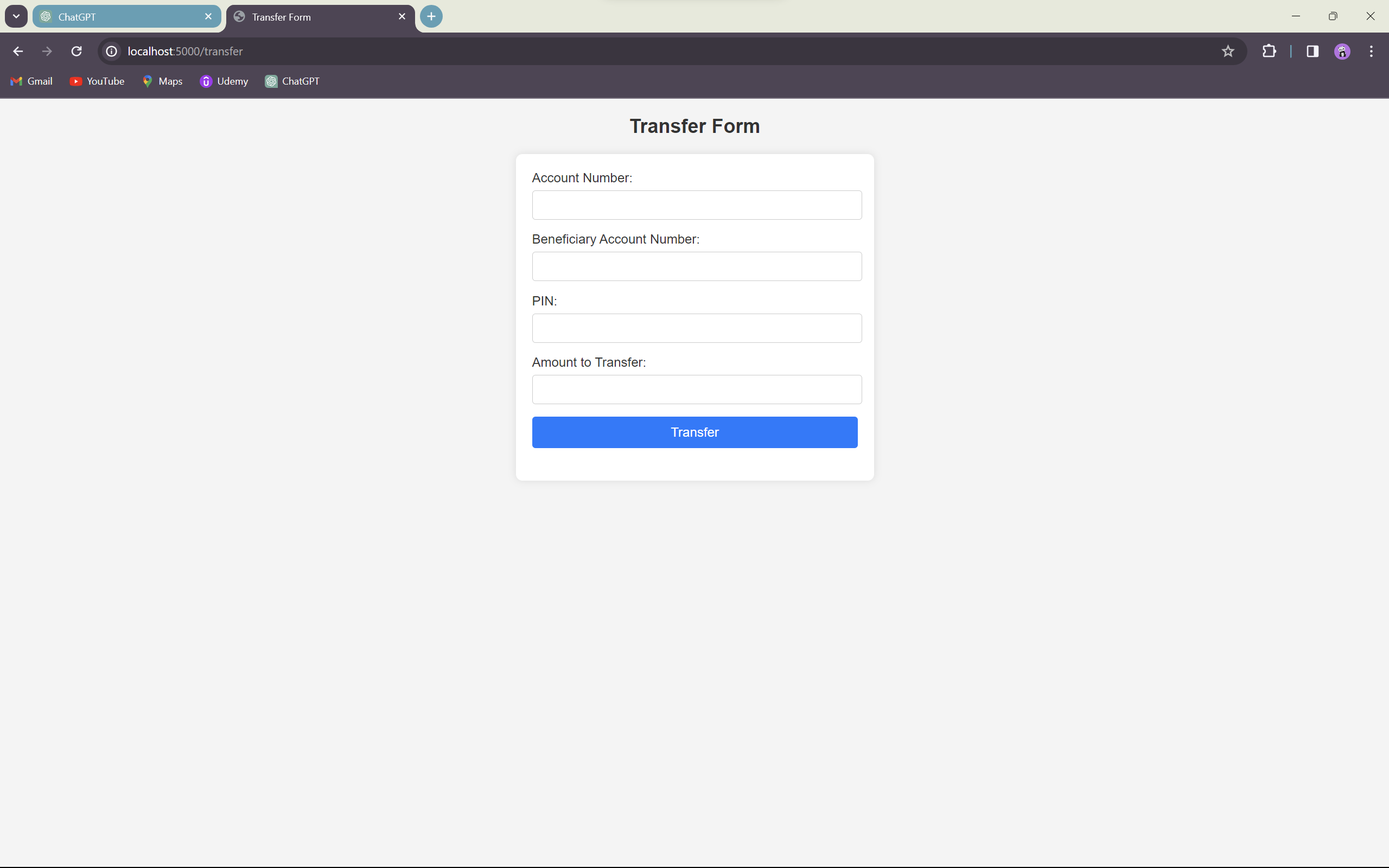


Figure 3 Transfer Amount form

# CONCLUSION:

Using Python's socket programming, the banking web app ensures secure, real-time interactions between users and the server. It offers easy access to banking services like transactions and account management. While managing connections directly improves responsiveness, ongoing security updates are vital for safeguarding user data in the ever-changing online banking environment.