**BANK**

**MANAGEMENT SYSTEM**

**A PROJECT REPORT**

**Submitted By :- Dheeraj Kumar**

**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 transformative approach to banking operations by consolidating various functions into a cohesive platform. This system is engineered to manage the intricate and essential tasks of financial institutions, ensuring optimal efficiency, security, and reliability. By handling customer accounts, transactions, loans, and other critical banking services, the BMS offers a streamlined and intuitive interface for both customers and bank employees, enhancing the overall banking experience.

**Bank Management System**

In today's fast-paced environment, the need for quick and secure financial transactions is greater than ever. Traditional, paper-based banking methods are often inefficient and error-prone. To meet the demands of digital banking, a robust system is required to handle millions of transactions daily while ensuring accuracy and security. A Bank Management System fulfills this need by automating core functions, reducing manual processes, and offering continuous 24/7 service.

****

**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**

CREDIT CARD SALES MANAGEMENT SYSTEM

ADD DEPOSIT

CLOSE ACCOUNT

ADMIN

WITHDRAW

CUSTOMER

ADD ACCOUNT VIEW BALANCE

VIEW TRANSACTION

DATA DATA

DATABASE

**PROGRAM CODE**

#include <iostream>

#include <fstream>

#include <string>

#include <iomanip>

using namespace std;

class Account {

public:

int accNo;

char accHolder[25];

float balance;

void addAcc();

void displayAccount() const;

void getAllAccounts() const;

void updateAcc();

void deleteAcc();

bool isAccountExists(int) const;

};

class Customer : public Account {

public:

void deposit();

void withdraw();

void viewStatement() const;

};

bool Account::isAccountExists(int id) const {

ifstream checkFile("account.dat", ios::binary);

Account a;

while (checkFile.read((char\*)&a, sizeof(a))) {

if (id == a.accNo) {

checkFile.close();

return true;

}

}

checkFile.close();

return false;

}

void Account::addAcc() {

cout << "\nEnter account holder name: ";

cin.ignore();

cin.getline(accHolder, 25);

cout << "Enter account number: ";

while (!(cin >> accNo)) {

cout << "Enter number only: ";

cin.clear();

cin.ignore(123, '\n');

}

if (isAccountExists(accNo)) {

cout << "Error: Account with number " << accNo << " already exists\n";

} else {

cout << "Enter initial balance: ";

while (!(cin >> balance)) {

cout << "Enter number only: ";

cin.clear();

cin.ignore(123, '\n');

}

ofstream file("account.dat", ios::binary | ios::app);

if (file) {

file.write((char\*)this, sizeof(\*this));

file.close();

cout << "\nAccount created successfully...\n";

} else {

cout << "Error: Could not open file to save account data.\n";

}

}

}

void Account::displayAccount() const {

cout << setw(5) << accNo << setw(25) << accHolder << setw(25) << balance << endl;

}

void Account::getAllAccounts() const {

ifstream file("account.dat", ios::binary);

if (!file) {

cerr << "Error: Could not open file.\n";

return;

}

Account a;

bool found = false; // To check if any account exists

while (file.read((char\*)&a, sizeof(a))) {

found = true;

a.displayAccount();

}

if (!found) {

cout << "There is no account.\n";

}

file.close();

}

void Account::updateAcc() {

int id, pos;

bool found = false;

cout << "\nEnter account number to update: ";

cin >> id;

fstream file("account.dat", ios::binary | ios::in | ios::out);

Account a;

while (file.read((char\*)&a, sizeof(a))) {

if (a.accNo == id) {

cout << "Enter new account holder name: ";

cin.ignore();

cin.getline(a.accHolder, 25);

cout << "Enter new balance: ";

while (!(cin >> a.balance)) {

cout << "Enter number only: ";

cin.clear();

cin.ignore(123, '\n');

}

pos = -1 \* static\_cast<int>(sizeof(a));

file.seekp(pos, ios::cur);

file.write((char\*)&a, sizeof(a));

found = true;

cout << "\nAccount updated successfully...\n";

break;

}

}

if (!found) {

cout << "Invalid Account Number\n";

}

file.close();

}

void Account::deleteAcc() {

int id;

bool found = false;

cout << "\nEnter account number to delete: ";

cin >> id;

ifstream file("account.dat", ios::binary);

ofstream tempFile("temp.dat", ios::binary);

Account a;

while (file.read((char\*)&a, sizeof(a))) {

if (a.accNo != id) {

tempFile.write((char\*)&a, sizeof(a));

} else {

found = true;

}

}

file.close();

tempFile.close();

remove("account.dat");

rename("temp.dat", "account.dat");

if (found) {

cout << "\nAccount deleted successfully...\n";

} else {

cout << "Invalid Account Number\n";

}

}

void Customer::deposit() {

int id;

bool found = false;

cout << "Enter account number: ";

cin >> id;

fstream file("account.dat", ios::binary | ios::in | ios::out);

Account a;

while (file.read((char\*)&a, sizeof(a))) {

if (a.accNo == id) {

found = true;

float amount;

cout << "Enter amount to deposit: ";

while (!(cin >> amount)) {

cout << "Enter number only: ";

cin.clear();

cin.ignore(123, '\n');

}

a.balance += amount;

int pos = -1 \* static\_cast<int>(sizeof(a));

file.seekp(pos, ios::cur);

file.write((char\*)&a, sizeof(a));

cout << "Deposit successful. New balance: " << a.balance << endl;

break;

}

}

if (!found) {

cout << "Invalid Account Number\n";

}

file.close();

}

void Customer::withdraw() {

int id;

bool found = false;

cout << "Enter account number: ";

cin >> id;

fstream file("account.dat", ios::binary | ios::in | ios::out);

Account a;

while (file.read((char\*)&a, sizeof(a))) {

if (a.accNo == id) {

found = true;

float amount;

cout << "Enter amount to withdraw: ";

while (!(cin >> amount)) {

cout << "Enter number only: ";

cin.clear();

cin.ignore(123, '\n');

}

if (a.balance >= amount) {

a.balance -= amount;

int pos = -1 \* static\_cast<int>(sizeof(a));

file.seekp(pos, ios::cur);

file.write((char\*)&a, sizeof(a));

cout << "Withdrawal successful. New balance: " << a.balance << endl;

} else {

cout << "Insufficient balance\n";

}

break;

}

}

if (!found) {

cout << "Invalid Account Number\n";

}

file.close();

}

void Customer::viewStatement() const {

int id;

bool found = false;

cout << "Enter account number: ";

cin >> id;

ifstream file("account.dat", ios::binary);

Account a;

while (file.read((char\*)&a, sizeof(a))) {

if (a.accNo == id) {

found = true;

cout << "Account Statement:\n";

cout << "Account Number: " << a.accNo << endl;

cout << "Account Holder: " << a.accHolder << endl;

cout << "Balance: " << a.balance << endl;

break;

}

}

if (!found) {

cout << "Invalid Account Number\n";

}

file.close();

}

void adminMenu() {

int choice;

Account account;

do {

cout << "\n\*\*\*\*\*\*\*\*\*\*\*|| ADMIN MENU ||\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "1. Create Account\n";

cout << "2. Update Account\n";

cout << "3. Delete Account\n";

cout << "4. View All Accounts\n";

cout << "5. Logout\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

account.addAcc();

break;

case 2:

account.updateAcc();

break;

case 3:

account.deleteAcc();

break;

case 4:

account.getAllAccounts();

break;

case 5:

cout << "Logging out...\n";

break;

default:

cout << "Invalid choice\n";

}

} while (choice != 5);

}

void customerMenu() {

int choice;

Customer customer;

do {

cout << "\n\*\*\*\*\*\*\*\*\*\*\*|| CUSTOMER MENU ||\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "1. Deposit\n";

cout << "2. Withdraw\n";

cout << "3. View Statement\n";

cout << "4. Logout\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

customer.deposit();

break;

case 2:

customer.withdraw();

break;

case 3:

customer.viewStatement();

break;

case 4:

cout << "Logging out...\n";

break;

default:

cout << "Invalid choice\n";

}

} while (choice != 4);

}

void login() {

int choice;

string password;

do {

cout << "\n\*\*\*\*\*\*\*\*\*\*\*|| BANK MANAGEMENT SYSTEM ||\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "1. Admin Login\n";

cout << "2. Customer Login\n";

cout << "3. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1: // Admin Login

cout << "Enter Admin Password: ";

cin >> password;

if (password == "dheeraj167") {

adminMenu();

} else {

cout << "Invalid password\n";

}

break;

case 2: // Customer Login

customerMenu();

break;

case 3: // Exit

cout << "Exiting...\n";

break;

default:

cout << "Invalid choice\n";

}

} while (choice != 3);

}

int main() {

login(); // Start with the login screen

return 0;

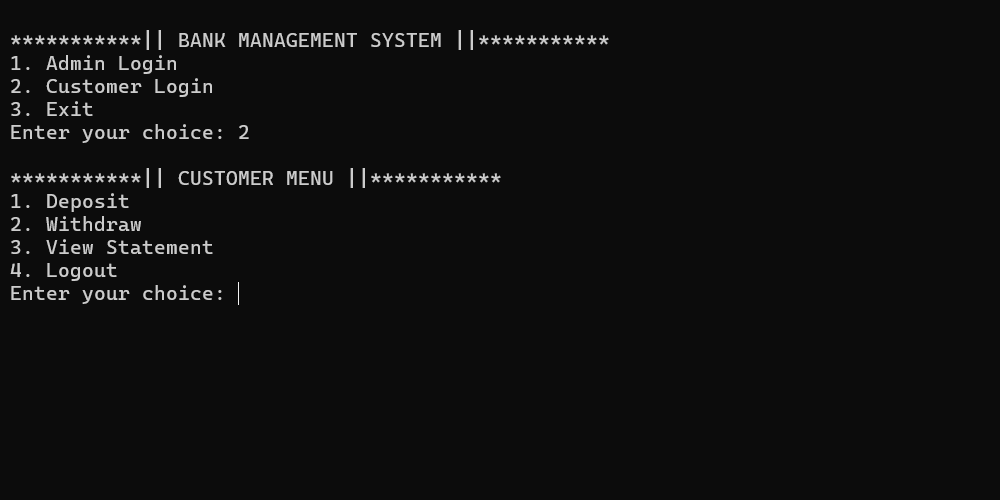
}

**INPUT / OUTPUT SCREENS**

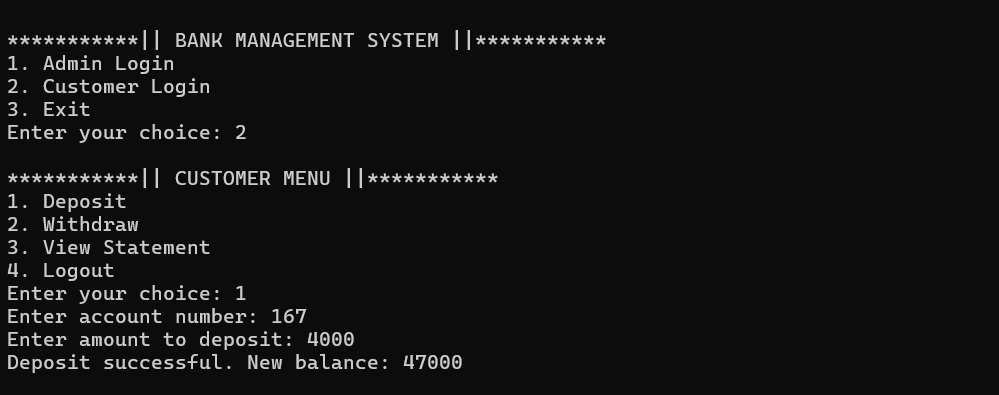
MAIN MENU



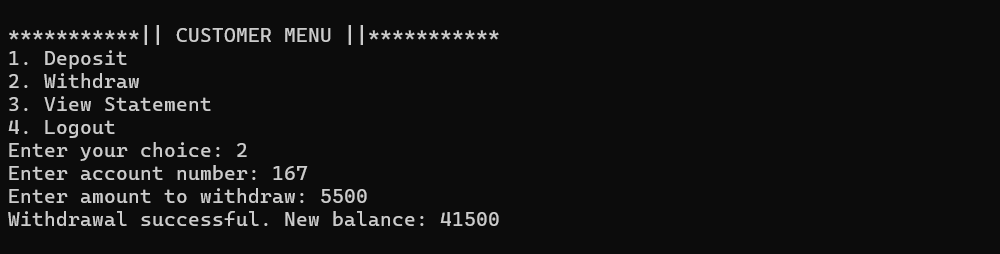
CUSTOMER LOGIN / CUSTOMER MENU



DEPOSIT



WITHDRAW



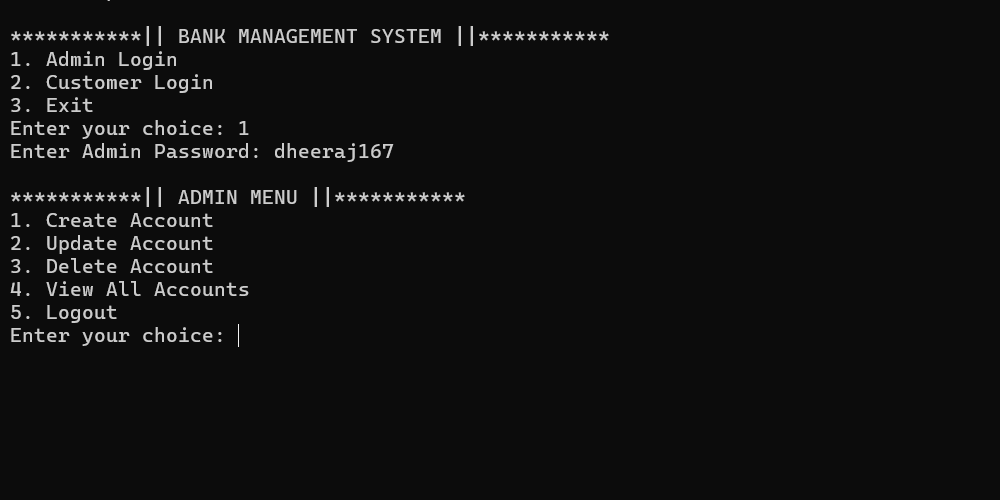
VIEW STATEMENT



LOGGING OUT AS A CUSTOMER



ADMIN LOGIN / ADMIN MENU



CREATING AN ACCOUNT

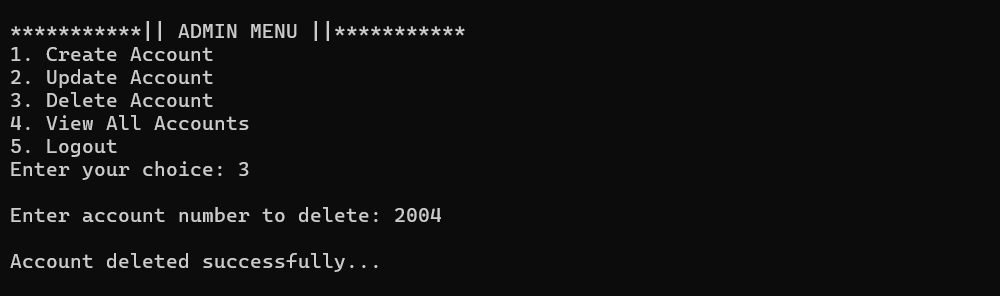
VIEW ALL ACCOUNTS



UPDATING AN ACCOUNT



DELETE 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)