**BANK**

**MANAGEMENT SYSTEM**

**Submitted By :- Abhishek**

**Date :- 10-08-2024**

**TABLE OF CONTENT**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **TOPIC** | **PAGE NO** |
| *1* | Introduction | 1 |
| *2* | Objective | 3 |
| *3* | Tools/Environment | 4 |
| *4* | Program Code | 7 |
| *5* | Input and Output Screens | 20 |
| *6* | Limitations of the Project | 26 |
| *7* | Future Applications of the Project | 27 |
| *8* | Bibliography | 28 |

**INTRODUCTION**

The Bank Management System (BMS) represents a fundamental shift in the way banking operations are conducted, offering an integrated platform that streamlines various banking activities into a unified system. It is designed to handle the complex and critical functions of a financial institution, ensuring efficiency, security, and reliability. A BMS manages customer accounts, transactions, loans, and other essential banking services, providing both customers and bank employees with a seamless and user-friendly interface.

**Bank Management System ..**

In today's fast-paced world, the demand for swift and secure financial transactions is higher than ever. Traditional banking methods, often reliant on paper-based processes, are inefficient and prone to errors. The advent of digital banking requires a robust system capable of handling millions of transactions daily while maintaining the highest standards of accuracy and security. A Bank Management System addresses these needs by automating core banking functions, reducing manual intervention, and enabling 24/7 service availability.



**Key Features of the Bank Management System**

The BMS is built to cater to various banking needs, integrating multiple features that support the smooth operation of a bank:

* **Customer Management:** The system provides comprehensive tools for managing customer data, including account creation, KYC (Know Your Customer) verification, and profile management.
* **Transaction Processing:** BMS automates the processing of deposits, withdrawals, transfers, and payments, ensuring real-time transaction updates and reducing the possibility of errors.
* **Loan and Credit Management:** The system manages loan applications, approvals, disbursements, and repayments, along with credit scoring and risk assessment.
* **Account Management:** Customers can open and manage various types of accounts, such as savings, checking, and fixed deposits, with detailed account statements and summaries.
* **Security Features:** The BMS incorporates advanced security measures, including multi-factor authentication, encryption, and fraud detection algorithms, ensuring the safety of customer data and financial transactions.
* **Reporting and Analytics:** Bank administrators can generate detailed reports on transactions, customer behavior, loan performance, and other critical metrics, aiding in decision-making and regulatory compliance.

As technology continues to evolve, the future of Bank Management Systems looks promising. The integration of artificial intelligence (AI) and machine learning (ML) into BMS platforms is expected to revolutionize banking operations. AI-driven chatbots and virtual assistants will enhance customer service, while predictive analytics will provide banks with deeper insights into customer behavior and market trends.

**OBJECTIVES**

**1.** **Enhance Operational Efficiency**

* Automation of Processes: Automate routine banking tasks such as account management, transaction processing, and customer service to reduce manual effort and improve speed.
* Streamline Workflows: Simplify and optimize internal workflows, ensuring that banking operations are conducted in an efficient, organized manner.

**2. Improve Customer Experience**

* User-Friendly Interface: Provide a simple, intuitive interface for customers to access banking services, including online and mobile banking.
* 24/7 Accessibility: Enable customers to perform banking activities at any time and from any location, enhancing convenience and satisfaction.

**3. Ensure Data Security and Integrity**

* Protect Sensitive Information: Implement robust security measures such as encryption, multi-factor authentication, and secure access controls to safeguard customer data and financial transactions.

**4. Enhance Decision-Making Capabilities**

* Real-Time Reporting: Provide real-time data analytics and reporting tools to help bank managers make informed decisions based on up-to-date information.
* Customer Insights: Analyze customer behavior and transaction patterns to identify trends, preferences, and potential risks, aiding in strategic planning.

**5. Support Financial Management.**

* Loan and Credit Management: Efficiently manage loan applications, disbursements, repayments, and credit assessments, reducing the risk of default and enhancing credit control.

**TOOLS AND ENVIORNMENT**

**HARDWARE REQUIREMENTS**

**Processor:** Minimum Pentium IV 2.4 GHZ

**RAM:** At Least 100 MB

**Disk Space:** At Least 500 MB

# **SOFTWARE REQUIREMENTS**

**Operating System:** Windows,IOS,LINUX,Etc.

**Code Compiler :** Visual Code Studio / Dev C++/ Turbo C++/Etc.

**ENTITY RELATIONSHIP DIAGRAM**

**HAS**

**BANK**

**ACCOUNTS**

**DATA FLOW DIAGRAM**

**BANK MANAGEMENT SYSTEM**

ADD DEPOSIT

CLOSE ACCOUNT

**ADMIN**

WITHDRAW

**CUSTOMER**

ADD ACCOUNT VIEW BALANCE

VIEW TRANSACTION

DATA DATA

**DATABASE**

**PROGRAM CODE**

#include <iostream>

#include <string>

#include <fstream>

#include <cstdio> // Include for sprintf

using namespace std;

const int MAX\_ACCOUNTS = 100;

const int MAX\_TRANSACTIONS = 10; // Maximum number of transactions to store

class Account {

public:

string account\_number;

string name;

double balance;

string transactions[MAX\_TRANSACTIONS]; // Array to store transaction history

int transaction\_count;

Account() {

account\_number = "";

name = "";

balance = 0;

transaction\_count = 0;

}

void create\_account(string acc\_number, string acc\_name, double initial\_deposit) {

account\_number = acc\_number;

name = acc\_name;

balance = initial\_deposit;

add\_transaction("Account created with initial deposit: ", initial\_deposit);

}

void deposit(double amount) {

balance += amount;

add\_transaction("Deposited: ", amount);

}

void withdraw(double amount) {

if (amount > balance) {

cout << "Insufficient balance!" << endl;

} else {

balance -= amount;

add\_transaction("Withdrew: ", amount);

}

}

void display\_balance() {

cout << "Account balance for " << name << ": " << balance << endl;

}

void display\_transactions() {

cout << "Transaction history for " << name << " (" << account\_number << "):" << endl;

for (int i = 0; i < transaction\_count; i++) {

cout << transactions[i] << endl;

}

}

void load\_from\_file(ifstream& file) {

file >> account\_number >> name >> balance;

file >> transaction\_count;

file.ignore(); // Ignore the newline character after the transaction count

for (int i = 0; i < transaction\_count; i++) {

getline(file, transactions[i]);

}

}

void save\_to\_file(ofstream& file) {

file << account\_number << " " << name << " " << balance << endl;

file << transaction\_count << endl;

for (int i = 0; i < transaction\_count; i++) {

file << transactions[i] << endl;

}

}

private:

void add\_transaction(const char\* action, double amount) {

if (transaction\_count == MAX\_TRANSACTIONS) {

// Shift all transactions up if the array is full

for (int i = 1; i < MAX\_TRANSACTIONS; i++) {

transactions[i - 1] = transactions[i];

}

transaction\_count--;

}

char buffer[100];

sprintf(buffer, "%s%.2f", action, amount);

transactions[transaction\_count] = buffer;

transaction\_count++;

}

};

class Bank {

public:

Account accounts[MAX\_ACCOUNTS];

int num\_accounts;

Bank() {

num\_accounts = 0;

load\_accounts();

}

~Bank() {

save\_accounts();

}

void create\_account(string account\_number, string name, double initial\_deposit) {

if (num\_accounts >= MAX\_ACCOUNTS) {

cout << "Bank is full, cannot create more accounts!" << endl;

return;

}

accounts[num\_accounts].create\_account(account\_number, name, initial\_deposit);

num\_accounts++;

cout << "Account created successfully!" << endl;

}

void deposit(string account\_number, double amount) {

for (int i = 0; i < num\_accounts; i++) {

if (accounts[i].account\_number == account\_number) {

accounts[i].deposit(amount);

cout << "Deposit successful! New balance: " << accounts[i].balance << endl;

return;

}

}

cout << "Account not found!" << endl;

}

void withdraw(string account\_number, double amount) {

for (int i = 0; i < num\_accounts; i++) {

if (accounts[i].account\_number == account\_number) {

accounts[i].withdraw(amount);

cout << "Withdrawal successful! New balance: " << accounts[i].balance << endl;

return;

}

}

cout << "Account not found!" << endl;

}

void enquire\_balance(string account\_number) {

for (int i = 0; i < num\_accounts; i++) {

if (accounts[i].account\_number == account\_number) {

accounts[i].display\_balance();

return;

}

}

cout << "Account not found!" << endl;

}

void close\_account(string account\_number) {

for (int i = 0; i < num\_accounts; i++) {

if (accounts[i].account\_number == account\_number) {

accounts[i].account\_number = "";

accounts[i].name = "";

accounts[i].balance = 0.0;

for (int j = i; j < num\_accounts - 1; j++) {

accounts[j] = accounts[j + 1];

}

num\_accounts--;

cout << "Account closed successfully!" << endl;

return;

}

}

cout << "Account not found!" << endl;

}

void show\_customer\_menu() {

system("cls");

cout << endl << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Customer Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "1. Deposit" << endl;

cout << "2. Withdraw" << endl;

cout << "3. Balance Inquiry" << endl;

cout << "4. Display Transaction History" << endl;

cout << "5. Exit" << endl;

}

void show\_admin\_menu() {

system("cls");

cout << endl << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Admin Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "1. Create Account" << endl;

cout << "2. Close Account" << endl;

cout << "3. Exit" << endl;

}

private:

void load\_accounts() {

ifstream file("accounts1.txt");

if (file.is\_open()) {

file >> num\_accounts;

for (int i = 0; i < num\_accounts; i++) {

accounts[i].load\_from\_file(file);

}

file.close();

} else {

cout << "Unable to open accounts file for reading." << endl;

}

}

void save\_accounts() {

ofstream file("accounts1.txt");

if (file.is\_open()) {

file << num\_accounts << endl;

for (int i = 0; i < num\_accounts; i++) {

accounts[i].save\_to\_file(file);

}

file.close();

} else {

cout << "Unable to open accounts file for writing." << endl;

}

}

};

int main() {

Bank bank;

string admin\_id;

string admin\_password;

string customer\_mobile\_no;

string customer\_password;

int choice;

string account\_number, name, id, password, mobile\_no;

double amount;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Welcome to the Bank Management System\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "1. Login as Customer" << endl;

cout << "2. Login as Admin" << endl;

cout << "Enter your choice: ";

cin >> choice;

if (choice == 1) {

do {

cout << endl << "Enter your mobile number: ";

cin >> customer\_mobile\_no;

if (customer\_mobile\_no.length() != 10) {

cout << "Mobile number must be exactly 10 digits long. Please try again." << endl;

}

} while (customer\_mobile\_no.length() != 10);

cout << "Enter your password: ";

cin >> customer\_password;

if (customer\_mobile\_no == "8595632225" && customer\_password == "209") {

do {

bank.show\_customer\_menu();

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter account number: ";

cin >> account\_number;

cout << "Enter amount to deposit: ";

cin >> amount;

bank.deposit(account\_number, amount);

break;

case 2:

cout << "Enter account number: ";

cin >> account\_number;

cout << "Enter amount to withdraw: ";

cin >> amount;

bank.withdraw(account\_number, amount);

break;

case 3:

cout << "Enter account number: ";

cin >> account\_number;

bank.enquire\_balance(account\_number);

break;

case 4:

cout << "Enter account number: ";

cin >> account\_number;

for (int i = 0; i < bank.num\_accounts; i++) {

if (bank.accounts[i].account\_number == account\_number) {

bank.accounts[i].display\_transactions();

break;

}

}

break;

case 5:

cout << "Exiting..." << endl;

return 0;

default:

cout << "Invalid choice! Please try again." << endl;

break;

}

} while (true);

} else {

cout << "Invalid mobile number or password!" << endl;

}

} else if (choice == 2) {

cout << endl << "Enter admin ID: ";

cin >> admin\_id;

cout << "Enter admin password: ";

cin >> admin\_password;

if (admin\_id == "shubham" && admin\_password == "209") {

do {

bank.show\_admin\_menu();

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter account number: ";

cin >> account\_number;

cout << "Enter account holder name: ";

cin.ignore();

getline(cin, name);

cout << "Enter initial deposit: ";

cin >> amount;

bank.create\_account(account\_number, name, amount);

break;

case 2:

cout << "Enter account number: ";

cin >> account\_number;

bank.close\_account(account\_number);

break;

case 3:

cout << "Exiting..." << endl;

return 0;

default:

cout << "Invalid choice! Please try again." << endl;

break;

}

} while (true);

} else {

cout << "Invalid admin ID or password!" << endl;

}

} else {

cout << "Invalid choice! Exiting..." << endl;

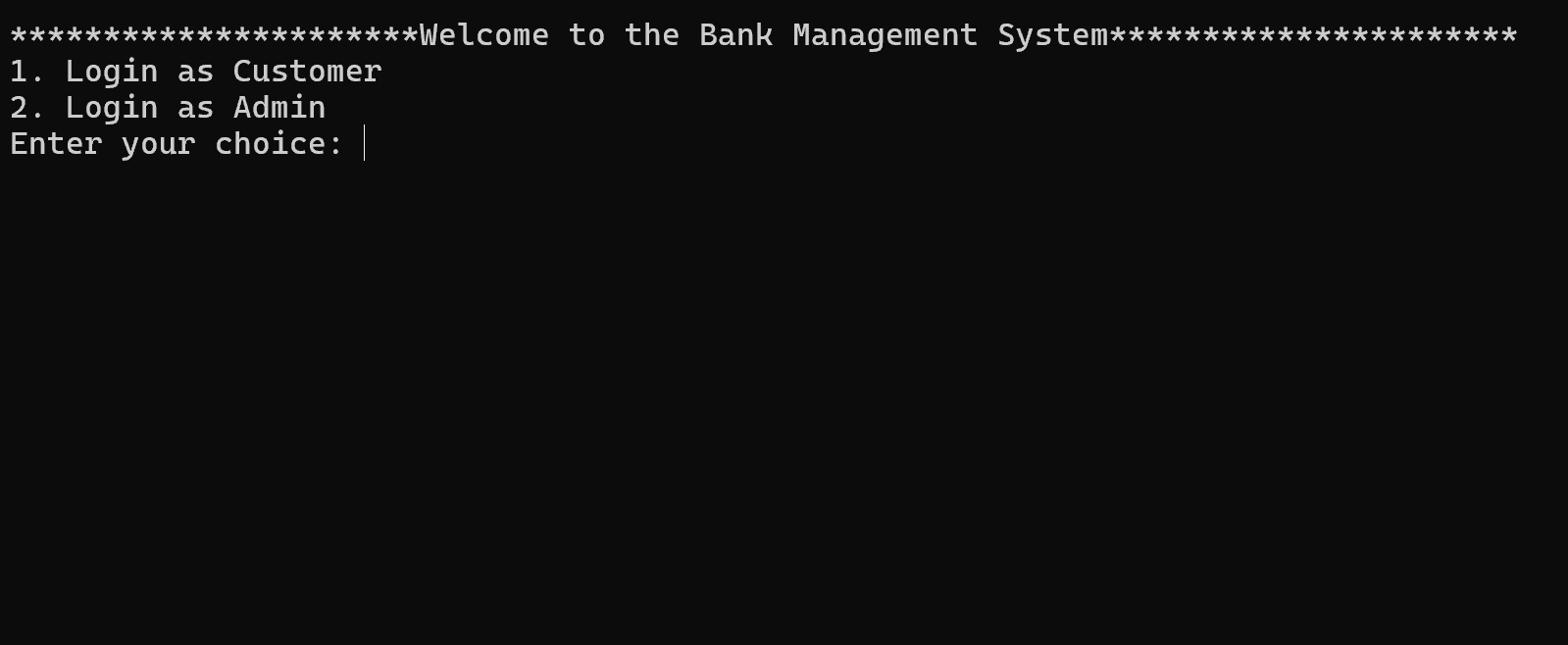
}

return 0;

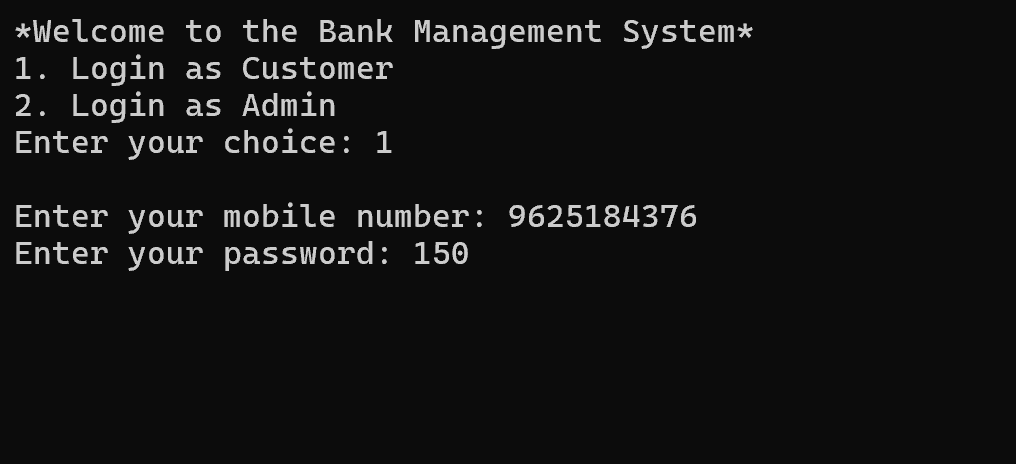
}

**INPUT / OUTPUT SCREENS**

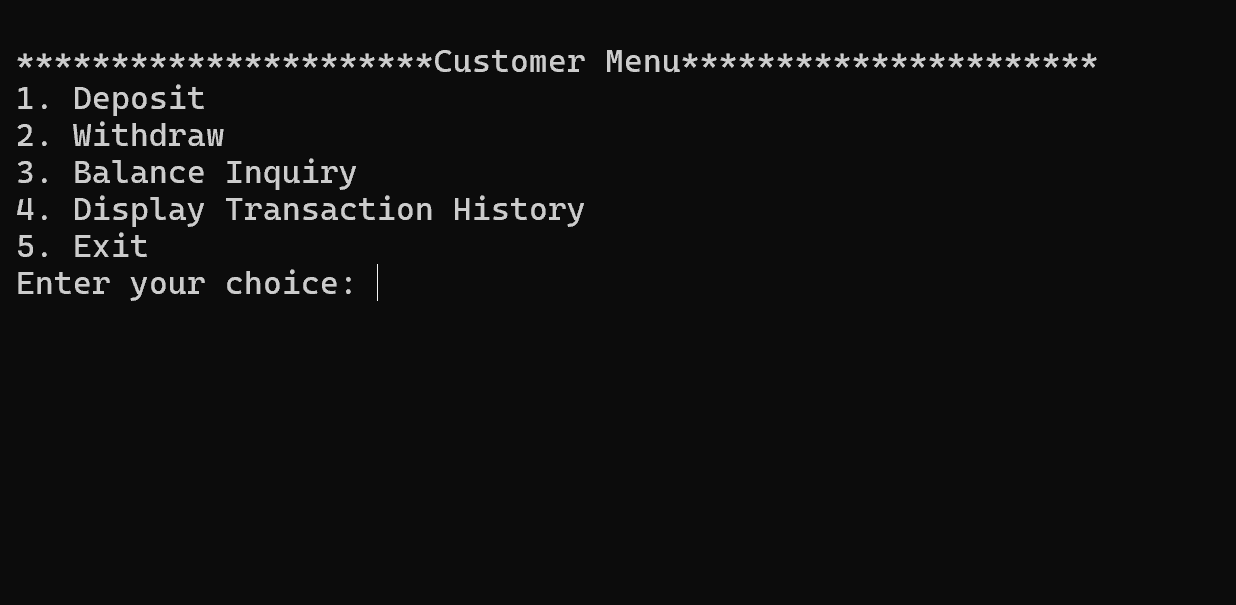
MAIN MENU



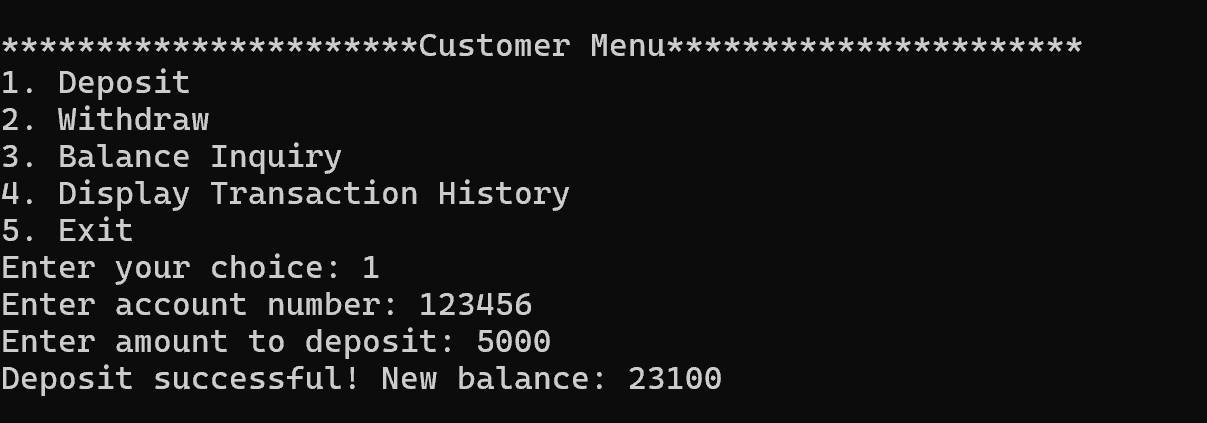
CUSTOMER LOGIN



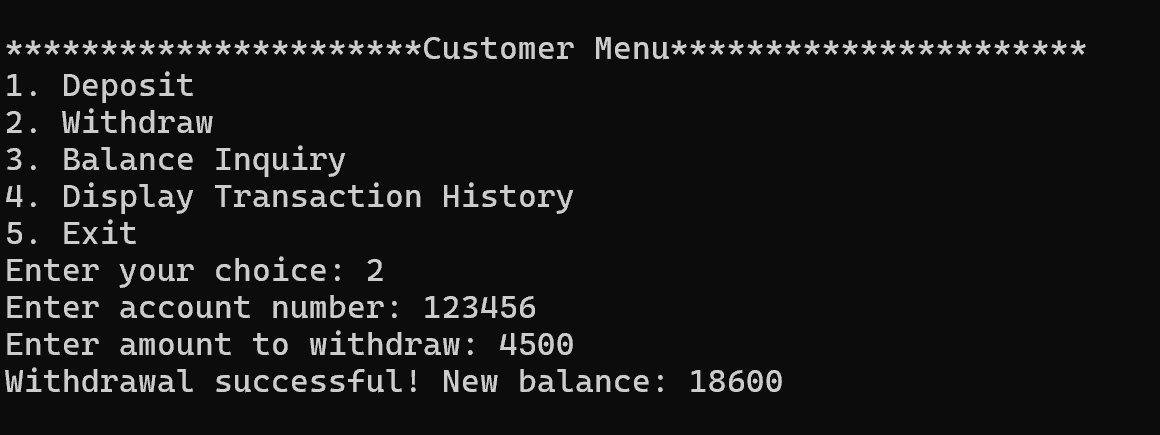
CUSTOMER MENU



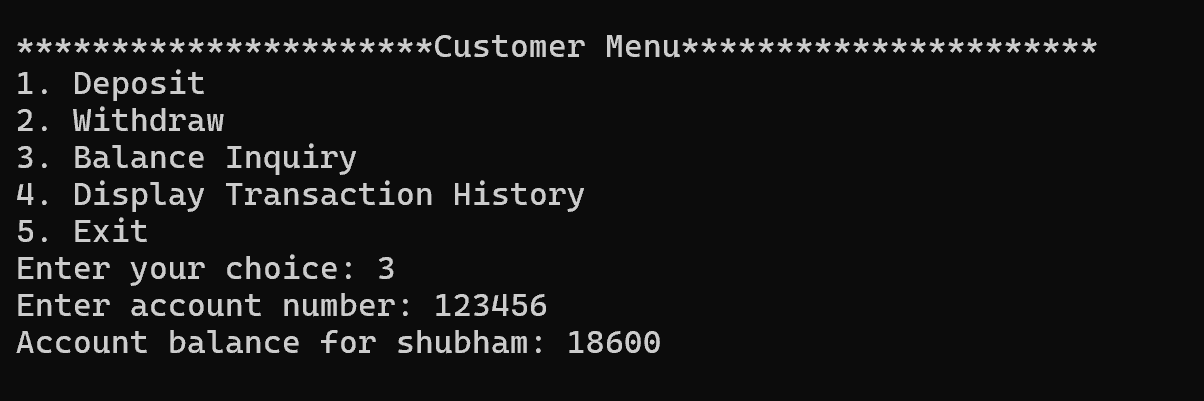
DEPOSIT



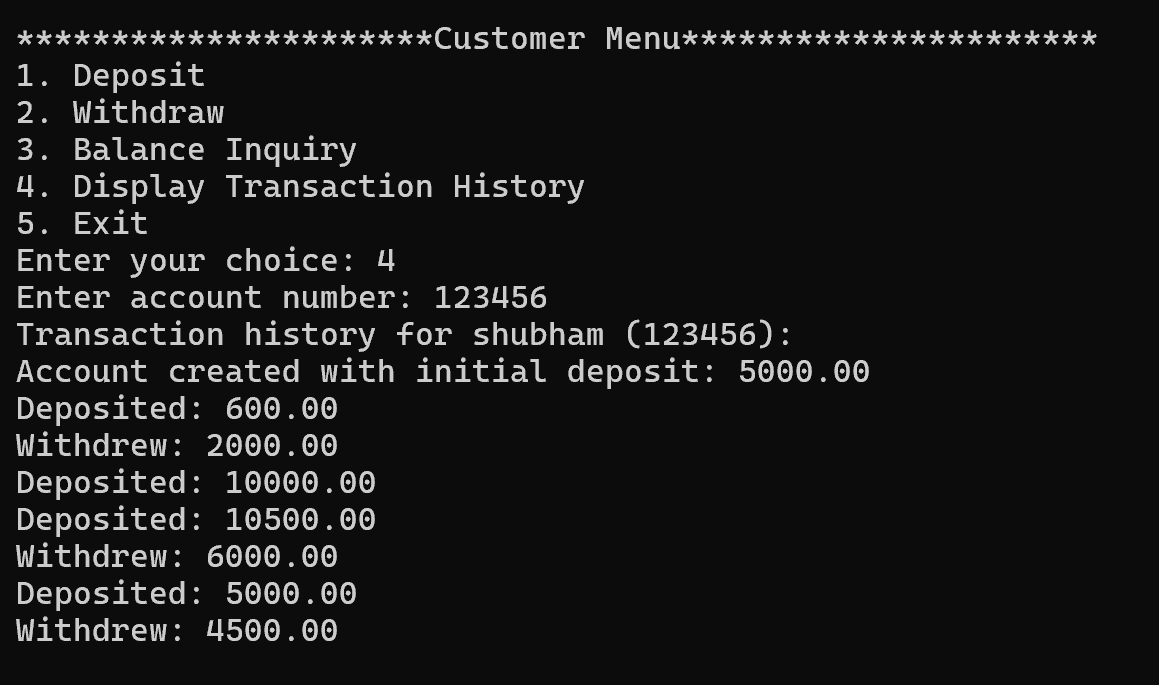
WITHDRAW



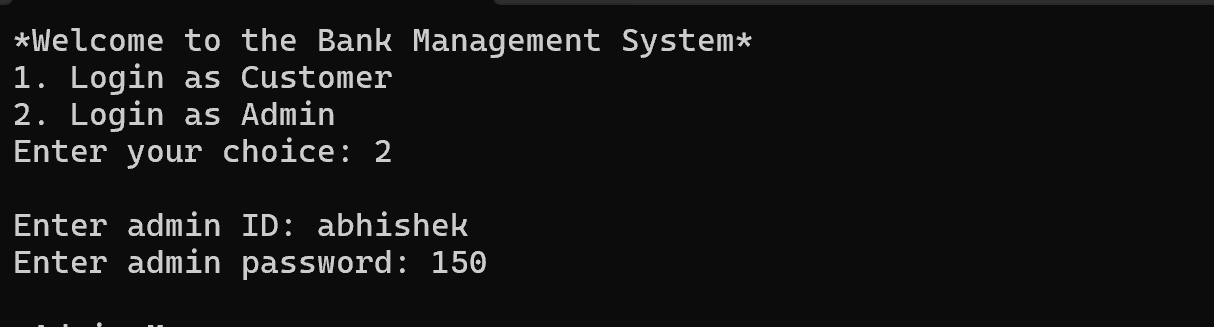
BALANCE ENQUIRY



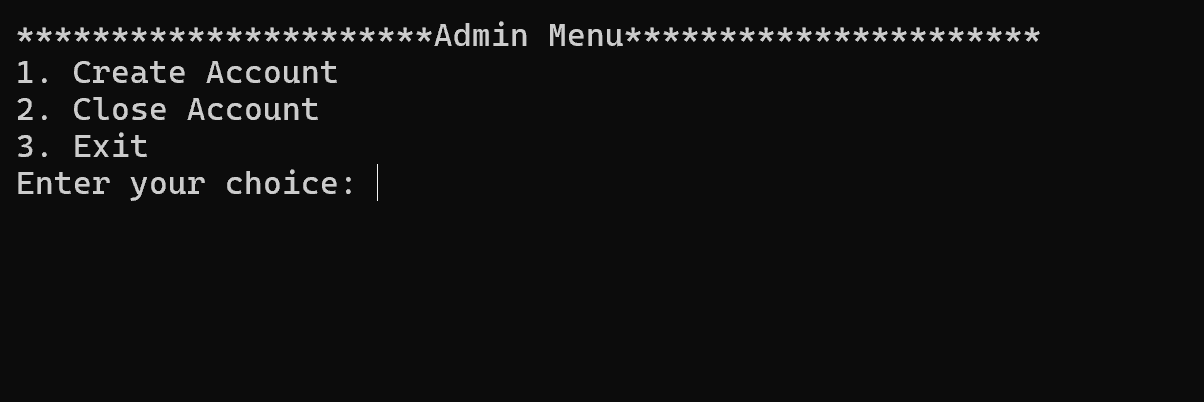
TRANSACTION HISTORY



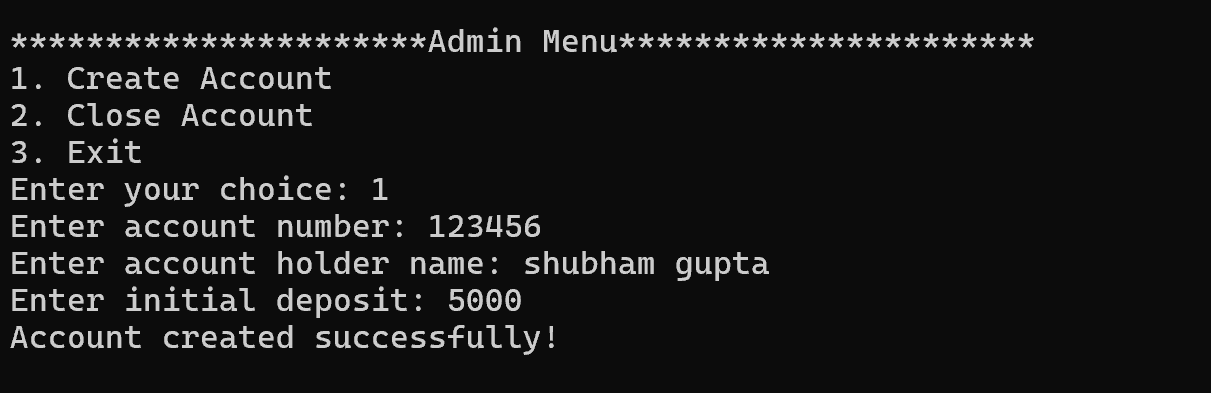
ADMIN ID AND PASSWORD



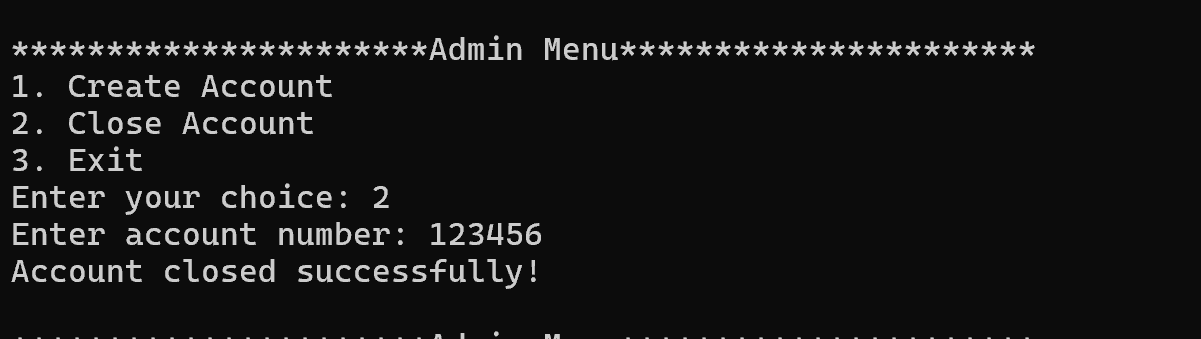
ADMIN MENU



ACCOUNT CREATION



CLOSE ACCOUNT



**LIMITATIONS OF BANK MANAGEMENT SYSTEM**

**1. Security Vulnerabilities**

* **Cybersecurity Threats:** BMS is susceptible to cyber-attacks, including hacking, phishing, and malware. If not properly secured, sensitive financial data can be compromised.
* **Data Breaches:** Unauthorized access to the system can lead to data breaches, exposing customer information and potentially leading to financial losses and reputational damage.

**2. System Downtime**

* **Technical Failures:** The system may experience technical issues, leading to downtime. This can disrupt banking operations, causing delays in transactions and customer dissatisfaction.
* **Maintenance Requirements:** Regular maintenance and updates are necessary to keep the system running smoothly. During maintenance, the system may be temporarily unavailable.

**3. High Initial Cost**

* **Development and Implementation:** Developing and implementing a BMS can be expensive, especially for smaller banks. The costs include software development, hardware procurement, and training.
* **Ongoing Costs:** In addition to the initial investment, there are ongoing costs for system maintenance, updates, and security measures.

**4. Complexity**

* **User Training:** The system can be complex, requiring extensive training for staff to use it effectively. This can be time-consuming and costly.
* **Integration Challenges:** Integrating the BMS with existing systems or third-party applications can be challenging, leading to potential compatibility issues.

**5. Data Privacy Concerns**

* **Regulatory Compliance:** Banks must ensure that the BMS complies with various data protection regulations. Failure to comply can result in legal penalties and loss of customer trust.
* **Data Ownership:** Issues related to data ownership and control can arise, especially when third-party vendors are involved in managing the BMS.

**FUTURE APPLICATION OF THE PROJECT**

The future applications of a Bank Management System (BMS) are driven by advancements in technology, evolving customer expectations, and the need for more efficient and secure banking operations. Here are some potential future applications of BMS:

**1. AI and Machine Learning Integration**

* **Personalized Banking Services:** AI can be used to analyze customer data and provide personalized banking experiences, such as customized loan offers, investment advice, and spending recommendations.
* **Fraud Detection:** Machine learning algorithms can enhance fraud detection by analyzing transaction patterns and identifying suspicious activities in real-time.

**2. Blockchain and Cryptocurrencies**

* **Secure Transactions:** Blockchain technology can be integrated into BMS to ensure more secure, transparent, and tamper-proof transactions.
* **Cryptocurrency Management:** As digital currencies become more mainstream, BMS could support cryptocurrency transactions, storage, and management.

**3. Robotic Process Automation (RPA)**

* **Operational Efficiency:** RPA can automate repetitive tasks such as data entry, account reconciliation, and compliance reporting, reducing errors and freeing up staff for more strategic activities.
* **Loan Processing:** Automation can streamline loan processing, reducing approval times and improving customer satisfaction.

**4. Enhanced Mobile Banking**

* **Biometric Authentication:** Future BMS may incorporate advanced biometric authentication methods, such as facial recognition or voice identification, to enhance security and user experience in mobile banking.
* **Augmented Reality (AR) Banking:** AR could be used to provide interactive, immersive banking experiences, such as virtual branch visits or real-time financial data visualization.

**5. Open Banking and APIs**

* **Integration with Third-Party Services:** Open banking will allow BMS to integrate with third-party financial services via APIs, offering customers more comprehensive financial management tools and services.

**BIBLIOGRAPHY**

**Books**

* Let Us C by Yashavant Kanetkar.
* Let us C++ by Yashavant Kanetkar.
* C in Depth by S.K Srivastava.
* The C++ Programming Language By Bjarne Stroustrup.

**Websites**

* [www.google.com](http://www.google.com)
* [www.youtube.com](http://www.youtube.com)
* [www.w3schools.com](http://www.w3schools.com)
* [www.geeksforgeeks.com](http://www.geeksforgeeks.com)