**BANK MANAGEMENT SYSTEM**

**A PROJECT REPORT**

Submitted by : - vishwas

Date :-10-8-2024

**TABLE OF CONTENT**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **TOPIC** | **PAGE NO** |
| 1 | Introduction | **3-4** |
| 2 | Objective | **5** |
| 3 | Tools/Environment | **8** |
| 4 | Program code | **9-17** |
| 5 | Input and output screen | **18-20** |
| 6 | Limitations of the project | **21-22** |
| 7 | Future applications of this project | **23-24** |
| 8 | biblography | **25** |

**INTRODUCTION**

A Bank Management System is a software application designed to manage various banking operations in a simple and efficient manner. Typically, this system is used to perform tasks such as creating and managing bank accounts, processing deposits and withdrawals, tracking account balances, and generating reports on account activity.



Key Features:

1. Account Creation: The system allows users to create new bank accounts by entering details such as account number, account holder's name, and initializing the balance.
2. Deposit and Withdrawal: Users can deposit money into their accounts or withdraw money from them. The system automatically updates the account balance after each transaction.
3. Balance Inquiry: Users can check the current balance of their accounts at any time. This feature displays account details, including the balance, in a clear and concise manner.
4. Account Management: The system stores all account details in a file, making it easy to access, update, and manage multiple accounts.
5. Display All Accounts: The system can generate a list of all accounts, showing essential information like account numbers, account holders' names, and their balances.

Advantages:

* Efficiency: Automates routine banking tasks, reducing the time and effort required to manage accounts.
* Accuracy: Minimizes human errors in financial calculations and record-keeping.
* User-Friendly: Designed with a simple interface, making it accessible even to users with minimal technical expertise.
* Data Management: Utilizes basic file handling to store and retrieve account information, ensuring data persistence.

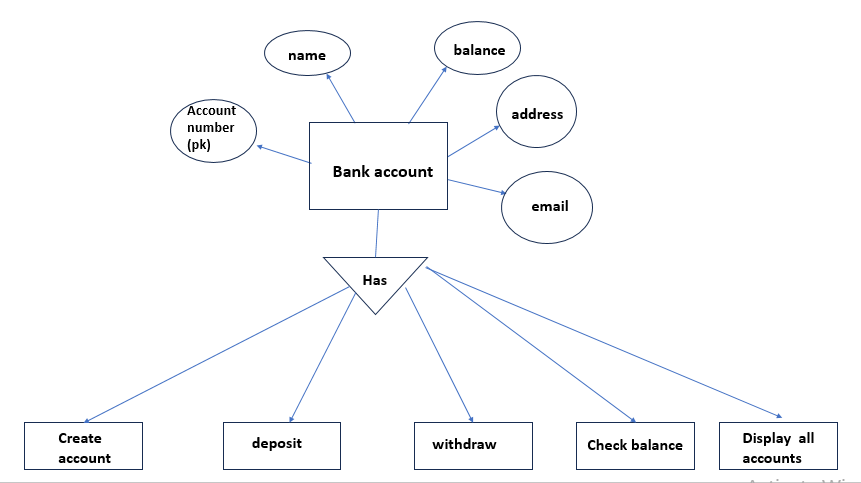
**OBJECTIVE**

The primary objective of the Bank Management System project is to develop a simple and efficient software application that can manage basic banking operations. This system aims to automate routine tasks involved in bank account management, ensuring accuracy and ease of use. Specifically, the objectives include:

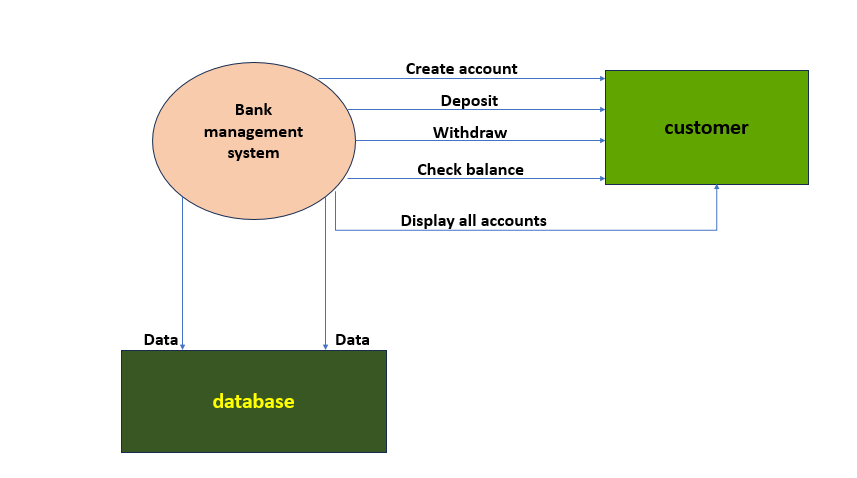
1. Account Creation: To provide a straightforward mechanism for creating new bank accounts, where users can input essential details like account numbers and names, and the system stores this information securely.
2. Transaction Handling: To facilitate seamless deposit and withdrawal transactions, updating account balances accurately and instantly, while maintaining records of all operations.
3. Balance Inquiry: To enable users to check their account balance at any time, displaying the information clearly and with precise formatting.
4. Data Persistence: To implement basic file handling techniques that allow the system to store, retrieve, and update account information efficiently, ensuring data remains consistent and accessible.
5. User-Friendly Interface: To design a simple and intuitive console-based interface that allows users, especially beginners, to interact with the system without requiring advanced technical knowledge.
6. Educational Purpose: To serve as a learning tool for students to understand the fundamentals of banking operations, file handling, and basic C++ programming, thereby reinforcing key concepts in software development.

The overall goal is to create a reliable and accessible system that meets the essential needs of banking management while providing an educational platform for learning and practicing C++ programming.

**ER DIAGRAM**



**Data flow diagram**



**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.

**PROGRAM CODE**

#include <iostream>

#include <fstream>

using namespace std;

// Account structure

struct Account {

int accountNumber;

char name[100];

float balance;

char address[100];

char phoneNumber[50];

char email[100];

};

// Function prototypes

void createAccount();

void deposit();

void withdraw();

void checkBalance();

void displayAllAccounts();

void showMenu();

int main() {

showMenu();

return 0;

}

void showMenu() {

int choice;

do {

cout << "\n--- Bank Management System ---\n";

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

cout << "2. Deposit\n";

cout << "3. Withdraw\n";

cout << "4. Check Balance\n";

cout << "5. Display All Accounts\n";

cout << "6. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1: createAccount(); break;

case 2: deposit(); break;

case 3: withdraw(); break;

case 4: checkBalance(); break;

case 5: displayAllAccounts(); break;

case 6: cout << "Exiting...\n"; break;

default: cout << "Invalid choice. Please try again.\n";

}

} while (choice != 6);

}

void createAccount() {

Account;

ofstream outFile;

cout << "\n--- Create Account ---\n";

cout << "Enter Account Number: ";

cin >> account.accountNumber;

cin.ignore(); // To ignore the newline character left in the buffer

cout << "Enter Name: ";

cin.getline(account.name, 100);

cout << "Enter Address: ";

cin.getline(account.address, 100);

cout << "Enter Phone Number: ";

cin.getline(account.phoneNumber,50);

cout << "Enter Email: ";

cin.getline(account.email, 100);

account.balance = 0.0;

outFile.open("accounts.txt", ios::app);

if (outFile) {

outFile << account.accountNumber << " " << account.name << " " << account.balance << " "

<< account.address << " " << account.phoneNumber << " " << account.email << endl;

outFile.close();

cout << "\t\t\tAccount created successfully!\n";

} else {

cout << "\t\t\tError creating account.\n";

}

}

void deposit() {

int accNum;

float amount;

Account;

ifstream inFile;

ofstream outFile;

cout << "\n--- Deposit ---\n";

cout << "Enter Account Number: ";

cin >> accNum;

cout << "Enter Amount to Deposit: ";

cin >> amount;

inFile.open("accounts.txt");

outFile.open("temp.txt");

bool found = false;

while (inFile >> account.accountNumber >> account.name >> account.balance >> account.address >> account.phoneNumber >> account.email) {

if (account.accountNumber == accNum) {

account.balance += amount;

found = true;

cout << "\t\t\tAmount deposited successfully!\n";

}

outFile << account.accountNumber << " " << account.name << " " << account.balance << " "

<< account.address << " " << account.phoneNumber << " " << account.email << endl;

}

inFile.close();

outFile.close();

if (found) {

remove("accounts.txt");

rename("temp.txt", "accounts.txt");

} else {

cout << "Account not found.\n";

remove("temp.txt");

}

}

void withdraw() {

int accNum;

float amount;

Account account;

ifstream inFile;

ofstream outFile;

cout << "\n--- Withdraw ---\n";

cout << "Enter Account Number: ";

cin >> accNum;

cout << "Enter Amount to Withdraw: ";

cin >> amount;

inFile.open("accounts.txt");

outFile.open("temp.txt");

bool found = false;

while (inFile >> account.accountNumber >> account.name >> account.balance >> account.address >> account.phoneNumber >> account.email) {

if (account.accountNumber == accNum) {

if (account.balance >= amount) {

account.balance -= amount;

found = true;

cout << "\t\t\tAmount withdrawn successfully!\n";

} else {

cout << "Insufficient balance.\n";

}

}

outFile << account.accountNumber << " " << account.name << " " << account.balance << " "

<< account.address << " " << account.phoneNumber << " " << account.email << endl;

}

inFile.close();

outFile.close();

if (found) {

remove("accounts.txt");

rename("temp.txt", "accounts.txt");

} else {

cout << "Account not found.\n";

remove("temp.txt");

}

}

void checkBalance() {

int accNum;

Account account;

ifstream inFile;

cout << "\n--- Check Balance ---\n";

cout << "Enter Account Number: ";

cin >> accNum;

inFile.open("accounts.txt");

bool found = false;

while (inFile >> account.accountNumber >> account.name >> account.balance >> account.address >> account.phoneNumber >> account.email) {

if (account.accountNumber == accNum) {

cout << "Account Number: " << account.accountNumber << endl;

cout << "Name: " << account.name << endl;

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

cout << "Address: " << account.address << endl;

cout << "Phone Number: " << account.phoneNumber << endl;

cout << "Email: " << account.email << endl;

found = true;

break;

}

}

if (!found) {

cout << "Account not found.\n";

}

inFile.close();

}

void displayAllAccounts() {

Account account;

ifstream inFile;

cout << "\n--- All Accounts ---\n";

inFile.open("accounts.txt");

while (inFile >> account.accountNumber >> account.name >> account.balance >> account.address >> account.phoneNumber >> account.email) {

cout << "Account Number: " << account.accountNumber << endl;

cout << "Name: " << account.name << endl;

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

cout << "Address: " << account.address << endl;

cout << "Phone Number: " << account.phoneNumber << endl;

cout << "Email: " << account.email << endl;

cout << "-------------------------\n";

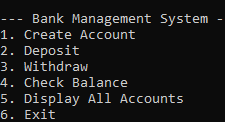
}

inFile.close();

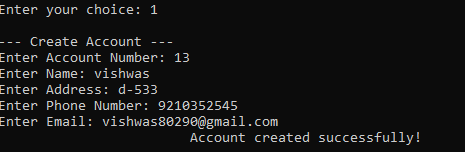
}

**INPUT/OUTPUT SCREEN**

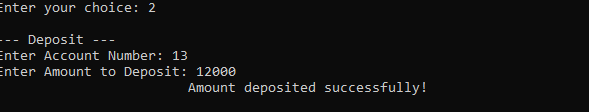
**1** Main menu



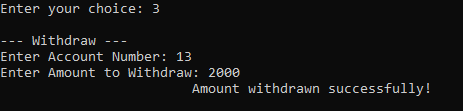
**2** Create account



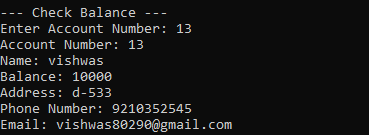
**3** deposit in account



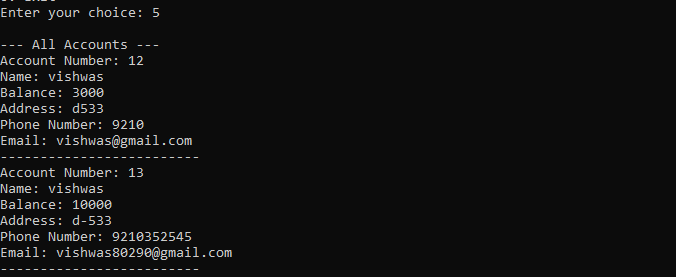
4 withdraw



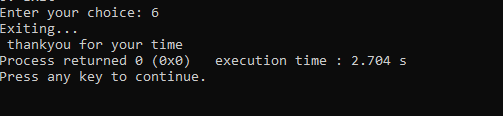
**5** check balance

****

**6** check all accounts



7 exit



**Limitations of this project**

While the Bank Management System provides a simple and functional approach to managing basic banking operations, it has several limitations due to its simplicity and educational purpose:

1. **Limited Functionality**:
   * The system only supports basic operations such as account creation, deposit, withdrawal, and balance inquiry. It lacks more advanced features such as fund transfers, interest calculations, and transaction history tracking.
2. **No User Authentication**:
   * There is no user authentication or security mechanism in place, meaning that anyone with access to the system can view and modify account details without any restrictions.
3. **No Error Handling for Invalid Inputs**:
   * The system does not robustly handle invalid inputs (e.g., entering non-numeric characters for account numbers or amounts), which could lead to unexpected behavior or crashes.
4. **No Real-Time Data Processing**:
   * The system relies on simple file handling, which means it does not support real-time data processing or multi-user access. Multiple users cannot interact with the system simultaneously without risking data inconsistencies.
5. **No Database Integration**:
   * The system uses basic text files to store data, which is not as secure or efficient as using a database. This limits scalability, data retrieval speed, and the ability to manage large volumes of data.
6. **Lack of Data Validation**:
   * The system does not perform comprehensive validation checks on user inputs, such as ensuring that an account number is unique or that a name does not contain invalid characters.
7. **Simplistic Interface**:
   * The console-based interface is very basic and may not be user-friendly for non-technical users. It lacks a graphical user interface (GUI) that could make the system more accessible and visually appealing.
8. **No Record of Transactions**:
   * The system does not maintain a detailed transaction history for each account, meaning users cannot review past transactions or monitor account activity over time.
9. **Data Integrity Issues**:
   * Since the system uses simple file handling techniques, there is a risk of data corruption or loss if the file is improperly accessed or modified. There is also no backup or recovery mechanism.
10. **Scalability**:
    * The system is not designed to handle a large number of accounts or complex banking operations, making it unsuitable for use in a real-world banking environment.

**Future applications of this project**

While the current version of the Bank Management System is a basic educational tool, it can be expanded and enhanced to serve more advanced purposes. Here are some potential future applications:

1. **Enhanced Educational Tool**:
   * The system can be further developed into a more comprehensive learning platform by incorporating additional banking features, such as loan management, interest calculations, and account transfers. This would provide students with a deeper understanding of banking operations and more complex programming concepts.
2. **Small-Scale Banking Solutions**:
   * With improvements, such as adding security features, a graphical user interface (GUI), and database integration, the system could be used by small financial institutions, cooperatives, or microfinance organizations to manage their operations efficiently.
3. **Personal Finance Management**:
   * The project can be adapted into a personal finance management tool, allowing users to track their expenses, manage savings, and set financial goals. It could include features like budgeting tools, spending analysis, and notifications for low balances or upcoming bills.
4. **Mobile Banking App**:
   * By transforming the project into a mobile application, it could serve as a simple banking or financial management app for users who need to manage their accounts on the go. Integration with mobile payment systems and online banking could be added to enhance its functionality.
5. **Banking Simulation Software**:
   * The project could be expanded into a banking simulation game or software used in educational institutions to teach students about financial literacy, banking operations, and decision-making in a controlled, risk-free environment.
6. **Integration with E-Commerce**:
   * The system could be integrated into an e-commerce platform to handle transactions, process payments, and manage customer accounts. This would involve adding features like payment gateways, transaction histories, and account management for online shoppers.
7. **Online Banking System**:
   * With the addition of internet connectivity and secure protocols, the project could evolve into an online banking system where users can access their accounts, perform transactions, and receive account statements remotely.
8. **AI-Powered Financial Advisor**:
   * The system could be integrated with artificial intelligence (AI) to offer personalized financial advice to users. The AI could analyze user data, provide insights on spending patterns, suggest investment opportunities, and help with financial planning.
9. **Integration with Cryptocurrency**:
   * The project could be extended to support cryptocurrency transactions, allowing users to manage both traditional and digital currencies within the same platform. This would require implementing blockchain technology and secure transaction processing.
10. **Corporate Banking Solution**:
    * With further development, the system could be tailored to meet the needs of corporate clients, including features like multi-user access, detailed financial reporting, payroll management, and integration with enterprise resource planning (ERP) systems.

**BIBLOGRAPHY**

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)