BECE209E

Structured and Object-Oriented Programming

A project report titled

BANK ACCOUNT MANAGER

By

22BEC1053	JAYANTH.S.B
22BEC1093	KISHOR.R

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Submitted to

Dr. R. KARTHIK

November 2023

School of Electronics Engineering

DECLARATION BY THE CANDIDATE

I hereby declare that the Report entitled “**Bank Account Manager”** submitted by me to VIT Chennai is a record of bonafide work undertaken by me under the supervision of **Dr. R. Karthik, Associate Professor, SENSE, VIT Chennai.**

Signature of the Candidates

Chennai

09/11/2021.

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. R. Karthik,** School of Electronics Engineering for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We are extremely grateful to **Dr. Susan Elias,** Dean of the School of Electronics Engineering (SENSE), VIT University Chennai, for extending the facilities of the school towards our project and for his unstinting support.

We express our thanks to our **Head of The Department Dr. Mohanaprasad (for B.Tech-ECE)** for his support throughout the course of this project.

We also take this opportunity to thank all the faculty of the school for their support and their wisdom imparted to us throughout the courses till date.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

BONAFIDE CERTIFICATE

Certified that this project report entitled “Bank Account Manager**”** is a bonafide work of **JAYANTH.S.B (22BEC1053) and KISHOR.R (19BEC1093)** carried out the project work under my supervision and guidance for BECE209E Structured and Object-Oriented Programming.

Dr. R. Karthik

School of Electronics Engineering

VIT University, Chennai

Chennai – 600 127.

TABLE OF CONTENTS

S.NO	Chapter	PAGE NO.
1	Chapter -1 Introduction	7-8
2	Chapter – 2 Requirements and proposed system	9
3	Chapter -3 Module description	10
4	Chapter4-Source Code	11-23
4	Chapter 5 – Results and Conclusion	24-25
5	Reference	26

ABSTRACT

In today's digital era, efficient and secure management of financial assets is crucial. The Bank Account Management System (BAMS) is a state-of-the-art C++ application designed to address these needs. BAMS offers a comprehensive suite of features for both individual and business account holders, providing a user-friendly platform for account creation, deposits, withdrawals, and account details management. Robust security features, including multi-factor authentication and transaction encryption, safeguard user data and protect against unauthorized access. BAMS' modular design and extensive customization capabilities make it adaptable to specific financial institutions' requirements. The system's intuitive administrative tools simplify account management and transaction oversight. BAMS enhances user experience, streamlines operations, and strengthens security, making it a valuable tool for modern banking and financial management.

CHAPTER – 1

INTRODUCTION

The Bank Account Management System is an advanced C++ application that offers a comprehensive, highly sophisticated, and state-of-the-art solution for efficiently and securely overseeing and managing complex financial accounts in the fast-evolving and ever-changing contemporary digital landscape. In today's modern and digitally driven era, the effective and robust management of diverse, substantial, and often intricate financial assets is of paramount importance. This necessitates a comprehensive system that not only caters to but also anticipates the diverse and complex needs, demands, and expectations of both individual and business account holders. The system stands as a beacon of reliability, dependability, and flexibility, providing a robust and dependable platform designed to seamlessly facilitate their ability to access, control, and effectively manage their bank accounts with the utmost convenience, efficiency, and security, thus addressing and exceeding the multifaceted requirements and expectations of today's ever-evolving financial sector.

In this comprehensive and in-depth document, we will thoroughly and comprehensively explore and scrutinize the integral components, aspects, and facets of the system. We will delve into and dissect the stipulated and meticulously detailed requirements, serving as the bedrock and foundation upon which the system's functionality is meticulously built. Additionally, we will navigate through and explore the proposed feature set, which offers a vast, extensive, and broad array of functionalities and capabilities, providing a versatile and powerful toolset for both users and administrators alike. To gain an even deeper, more profound, and comprehensive understanding of the system's internal workings and intricacies, we will meticulously and methodically dissect and navigate through intricate module descriptions, shedding light on how various components, subcomponents, and elements interact, integrate, and function harmoniously and cohesively as part of the greater whole.

Furthermore, to demystify and elucidate the technical aspects and implementation details, we will provide an overarching and detailed glimpse and insight into the source code that brings the system to life, showcasing and highlighting the nuanced, precise, and elegant implementation of its features, functionalities, and capabilities. Finally, we will present a succinct yet comprehensive summary encapsulating and distilling the results, outcomes, and conclusions derived from the system's operational functionality. This will underscore and emphasize the system's significance, potential, and its role as a pioneering and transformative force in the realm of modern banking and financial management. The Bank Account Management System stands ready to deliver highly effective, secure, and innovative solutions to meet the intricate and ever-evolving demands of the world of financial management.

CHAPTER – 2

REQUIREMENTS AND PROPOSED SYSTEM

Requirements

1. Account Creation: The system should allow users to create a new bank account, capturing crucial information such as the account number, account holder name, initial balance, password, email address, and mobile number.
2. Deposits and Withdrawals: Users should be able to deposit and withdraw funds from their accounts, provided that they enter the correct password and have a sufficient balance.
3. View Account Details: Account holders should be able to view their account details, including the current balance, account holder name, email address, and mobile number.
4. Security Features: The system should implement security features such as account locking after a specified number of failed login attempts.
5. Administrative Privileges: Administrators should have access to a list of all accounts, the ability to unlock locked accounts, and the option to modify account details.

Proposed System

The proposed system is an object-oriented C++ program designed to fulfil the requirements outlined above. It utilizes data structures like vectors, maps, and structs to manage and organize bank account data efficiently. The user interface is user-friendly, guiding users through various account management actions.

CHAPTER – 3

MODULE DESCRIPTION

- Account Class

The core of the system is the Account class, responsible for encapsulating the attributes and functionalities of a bank account. It offers methods for depositing, withdrawing, displaying account details, and more. To maintain data privacy, the class utilizes a private member variable, BAI (Bank Account Information), to store account data securely.

- BAI Struct

The BAI struct (Bank Account Information) is responsible for storing critical information related to a bank account, including the account number, account holder name, balance, password, email address, mobile number, and password retry count. This encapsulation ensures that data remains secure and accessible through controlled access points.

- Main Function

The main function acts as the entry point for the program. It provides a user-friendly command-line interface to guide users through various account management actions. Users can create accounts, deposit or withdraw funds, view their account details, and access administrative functions to unlock accounts and modify account details.

CHAPTER – 4
SOURCE CODE

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <time.h>
#include <map>
#include <iomanip>
using namespace std;

/*
BAI = Bank Account Information

Variables:
AN = Account Number
AHN = Account Holder Name
Bal = Balance
Pass = Password
Email = User Email ID
MN = Mobile Number
*/

struct BAI
{
    unsigned int AN;
    string AHN;
    double Bal;
    string Pass;
    string EmailID;
    string MN;
    int passtry;
};

/*
Account = Bank Account

Private Member Variable:
an = account number!
ahn = account holder name!
ib = initial balance!
am = amount!
```

```
pass = password!
email = email id!
mn = mobile number!

Member Functions:
deposit = +$
withdraw = -$
display = <$>
retan = Account Number
retpass = Password
retacc = Account Holder Name
retbal = Balance
retem = Email ID
retmob = Mobile Number
*/

class Account
{
private:
   BAI bai;

public:
   Account(unsigned int an, string ahn, double ib, string pass, string email, string mn, int
atpass = 3)
   {
      bai.AN = an;
      bai.AHN = ahn;
      bai.Bal = ib;
      bai.Pass = pass;
      bai.EmailID = email;
      bai.MN = mn;
      bai.passtry = atpass;
   }

   void deposit(double am)
   {
      if (am > 0)
      {
         bai.Bal += am;
         cout << "\nDeposited Amount = $" << am << "\\-" << endl;
      }
      else
      {
        cout << "\nInvalid Deposit." << endl;
    }
}

void withdraw(double am)
{
    if (am > 0 && am <= bai.Bal)
    {
        bai.Bal -= am;
        cout << "\nWithdrawn Amount = $" << am << "\\-" << endl;
        cout << "Total Balance = $" << bai.Bal <<"\\-" << endl;
    }
    else
    {
        cout << "\nInvalid withdrawal Amount or Insufficient Bal." << endl;
    }
}

void display()
{
    cout << "\nAccount Number : " << bai.AN << endl;
    cout << "Account Holder : " << bai.AHN << endl;
    cout << "Account Balance = $" << bai.Bal << endl;
    cout << "Email ID : " << bai.EmailID << endl;
    cout << "Mobile Number : " << bai.MN << endl;
}

void admindisplay()
{
    cout << bai.AN << "\t:::::\t" << bai.AHN << "\t:::::\t" << bai.Bal;
    cout << "\t:::::\t" << bai.EmailID << "\t:::::\t" << bai.MN << endl;
}

void tminus()
{
    bai.passtry--;
}

void reset()
{
    bai.passtry = 3;
}
```

```
unsigned int retan() const
{
   return bai.AN;
}
string retpass() const
{
   return bai.Pass;
}
string retacc() const
{
   return bai.AHN;
}
double retbal() const
{
   return bai.Bal;
}
string retem() const
{
   return bai.EmailID;
}
string retmob() const
{
   return bai.MN;
}
int rettry() const
{
   return bai.passtry;
}

void modify(char c, string n)
{
   switch (c)
   {
      case 'N':
         bai.AHN = n;
         break;
      case 'P':
         bai.Pass = n;
         break;
      case 'M':
         bai.MN = n;
         break;
      case 'E':
```

```
                bai.EmailID = n;
                break;
            default:
                cout << "INVALID CHOICE!!!" << endl;
        }
    }
};

/*
as = Bank Accounts
choice = Action on Account
Action List = List of choices

an = account number!!
ahn = account holder name!!
ib = initial balance!!
am = amount!!
pass = password!!
email = email id!!
mn = mobile number!!

adam = add amount
subam = subract amount
*/

int main()
{
    cout << fixed << setprecision(0);
    vector<Account> as;

    map<string, string> addata;
    addata["Jayanth"] = "jayrocks";
    addata["Kishor"] = "kish";
    addata["Karthick"] = "prof.c++";

    cout << "\nBank Account Manager\n";
    cout << "Press 0 for 'Action List'" << endl;
    cout << "Press 1 to 'Create an Account'" << endl;
    cout << "Press 2 to 'Deposit Amount'" << endl;
    cout << "Press 3 to 'Withdraw Amount'" << endl;
    cout << "Press 4 to 'View Account Details'" << endl;
    cout << "Press 5 to 'Exit'" << endl;
    while (true)
    {
        int choice;
        cout << "\nOption : ";
        cin >> choice;

        switch (choice)
        {
            case 0:
                cout << "\nPress 1 to 'Create an Account'" << endl;
                cout << "Press 2 to 'Deposit Amount'" << endl;
                cout << "Press 3 to 'Withdraw Amount'" << endl;
                cout << "Press 4 to 'View Account Details'" << endl;
                cout << "Press 5 to 'Exit'" << endl;
                break;

            case 1:
            {
                string ahn;
                unsigned int an;
                double ib;
                string pass, pass_;
                string email;
                string mn;

                srand(time(0));
                an = rand() * 7137;

                cout << "\nAccount Holder (All Caps) : ";
                cin.ignore();
                getline(cin, ahn);

                cout << "Password : ";
                cin >> pass;
                cout << "Re-enter Password : ";
                cin >> pass_;
                if(pass != pass_)
                {
                    cout << "PASSWORDS DO NOT MATCH.\nRETRY ACCOUNT
REGISTRATION.";
                    break;
                }
                cout << "Email ID : ";
                cin >> email;

                cout << "Mobile Number: ";
                cin >> mn;

                cout << "Initial Balance Deposit = $";
                cin >> ib;

                Account account(an, ahn, ib, pass, email, mn);
                as.push_back(account);
                cout << "\nACCOUNT CREATED." << endl;
                cout << "\nYour Account:-" << endl;
                account.display();
                break;
            }

            case 2:
            {
                unsigned int an;
                double adam;
                bool note = false;

                cout << "\nAccount Number : ";
                cin >> an;
                cout << "Deposit Amount = $";
                cin >> adam;

                for (Account& account : as)
                {
                    if (account.retan() == an)
                    {
                        account.deposit(adam);
                        note = true;
                        break;
                    }
                }

                if(!note){
                    cout << "\nACCOUNT NUMBER INVALID." << endl;
                }
                break;
            }
```

```
case 3:
{
    unsigned int an;
    double subam;
    string pass;
    bool note = false;

    cout << "\nAccount Number : ";
    cin >> an;
    cout << "Password : ";
    cin >> pass;

    for (Account& account : as)
    {
        if(account.rettry() == 0)
        {
            cout << "\nMAXIMUM PASSWORD TRIES REACHED.\nACCOUNT
LOCKED" << endl;
            return 0;
        }

        if((account.retan() == an) && (account.retpass() == pass))
        {
            cout << "Withdrawal Amount = $";
            cin >> subam;
            account.withdraw(subam);
            note = true;
            break;
        }
        else if((account.retan() == an) && (account.retpass() != pass))
        {
            account.tminus();
            cout << "\nINVALID PASSWORD" << endl;
            note = true;
            break;
        }
    }

    if(!note){
        cout << "\nACCOUNT NUMBER INVALID." << endl;
    }
    break;
```

```
            }

            case 4:
            {
               unsigned int an;
               string pass;
               bool note = false;

               cout << "\nAccount Number : ";
               cin >> an;
               cout << "Password : ";
               cin >> pass;

               for (Account& account : as)
               {
                  if(account.rettry() == 0)
                  {
                     cout << "\nMAXIMUM PASSWORD TRIES REACHED.\nACCOUNT
LOCKED" << endl;
                     return 0;
                  }

                  if ((account.retan() == an) && (account.retpass() == pass))
                  {
                     account.display();
                     note = true;
                     break;
                  }
                  else if((account.retan() == an) && (account.retpass() != pass))
                  {
                     account.tminus();
                     cout << "\nINVALID PASSWORD" << endl;
                     note = true;
                     break;
                  }
               }

               if(!note){
                  cout << "\nACCOUNT NUMBER INVALID." << endl;
               }
               break;
            }
```

```
case -1:
{
    string adid, adpass;
    bool note = false;

    cout << "\nAdmin User ID: ";
    cin >> adid;
    cout << "Admin Password: ";
    cin >> adpass;

    map<string, string>::reverse_iterator it = addata.rbegin();
    while (it != addata.rend()) {
        if (it->first == adid) {
            note = true;
            if (it->second != adpass) {
                return 0;
            }
            else {
                cout << "\nAccount List:\n";
                for (Account& account : as) {
                    account.admindisplay();
                }
                break;
            }
        }
        ++it;
    }

    if (!note) {
        cout << "\nADMIN USERNAME INVALID" << endl;
        break;
    }
    break;
}

case -2:
{
    string adid, adpass;
    bool note = false;
    unsigned int an;

    cout << "\nAdmin User ID: ";
    cin >> adid;
```

```
        cout << "Admin Password: ";
        cin >> adpass;

        map<string, string>::reverse_iterator it = addata.rbegin();
        while (it != addata.rend()) {
            if (it->first == adid) {
                note = true;
                if (it->second != adpass) {
                    return 0;
                }
                else {
                    cout << "\nEnter the Account Number to Unlock: ";
                    cin >> an;
                    for (Account& account : as) {
                        if (account.retan() == an) {
                            account.reset();
                            cout << "Account with number " << an << " is now unlocked." <<
endl;
                            break;
                        }
                    }
                    note = true;
                    break;
                }
            }
            ++it;
        }

        if (!note) {
            cout << "\nADMIN USERNAME INVALID" << endl;
            break;
        }
        break;
    }

    case -3:
    {
        string adid, adpass, newval;
        char mod;
        bool note = false;
        unsigned int an;

        cout << "\nAdmin User ID: ";
        cin >> adid;
        cout << "Admin Password: ";
        cin >> adpass;

        map<string, string>::reverse_iterator it = addata.rbegin();
        while (it != addata.rend()) {
            if (it->first == adid) {
                note = true;
                if (it->second != adpass) {
                    return 0;
                }
                else {
                    cout << "\nEnter the Account Number to Modify: ";
                    cin >> an;
                    break;
                }
            }
            ++it;
        }

        if (!note) {
            cout << "\nADMIN USERNAME INVALID" << endl;
            break;
        }

        cout << "\nModify:\n";
        cout << "N. Name\nP. Password\nE. Email ID\nM. Mobile Number\n\nChoice: ";
        cin >> mod;
        cout << "\nNew Value: ";
        cin >> newval;

        for (Account& account : as) {
            if (account.retan() == an) {
                account.modify(mod, newval);
                cout << "Account with number " << an << " has been modified." << endl;
                break;
            }
        }
        break;
    }

    case 5:
        cout << "\n(-: THANK YOU :-)\n" << endl;
```

```
                return 0;

            default:
                cout << "\nINVALID ACTION\n" << endl;
                break;
        }
    }

    return 0;
}
```

CHAPTER – 5
RESULTS

The Bank Account Management System has yielded several positive outcomes, including:

- Enhanced Security: The system's robust security features, such as multi-factor authentication and transaction encryption, provide a high level of protection against unauthorized access and fraudulent activities.

- Improved Efficiency: The system's streamlined processes and user-friendly interface enable account holders to perform various transactions, such as deposits, withdrawals, and account transfers, with greater ease and efficiency.

- Enhanced Flexibility: The system's modular design and extensive customization capabilities allow it to be adapted to the specific needs and requirements of various financial institutions and their clientele.

- Streamlined Administration: The system's intuitive administrative tools simplify the task of managing a large volume of accounts, overseeing transactions, and maintaining financial records.

- Enhance User Experience: The system's user-centric approach and intuitive interface provide a seamless and enjoyable experience for both individual and business account holders.

Overall, the Bank Account Management System has proven to be a valuable tool for effectively managing financial transactions, enhancing security, and improving the overall user experience. Its continued development and refinement will solidify its position as a leading solution in the field of financial technology.

CONCLUSION

In conclusion, the Bank Account Management System is a comprehensive and sophisticated application that addresses the intricate and ever-evolving demands of modern banking and financial management. Its robust functionality, security features, and user-friendly interface make it an invaluable tool for both individual and business account holders. The system's ability to seamlessly integrate with various components and subcomponents ensures a seamless and user-centric experience, while its modular design facilitates effortless customization and adaptation to evolving user needs. The system's thorough documentation and comprehensive source code provide a clear understanding of its inner workings and functionalities, enabling developers and administrators to effectively maintain and enhance its performance. As a result of its meticulous design, rigorous testing, and comprehensive functionality, the Bank Account Management System stands as a testament to the power of technology in revolutionizing financial management and ensuring the security and accessibility of financial resources.

REFERENCES

1. www.geeksforgeeks.org
2. www.programiz.com
3. BECE209E Notes