**Bank Management system**

**Introduction:**

The Bank Management System is a software application that helps a bank to manage its customer's financial transactions. The system is designed to automate many of the tasks that are performed manually by bank employees. The Bank Management System consists of three Python scripts: app.py, menu.py, and tables.py. These scripts work together to provide a comprehensive system that can be used by the bank to manage its customers' financial transactions.

**Objective:**

The objective of this project is to create a Bank Management System that can be used by the bank to manage its customer's financial transactions. The system should be able to perform tasks such as opening new accounts, depositing and withdrawing money, transferring funds between accounts, and generating reports.

**Project Overview:**

The project consists of three Python scripts:

app.py: This script contains the main code for the Bank Management System. It provides a user interface that allows customers to access their accounts, view their details, and perform banking transactions.

menu.py: This script contains the menu options that users can choose from to perform various banking transactions.

tables.py: This script contains the database tables used to store customer account information and transaction details.

Values.py: This script contains the code to insert values in the tables

**Technologies Used :**

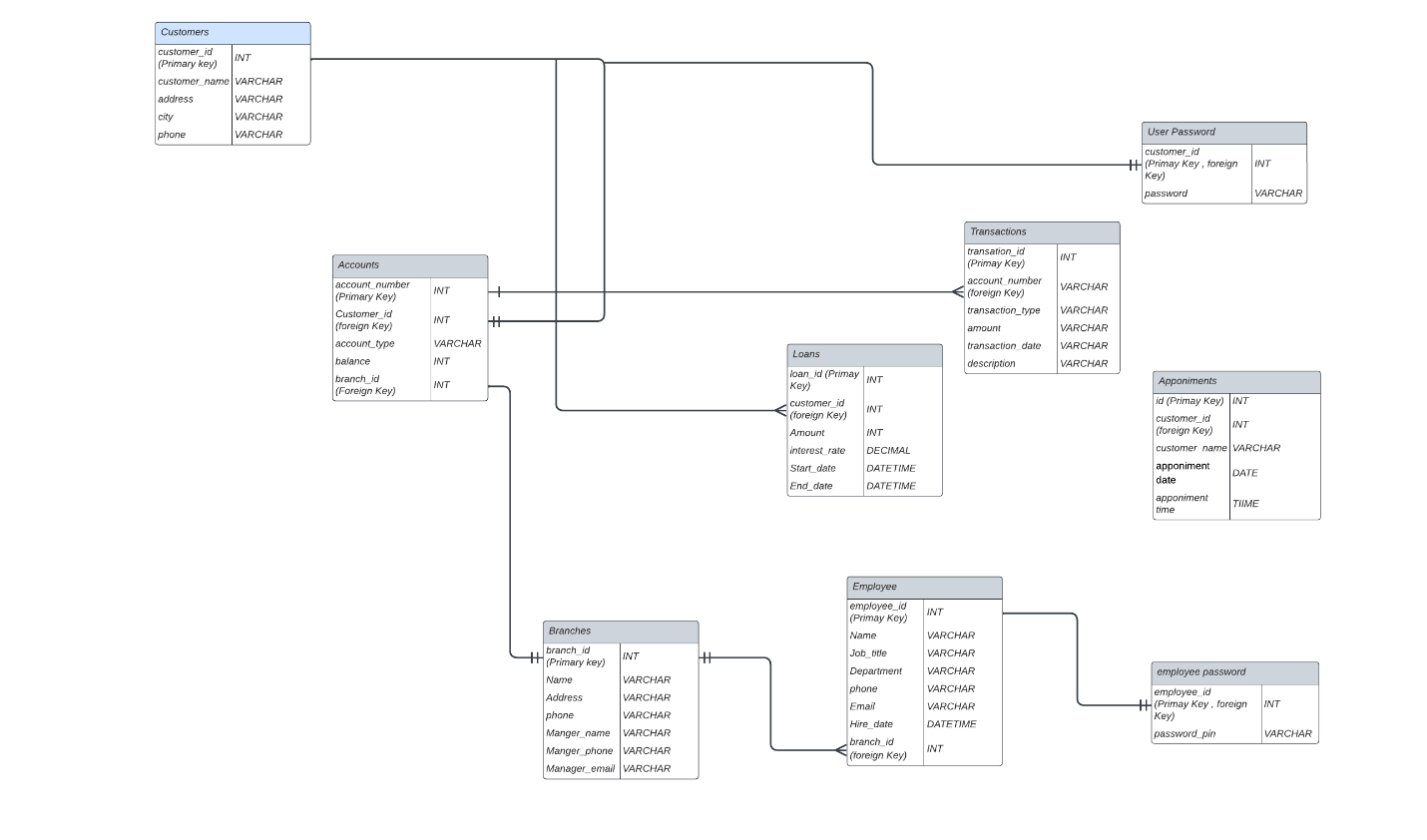
* Python 3.8.10
* MySQL 8.0.26
* PyMySQL 1.0.2
* Streamlit 1.4.0

**Database Design:**

The Bank Management System uses a MySQL database to store and retrieve data. The database contains 9 tables:

* accounts - This table stores information about the bank accounts, including the account number, customer ID, account type, balance, and branch ID.
* transactions - This table stores information about the transactions made on the bank accounts, including the transaction ID, account number, transaction type (national or international), transaction amount, and transaction date.
* branches - This table stores information about the bank branches, including the branch ID, branch name, branch city, and branch state.
* customers - This table stores information about the bank customers, including the customer ID, customer name, customer address, and customer phone number.
* employee - This table stores information about the bank employees, including the employee ID, employee name, employee position, and employee branch ID.
* loans - This table stores information about the loans given by the bank, including the loan ID, customer ID, loan amount, interest rate, start date, and end date.
* user\_password - This table stores the login credentials (username and password) for the bank customers.
* employee\_password - This table stores the login credentials (username and password) for the bank employees.
* appointments - This table stores information about the appointments scheduled by the bank customers, including the appointment ID, customer ID, customer name, appointment date, and appointment time.

**EDR:**



**Tablee.py:**

import mysql.connector as connection

from constants import host , user , passwd , database

class Database\_creation():

    def \_\_init\_\_(self , database  , host , user , passwd):

        try:

            self.database = database

            self.mydb = connection.connect(host  = host , user = user , passwd =passwd)

            self.cursor = self.mydb.cursor()

        except Exception as e :

            print("Issue with database connection , Please check the user name , password and host")

            raise e

    def is\_database\_created(self , database):

        is\_database = False

        query = f"SHOW DATABASES LIKE '{database}'"

        self.cursor.execute(query)

        if len(self.cursor.fetchall()) > 0:

            is\_database = True

        return is\_database

    def create\_database(self , database):

        if self.is\_database\_created(database):

            #print(f"{database} is already present")

            pass

        else:

            query = f"CREATE DATABASE {database}"

            self.cursor.execute(query)

            if self.is\_database\_created(database):

                print("Sucessfully create the database")

            else:

                print("database is not create please check")

    def  is\_table\_created(self, database, table\_name):

        is\_table = False

        query = f"USE {database}"

        self.cursor.execute(query)

        query\_table = f"SHOW TABLES LIKE '{table\_name}'"

        self.cursor.execute(query\_table)

        if len(self.cursor.fetchall()) > 0:

            is\_table = True

        return is\_table

    def create\_tables(self , database):

        try:

            if self.is\_table\_created(database , "customers"):

                pass

                #print("Customers table is already present in the database")

            else:

                customer\_query = "CREATE TABLE customers ( customer\_id INT PRIMARY KEY AUTO\_INCREMENT,\

                         customer\_name VARCHAR(100) ,\

                         address VARCHAR(255) ,\

                         city VARCHAR(255) ,\

                         phone VARCHAR(20))"

                self.cursor.execute(customer\_query)

            if self.is\_table\_created(database , "Transactions"):

                pass

                #print("Transactions table is already present in the database")

            else:

                transactions\_query = "CREATE TABLE Transactions ( \

                    transaction\_id INT PRIMARY KEY ,\

                    account\_number INT NOT NULL,\

                    transaction\_type VARCHAR(20) NOT NULL,\

                    amount DECIMAL(10,2) NOT NULL,\

                    transaction\_date DATETIME NOT NULL,\

                    description VARCHAR(255) ,\

                    FOREIGN KEY (account\_number) REFERENCES Accounts(account\_number) )"

                self.cursor.execute(transactions\_query)

            if self.is\_table\_created(database , "Branches"):

                pass

                #print("Branches table is already present in the database")

            else:

                branches\_query = "CREATE TABLE Branches (\

                    branch\_id INT PRIMARY KEY,\

                    name VARCHAR(255) NOT NULL,\

                    address VARCHAR(255) NOT NULL,\

                    phone VARCHAR(20),\

                    manager\_name VARCHAR(255) NOT NULL,\

                    manager\_phone VARCHAR(20),\

                    manager\_email VARCHAR(255))"

                self.cursor.execute(branches\_query)

            if self.is\_table\_created(database , "Accounts"):

                pass

                #print("Accounts table sis already present in the database")

            else:

                accounts\_query = "CREATE TABLE Accounts ( account\_number INT PRIMARY KEY , \

                    customer\_id INT NOT NULL , \

                    account\_type VARCHAR(20) NOT NULL , \

                    balance DECIMAL(10,2) NOT NULL , \

                    branch\_id INT , \

                    FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)\

                    FOREIGN KEY (branch\_id) REFERENCES Branches(branch\_id))"

                self.cursor.execute(accounts\_query)

            if self.is\_table\_created(database , "Employees"):

                pass

                #print("Employees table is present in the database")

            else:

                employees\_query = "CREATE TABLE Employees ( \

                    employee\_id INT PRIMARY KEY,\

                    branch\_id INT,\

                    name VARCHAR(255) NOT NULL,\

                    job\_title VARCHAR(255) NOT NULL,\

                    department VARCHAR(255) NOT NULL,\

                    phone VARCHAR(20),\

                    email VARCHAR(255),\

                    hire\_date DATE NOT NULL,\

                    FOREIGN KEY (branch\_id) REFERENCES Branches(branch\_id))"

                self.cursor.execute(employees\_query)

            if self.is\_table\_created(database , "Loans"):

                pass

                #print("Loan table is already present in the database")

            else:

                loan\_query = "CREATE TABLE Loans ( \

                    loan\_id INT PRIMARY KEY,\

                    customer\_id INT NOT NULL,\

                    amount DECIMAL(10, 2) NOT NULL,\

                    interest\_rate DECIMAL(5, 2) NOT NULL,\

                    start\_date DATE NOT NULL,\

                    end\_date DATE NOT NULL,\

                    FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id))"

                self.cursor.execute(loan\_query)

            if self.is\_table\_created(database , "User\_password"):

                pass

                #print("The User password table is already present in the database")

            else:

                user\_password\_query = "CREATE TABLE User\_password ( \

                    customer\_id INT PRIMARY KEY,\

                    password\_pin VARCHAR NOT NULL , \

                    FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id))"

                self.cursor.execute(user\_password\_query)

            if self.is\_table\_created(database , "employee\_password"):

                pass

                #print("The User password table is already present in the database")

            else:

                user\_password\_query = "CREATE TABLE employee\_password ( \

                    employee\_id INT PRIMARY KEY,\

                    password\_pin INT NOT NULL , \

                    FOREIGN KEY (employee\_id) REFERENCES Employees(employee\_id))"

                self.cursor.execute(user\_password\_query)

            if self.is\_table\_created(database , "appointment"):

                pass

            else:

                appointment\_query = "CREATE TABLE appointment (\

                                    id INT NOT NULL AUTO\_INCREMENT,\

                                    customer\_id INT NOT NULL,\

                                    customer\_name VARCHAR(50) NOT NULL,\

                                    appointment\_date DATE NOT NULL,\

                                    appointment\_time TIME NOT NULL,\

                                    PRIMARY KEY (id),\

                                    FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)\

                                    )"

                self.cursor.execute(appointment\_query)

        except Exception as e:

            raise e

    def initiate\_database(self):

        self.create\_database(self.database)

        self.create\_tables(self.database)

        print("=================================The Bank Database is up and ready====================================================")

**User Interface**

The Bank Management System uses the Streamlit library to create a user interface. The user interface provides the following options:

Customer’s functions :

* login with customer\_id: This function allows a customer to log in to their account using their customer ID.
* withdraw amount: This function allows a customer to withdraw a certain amount of money from their account.
* add amount: This function allows a customer to deposit a certain amount of money into their account.
* transfer amount to other account: This function allows a customer to transfer a certain amount of money to another account.
* Transaction details: This function displays all the transaction details for the customer's account.
* view customer details: This function displays all the details of the customer, such as name, address, phone number, etc.
* loan details: This function displays all the details of the customer's loan, such as loan amount, interest rate, repayment period, etc.
* book an appointment: This function allows the customer to book an appointment with the bank.

Employee’s functions :

* login with employee\_id: This function allows a employee to log in to their account using their employee ID.
* Cash deposit: This function allows the employee to deposit cash into the customer's account.
* view customers in branch: This function allows the employee to view all the customers in the branch.
* Report (total international transaction vs total national transaction): This function generates a report that shows the total amount of international transactions vs. national transactions.
* see all the appointments: This function allows the employee to see all the appointments that have been booked by customers.
* delete customer\_id account: This function allows the employee to delete a customer's account.
* loan approval: This function allows the employee to approve loan for a customer Top of Form

**App.py:**

The app.py script is the main script of the Bank Management System. It is responsible for running the system and providing a user interface for the bank employees. The app.py script imports the other two scripts, menu.py, and tables.py. The app.py script contains the main function that is responsible for running the system. The main function calls the menu function from the menu.py script to display the main menu of the system. The menu function is responsible for displaying the different options that are available in the system, such as opening a new account or depositing money into an existing account.

This is a Python script that implements a bank management system using the Streamlit library. It allows customers and employees to log in and perform various tasks related to their accounts.

The script starts by importing the necessary libraries and modules, including Streamlit, mysql-connector, and several custom modules for database creation, value insertion, and menu navigation.

It then initializes the database and sets up a menu object based on the user's role (customer or employee) using the Streamlit option\_menu widget. Based on the user's selection, the menu object will display various options for performing transactions, checking account balances, updating account details, approving loans, generating reports, and more.

The user's input is validated and checked against the database, and appropriate warning messages are displayed if the input is incorrect or missing.

Finally, the script defines a series of conditional statements that determine which menu options should be displayed based on the user's selection.

Overall, this script provides a user-friendly interface for managing bank accounts and performing various banking tasks. However, it is important to note that it relies heavily on external libraries and modules, and may require additional setup and configuration to run properly.

Command to run the app.py is – “streamlit run app.py”

Code:

import streamlit as st

from streamlit\_option\_menu import option\_menu

from constants import host , user , passwd , database

from tables import Database\_creation

from values import Insert\_values

import numpy as np

import  mysql.connector as sql

from menu import menu

import datetime as dt

import streamlit\_authenticator as stauth

st.title("Bank management system")

mydb = Database\_creation(database , host , user , passwd)

mydb.initiate\_database()

#values = Insert\_values(database , host , user , passwd)

#values.initiate\_inserting\_values()

conn=sql.connect(host=host,user=user,passwd=passwd,database=database)

cur = conn.cursor()

options = st.selectbox("Customers or Employee" , ("Select a option" , "customer" , "employee") , label\_visibility = "hidden")

if not options:

    st.stop()

if options == "customer":

    customer\_id = st.number\_input("Please enter the customer\_id")

    password = st.text\_input("Please enter the password" , type="password")

    #button\_status = st.button("Login")

    if not customer\_id or not password:

        st.stop()

    cur.execute(f"SELECT password\_pin FROM User\_password where customer\_id = {customer\_id}")

    state = (password == cur.fetchall()[0][0])

    #login\_state = st.button("Login")

    if not state:

        st.warning("Password is incorrect")

        st.stop()

    else:

        menu\_object = menu(host=host,user=user,passwd=passwd,database=database , customer\_id= customer\_id)

        with st.sidebar:

            selected = option\_menu(

                menu\_title = "Main Menu",

                options = ['Transaction', 'Customer details', 'Transaction details', 'Check your account balance', 'Update account details', 'Loan details', 'Appointment'] , default\_index=1)

        if selected == "Customer details":

            menu\_object.options(2)

        elif selected == "Transaction":

            menu\_object.options(1)

        elif selected == "Transaction details":

            menu\_object.options(3)

        elif selected == "Check your account balance":

            menu\_object.options(5)

        #elif selected == "DELETE ACCOUNT":

        #    menu\_object.options(4)

        elif selected == "Update account details":

            menu\_object.options(6)

        elif selected == "Loan details":

            menu\_object.options(7)

        elif selected == "Appointment":

            menu\_object.options(8)

elif options == "employee":

    employee\_id = st.number\_input("Please enter your employee id")

    password = st.text\_input("Please enter the password" , type="password")

    if not employee\_id or not password:

        st.stop()

    cur.execute(f"SELECT password FROM employee\_password where employee\_id = {employee\_id}")

    state = (password == cur.fetchall()[0][0])

    #login\_state = st.button("Login")

    if not state:

        st.warning("Password is incorrect")

        st.stop()

    else:

        menu\_object = menu(host=host,user=user,passwd=passwd,database=database)

        with st.sidebar:

            selected = option\_menu(

                menu\_title = "Main Menu",

                options = ["Cash" , "Customers in a Branch" , "Loan approval" , "Report" , "Appointment" , "Delet Account"])

        if selected == "cash":

            menu\_object.options\_employee(1)

        elif selected == "customers":

            menu\_object.options\_employee(2)

        elif selected == "Loan approval":

            menu\_object.options\_employee(3)

        elif selected == "report":

            menu\_object.options\_employee(4)

        elif selected == "Appointment":

            menu\_object.options\_employee(5)

        elif selected == "Delet Account":

            menu\_object.options\_employee(6)

**Menu.py:**

The menu.py script is responsible for displaying the main menu of the system. It is imported by the app.py script and contains the menu function. The menu function is responsible for displaying the different options that are available in the system. When the user selects an option, the menu function calls the appropriate function from the tables.py script to perform the task.

Python script that provides a menu to interact with a banking system. The script uses Streamlit for the frontend, and MySQL as the backend to store and retrieve customer and transaction data.

The menu() class takes in the necessary credentials to connect to the database and initializes a connection. It also takes in a customer\_id parameter, which is used to fetch customer details and transaction history.

Menu.py has a class called menu() , which has 2 main function that is options() and options\_employee(), both the functions take a number a input and this is coming the app.py file where the user is select differ options and each option is given a number.

Customer has 7 options to select in the website,

* Transactions

1. With draw
2. Add amount
3. Account to account transfer

* Customer details
* Transactions details
* Check your balance
* Update account details
* Loan details
* Appointments

**Transactions**:  
  
The transactions code starts with if statement that checks if the input parameter n equals 1. If so, it fetches the current datetime, prompts the user to enter their account number, and fetches the account details for that account number from the database. If the account number is invalid, the script displays an error message. Otherwise, it presents the user with three options to select: "with draw", "add amount", and "account to account transfer".

If the user selects "with draw", the script prompts the user to enter the amount to withdraw. If the entered amount is greater than the account balance, it displays a message indicating insufficient funds. Otherwise, it updates the account balance in the database and inserts a new transaction record into the "Transactions" table.

If the user selects "add amount", the script prompts the user to enter the amount to add. It then updates the account balance in the database and inserts a new transaction record into the "Transactions" table. Finally, it displays the updated account balance.

If the user selects "account to account transfer", the script prompts the user to enter the recipient account number and the amount to transfer. It then inserts two new transaction records into the "Transactions" table (one for the sender and one for the recipient), updates the account balances in the database for both accounts, and displays the updated account balance for the sender's account.Top of Form

**Customer details:**  
  
option 2 from the main menu, which is to view their account details. The script queries the customer table in the database for the user's account details using their customer\_id.

If the account is found, the user's details are fetched from the database and displayed using the st.write() function. Otherwise, a message is displayed indicating that the account does not exist.

if any rows were returned by the SQL query. If the length of data is 0, then it means that the user does not exist in the customers table, and a message is displayed using the st.write() function.

If the user exists, then their details are displayed. A list called details is created with the names of the columns in the customers table that contain the user's details. The zip() function is used to iterate over the elements of the details list and the corresponding row fetched from the database. For each element in the details list and the corresponding element in the row fetched from the database, the element name and value are concatenated into a string using the + operator and displayed using the st.write() function.

**Transactions details:**

Option 3 from the main menu , It creates a SQL query to select all transactions for the accounts associated with the customer whose customer\_id matches the customer\_id value.

It executes the SQL query using self.cursor.execute(transactions) method to retrieve data from the database. It fetches all the data retrieved by the SQL query using self.cursor.fetchall().

It checks the length of the retrieved data using len(data). If the length of the data is zero, then it prints "No Transactions", indicating that there are no transactions associated with the customer.

Otherwise, it creates a list of transaction details and creates a Pandas DataFrame from the retrieved data using pd.DataFrame(data, columns=transactions\_details).

Finally, it displays the DataFrame using st.table(df) function from Streamlit.

**Check your balance:**

Option 4 from main menu , It creates a SQL query to select the balance value for the accounts associated with the customer whose customer\_id matches the customer\_id value.

It executes the SQL query using self.cursor.execute(balance\_query) method to retrieve data from the database. It fetches all the data retrieved by the SQL query using self.cursor.fetchall().

It displays the balance value using st.write(f"Balance - {data[0][0]}") function from Streamlit library.

**Update account details:**

if the value of the variable n is equal to 6. If n is equal to 6, then it performs the following operations:

It prompts the user to enter the updated name, address, city, and phone number using Streamlit text\_input function.

It checks if any of the user input values are empty, and if so, it stops the program using st.stop() function from Streamlit library.

It creates a SQL query to update the customer details (name, address, city, and phone number) for the customer whose customer\_id matches the self.customer\_id value.

It executes the SQL query using self.cursor.execute() method to update the customer details in the database.

It creates a SQL query to retrieve the updated customer details for the customer whose customer\_id matches the self.customer\_id value.

It executes the SQL query using self.cursor.execute() method to retrieve the updated customer details from the database.

It fetches all the data retrieved by the SQL query using self.cursor.fetchall().

It displays the updated customer details using Streamlit write function, which includes the customer\_id, customer\_name, address, city, and phone for the updated customer.

Overall, this code is checking if the value of n is 6, and if it is, it prompts the user to enter updated customer details, updates the customer details in the database, retrieves the updated customer details from the database, and displays the updated customer details to the user.

**Loan details:**

if the value of the variable n is equal to 7. If n is equal to 7, then it performs the following operations:

It creates a SQL query to select all loans associated with the customer whose customer\_id matches the self.customer\_id value.

It executes the SQL query using self.cursor.execute() method to retrieve loan data from the database.

It fetches all the data retrieved by the SQL query using self.cursor.fetchall().

It checks if the length of data retrieved is greater than 0, and if so, it creates a Pandas DataFrame with the loan data and displays it using Streamlit table function.

If there are no loans associated with the customer, it displays a message using st.write() function that the customer doesn't have any loans.

Overall, this code is checking if the value of n is 7, and if it is, it retrieves all the loans associated with the customer whose customer\_id matches the self.customer\_id value, creates a DataFrame to display the loan details if there are any loans, or displays a message if there are no loans.

**Appointments:**

If the value of the variable n is equal to 8. If n is equal to 8, then it performs the following operations:

It creates an input field using Streamlit's date\_input() function to allow the user to select the appointment date.

It creates an input field using Streamlit's time\_input() function to allow the user to select the appointment time.

It creates an input field using Streamlit's number\_input() function to allow the user to enter their customer ID.

It creates an input field using Streamlit's text\_input() function to allow the user to enter their name.

It checks if all the input fields have been filled by the user, and if any of the fields are empty, it stops the execution of the code using st.stop() function.

It creates a SQL query to insert the appointment details, including the appointment date, appointment time, customer ID, and customer name into the appointment table in the database.

It executes the SQL query using self.cursor.execute() method to insert the appointment details into the database.

It displays a message to the user using st.write() function that their appointment is booked on the selected date.

Overall, this code is checking if the value of n is 8, and if it is, it allows the user to select the appointment date and time, enter their customer ID and name, inserts the appointment details into the database, and displays a message to the user that their appointment is booked.

Employee has 7 options to select in the website

* Cash
* Customers
* Loan approval
* Report
* Appointment
* Delete an account

**Cash:**

This code is a part of a bank management system that allows the employee to add cash credit to a particular account.

The employee is prompted to enter the account number for which they want to add cash credit. Then, the code checks if the account number entered by the user exists in the database or not. If the account number does not exist, a warning message is displayed and the execution of the code is stopped.

If the account number exists, the user is prompted to enter the amount they want to add to the account. If the user does not enter any value, the execution of the code is stopped.

After the user enters the amount, the code updates the account balance in the Accounts table of the database by adding the entered amount to the existing account balance using the UPDATE query. Finally, the changes are committed to the database using the COMMIT statement.

**Customers:**

The employee is prompted to enter the branch ID. Then, the code fetches all the account details from the Accounts table of the database that match the given branch ID using a SELECT query.

After fetching the data, it is stored in a Pandas DataFrame object with column names set to ["account\_number" , "customer\_id" , "account\_type" , "balance" , "branch\_id"].

Finally, the account information is displayed to the user in a table format using the Streamlit st.table() function.

**Loan approval:**

The code first prompts the employee to input the customer\_id and then creates a SQL query to fetch all the transactions associated with the accounts of that customer.

After executing the query and fetching the data, it checks whether any transactions are returned or not. If no transactions are returned, it prints a message saying "No Transactions" otherwise it creates a Pandas DataFrame of the fetched data and displays it using st.table() function.

**Report:**

For generating a report that compares the total amount of money involved in international transactions vs. national transactions. It is a part of a banking application that allows the user to perform various banking operations like cash credit, loan approval, appointment booking, etc.

When the user selects the option number 4, the code executes a SQL query to fetch the total sum of money involved in international transactions and stores it in the variable "international". Similarly, it fetches the total sum of money involved in national transactions and stores it in the variable "nation".

The fetched data is then used to create a pandas dataframe that contains two columns: "total sum" and "type". The "total sum" column contains the sum of money involved in transactions, and the "type" column contains the transaction type, which is either "International" or "National".

Finally, the code generates a bar chart using matplotlib that compares the total sum of international transactions and national transactions. The chart is displayed using the "st.pyplot()" function provided by the Streamlit library.

**Appointment:**

First, the SELECT query is executed to fetch all the rows from the "appointment" table, and the retrieved data is stored in the "data" variable.

Next, the retrieved data is converted to a Pandas DataFrame with appropriate column names, and this DataFrame is displayed using the "st.write()" function.

Finally, a loop is used to display each row of the DataFrame in a readable format, with each row separated by a line of asterisks.

**Delete an account:**

It first prompts the user to enter a customer ID, then checks if the input is valid. It then selects the balance from the Accounts table where the customer ID matches the input.

Next, it sets FOREIGN\_KEY\_CHECKS to 0 to temporarily disable foreign key constraints for the current session, and deletes the account and customer from the Accounts and Customers tables, respectively, where the customer ID matches the input.

Finally, it displays a message indicating that the account has been deleted successfully.

import streamlit as st

import  mysql.connector as connection

import random

import pandas as pd

import matplotlib.pyplot as plt

from datetime import datetime

from constants import host , passwd , user , database

class menu():

    def \_\_init\_\_(self ,host , user , passwd , database , customer\_id=None):

        try:

            self.customer\_id = customer\_id

            self.mydb = connection.connect(host  = host , user = user , passwd =passwd , database = database)

            self.cursor = self.mydb.cursor()

            self.mydb.autocommit = True

        except Exception as e:

            raise e

    def check\_account\_number(self):

         self.cursor.execute(f"select account\_number from Accounts where customer\_id = {self.customer\_id}")

         data = self.cursor.fetchall()

         return data[0][0]

    def options(self , n):

        if n == 1:

            date\_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

            self.account\_no = st.number\_input("Please enter your account number")

            if not self.account\_no:

                 st.stop()

            if self.account\_no != self.check\_account\_number():

                 st.warning("Please check your account number")

                 st.stop()

            self.cursor.execute(f"select \* from Accounts where account\_number = {int(self.account\_no)}")

            data = self.cursor.fetchall()

            account\_balance = data[0][3]

            if len(data) == 0:

                    print("Invalid Account Number")

            else:

                    options = st.selectbox("Please select the action" , ("with draw" , "add amount" , "account to account transfer"))

                    if options == "with draw" :

                        with\_draw\_amount = int(st.number\_input("Please enter the amount"))

                        if not with\_draw\_amount:

                             st.stop()

                        if with\_draw\_amount > account\_balance:

                            st.write("your balance is low")

                        else:

                            st.write("Please collect your cash at the bank")

                            update\_query = f"update Accounts set  balance= {account\_balance - with\_draw\_amount}  where account\_number={self.account\_no}"

                            self.cursor.execute(update\_query)

                            transactions\_query = f"INSERT INTO Transactions\

                                (transaction\_id , account\_number , transaction\_type , amount , transaction\_date , description)\

                                VALUES ({random.randint(0,899999)} , {self.account\_no} , 'international' , {with\_draw\_amount} , '%s' , 'nothing')\

                                " %(date\_time)

                            self.cursor.execute(transactions\_query)

                            self.mydb.commit()

                    if options =="add amount":

                        amount2 = int(st.number\_input("Pleae enter the amount you to add - "))

                        if not amount2:

                             st.stop()

                        add\_update\_query = f"update Accounts set   balance= {account\_balance + amount2}  where account\_number={self.account\_no}"

                        self.cursor.execute(add\_update\_query)

                        date\_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

                        transactions\_query = f"INSERT INTO Transactions\

                                (transaction\_id , account\_number , transaction\_type , amount , transaction\_date , description)\

                                VALUES ({random.randint(0,899999)} , {self.account\_no} , 'international' , {amount2} , '%s' , 'nothing')\

                                " % (date\_time)

                        self.cursor.execute(transactions\_query)

                        self.mydb.commit()

                        self.cursor.execute(f"select balance from Accounts where account\_number={self.account\_no}")

                        st.write(f"The Updated balance is {self.cursor.fetchall()[0][0]}")

                    if options == "account to account transfer":

                        other\_account\_number = st.number\_input("Please enter the number account number you want to send money to - " , value= 0)

                        amount3 = st.number\_input("Please enter the amount you want to send - " , value=0)

                        if (not other\_account\_number) or (not amount3):

                             st.stop()

                        transactions\_query = f"INSERT INTO Transactions\

                                (transaction\_id , account\_number , transaction\_type , amount , transaction\_date , description)\

                                VALUES ({random.randint(0,899999)} , {self.account\_no} , 'international' , {amount3} , '%s' , 'Account to Account transfer') ,\

                                ({random.randint(0,899999)} , {other\_account\_number} , 'international' , {amount3} , '%s' , 'Account to Account transfer')" % (date\_time ,date\_time)

                        self.cursor.execute(transactions\_query)

                        update\_query1 = f"update Accounts set  balance= {account\_balance - amount3}  where account\_number={self.account\_no}"

                        self.cursor.execute(update\_query1)

                        self.cursor.execute(f"select \* from Accounts where account\_number = {other\_account\_number}")

                        other\_account\_amount = self.cursor.fetchall()[0][0]

                        update\_query2 = f"update Accounts set  balance= {other\_account\_amount + amount3}  where account\_number={other\_account\_number}"

                        self.cursor.execute(update\_query2)

                        self.mydb.commit()

                        self.cursor.execute(f"select balance from Accounts where account\_number={self.account\_no}")

                        st.write(f"The Updated balance is {self.cursor.fetchall()[0][0]}")

        if n == 2:

                details\_query = f"select \* from customers where customer\_id = {self.customer\_id}"

                self.cursor.execute(details\_query)

                data = self.cursor.fetchall()

                if len(data) == 0:

                    st.write("Account doesnot exist")

                else:

                    details = ["customer\_id" , "customer\_name" , "address" , "city" , "phone"]

                    for i , j in zip(details , data[0]):

                        st.write(i + " - " + str(j))

        if n ==3 :

                transactions = f"select \* from Transactions where account\_number in (select account\_number from Accounts where customer\_id = {self.customer\_id})"

                self.cursor.execute(transactions)

                data= self.cursor.fetchall()

                if len(data) == 0 :

                    print("No Transactions")

                else:

                   transactions\_details = ["transaction\_id" , "account\_number" , "transaction\_type" , "amount" , "transaction\_date" , "description"]

                   df = pd.DataFrame(data , columns=transactions\_details)

                   st.table(df)

        if n == 5:

                balance\_query = f"SELECT balance FROM Accounts where customer\_id = {self.customer\_id}"

                self.cursor.execute(balance\_query)

                data = self.cursor.fetchall()

                st.write(f"Balance - {data[0][0]}")

        if n == 6 :

                updated\_name = st.text\_input("Please enter the new name - ")

                updated\_address = st.text\_input("Please enter the updated address - ")

                updated\_city = st.text\_input("Please enter the updated city - ")

                updated\_phone = st.text\_input("Please enter the updated phone - ")

                if not updated\_name or not updated\_address or not updated\_city or not updated\_phone:

                     st.stop()

                self.cursor.execute(f"Update customers set customer\_name = '%s' , address = '%s' , city = '%s',\

                                    phone = '%s' where customer\_id = {self.customer\_id}" % (updated\_name , updated\_address , updated\_city , updated\_phone))

                updated\_details\_query = f"select \* from customers where customer\_id = {self.customer\_id}"

                self.cursor.execute(updated\_details\_query)

                data = self.cursor.fetchall()

                st.write("UPDATED DETAILS - ")

                details = ["customer\_id" , "customer\_name" , "address" , "city" , "phone"]

                for i , j in zip(details , data[0]):

                    st.write(i + " - " + str(j))

        if n == 7 :

                self.cursor.execute(f"select \* from Loans where customer\_id = {self.customer\_id}")

                data = self.cursor.fetchall()

                if len(data) > 0:

                    df = pd.DataFrame(data , columns=["Loan id" , "Customer\_id" , "amount" , "rate" , "start date" , "end date"])

                    st.table(df)

                else:

                    st.write("You dont have any loans")

        if n==8 :

             appointment\_date = st.date\_input("Please enter the day for which you what to book the appointement")

             appointment\_time = st.time\_input("Please enter the time of appointment")

             customer\_id = st.number\_input("Please enter your customer\_id")

             name = st.text\_input("Please enter your name")

             if not appointment\_date or not appointment\_time or not customer\_id or not name:

                  st.stop()

             self.cursor.execute(f"INSERT INTO appointment (appointment\_date , appointment\_time , customer\_id , customer\_name) VALUES ('%s' , '%s' , {customer\_id} , '%s')"%(appointment\_date , appointment\_time ,name))

             st.write(f"Your appointment is book on {appointment\_date}")

    def options\_employee(self , n):

        if n==1 :

            # cash credit

            account\_number = st.number\_input("enter the account number")

            if not account\_number:

                 st.stop()

            self.cursor.execute(f"select \* from Accounts where account\_number = {int(account\_number)}")

            data = self.cursor.fetchall()

            account\_balance = data[0][3]

            if len(data) ==  0:

                st.warning("Account is incorrect")

                st.stop()

            amount2 = int(st.number\_input("Pleae enter the amount you to add - "))

            if not amount2:

                    st.stop()

            add\_update\_query = f"update Accounts set   balance= {account\_balance + amount2}  where account\_number={account\_number}"

            self.cursor.execute(add\_update\_query)

            self.mydb.commit()

            st.write(f"Updated Balance - {account\_balance + amount2}")

        if n ==2:

            branch\_id = st.number\_input("Enter the branch id")

            self.cursor.execute(f"select \* from Accounts where branch\_id={branch\_id}")

            data = self.cursor.fetchall()

            df = pd.DataFrame(data , columns=["account\_number" , "customer\_id" , "account\_type" , "balance" , "branch\_id"])

            st.table(df)

        if n==3:

            # loan approval

            customer\_id = st.number\_input("Please the customer number for who you approved the loan")

            amount = st.number\_input("Please the loan amount")

            interset\_rate = st.number\_input("Enter the rate of interset")

            start\_date = st.date\_input("Please enter the strat date")

            end\_date = st.date\_input("Please enter the end date of the loan")

            if not customer\_id or not amount or not interset\_rate:

                 st.stop()

            if start\_date != end\_date:

                self.cursor.execute(f"INSERT INTO Loans (customer\_id , amount , interest\_rate ,start\_date ,end\_date)\

                                VALUES ({customer\_id} , {amount} , {interset\_rate} , '%s' , '%s')" % (start\_date, end\_date))

                st.write("Loan tabel is update")

                self.mydb.commit()

        if n==4 :

            st.write("Report")

            self.cursor.execute("select sum(amount) from Transactions where transaction\_type =  'international'")

            internation = self.cursor.fetchall()[0][0]

            self.cursor.execute("select sum(amount) from Transactions where transaction\_type =  'national'")

            nation = self.cursor.fetchall()[0][0]

            df = pd.DataFrame([(internation , "International") , (nation , "National")] , columns = ["total sum", "type"])

            fig, ax = plt.subplots()

            ax.bar(df["type"] , df["total sum"])

            plt.title("Total amount in international Transactions vs national Transactions")

            st.pyplot(fig)

        if n==5 :

             self.cursor.execute("select \* from appointment")

             data = self.cursor.fetchall()

             df = pd.DataFrame(data , columns = ["Appointment\_ID" , "Customer\_ID" , "Customer\_Name" , "Appointment\_Date" , "Appointment\_Time"])

             st.write('Appointments: '+"\n")

             for i in data :

                for j, k in zip(["Appointment\_ID" , "Customer\_ID" , "Customer\_Name" , "Appointment\_Date" , "Appointment\_Time"] , i):

                     st.write(j + "-"+ str(k))

                st.write("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

        if n==6:

             customer\_id = st.number\_input("Please enter the customer id")

             if not customer\_id:

                  st.stop()

             self.cursor.execute(f"select balance from Accounts where customer\_id = {customer\_id}")

             data = self.cursor.fetchall()

             st.write("Balance - ", data[0][0])

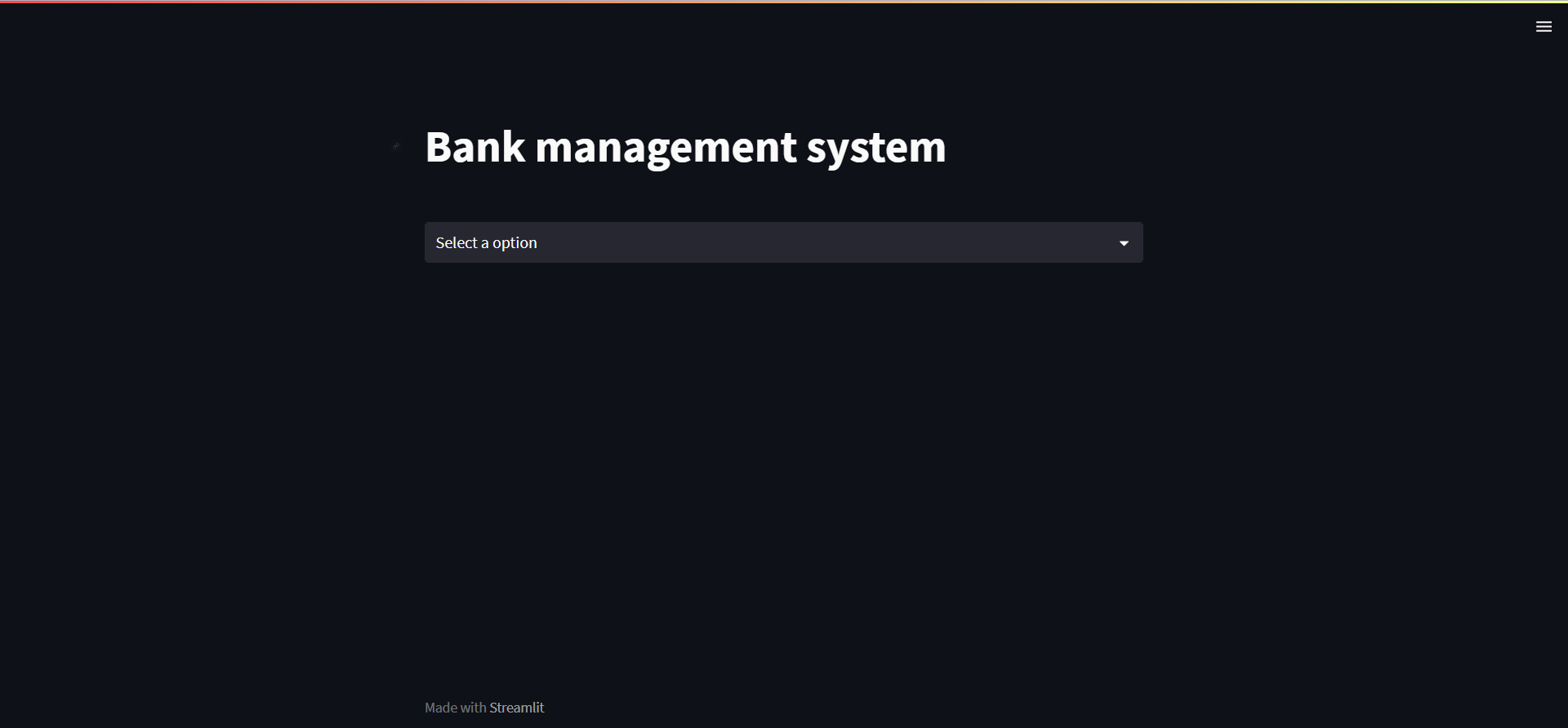
             self.cursor.execute("SET FOREIGN\_KEY\_CHECKS=0")

             self.cursor.execute("delete from Accounts where customer\_id="+str(customer\_id))

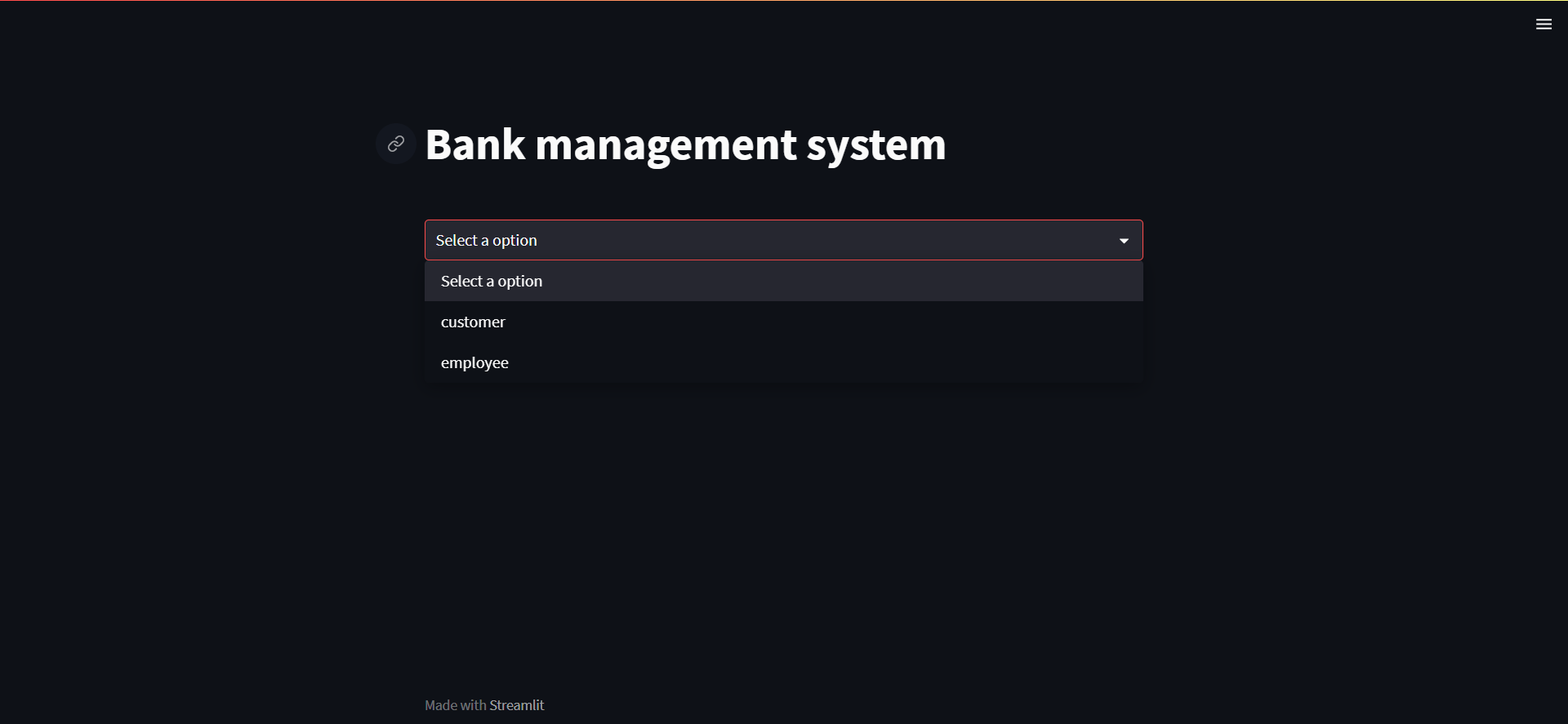
             self.cursor.execute('delete from Customers where customer\_id='+str(customer\_id))

             st.write('ACCOUNT DELETED SUCCESFULLY')

**Output:**

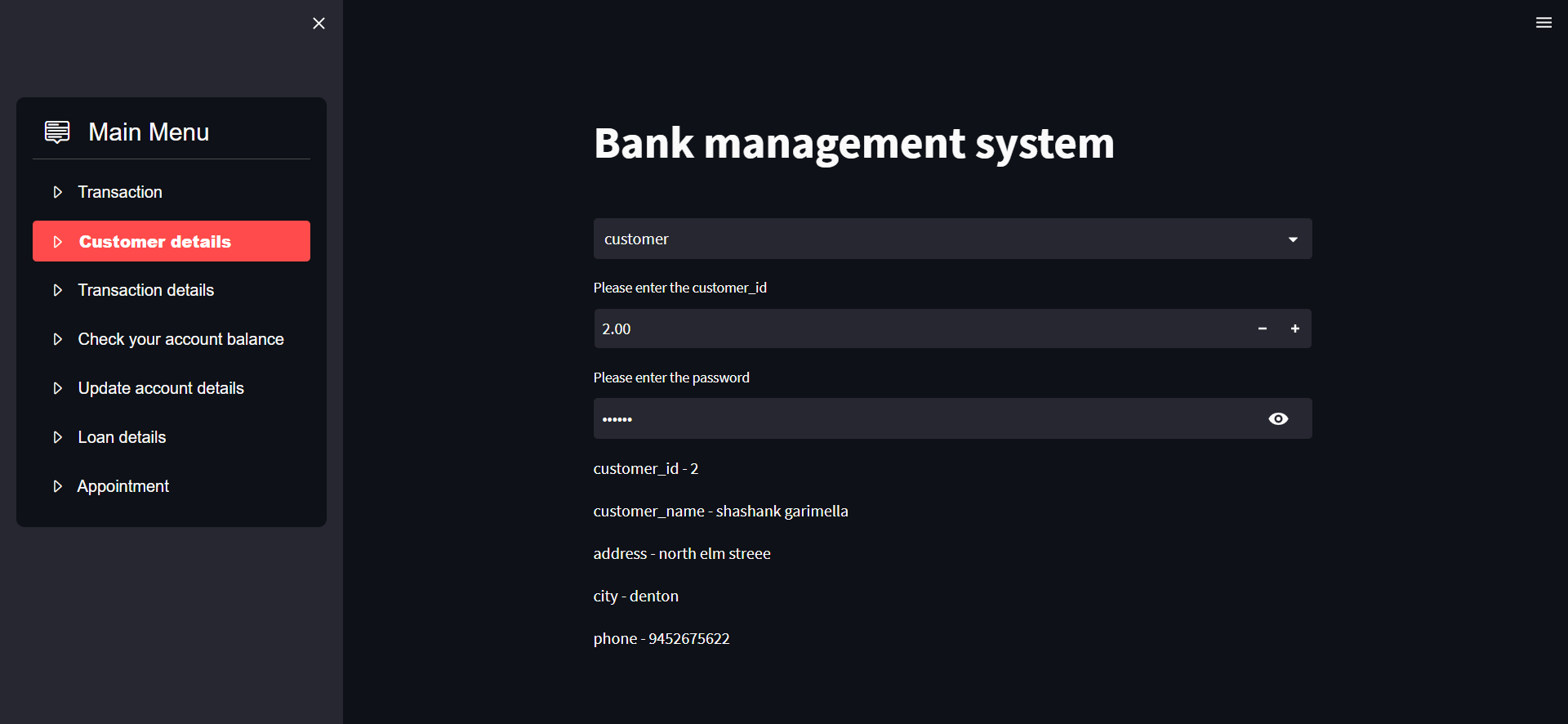


**Customer id Login:**



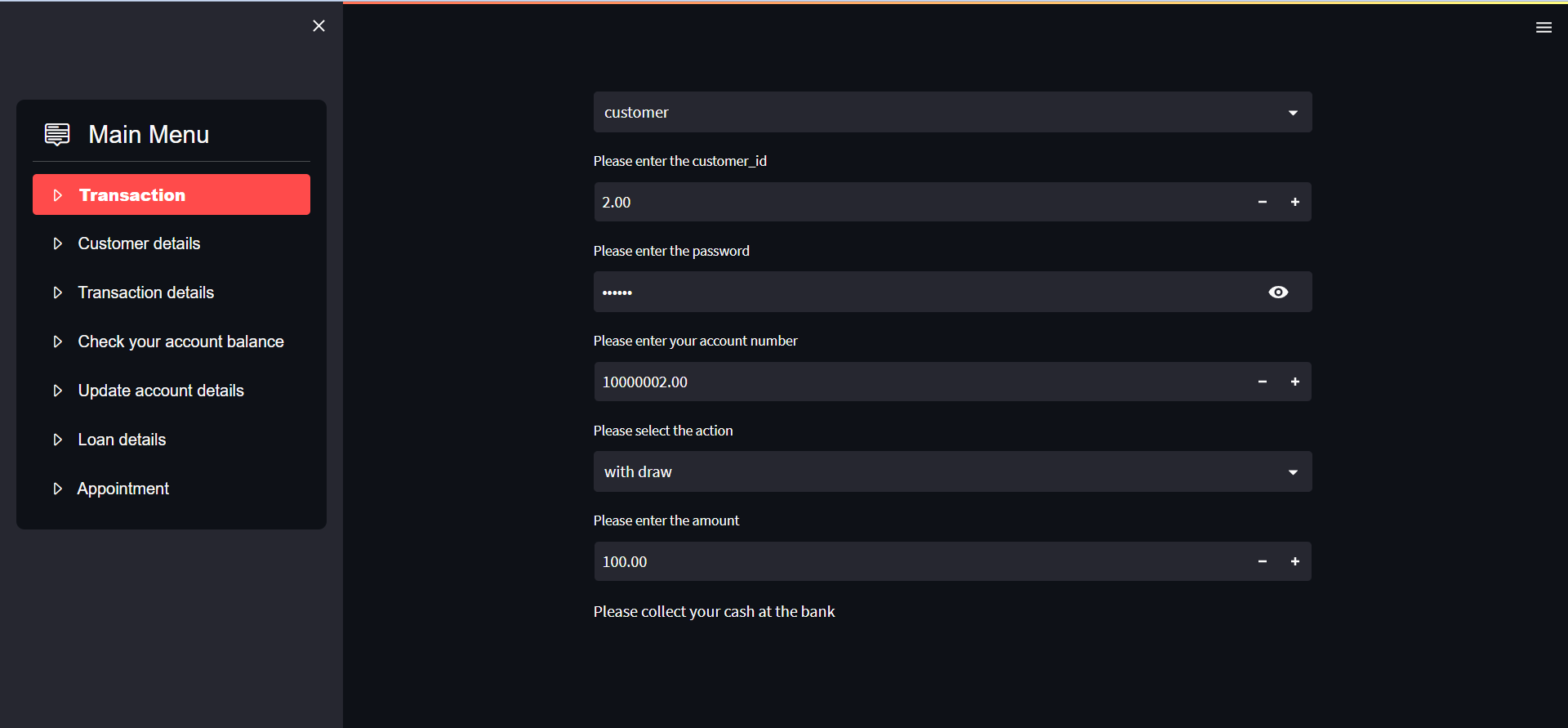
Customer menu:

Customer details

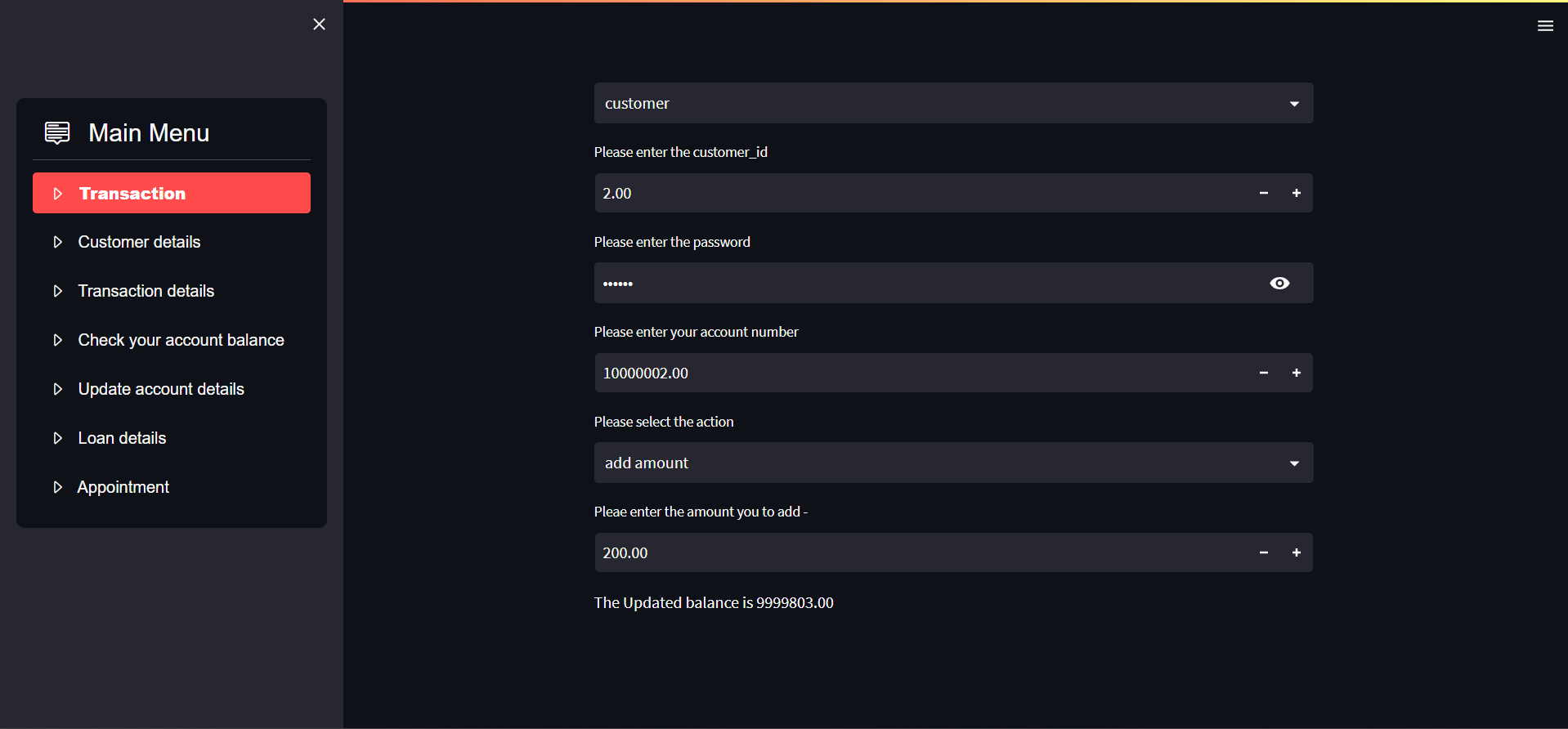


Transaction:

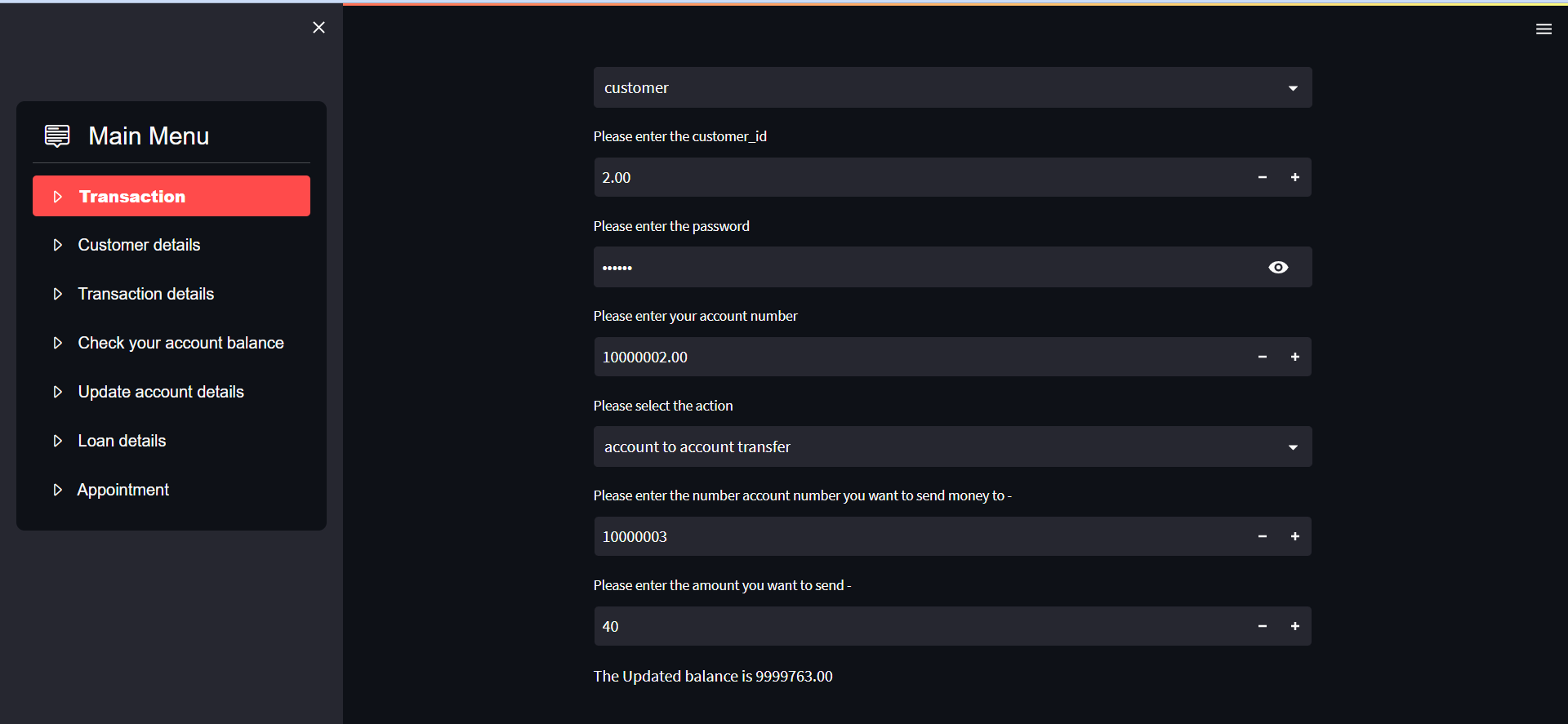
With draw



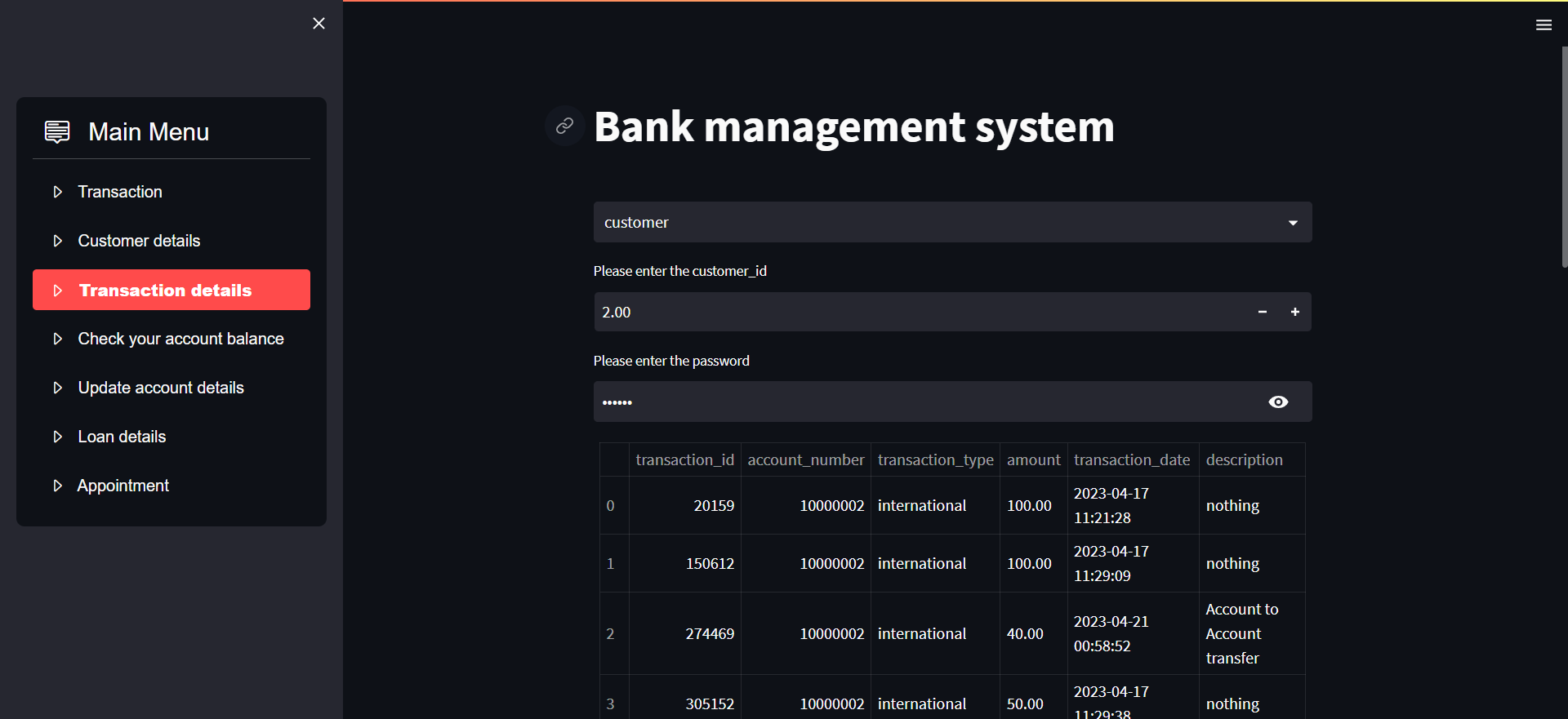
Add amount



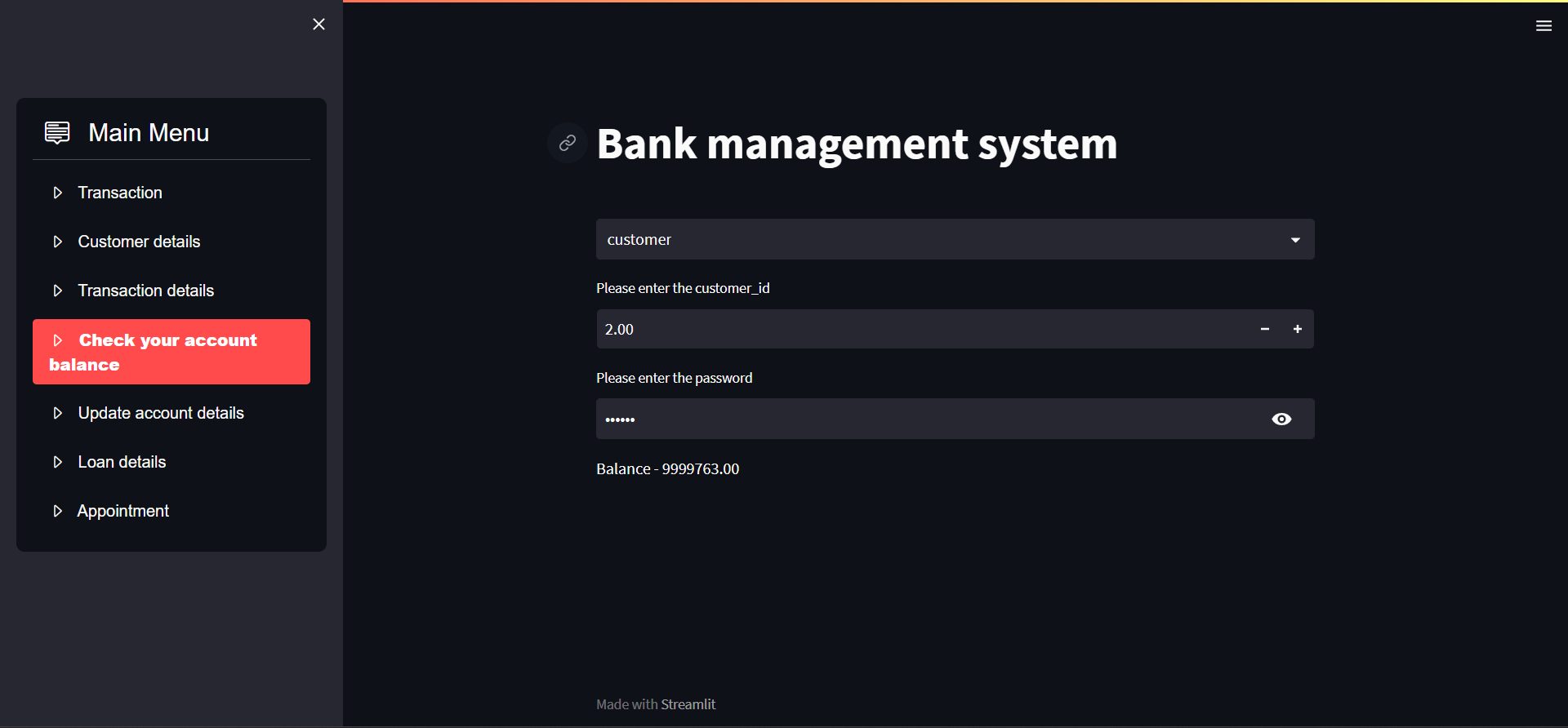
Account to account transfer



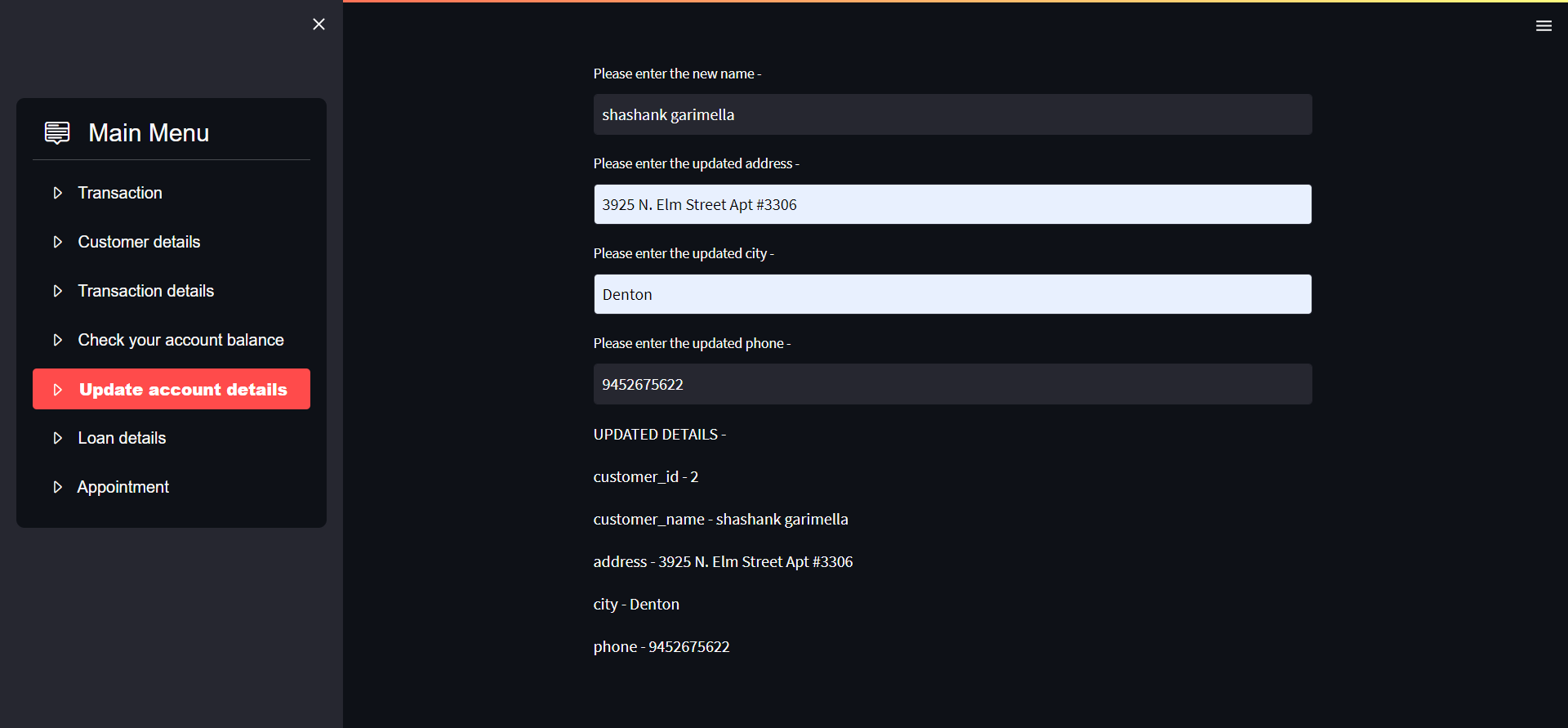
Transaction Details



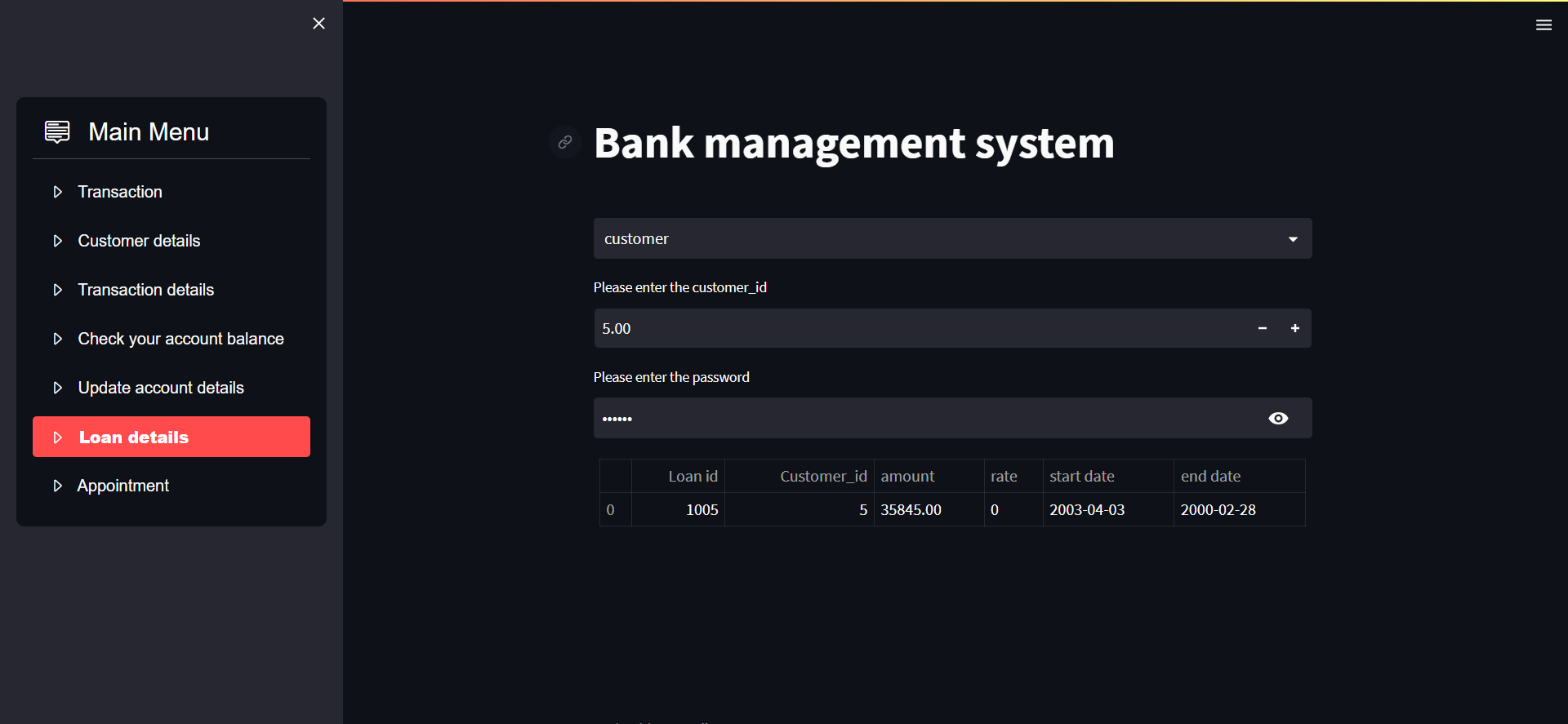
Balance



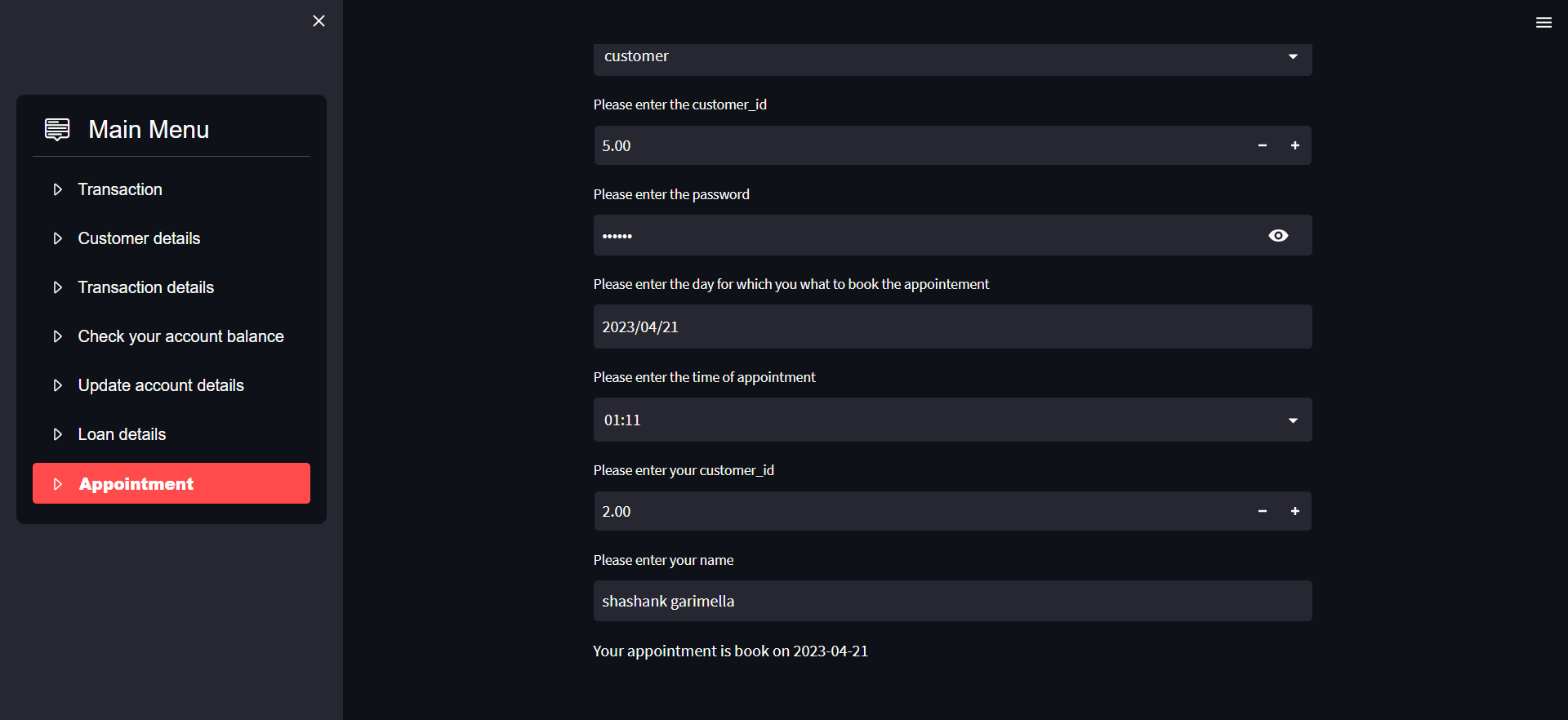
Update account details



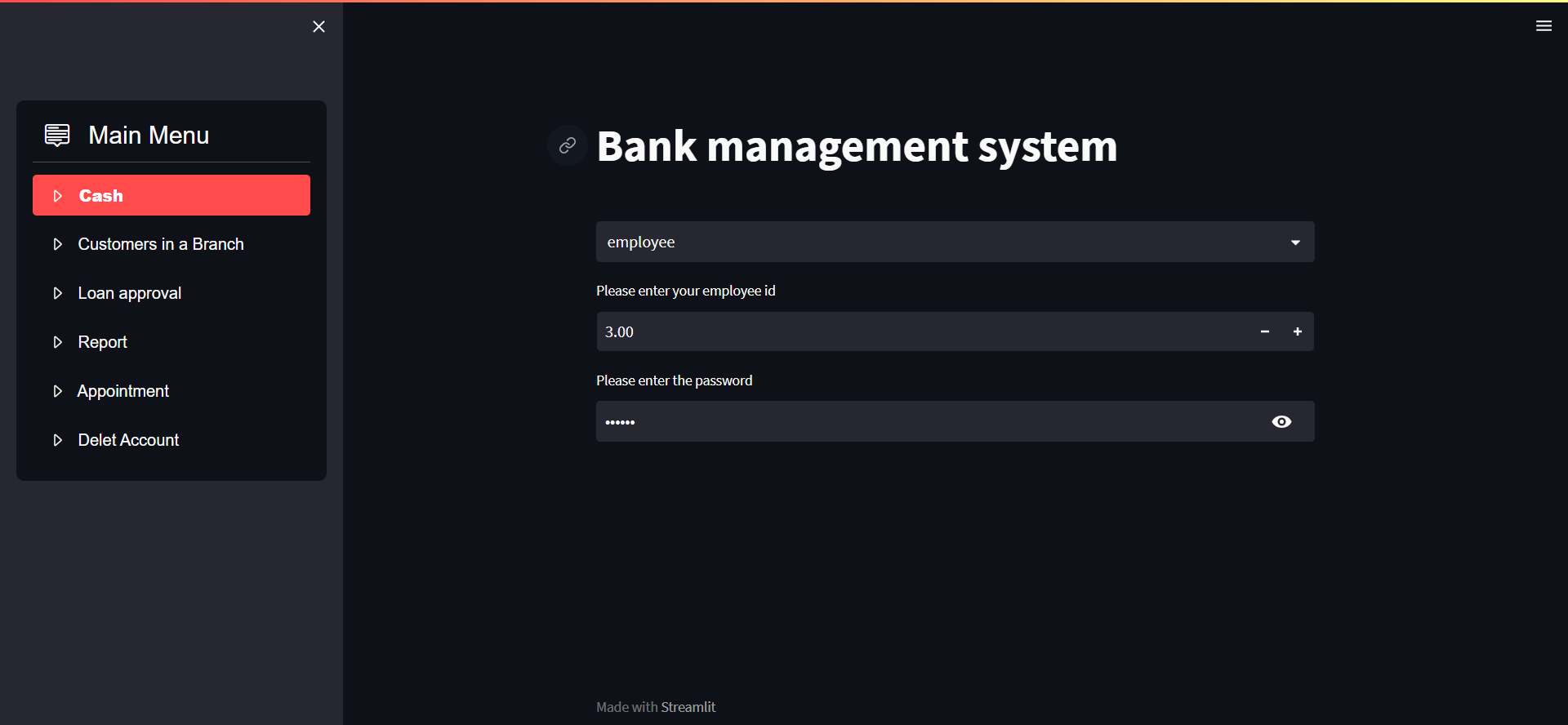
Loan details



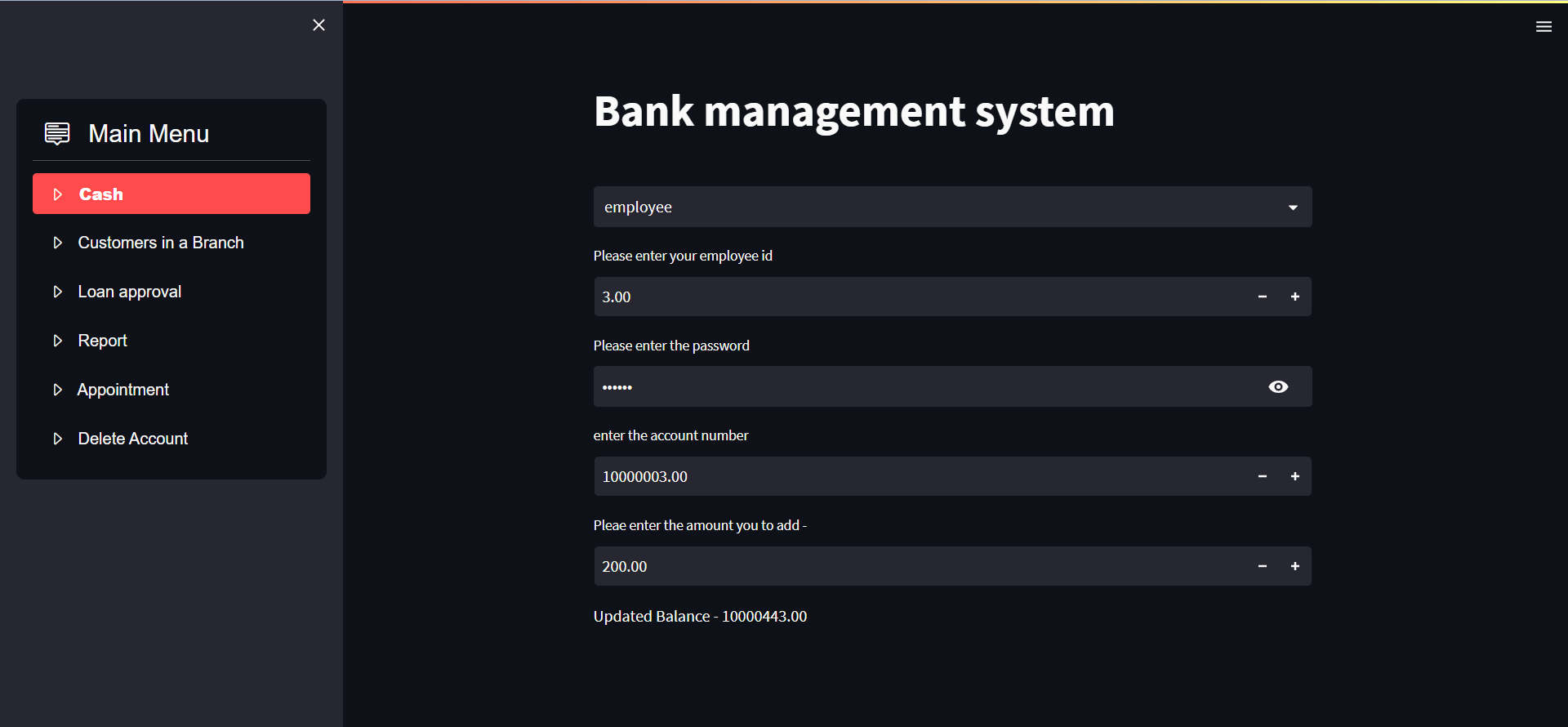
Appointment:



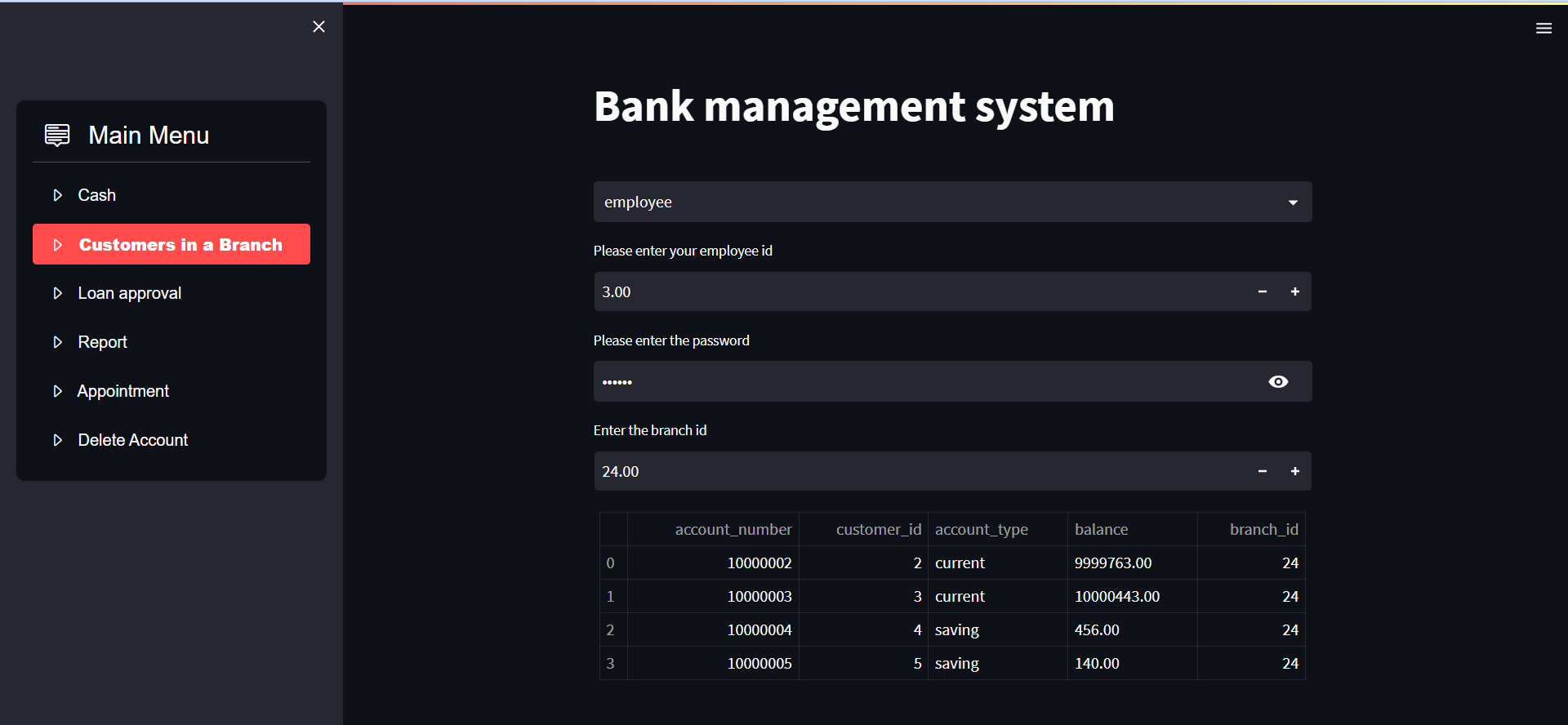
Employee



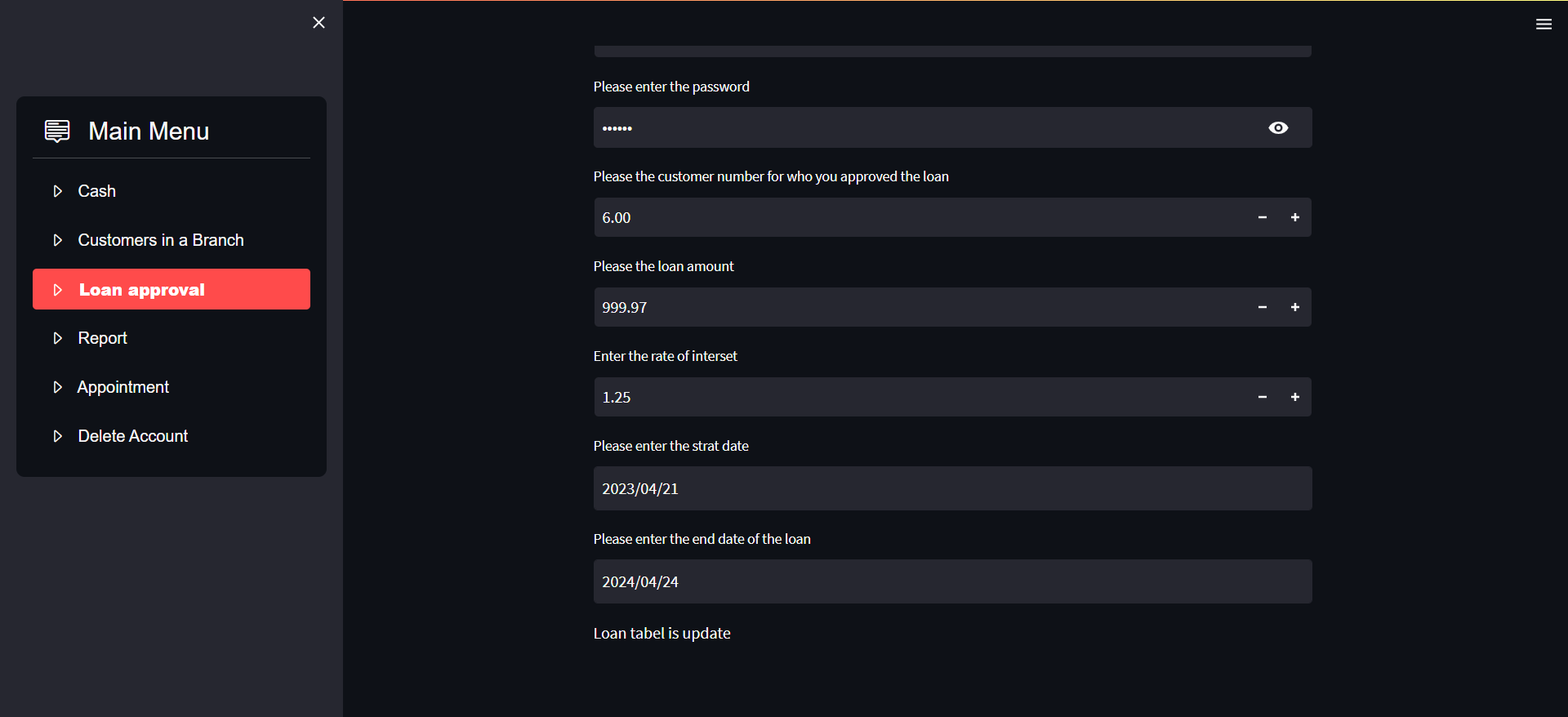
Cash



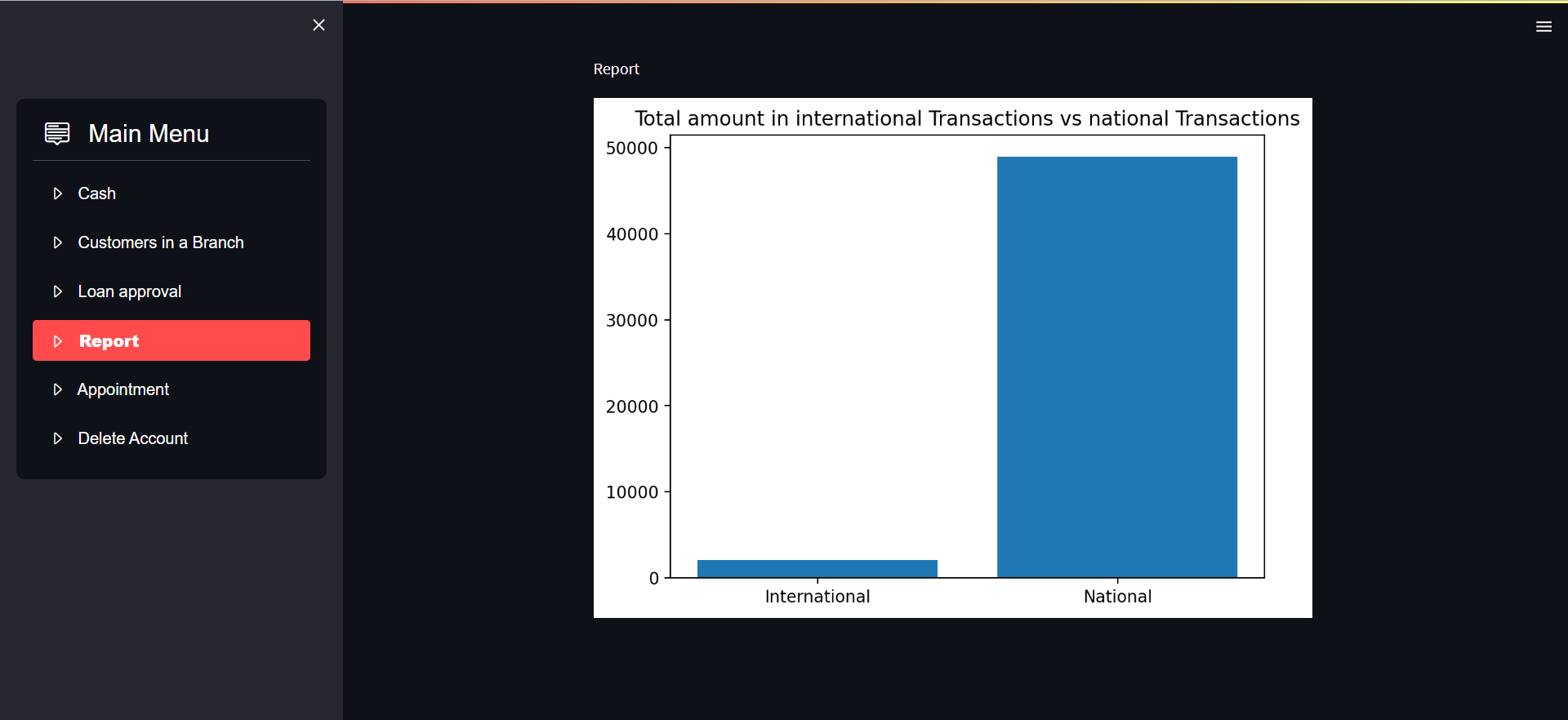
Customer in branch



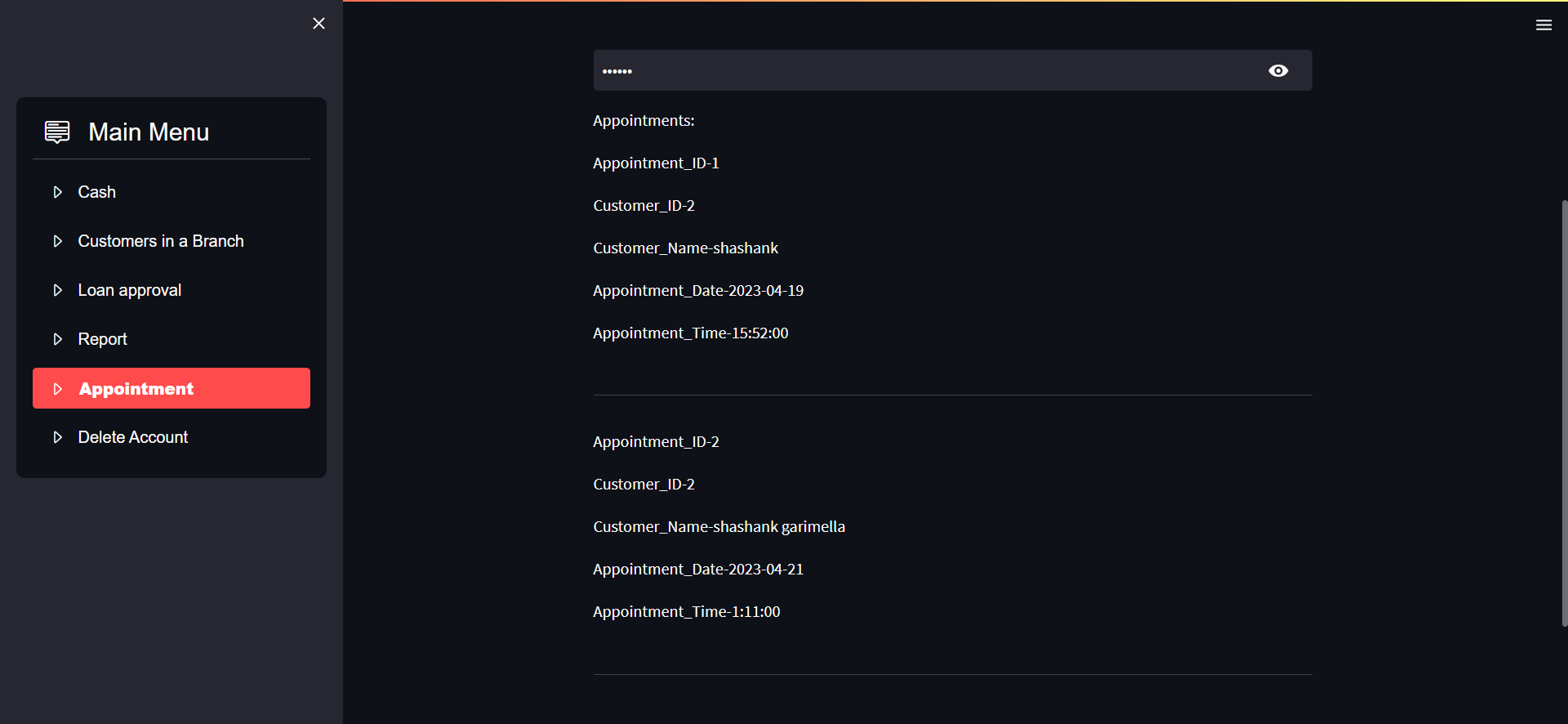
Loan approval



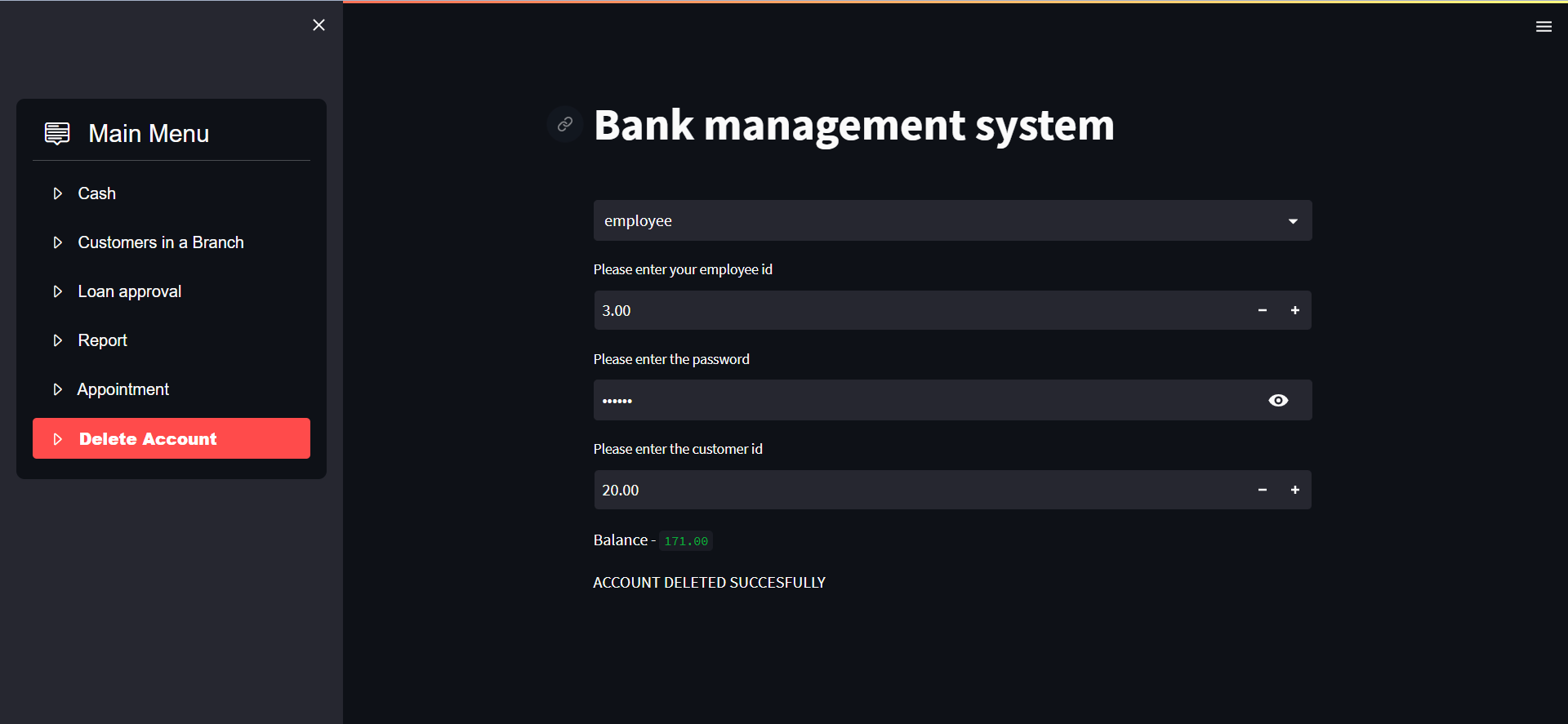
Report



Appointments:



Delete Account



Top of Form

Top of Form