Banking Management System - Documentation

Overview:

This is a Python-based console Banking Management System using SQLite for data storage. It provides role-based access for **Admin** and **Customer** users.

User Roles & Functionalities:

👤 Customer Functionalities:

1. Deposit

- Allows user to deposit money into their account.
- Updates balance and logs the transaction.

2. Withdraw

- User can withdraw if the balance is sufficient.
- Balance is updated and transaction recorded.
- Displays new balance post withdrawal.

3. Transfer Funds

- Transfer money to another user.
- Checks for sufficient balance.
- Logs the transfer in the transactions table.

4. View Transactions

- Displays the user's transaction history.
- Sorted by type, amount, and date.

5. Apply for Loan

- Allows customer to request a loan.
- Loan is recorded with status as 'pending'.

6. Logout

- Exit from customer dashboard.

💻 Admin Functionalities:

1. Transfer Funds (on behalf of sender)

- Admin can manually transfer funds from one user to another.
- Useful for assisting customers.

2. View Transactions

- View transaction history of a specific customer.

3. Create Customer

- Adds a new customer with a username and password.
- Automatically assigns customer role.

4. Delete Customer

- Permanently removes a customer and their data.

5. Update Customer

- Changes the customer's username.
- Useful for correcting input or renaming accounts.

6. List Customers

- Displays all customers with ID, username, and balance.

7. Search Customers by Balance

- Filter customers based on a balance range.

8. View Customer Transactions

- View detailed transaction logs of a selected customer.

9. Manage Loans

- View all loans and their statuses.
- Approve or reject a loan request.

10. Generate Report

- Shows summary stats:
    - Total number of customers
    - Total balance across all accounts
    - Number of loans by status (approved/rejected/pending)

11. Logout

- Exit from admin dashboard.

Key Points:

- All passwords are hashed using **bcrypt**.
- SQLite used as backend DB for storing users, transactions, and loans.
- Role-based interface with restricted access.
- Each transaction is timestamped for tracking.
- Built-in loan management for customers.
- Admin can manage users and generate reports.

Security & Data Integrity:

- Passwords are not stored in plain text.
- Foreign key constraints ensure valid user IDs in transactions and loans.
- Input validation ensures roles are correctly assigned.

Dependencies:

- `bcrypt`
- `sqlite3`
- `datetime`

Run this script in a Python environment to initialize the database and start using the system.

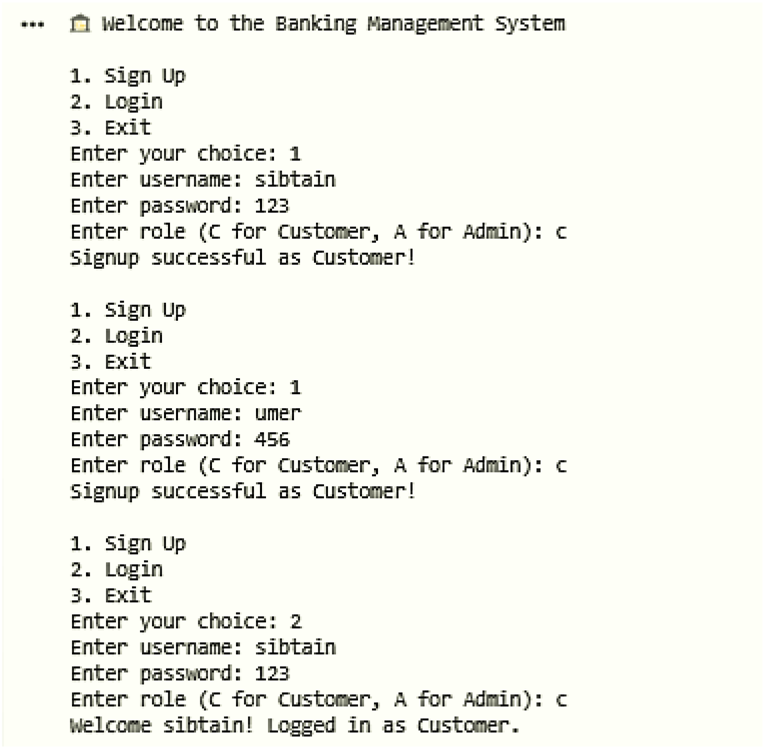
CODE:

```
!pip install bcrypt

import bcrypt
import sqlite3
from datetime import datetime

# Database Setup
def init_db():
    conn = sqlite3.connect('banking.db')
    c = conn.cursor()
    # Users
    c.execute('''CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT UNIQUE,
        password TEXT,
        role TEXT,
        balance REAL DEFAULT 0.0
    )''')
    # Transactions
    c.execute('''CREATE TABLE IF NOT EXISTS transactions (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        user_id INTEGER,
        type TEXT,
        amount REAL,
        date TEXT,
        FOREIGN KEY(user_id) REFERENCES users(id)
    )''')
    # Loans
    c.execute('''CREATE TABLE IF NOT EXISTS loans (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        user_id INTEGER,
        amount REAL,
        status TEXT,
        date_applied TEXT,
        FOREIGN KEY(user_id) REFERENCES users(id)
    )''')
    conn.commit()
    conn.close()

# Authentication
def signup(username, password, role_input):
    role_input = role_input.strip().lower()
    if role_input in ['c', 'customer']:
        role = 'customer'
    elif role_input in ['a', 'admin']:
        role = 'admin'
    else:
        print("Invalid input. Please enter 'C' for Customer or 'A' for
Admin.")
        return

    hashed = bcrypt.hashpw(password.encode(), bcrypt.gensalt())
    conn = sqlite3.connect('banking.db')
    c = conn.cursor()
    try:
        c.execute("INSERT INTO users (username, password, role) VALUES (?,
?, ?)", (username, hashed, role))
        conn.commit()
        print(f"Signup successful as {role.capitalize()}!")
    except sqlite3.IntegrityError:
        print("Username already exists!")
    conn.close()

def login(username, password, role_input):
    role_input = role_input.strip().lower()
    if role_input in ['c', 'customer']:
        role = 'customer'
    elif role_input in ['a', 'admin']:
        role = 'admin'
    else:
        print("Invalid input. Please enter 'C' for Customer or 'A' for
Admin.")
        return None

    conn = sqlite3.connect('banking.db')
    c = conn.cursor()
    c.execute("SELECT id, password FROM users WHERE username=? AND
role=?", (username, role))
    user = c.fetchone()
    conn.close()
    if user and bcrypt.checkpw(password.encode(), user[1]):
        print(f"Welcome {username}! Logged in as {role.capitalize()}.")
        return (user[0], username, role)
    else:
        print("Invalid credentials or role.")
        return None

# Admin Functionalities

def create_customer(username, password):
    signup(username, password, 'c')

def delete_customer(username):
    conn = sqlite3.connect('banking.db')
    c = conn.cursor()
    c.execute("DELETE FROM users WHERE username=? AND role='customer'",
(username,))
    conn.commit()
    conn.close()
    print("Customer account deleted.")

def update_customer(username, new_contact=None):
    conn = sqlite3.connect('banking.db')
    c = conn.cursor()
    if new_contact:
        c.execute("UPDATE users SET username=? WHERE username=? AND
role='customer'", (new_contact, username))
        conn.commit()
        print("Customer information updated.")
    else:
        print("No updates made.")
    conn.close()
```

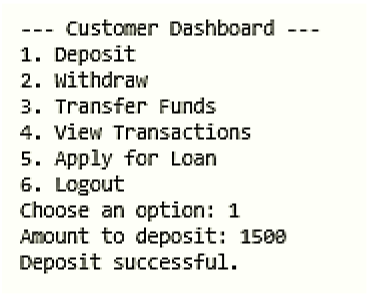
```
def list_customers():
    conn = sqlite3.connect('banking.db')
    c = conn.cursor()
    c.execute("SELECT id, username, balance FROM users WHERE
role='customer'")
    for row in c.fetchall():
        print(row)
    conn.close()

def search_customers_by_balance(min_bal, max_bal):
    conn = sqlite3.connect('banking.db')
    c = conn.cursor()
    c.execute("SELECT username, balance FROM users WHERE balance BETWEEN ?
AND ?", (min_bal, max_bal))
    for row in c.fetchall():
        print(row)
    conn.close()

def view_customer_transactions(username):
    conn = sqlite3.connect('banking.db')
    c = conn.cursor()
    c.execute("SELECT id FROM users WHERE username=?", (username,))
    uid = c.fetchone()
    if uid:
        c.execute("SELECT type, amount, date FROM transactions WHERE
user_id=?", (uid[0],))
        for row in c.fetchall():
            print(f"{row[0]} - ₹{row[1]} on {row[2]}")
    else:
        print("Customer not found.")
    conn.close()

def manage_loans():
    conn = sqlite3.connect('banking.db')
    c = conn.cursor()
    c.execute("SELECT loans.id, users.username, loans.amount, loans.status
FROM loans JOIN users ON loans.user_id = users.id")
    loans = c.fetchall()
    for loan in loans:
        print(f"Loan ID: {loan[0]}, User: {loan[1]}, Amount: ₹{loan[2]},
Status: {loan[3]}")

    loan_id = input("Enter Loan ID to approve/reject (or press Enter to
skip): ")
```

```
    if loan_id:
        action = input("Approve (A) or Reject (R): ").strip().lower()
        status = 'approved' if action == 'a' else 'rejected'
        c.execute("UPDATE loans SET status=? WHERE id=?", (status,
loan_id))
        conn.commit()
        print(f"Loan {loan_id} has been {status}.")
    conn.close()

def generate_report():
    conn = sqlite3.connect('banking.db')
    c = conn.cursor()
    c.execute("SELECT COUNT(*) FROM users WHERE role='customer'")
    users_count = c.fetchone()[0]
    c.execute("SELECT SUM(balance) FROM users WHERE role='customer'")
    total_balance = c.fetchone()[0]
    c.execute("SELECT status, COUNT(*) FROM loans GROUP BY status")
    loan_stats = c.fetchall()

    print(f"Total Customers: {users_count}")
    print(f"Total Balance Held: ₹{total_balance}")
    for stat in loan_stats:
        print(f"Loans {stat[0].capitalize()}: {stat[1]}")
    conn.close()

# Customer Functionalities

def deposit(user, amount):
    uid, _, _ = user
    conn = sqlite3.connect('banking.db')
    c = conn.cursor()
    c.execute("UPDATE users SET balance = balance + ? WHERE id=?",
(amount, uid))
    c.execute("INSERT INTO transactions (user_id, type, amount, date)
VALUES (?, 'deposit', ?, ?)",
              (uid, amount, datetime.now().strftime("%Y-%m-%d %H:%M:%S")))
    conn.commit()
    conn.close()
    print("Deposit successful.")

def withdraw(user, amount):
    uid, _, _ = user
    conn = sqlite3.connect('banking.db')
    c = conn.cursor()
    c.execute("SELECT balance FROM users WHERE id=?", (uid,))
    balance = c.fetchone()[0]
    if balance >= amount:
        c.execute("UPDATE users SET balance = balance - ? WHERE id=?",
(amount, uid))
        c.execute("INSERT INTO transactions (user_id, type, amount, date)
VALUES (?, 'withdraw', ?, ?)",
                  (uid, amount, datetime.now().strftime("%Y-%m-%d
%H:%M:%S")))
        conn.commit()
        print("Withdrawal successful.")
    else:
        print("Insufficient balance.")
    conn.close()

def transfer_funds(sender, receiver_username, amount):
    # Get sender's information from database instead of assuming sender ID
is passed
    conn = sqlite3.connect('banking.db')
    c = conn.cursor()
    c.execute("SELECT id, balance FROM users WHERE username=?",
(sender[1],)) # sender[1] holds the username
    sender_data = c.fetchone()

    if sender_data:
        sid, sender_balance = sender_data  # Get sender ID and balance
    else:
        print("Sender not found.")
        conn.close()
        return

    c.execute("SELECT id, balance FROM users WHERE username=?",
(receiver_username,))
    receiver = c.fetchone()

    if receiver:
        rid, _ = receiver
        # sender_balance is already fetched above
        if sender_balance >= amount:
            c.execute("UPDATE users SET balance = balance - ? WHERE id=?",
(amount, sid))
            c.execute("UPDATE users SET balance = balance + ? WHERE id=?",
(amount, rid))
            c.execute("INSERT INTO transactions (user_id, type, amount,
date) VALUES (?, 'transfer', ?, ?)",
                      (sid, amount, datetime.now().strftime("%Y-%m-%d
%H:%M:%S")))
            conn.commit()
            print("Transfer successful.")
        else:
            print("Insufficient funds.")
    else:
        print("Receiver not found.")
    conn.close()
def view_transactions(user):
    uid, _, _ = user
    conn = sqlite3.connect('banking.db')
    c = conn.cursor()
    c.execute("SELECT type, amount, date FROM transactions WHERE
user_id=?", (uid,))
    records = c.fetchall()
    if records:
        for row in records:
            print(f"{row[0].capitalize()} - ₹{row[1]} on {row[2]}")
    else:
        print("No transactions found.")
    conn.close()

def apply_loan(user, amount):
    uid, _, _ = user
    conn = sqlite3.connect('banking.db')
    c = conn.cursor()
    c.execute("INSERT INTO loans (user_id, amount, status, date_applied)
VALUES (?, ?, 'pending', ?)",
              (uid, amount, datetime.now().strftime("%Y-%m-%d %H:%M:%S")))
    conn.commit()
    conn.close()
    print("Loan application submitted.")
    #updated balance of the user after the transaction.
def withdraw(user, amount):
    uid, _, _ = user
    conn = sqlite3.connect('banking.db')
    c = conn.cursor()
    c.execute("SELECT balance FROM users WHERE id=?", (uid,))
    balance = c.fetchone()[0]
    if balance >= amount:
        c.execute("UPDATE users SET balance = balance - ? WHERE id=?",
(amount, uid))
        c.execute("INSERT INTO transactions (user_id, type, amount, date)
VALUES (?, 'withdraw', ?, ?)",
                  (uid, amount, datetime.now().strftime("%Y-%m-%d
%H:%M:%S")))
        conn.commit()
        c.execute("SELECT balance FROM users WHERE id=?", (uid,))
        updated_balance = c.fetchone()[0]
        print(f"Withdrawal successful. New Balance: ₹{updated_balance}")
    else:
        print("Insufficient balance.")
    conn.close()

# Main Application Loop
def main():
    init_db()
    print("\U0001F3E6 Welcome to the Banking Management System")
    while True:
        print("\n1. Sign Up\n2. Login\n3. Exit")
        choice = input("Enter your choice: ").strip()

        if choice == '1':
            uname = input("Enter username: ")
            pwd = input("Enter password: ")
            role = input("Enter role (C for Customer, A for Admin): ")
            signup(uname, pwd, role)

        elif choice == '2':
            uname = input("Enter username: ")
            pwd = input("Enter password: ")
            role = input("Enter role (C for Customer, A for Admin): ")
            user = login(uname, pwd, role)
            if user:
                uid, username, role = user
                while True:
                    print(f"\n--- {role.capitalize()} Dashboard ---")
                    if role == 'admin':
                        print("1. Transfer Funds")
                        print("2. View Transactions")
                        print("3. Create Customer")
                        print("4. Delete Customer")
                        print("5. Update Customer")
                        print("6. List Customers")
                        print("7. Search Customers by Balance")
                        print("8. View Customer Transactions")
                        print("9. Manage Loans")
                        print("10. View Reports")
                        print("11. Logout")
                        dash_choice = input("Choose an option: ").strip()

                        if dash_choice == '1':
                            sender = input("Sender Username: ")
                            receiver = input("Receiver Username: ")
                            amount = float(input("Amount to transfer: "))
                            transfer_funds((None, sender, 'admin'),
receiver, amount)
                        elif dash_choice == '2':
                            uname = input("Username to view transactions:
")
                            view_customer_transactions(uname)
                        elif dash_choice == '3':
                            uname = input("New customer username: ")
                            pwd = input("Password: ")
                            create_customer(uname, pwd)
                        elif dash_choice == '4':
                            uname = input("Customer username to delete: ")
                            delete_customer(uname)
                        elif dash_choice == '5':
                            uname = input("Existing username: ")
                            newname = input("New username: ")
                            update_customer(uname, newname)
                        elif dash_choice == '6':
                            list_customers()
                        elif dash_choice == '7':
                            minb = float(input("Minimum balance: "))
                            maxb = float(input("Maximum balance: "))
                            search_customers_by_balance(minb, maxb)
                        elif dash_choice == '8':
                            uname = input("Customer username: ")
                            view_customer_transactions(uname)
                        elif dash_choice == '9':
                            manage_loans()
                        elif dash_choice == '10':
                            generate_report()
                        elif dash_choice == '11':
                            print("Logging out...")
                            break
                        else:
                            print("Invalid admin option. Try again.")

                    else:  # Customer dashboard
                        print("1. Deposit")
                        print("2. Withdraw")
                        print("3. Transfer Funds")
                        print("4. View Transactions")
                        print("5. Apply for Loan")
                        print("6. Logout")
                        dash_choice = input("Choose an option: ").strip()

                        if dash_choice == '1':
                            amt = float(input("Amount to deposit: "))
                            deposit(user, amt)
                        elif dash_choice == '2':
                            amt = float(input("Amount to withdraw: "))
                            withdraw(user, amt)
                        elif dash_choice == '3':
                            receiver = input("Receiver username: ")
                            amt = float(input("Amount to transfer: "))
                            transfer_funds(user, receiver, amt)
                        elif dash_choice == '4':
                            view_transactions(user)
                        elif dash_choice == '5':
                            amt = float(input("Loan amount: "))
                            apply_loan(user, amt)
                        elif dash_choice == '6':
                            print("Logging out...")
                            break
                        else:
                            print("Invalid option. Try again.")

        elif choice == '3':
            print("Exiting... Goodbye!")
            break
        else:
            print("Invalid choice. Try again.")

if __name__ == "__main__":
    main()
```

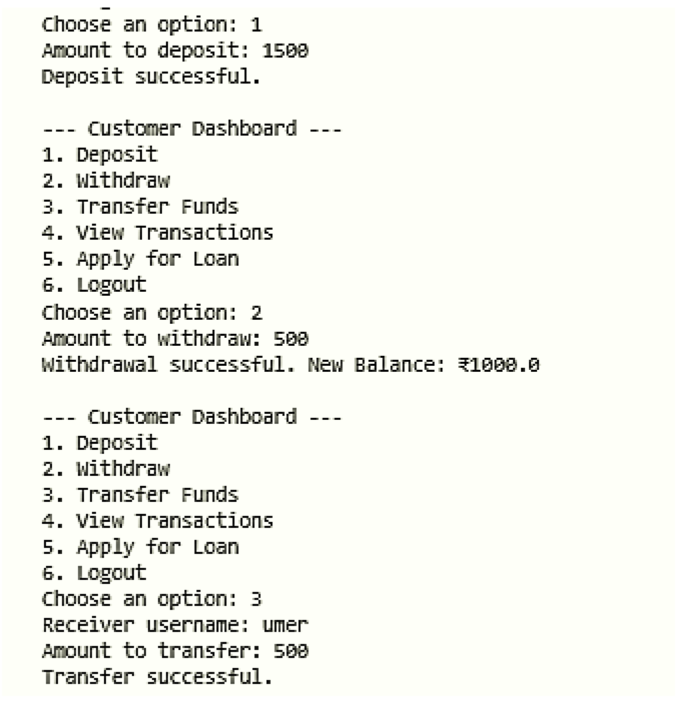
OUTPUT:



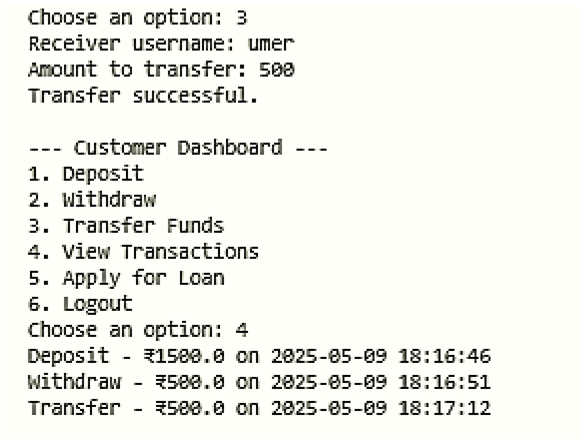
```
••• 🏦 Welcome to the Banking Management System

    1. Sign Up
    2. Login
    3. Exit
    Enter your choice: 1
    Enter username: sibtain
    Enter password: 123
    Enter role (C for Customer, A for Admin): c
    Signup successful as Customer!

    1. Sign Up
    2. Login
    3. Exit
    Enter your choice: 1
    Enter username: umer
    Enter password: 456
    Enter role (C for Customer, A for Admin): c
    Signup successful as Customer!

    1. Sign Up
    2. Login
    3. Exit
    Enter your choice: 2
    Enter username: sibtain
    Enter password: 123
    Enter role (C for Customer, A for Admin): c
    Welcome sibtain! Logged in as Customer.
```



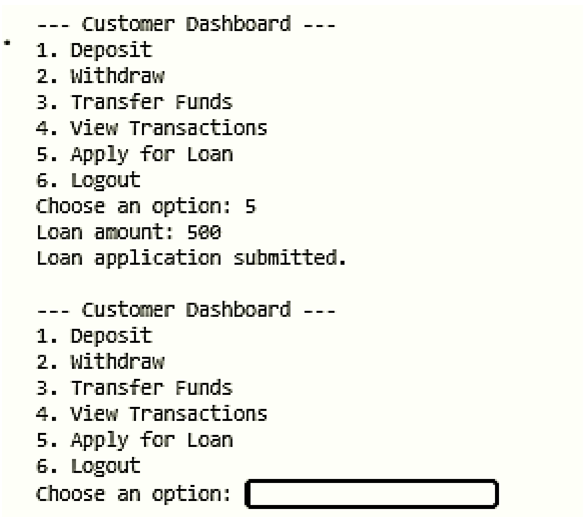
```
--- Customer Dashboard ---
1. Deposit
2. Withdraw
3. Transfer Funds
4. View Transactions
5. Apply for Loan
6. Logout
Choose an option: 1
Amount to deposit: 1500
Deposit successful.
```



```
Choose an option: 1
Amount to deposit: 1500
Deposit successful.

--- Customer Dashboard ---
1. Deposit
2. Withdraw
3. Transfer Funds
4. View Transactions
5. Apply for Loan
6. Logout
Choose an option: 2
Amount to withdraw: 500
Withdrawal successful. New Balance: ₹1000.0

--- Customer Dashboard ---
1. Deposit
2. Withdraw
3. Transfer Funds
4. View Transactions
5. Apply for Loan
6. Logout
Choose an option: 3
Receiver username: umer
Amount to transfer: 500
Transfer successful.
```

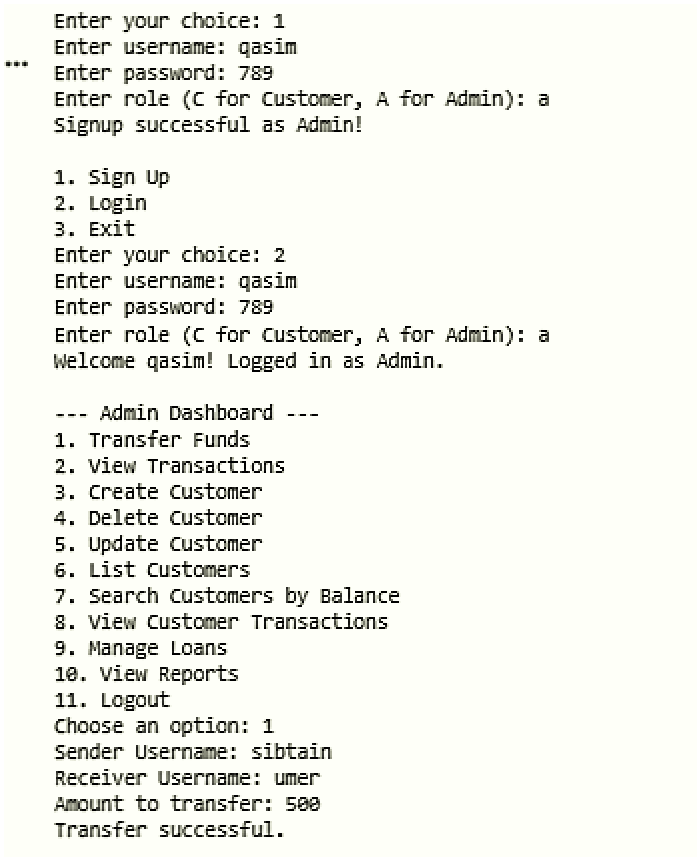


```
Choose an option: 3
Receiver username: umer
Amount to transfer: 500
Transfer successful.

--- Customer Dashboard ---
1. Deposit
2. Withdraw
3. Transfer Funds
4. View Transactions
5. Apply for Loan
6. Logout
Choose an option: 4
Deposit - ₹1500.0 on 2025-05-09 18:16:46
Withdraw - ₹500.0 on 2025-05-09 18:16:51
Transfer - ₹500.0 on 2025-05-09 18:17:12
```

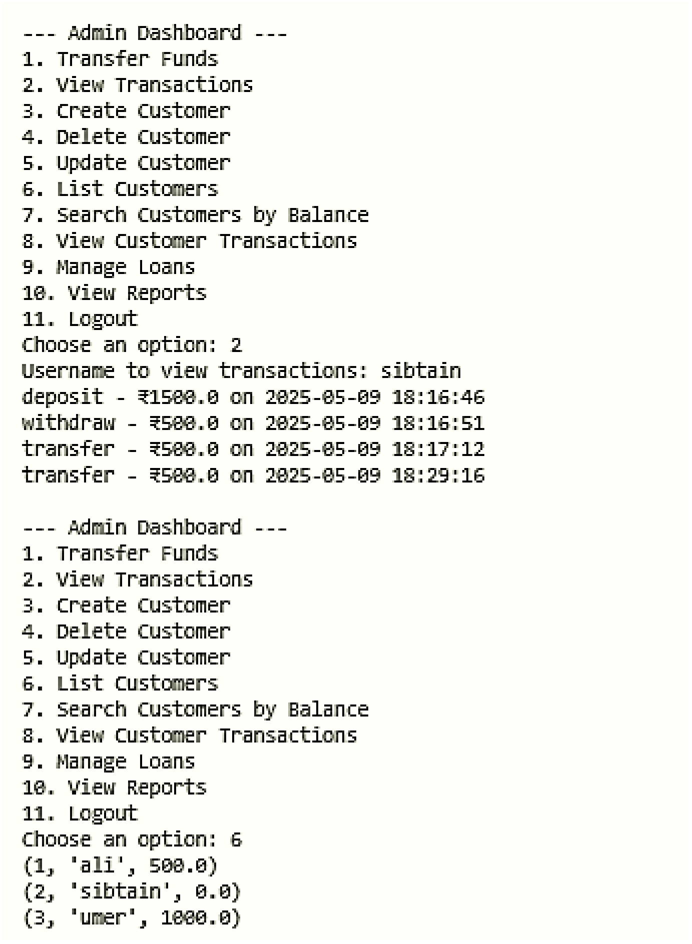


```
--- Customer Dashboard ---
1. Deposit
2. Withdraw
3. Transfer Funds
4. View Transactions
5. Apply for Loan
6. Logout
Choose an option: 5
Loan amount: 500
Loan application submitted.

--- Customer Dashboard ---
1. Deposit
2. Withdraw
3. Transfer Funds
4. View Transactions
5. Apply for Loan
6. Logout
Choose an option:
```

Admin Part:



```
...  Enter your choice: 1
     Enter username: qasim
     Enter password: 789
     Enter role (C for Customer, A for Admin): a
     Signup successful as Admin!

     1. Sign Up
     2. Login
     3. Exit
     Enter your choice: 2
     Enter username: qasim
     Enter password: 789
     Enter role (C for Customer, A for Admin): a
     Welcome qasim! Logged in as Admin.

     --- Admin Dashboard ---
     1. Transfer Funds
     2. View Transactions
     3. Create Customer
     4. Delete Customer
     5. Update Customer
     6. List Customers
     7. Search Customers by Balance
     8. View Customer Transactions
     9. Manage Loans
     10. View Reports
     11. Logout
     Choose an option: 1
     Sender Username: sibtain
     Receiver Username: umer
     Amount to transfer: 500
     Transfer successful.
```


```
--- Admin Dashboard ---
1. Transfer Funds
2. View Transactions
3. Create Customer
4. Delete Customer
5. Update Customer
6. List Customers
7. Search Customers by Balance
8. View Customer Transactions
9. Manage Loans
10. View Reports
11. Logout
Choose an option: 2
Username to view transactions: sibtain
deposit - ₹1500.0 on 2025-05-09 18:16:46
withdraw - ₹500.0 on 2025-05-09 18:16:51
transfer - ₹500.0 on 2025-05-09 18:17:12
transfer - ₹500.0 on 2025-05-09 18:29:16

--- Admin Dashboard ---
1. Transfer Funds
2. View Transactions
3. Create Customer
4. Delete Customer
5. Update Customer
6. List Customers
7. Search Customers by Balance
8. View Customer Transactions
9. Manage Loans
10. View Reports
11. Logout
Choose an option: 6
(1, 'ali', 500.0)
(2, 'sibtain', 0.0)
(3, 'umer', 1000.0)
```