**MAIN.PY**

from register import \*

from bank import \*

status = False

print("Welcome to Banking Project")

while True:

try:

register = int(input("1. SignUp\n"

"2. SignIn"))

if register == 1 or register == 2:

if register == 1:

SignUp()

if register == 2:

user = SignIn()

status = True

break

else:

print("Please Enter Valid Input From Options")

except ValueError:

print("Invalid Input Try Again with Numbers")

account\_number = db\_query(

f"SELECT account\_number FROM customers WHERE username = '{user}';")

while status:

print(f"Welcome {user.capitalize()} Choose Your Banking Service\n")

try:

facility = int(input("1. Balance Enquiry\n"

"2. Cash Deposit\n"

"3. Cash Withdraw\n"

"4. Fund Transfer\n"

"5. Exit\n "

))

if facility >= 1 and facility <= 5:

if facility == 1:

bobj = Bank(user, account\_number[0][0])

bobj.balanceequiry()

elif facility == 2:

while True:

try:

amount = int(input("Enter Amount to Deposit"))

bobj = Bank(user, account\_number[0][0])

bobj.deposit(amount)

mydb.commit()

break

except ValueError:

print("Enter Valid Input ie. Number")

continue

elif facility == 3:

while True:

try:

amount = int(input("Enter Amount to Withdraw"))

bobj = Bank(user, account\_number[0][0])

bobj.withdraw(amount)

mydb.commit()

break

except ValueError:

print("Enter Valid Input ie. Number")

continue

elif facility == 4:

while True:

try:

receive = int(input("Enter Receiver Account Number"))

amount = int(input("Enter Money to Transfer"))

bobj = Bank(user, account\_number[0][0])

bobj.fundtransfer(receive, amount)

mydb.commit()

break

except ValueError:

print("Enter Valid Input ie. Number")

continue

elif facility == 5:

print("Thanks For Using Banking Services")

status = False

else:

print("Please Enter Valid Input From Options")

continue

except ValueError:

print("Invalid Input Try Again with Numbers")

continue

The **Banking Management System** is a console-based Python application designed to mimic basic banking functionalities. The system allows users to create an account, log in, and perform various banking services securely. The program integrates user authentication and core banking operations, ensuring an interactive and practical user experience.

**Register.py Page**

#User Registration Signin Signup

from customer import \*

from bank import Bank

import random

def SignUp():

username = input("Create Username: ")

temp = db\_query(f"SELECT username FROM customers where username = '{username}';")

if temp:

print("Username Already Exists")

SignUp()

else:

print("Username is Available Please Proceed")

password = input("Enter Your Password: ")

name = input("Enter Your Name: ")

age = input("Enter Your Age: ")

city = input("Enter Your City: ")

while True:

account\_number = int(random.randint(10000000, 99999999))

temp = db\_query(f"SELECT account\_number FROM customers WHERE account\_number = '{account\_number}';")

if temp:

continue

else:

print("Your Account Number",account\_number)

break

cobj = Customer(username, password, name, age, city, account\_number)

cobj.createuser()

bobj = Bank(username, account\_number)

bobj.create\_transaction\_table()

def SignIn():

username = input("Enter Username: ")

temp = db\_query(f"SELECT username FROM customers where username = '{username}';")

if temp:

while True:

password = input(f"Welcome {username.capitalize()} Enter Password: ")

temp = db\_query(f"SELECT password FROM customers where username = '{username}';")

# print(temp[0][0])

if temp[0][0] == password:

print("Sign IN Successfully")

return username

else:

print("Wrong Password Try Again")

continue

else:

print("Enter Correct Username")

SignIn()

This Python code provides the core functionality for **user registration (SignUp)** and **user authentication (SignIn)** in a banking management system. It ensures that user accounts are created and managed securely, with proper validation and database integration.

***Oustamer.py***

#Customer Details

from database import \*

class Customer:

def \_\_init\_\_(self, username, password, name, age, city, account\_number):

self.\_\_username = username

self.\_\_password = password

self.\_\_name = name

self.\_\_age = age

self.\_\_city = city

self.\_\_account\_number = account\_number

def createuser(self):

db\_query(f"INSERT INTO customers VALUES ('{self.\_\_username}', '{self.\_\_password}', '{self.\_\_name}', '{self.\_\_age}', '{self.\_\_city}', 0 , '{self.\_\_account\_number}', 1 );")

mydb.commit()

The **Customer** class defines the structure and behavior for handling customer data in a banking management system. It encapsulates user information and provides a method to store customer details securely in a database.

***Database.py***

#Database Management Banking

import mysql.connector as sql

mydb = sql.connect(

host="localhost",

user="root",

passwd="7877",

database="bank"

)

cursor = mydb.cursor()

def db\_query(str):

cursor.execute(str)

result = cursor.fetchall()

return result

def createcustomertable():

cursor.execute('''

CREATE TABLE IF NOT EXISTS customers

(username VARCHAR(20) NOT NULL,

password VARCHAR(20) NOT NULL,

name varchar(20) NOT NULL,

age INTEGER NOT NULL,

city VARCHAR(20) NOT NULL,

balance INTEGER NOT NULL,

account\_number INTEGER NOT NULL,

status BOOLEAN NOT NULL)

''')

mydb.commit()

if \_\_name\_\_ == "\_\_main\_\_":

createcustomertable()

The provided code defines a database management module for a banking application using Python and MySQL. Here's a detailed explanation of the code:

**Bank.py**

# Bank Services

from database import \*

import datetime

class Bank:

def \_\_init\_\_(self, username, account\_number):

self.\_\_username = username

self.\_\_account\_number = account\_number

def create\_transaction\_table(self):

db\_query(f"CREATE TABLE IF NOT EXISTS {self.\_\_username}\_transaction "

f"( timedate VARCHAR(30),"

f"account\_number INTEGER,"

f"remarks VARCHAR(30),"

f"amount INTEGER )")

def balanceequiry(self):

temp = db\_query(

f"SELECT balance FROM customers WHERE username = '{self.\_\_username}';")

print(f"{self.\_\_username} Balance is {temp[0][0]}")

def deposit(self, amount):

temp = db\_query(

f"SELECT balance FROM customers WHERE username = '{self.\_\_username}';")

test = amount + temp[0][0]

db\_query(

f"UPDATE customers SET balance = '{test}' WHERE username = '{self.\_\_username}'; ")

self.balanceequiry()

db\_query(f"INSERT INTO {self.\_\_username}\_transaction VALUES ("

f"'{datetime.datetime.now()}',"

f"'{self.\_\_account\_number}',"

f"'Amount Deposit',"

f"'{amount}'"

f")")

print(f"{self.\_\_username} Amount is Sucessfully Depositted into Your Account {self.\_\_account\_number}")

def withdraw(self, amount):

temp = db\_query(

f"SELECT balance FROM customers WHERE username = '{self.\_\_username}';")

if amount > temp[0][0]:

print("Insufficient Balance Please Deposit Money")

else:

test = temp[0][0] - amount

db\_query(

f"UPDATE customers SET balance = '{test}' WHERE username = '{self.\_\_username}'; ")

self.balanceequiry()

db\_query(f"INSERT INTO {self.\_\_username}\_transaction VALUES ("

f"'{datetime.datetime.now()}',"

f"'{self.\_\_account\_number}',"

f"'Amount Withdraw',"

f"'{amount}'"

f")")

print(

f"{self.\_\_username} Amount is Sucessfully Withdraw from Your Account {self.\_\_account\_number}")

def fundtransfer(self, receive, amount):

temp = db\_query(

f"SELECT balance FROM customers WHERE username = '{self.\_\_username}';")

if amount > temp[0][0]:

print("Insufficient Balance Please Deposit Money")

else:

temp2 = db\_query(

f"SELECT balance FROM customers WHERE account\_number = '{receive}';")

if temp2 == []:

print("Account Number Does not Exists")

else:

test1 = temp[0][0] - amount

test2 = amount + temp2[0][0]

db\_query(

f"UPDATE customers SET balance = '{test1}' WHERE username = '{self.\_\_username}'; ")

db\_query(

f"UPDATE customers SET balance = '{test2}' WHERE account\_number = '{receive}'; ")

receiver\_username = db\_query(

f"SELECT username FROM customers where account\_number = '{receive}';")

self.balanceequiry()

db\_query(f"INSERT INTO {receiver\_username[0][0]}\_transaction VALUES ("

f"'{datetime.datetime.now()}',"

f"'{self.\_\_account\_number}',"

f"'Fund Transfer From {self.\_\_account\_number}',"

f"'{amount}'"

f")")

db\_query(f"INSERT INTO {self.\_\_username}\_transaction VALUES ("

f"'{datetime.datetime.now()}',"

f"'{self.\_\_account\_number}',"

f"'Fund Transfer -> {receive}',"

f"'{amount}'"

f")")

print(

f"{self.\_\_username} Amount is Sucessfully Transaction from Your Account {self.\_\_account\_number}")

***Project details***

## ****Features and Functionalities****

### ****1. User Registration and Authentication****

#### ****SignUp:****

* **Purpose:** Allows new users to register and create an account.
* **Process:**
  1. Users provide a unique username.
  2. The system checks the database to ensure the username is not already taken.
  3. If available:
     + Users set a password and provide personal details (e.g., full name, age, city).
     + A unique account number is auto-generated and displayed to the user.
     + User data is securely stored in the database.
     + A transaction table specific to the user is initialized for future transactions.

#### ****SignIn:****

* **Purpose:** Enables registered users to access their account.
* **Process:**
  1. Users enter their username.
  2. The system verifies the username against the database.
  3. If the username exists:
     + Users provide their password.
     + The system validates the password against stored credentials.
  4. On successful login:
     + The username is returned to the main program for session handling.
     + Users gain access to banking services.

### ****2. Banking Services****

#### ****Balance Enquiry:****

* Users can check their current account balance.
* The balance is fetched from the database and displayed.

#### ****Cash Deposit:****

* Allows users to deposit money into their account.
* The deposited amount is added to the current balance and updated in the database.
* A transaction record is created with details of the deposit.

#### ****Cash Withdrawal:****

* Users can withdraw a specified amount, provided they have sufficient funds.
* The withdrawn amount is deducted from the current balance and updated in the database.
* A transaction record is created with details of the withdrawal.
* If insufficient funds are available, an error message is displayed.

#### ****Fund Transfer:****

* Users can transfer money to another account by providing:
  + Recipient’s account number.
  + Transfer amount.
* The system validates:
  + Sufficient balance in the sender’s account.
  + Existence of the recipient’s account.
* On successful validation:
  + Deducts the transfer amount from the sender’s account.
  + Credits the amount to the recipient’s account.
  + Creates transaction records for both sender and recipient.

#### ****Exit:****

* Allows users to securely log out of the system, ending their session.

## ****SignUp Process****

1. **Unique Username:** Users choose a unique identifier.
2. **Database Check:** The system ensures the username is not already in use.
3. **Successful Registration:**
   * Users set a password and provide details like name, age, and city.
   * A unique account number is generated.
   * User data is saved to the database.
   * A transaction table is initialized for the user.

## ****SignIn Process****

1. **Username Verification:** Users provide their username, which is checked against the database.
2. **Password Validation:** If the username exists, users input their password, which is validated.
3. **Access Granted:** On successful login, users access banking services.

## ****Attributes****

The Bank class stores user information using private attributes to ensure security and adhere to encapsulation principles:

### ****Private Attributes****

* **\_\_username:** Unique identifier chosen by the user.
* **\_\_password:** User-set password for secure authentication.
* **\_\_name:** Full name of the user.
* **\_\_age:** Age of the user.
* **\_\_city:** City where the user resides.
* **\_\_account\_number:** System-generated unique account number.

### ****Encapsulation:****

* These attributes cannot be accessed or modified directly from outside the class.
* Ensures data integrity and security.