**Bank Management System.**



**Session 2023 - 2027**

**Submitted by:**

Sher Muhammad 2023-CS-15

**Supervised by:**

Prof. Dr. Muhammad Awais Hassan

& Sir. Laeeq Khan Niazi

**Course:**

CSC-102 Object Oriented Programming

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

**Table of Content**

[1. Introduction: 3](#_Toc164974813)

[i. Objective: 3](#_Toc164974814)

[ii. Output Expectations: 3](#_Toc164974815)

[2. Users of Application: 3](#_Toc164974816)

[iii. Admin 3](#_Toc164974817)

[iv. User 3](#_Toc164974818)

[3. Functional Requirements 3](#_Toc164974819)

[4. Wireframes For GUI Application 4](#_Toc164974820)

[5. Wireframes For Console Application 9](#_Toc164974821)

[6. Class Design (CRC) 12](#_Toc164974822)

[7. Complete Code 12](#_Toc164974823)

# Introduction:

## Objective:

This app is an efficient and secure bank management system that caters to the functionalities required by both administrator and regular users.

## Output Expectations:

##### User-Friendly Interface

The interface for Console as well as GUI is designed to be easily understandable for users, they do not require any additional information before using the software.

# Users of Application:

There are two types of users in this application. An Admin who has the authority over the application and the Client who can access his/her account info and conduct different transactions easily.

## Admin

The admin or manager holds full authority over the application. They can add or remove clients, view client records, inspect the assets held by the bank, and even add new assets if necessary.

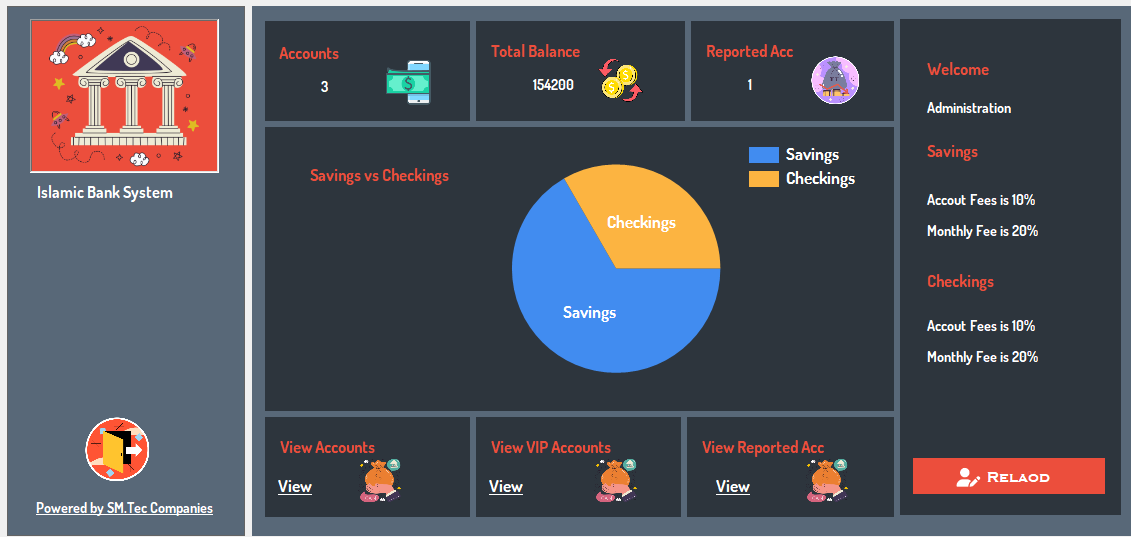
## User

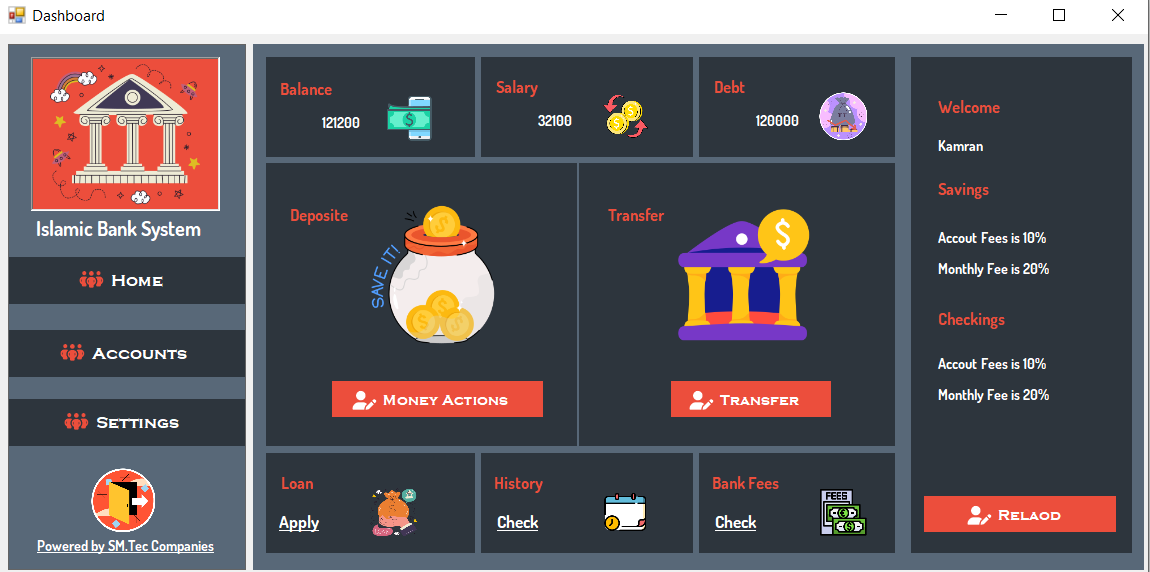
The user has the capability to register their account, enabling them to perform various transactions such as deposit, transfer, and withdraw Additionally, they can view record of their transactions and possess the authority to modify their personal information.

# Functional Requirements

|  |  |  |
| --- | --- | --- |
| **User Type** | **Functions** | **Action Performed** |
| **Admin** | View VIP Users | View some of the VIP Users from |
| View Users | View all users and their progress |
| View Single User | View the details of the single user |
| Give Loan | Give Loan to the Applicants |
| Report Account | Report the Suspicious User |
| Un Report Account | This will Un report the reported account |
| Delete Permanently | This will delete the user and its account permanently |
|  | | |
| **User** | Create Account | User can create account for interactions in the app |
| Deposit Money | Deposit money into the account |
| Withdraw Money | Withdraw money from the account |
| Transfer Money | Transfer money to another account |
| View Transactions | Can see his transactions of the account |
| Apply Loan | Users can apply for Loan |
| Change Password | To Change the Password |
| Bank Fees and Interest Rate | Users can see Bank Fees and Interest Rate applied |

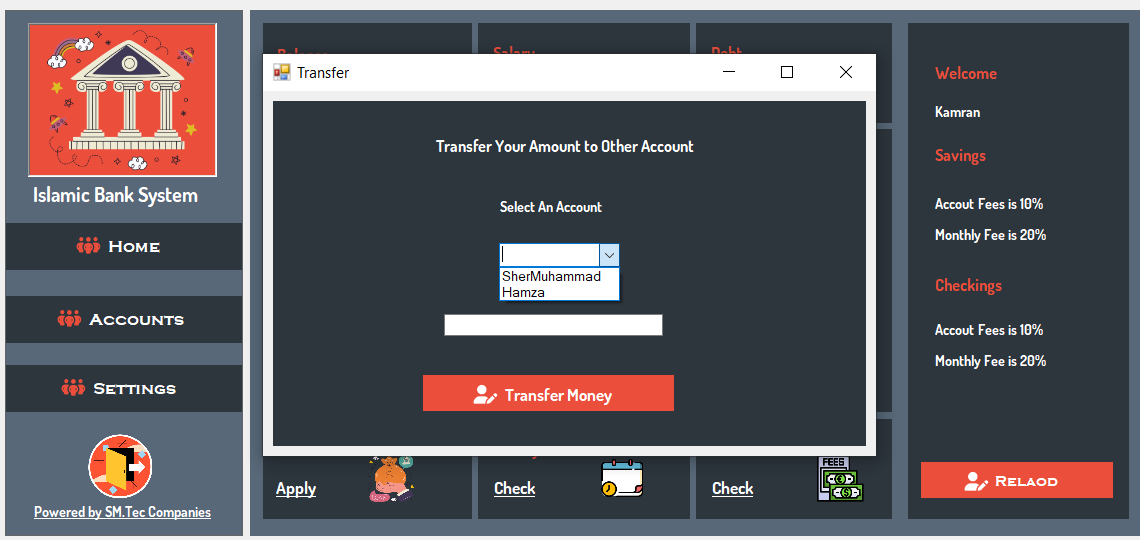
# Wireframes For GUI Application

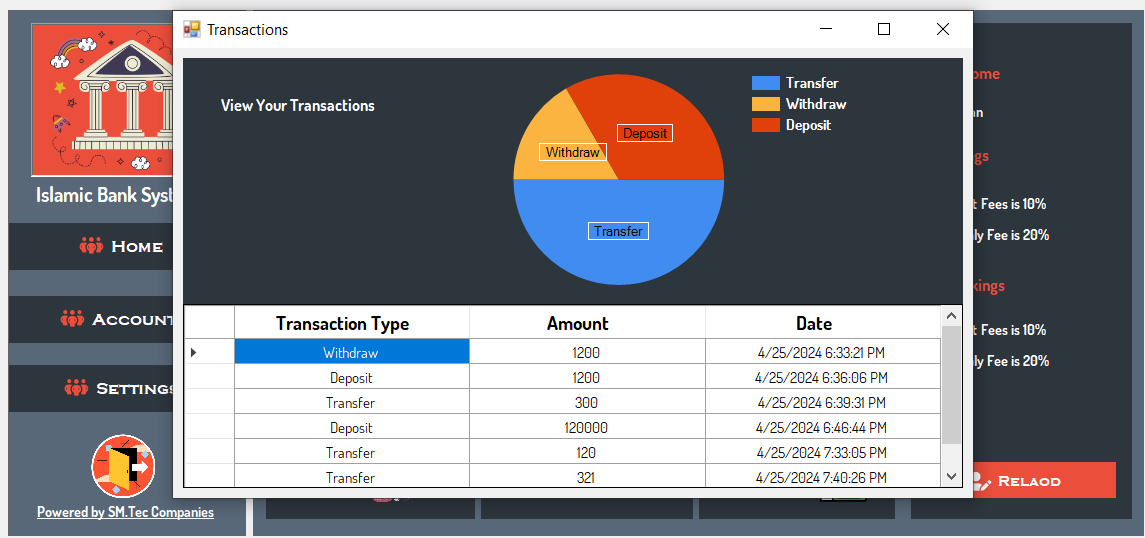
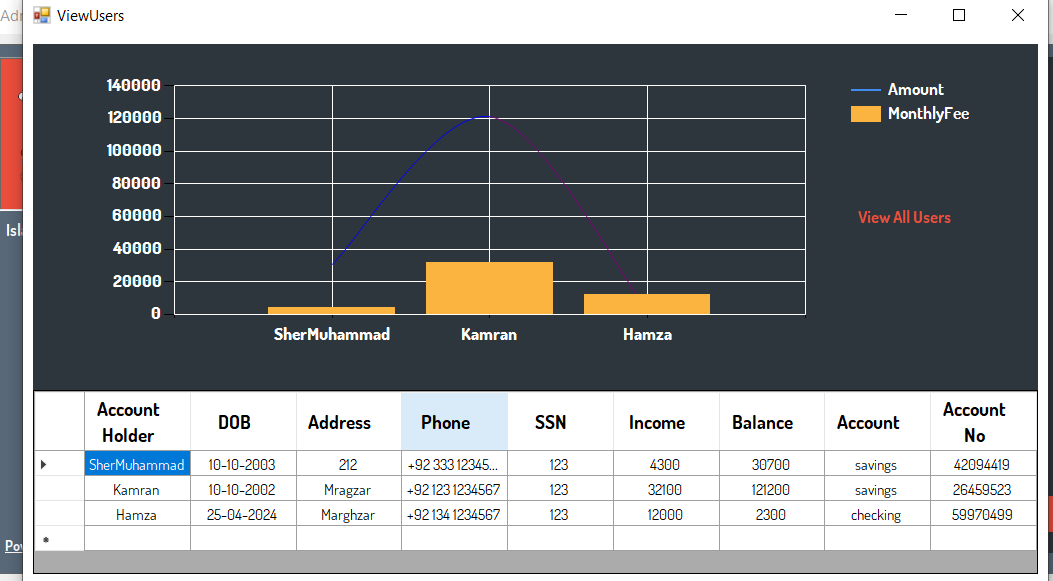
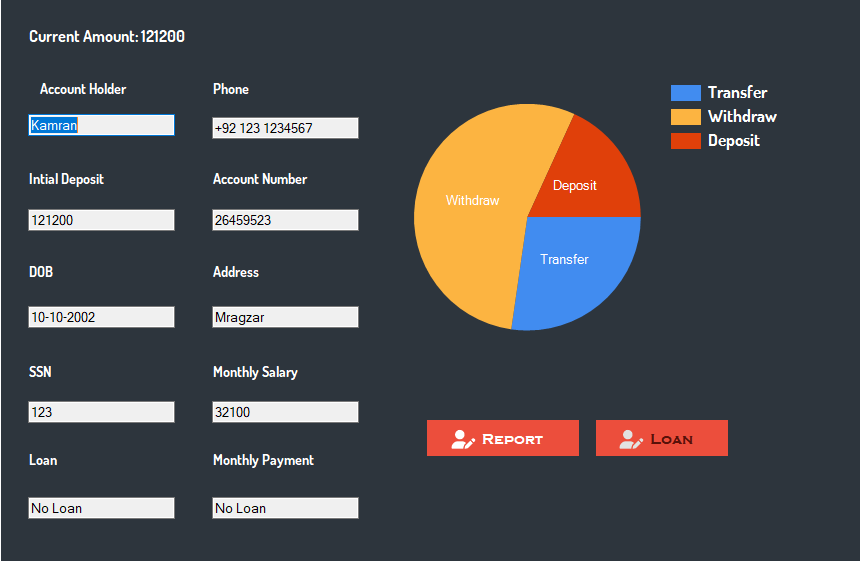




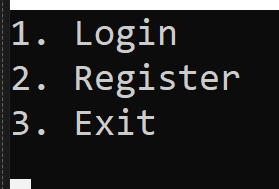
***Admin Pannel***

***User Pannel***



# Wireframes For Console Application

****

Main Page

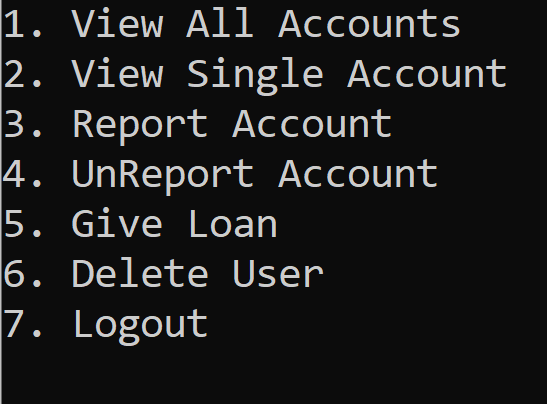
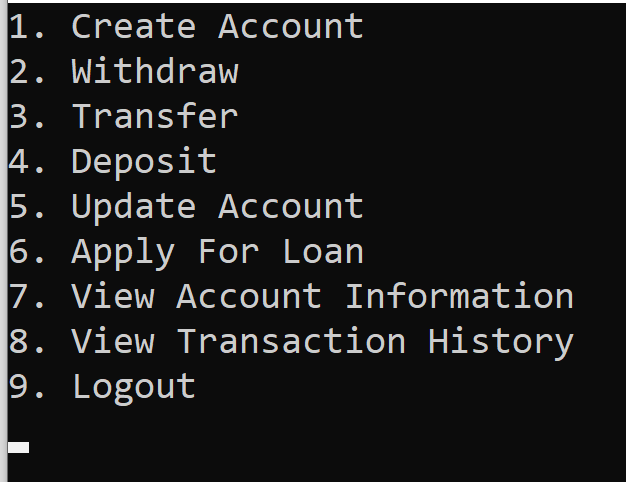


Figure -Console-UserPage

User’s Pannel

****

Admin Pannel

# Class Design (CRC)

# Complete Code

## UI:

public static void UpdateAccountHolder(string prevAccountHolder, string newAccountHolder)

{

MUser currentUser = ObjectHandler.GetUserDL().GetCurrentUser();

Account currentAccount = currentUser.GetAccount();

ITransactionDL transactionDL = ObjectHandler.GetTransactionDL();

// check if the transactions for the current acccount exist

if (currentAccount.GetTransactions().Count > 0)

{

transactionDL.UpdtateAccountHolder(prevAccountHolder, newAccountHolder);

}

if (currentAccount.GetLoan() != null)

{

currentAccount.GetLoan().SetAccountHolder(newAccountHolder);

ObjectHandler.GetLoanDL().UpdateLoanInfo(newAccountHolder, prevAccountHolder);

}

}

public static bool isAccountAlreadyExist() // for current user account

{

MUser currentUser = ObjectHandler.GetUserDL().GetCurrentUser();

Account account = ObjectHandler.GetAccountDL().isAccountExists(currentUser.GetUsername());

if (account != null)

{

return true;

}

return false;

}

public static bool Deposit()

{

MUser currentUser = ObjectHandler.GetUserDL().GetCurrentUser();

Account currentAccount = currentUser.GetAccount();

Console.WriteLine("Enter the amount to deposit: ");

int amount;

do

{

try

{

amount = Convert.ToInt32(Console.ReadLine());

if (UserInterface.ValidateNumber(amount))

{

break;

}

else

{

Console.WriteLine("Invalid amount. Please enter a positive number.");

}

}

catch (FormatException)

{

Console.WriteLine("Invalid input. Please enter a number.");

}

} while (true);

bool isDeposited = currentAccount.Deposit(amount);

if (!isDeposited)

{

Console.WriteLine("Error depositing funds. Please try again.");

return false;

}

bool isUpdated = ObjectHandler.GetAccountDL().UpdateAccountInfo(currentAccount, currentAccount.GetAccountHolder());

if (!isUpdated)

{

Console.WriteLine("Error updating account information. Deposit may not be reflected.");

}

bool isSaved = SaveTransaction("Deposit", amount, currentAccount.GetAccountHolder());

if (!isSaved)

{

Console.WriteLine("Error saving transaction details.");

}

return isDeposited && isUpdated && isSaved;

}

public static bool Withdraw()

{

MUser currentUser = ObjectHandler.GetUserDL().GetCurrentUser();

Account currentAccount = currentUser.GetAccount();

int amount;

do

{

Console.WriteLine("Enter the amount to withdraw: ");

try

{

amount = Convert.ToInt32(Console.ReadLine());

if (UserInterface.ValidateNumber(amount))

{

break;

}

else

{

Console.WriteLine("Invalid amount. Please enter a number.");

}

}

catch (FormatException)

{

Console.WriteLine("Invalid input. Please enter a number.");

}

} while (true);

bool isWithdraw = currentAccount.Withdraw(amount);

if (!isWithdraw)

{

Console.WriteLine("Insufficient funds for withdrawal. Please try a lower amount.");

return false;

}

bool isUpdated = ObjectHandler.GetAccountDL().UpdateAccountInfo(currentAccount, currentAccount.GetAccountHolder());

if (!isUpdated)

{

Console.WriteLine("Error updating account information. Withdrawal may not be reflected.");

}

bool isSaved = SaveTransaction("Withdraw", amount, currentAccount.GetAccountHolder());

if (!isSaved)

{

Console.WriteLine("Error saving transaction details.");

}

return isWithdraw && isUpdated && isSaved;

}

public static bool Transfer()

{

MUser currentUser = ObjectHandler.GetUserDL().GetCurrentUser();

Account currentAccount = currentUser.GetAccount();

Console.WriteLine("Enter the amount to transfer: ");

int amount;

do

{

try

{

amount = Convert.ToInt32(Console.ReadLine());

if (UserInterface.ValidateNumber(amount))

{

break;

}

else

{

Console.WriteLine("Invalid amount. Please enter a positive number.");

}

}

catch (FormatException)

{

Console.WriteLine("Invalid input. Please enter a number.");

}

} while (true);

Console.WriteLine("Enter the account holder name to transfer to: ");

string accountHolder = Console.ReadLine();

Account accountToTransfer = ObjectHandler.GetAccountDL().isAccountExists(accountHolder);

if (accountToTransfer == null)

{

return false;

}

AccountTransaction transaction1 = new AccountTransaction("Deposit", amount, accountToTransfer.GetAccountHolder()); // deposit in the account to transfer

bool isTransaction = ObjectHandler.GetTransactionDL().SaveTransactionInfo(transaction1);

bool isSaved = ObjectHandler.GetAccountDL().UpdateBalanceOnTransactions(accountToTransfer.GetIntialDeposit(), accountToTransfer.GetAccountHolder());

bool isTransfered = currentAccount.Transfer(amount); // Withdraws from current account

if (!isTransfered && !(isTransaction && isSaved))

{

return false;

}

bool isTransactionSaved = SaveTransaction("Transfer", amount, currentAccount.GetAccountHolder());

return isTransactionSaved;

}

public static bool SaveTransaction(string type, int amount, string accountHolder)

{

AccountTransaction transaction = new AccountTransaction(type, amount, accountHolder);

bool isSaved = ObjectHandler.GetTransactionDL().SaveTransactionInfo(transaction);

return isSaved;

}

## FH:

public class ReportedAccountFH : IReportedAccount

{

private static List<ReportedAccount> reportedAccounts = new List<ReportedAccount>();

private string path = "C:\\Users\\dell\\Pictures\\OOP\\IslamicBank\\BankData\\reportedAccounts.txt";

private ReportedAccountFH() { }

private static ReportedAccountFH instance;

public static ReportedAccountFH GetInstance()

{

if (instance == null)

{

instance = new ReportedAccountFH();

}

return instance;

}

public bool SaveReportedAccountInfo(ReportedAccount reportedAccount)

{

reportedAccounts.Add(reportedAccount);

// saving to file

SaveDataToFile(reportedAccount);

return true;

}

public List<ReportedAccount> GetReportedAccounts()

{

return reportedAccounts;

}

public void LoadReportedAccountData()

{

try

{

using (StreamReader reader = new StreamReader(path))

{

string line;

while ((line = reader.ReadLine()) != null)

{

string[] data = line.Split(',');

try

{

ReportedAccount reportedAccount = new ReportedAccount(data[0], data[1]);

// add to list

reportedAccounts.Add(reportedAccount);

}

catch (Exception ex)

{

}

}

}

}

catch (Exception ex)

{

}

}

public bool SaveDataToFile(ReportedAccount reportedAccount)

{

try

{

using (StreamWriter writer = new StreamWriter(path, true))

{

writer.WriteLine($"{reportedAccount.GetAccountHolder()},{reportedAccount.GetReason()}");

}

return true;

}

catch (Exception ex)

{

return false;

}

}

public ReportedAccount isReported(string accountHolder)

{

foreach (ReportedAccount reportedAccount in reportedAccounts)

{

if (reportedAccount.GetAccountHolder() == accountHolder)

{

return reportedAccount;

}

}

return null;

}

public bool UnreportAccount(string accountHolder)

{

ReportedAccount reportedAccount = isReported(accountHolder);

if (reportedAccount != null)

{

reportedAccounts.Remove(reportedAccount);

bool isUnreported = UnreportAccountInFile(accountHolder);

return isUnreported;

}

return false;

}

public bool UnreportAccountInFile(string accountHolder)

{

try

{

List<string> lines = new List<string>();

using (StreamReader reader = new StreamReader(path))

{

string line;

while ((line = reader.ReadLine()) != null)

{

string[] data = line.Split(',');

if (data[0] != accountHolder)

{

lines.Add(line);

}

}

}

using (StreamWriter writer = new StreamWriter(path))

{

foreach (string line in lines)

{

writer.WriteLine(line);

}

}

return true;

}

catch (Exception ex)

{

return false;

}

## DB:

# public class ReportedAccountDB : IReportedAccount

# {

# private static List<ReportedAccount> reportedAccounts = new List<ReportedAccount>();

# private ReportedAccountDB() { }

# private static ReportedAccountDB instance;

# public static ReportedAccountDB GetInstance()

# {

# if (instance == null)

# {

# instance = new ReportedAccountDB();

# }

# return instance;

# }

# public bool SaveReportedAccountInfo(ReportedAccount reportedAccount)

# {

# reportedAccounts.Add(reportedAccount);

# return SaveDataToDb(reportedAccount);

# }

# public List<ReportedAccount> GetReportedAccounts()

# {

# return reportedAccounts;

# }

# public bool SaveDataToDb(ReportedAccount reportedAccount)

# {

# string Query = "INSERT INTO ReportedAccounts (AccountHolder, Reason) VALUES ('{0}','{1}')";

# Query = string.Format(Query, reportedAccount.GetAccountHolder(), reportedAccount.GetReason());

# int rowsAffected = utills.SetData(Query);

# if (rowsAffected > 0)

# {

# return true;

# }

# return false;

# }

# public void LoadReportedAccountData()

# {

# string Query = "SELECT \* FROM ReportedAccounts";

# DataTable dt = utills.GetData(Query);

# if (dt != null)

# {

# foreach (DataRow row in dt.Rows)

# {

# ReportedAccount reportedAccount = new ReportedAccount(row["AccountHolder"].ToString(), row["Reason"].ToString());

# // add to list

# reportedAccounts.Add(reportedAccount);

# }

# }

# }

# public ReportedAccount isReported(string accountHolder)

# {

# foreach (ReportedAccount reportedAccount in reportedAccounts)

# {

# if (reportedAccount.GetAccountHolder() == accountHolder)

# {

# return reportedAccount;

# }

# }

# return null;

# }

# public bool UnreportAccount(string accountHolder)

# {

# ReportedAccount reportedAccount = isReported(accountHolder);

# if (reportedAccount != null)

# {

# reportedAccounts.Remove(reportedAccount);

# bool isUnreported = UnreportAccountInDB(accountHolder);

# return isUnreported;

# }

# return false;

# }

# public bool UnreportAccountInDB(string accountHolder)

# {

# string Query = $"DELETE FROM ReportedAccounts WHERE AccountHolder = '{accountHolder}'";

# int rowsAffected = utills.SetData(Query);

# if (rowsAffected > 0)

# {

# return true;

# }

# return false;

# }

# }