**Azure Bank Inc.**



**Session 2023 - 2027**

**Submitted by:**

Munees Tariq 2023-CS-32

**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](#_Toc164837024)

[i. Objective: 3](#_Toc164837025)

[ii. Output Expectations: 3](#_Toc164837026)

[2. Users of Application: 3](#_Toc164837027)

[i. Admin/Manager 3](#_Toc164837028)

[ii. Client/User 3](#_Toc164837029)

[3. Functional Requirements 4](#_Toc164837030)

[4. Wireframes For GUI Application 5](#_Toc164837031)

[5. Wireframes For Console Application 9](#_Toc164837032)

[6. Class Design (CRC) 12](#_Toc164837033)

[7. Complete Code 13](#_Toc164837034)

[i. UI: 13](#_Toc164837035)

[ii. FH: 14](#_Toc164837036)

[iii. DB: 15](#_Toc164837037)

# 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.

##### Security Measures

Ensuring the safety of user’s data is crucial. To enhance security, Password masking is used in both of the projects.

# 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/Manager

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.

## Client/User

The client 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/Manager** | View Liquidity | View the total balance collectively held by all clients. |
| Add User | Creates new user account |
| View Users | View the records of clients |
| Add An Asset | To add new asset and its worth |
| View Assets | View the asset’s that the bank holds |
| Remove User | Remove the account of a client |
| Reset Password | Resets the password for admin |
|  | | |
| **Client/User** | Check Portfolio | Check the Balance and other assets |
| 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 |
| Lock/Unlock Transaction | To block or unblock the transactions |
| Change Password | To Change the Password |
| Delete Account | Allow user to remove his account |

# Wireframes For GUI Application

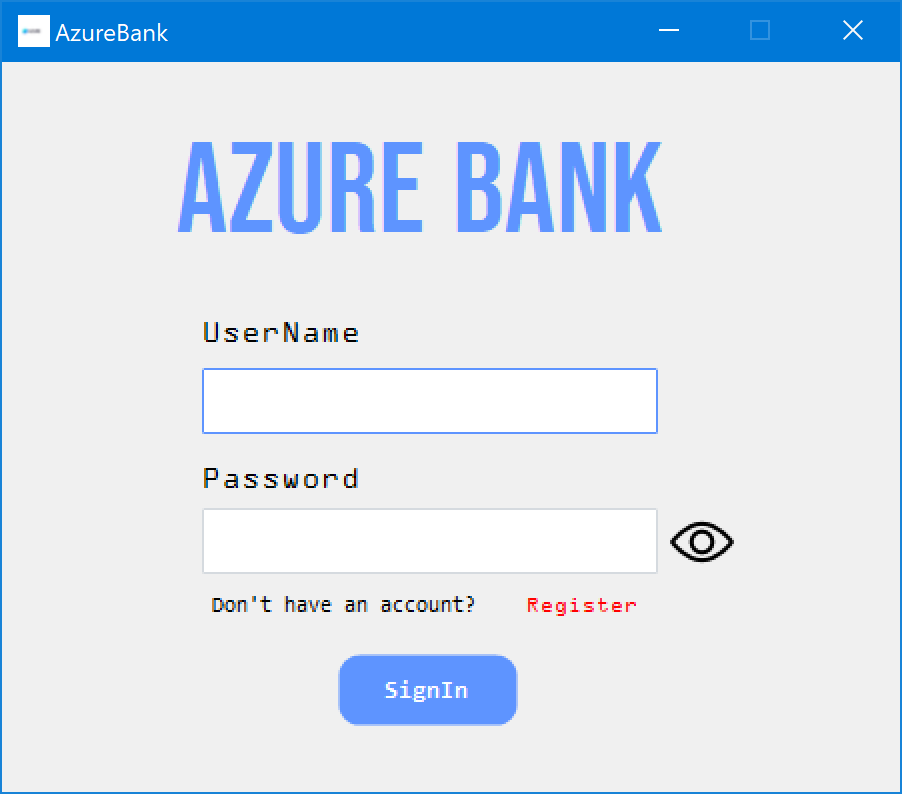


Figure 1-Gui-MainPage

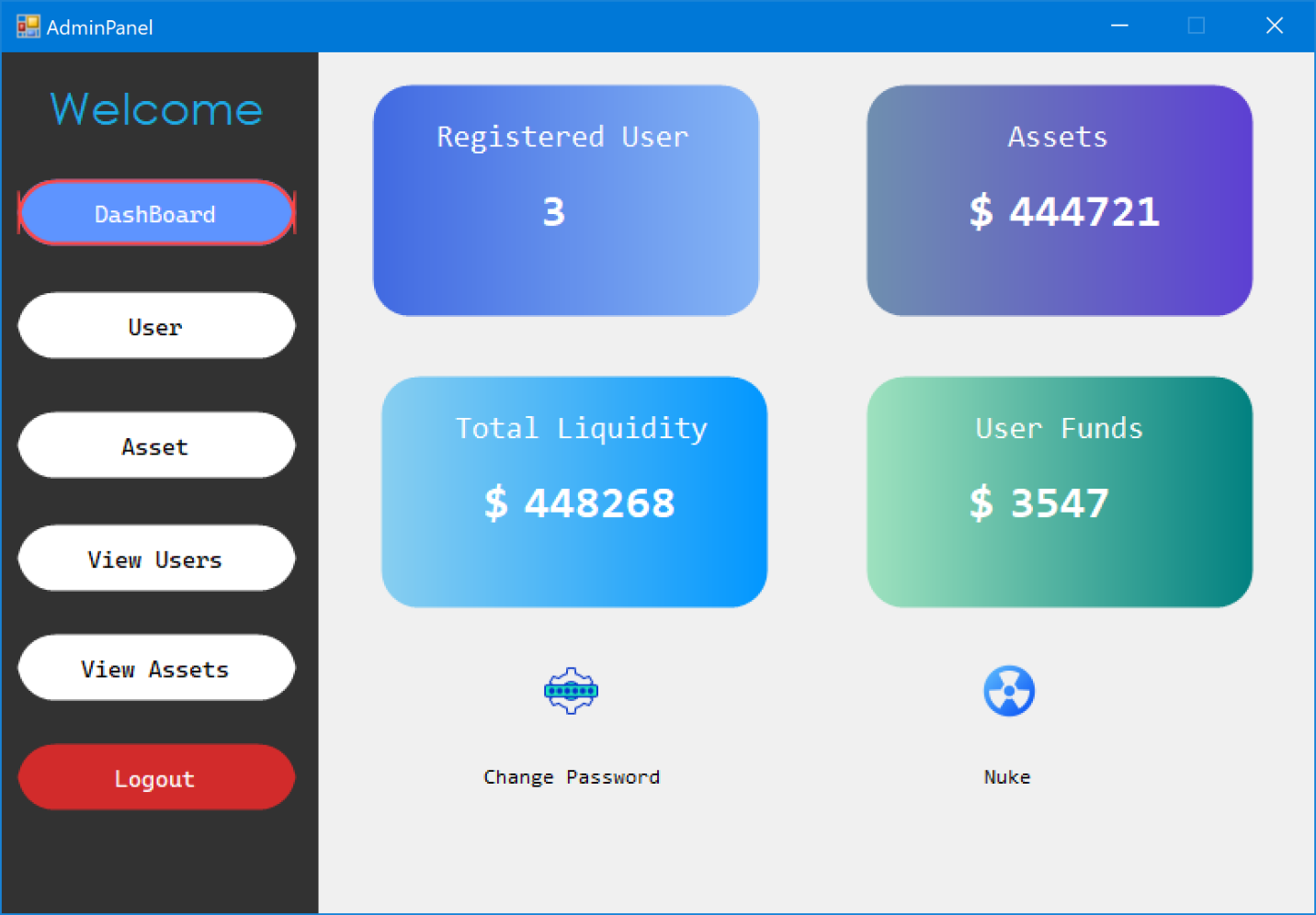


Figure 2-Gui-AdminPage

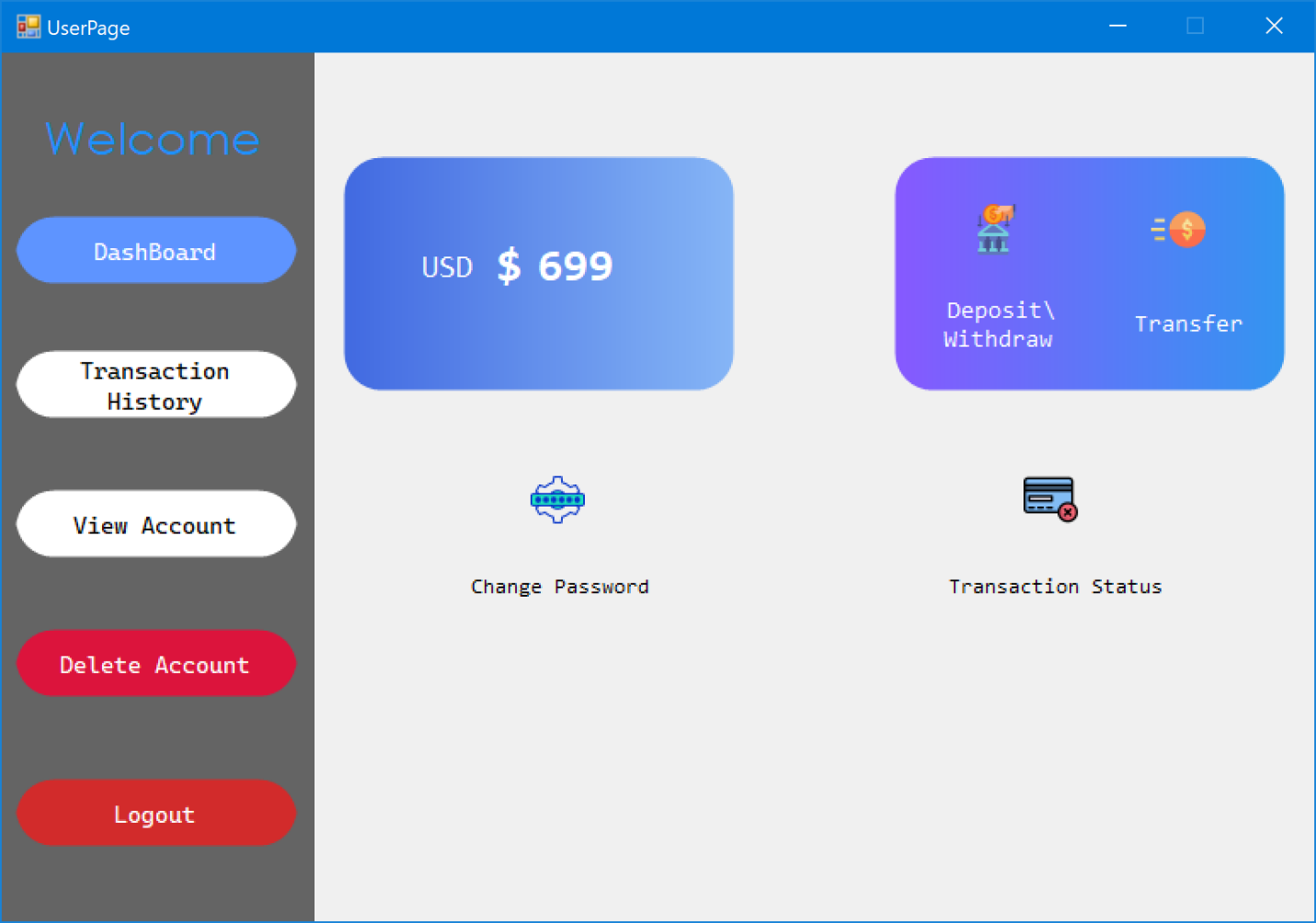


Figure 3-Gui-UserPage

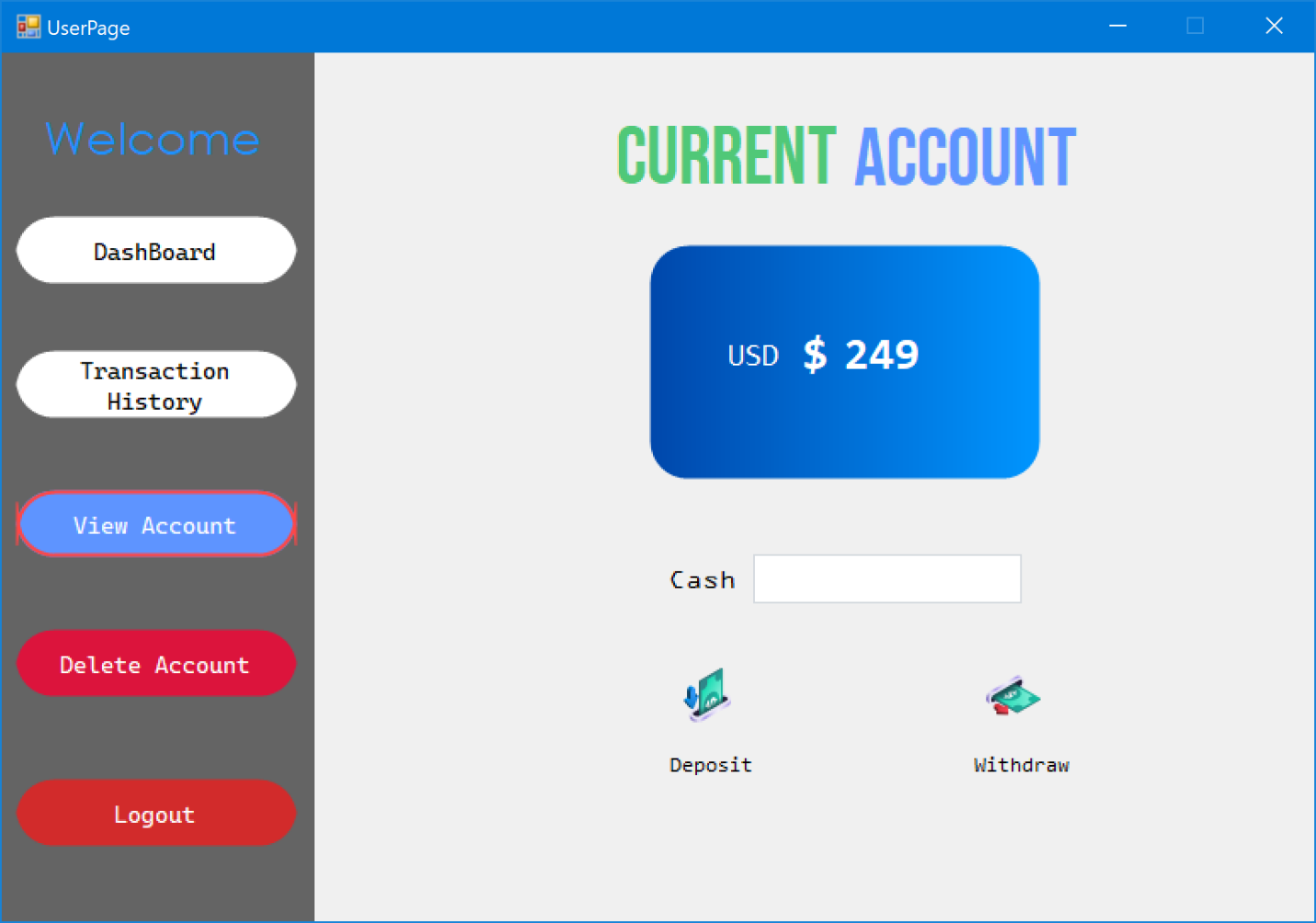


Figure 4-Gui-AccountPage

# Wireframes For Console Application

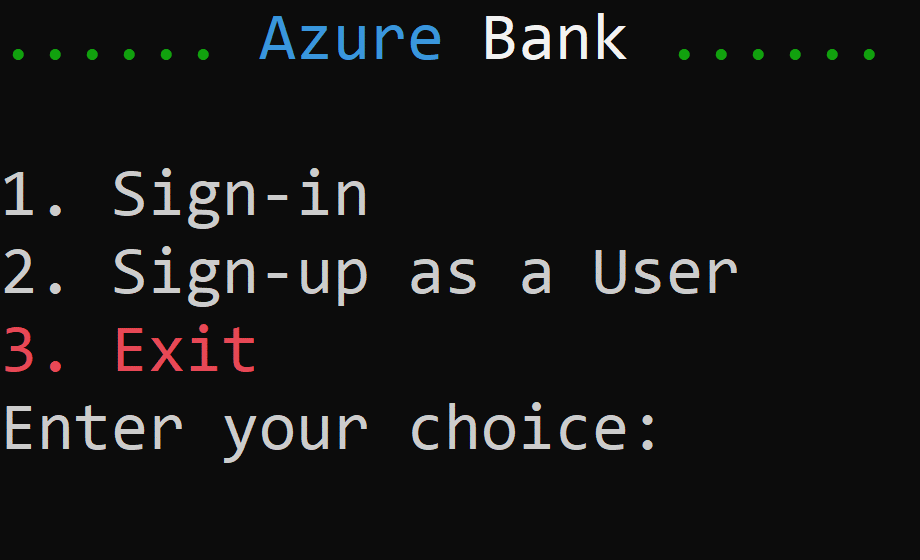


Figure 5-Console-MainPage

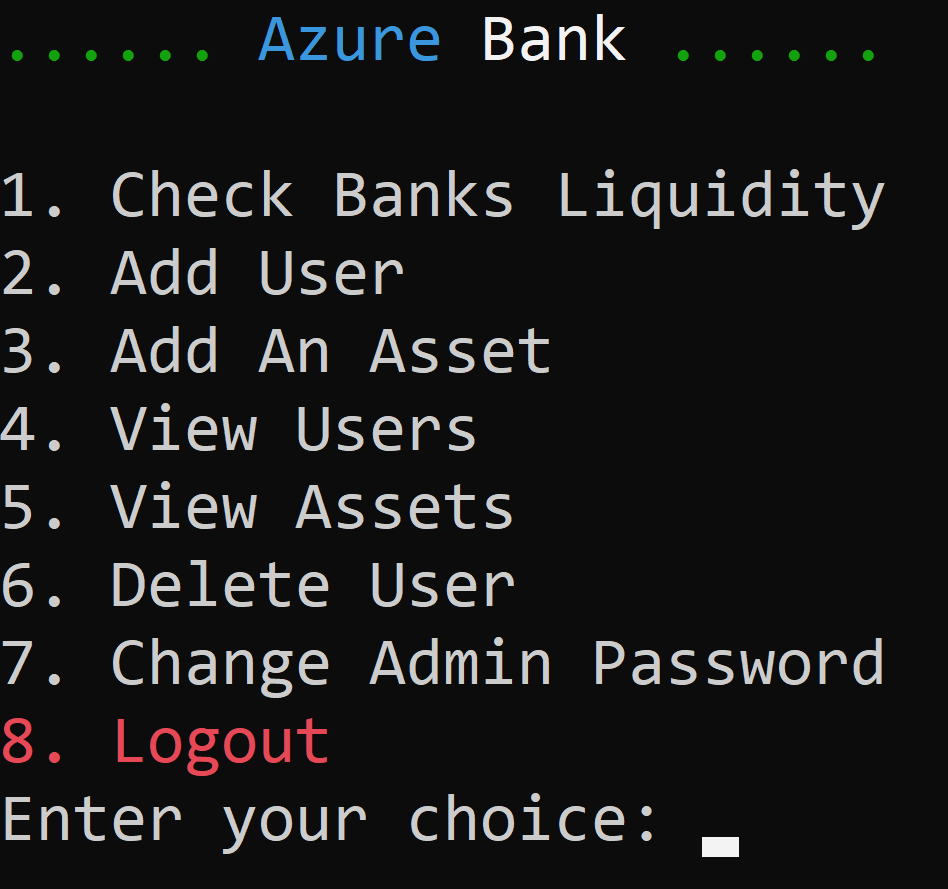


Figure 6-Console-AdminPage

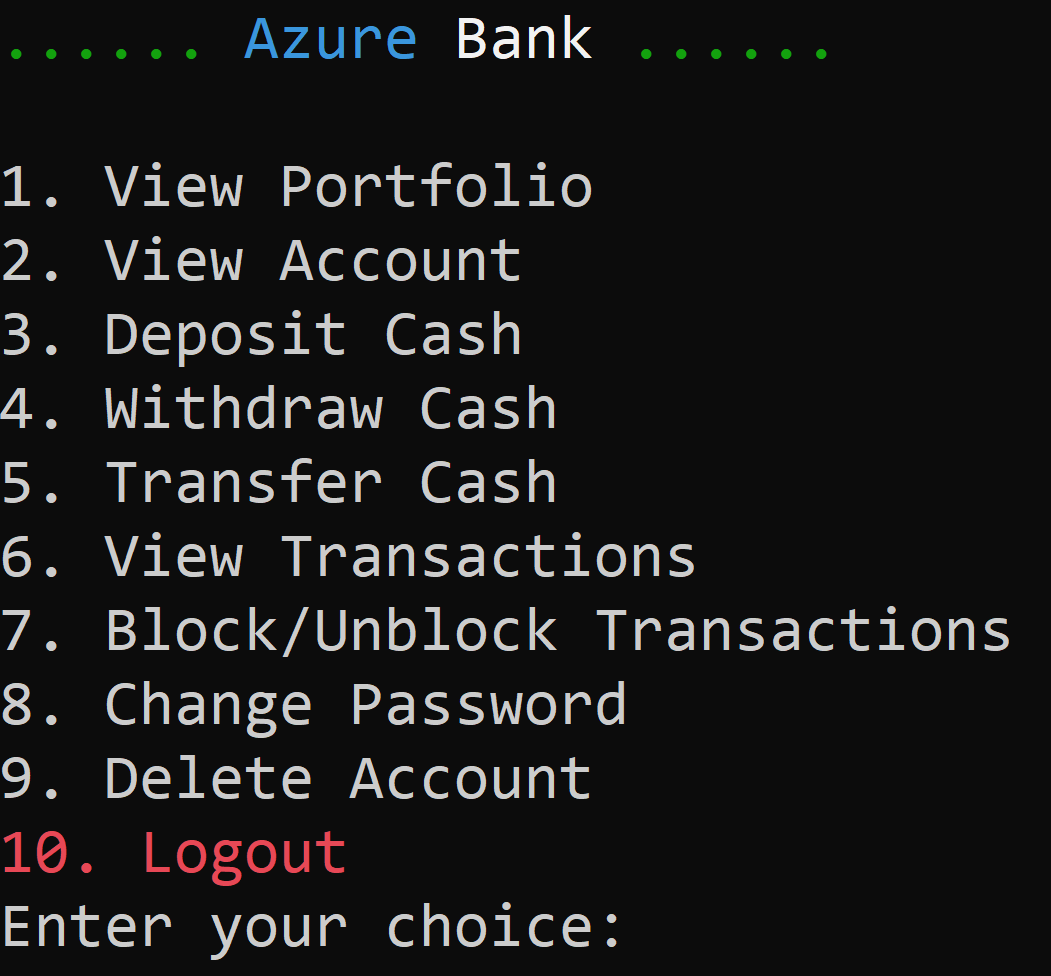
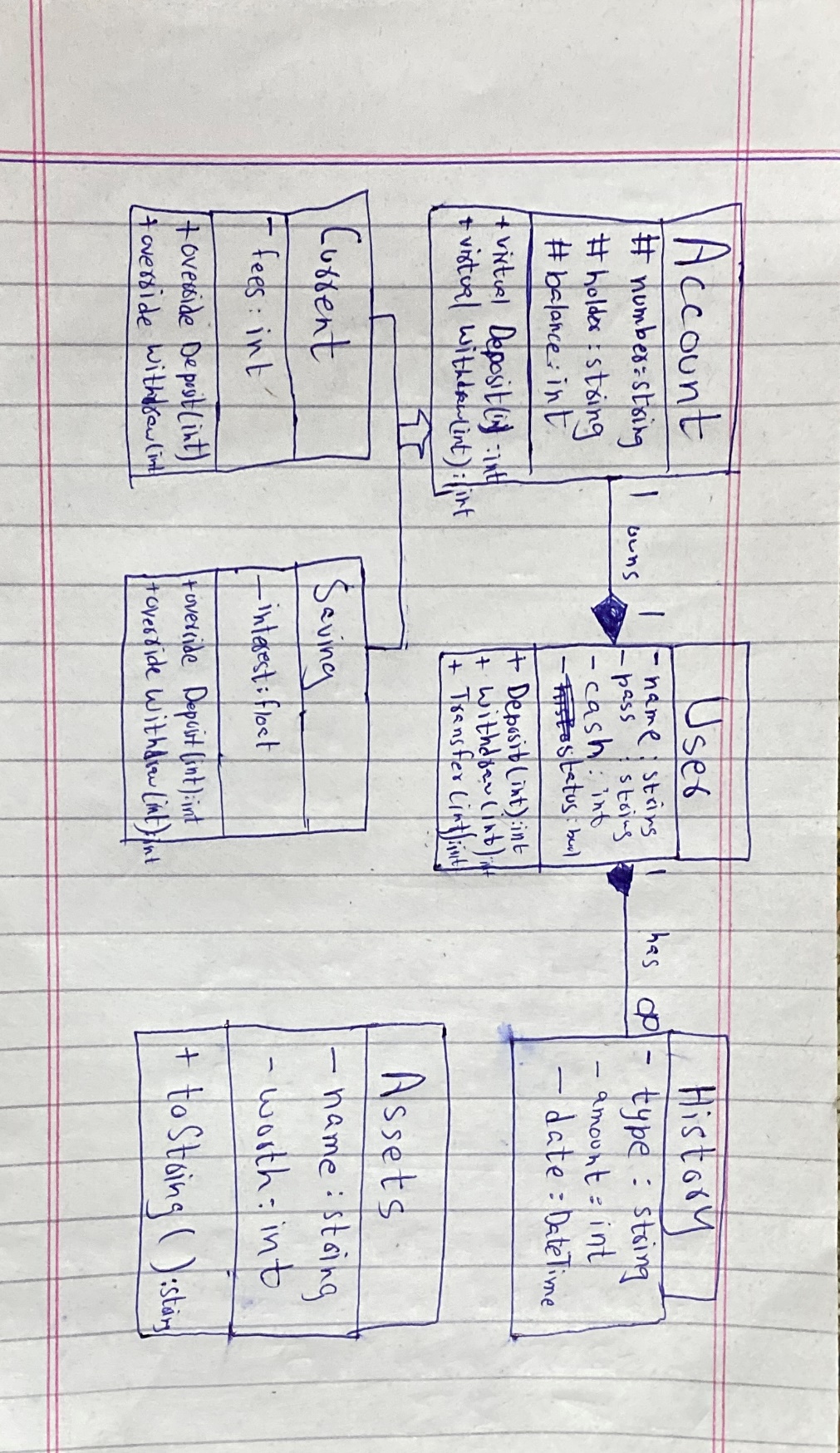


Figure 7-Console-UserPage

# Class Design (CRC)



# Complete Code

## UI:

public static int GetAmount(string type)

{

int num = 0;

Again:

Console.Write($"Enter the amount you want to {type}: $");

if (Validation.IsValidNumber(ref num))

return num; // return amount if all conditions pass

else

goto Again;

}

public static string GetAccountType()

{

Again:

Console.Write("Enter the type of account Saving(1) or Current(2)\nPress the number key: ");

char key = Console.ReadKey().KeyChar;

Console.WriteLine(); // new line

if (key == '1')

return "Saving";

else if (key == '2')

return "Current";

else

goto Again;

}

public static Account CreateAcc(string name)

{

string randomNums = UtilUi.GenerateRandomString(3);

string number = $"{name.ToLower()}{randomNums}@AzureBank";

string holderName = name;

string accType = UserUi.GetAccountType();

Again:

int amount = UserUi.GetAmount("Deposit initially: ");

if (!ValidAmount(amount))

goto Again;

UtilUi.Success("Your account has been created successfully");

if (accType == "Saving")

return new SavingAccount(number + "SAV", holderName, amount);

else

return new CurrentAccount(number + "CUR" , holderName, amount);

}

public static void CheckAccount(IAccount accountDL, User user)

{

MainUi.Header();

Account acc = accountDL.Read(user.getName());

if (acc != null)

user.setAccount(acc);

else

{

Account account = CreateAcc(user.getName());

user.setAccount(account);

accountDL.Create(account);

}

}

private static bool ValidAmount(int amount)

{

if (amount < 0 || amount == 0)

{

UtilUi.Error("Please provide valid amount", false);

return false;

}

else if (amount > 100)

{

UtilUi.Error("Initial deposit must be less than $100", false);

return false;

}

else

return true;

}

## FH:

public class AssetFH : IAsset

{

internal static string fileName;

private static AssetFH instance = null;

private AssetFH(string file)

{

fileName = file;

}

public static AssetFH getInstance(string fileName)

{

if (instance == null)

instance = new AssetFH(fileName);

return instance;

}

public bool Create(Asset asset)

{

try

{

// Open the file for appending

using (StreamWriter writer = File.AppendText(fileName))

{

// Write asset information to the file

writer.WriteLine($"{asset.getName()}, {asset.getWorth()}");

}

return true; // Successfully written

}

catch (IOException)

{

return false; // Failed to write

}

}

public List<Asset> ReadAll()

{

List<Asset> assets = new List<Asset>();

try

{

// Open the file for reading

using (StreamReader reader = new StreamReader(fileName))

{

string line;

// Read each line until the end of the file

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

{

// Split the line into name and worth

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

if (parts.Length == 2)

{

string name = parts[0].Trim();

int worth = int.Parse(parts[1].Trim());

assets.Add(new Asset(name, worth));

}

}

}

}

catch (IOException)

{

return assets;

}

return assets;

}

}

## DB:

public class AssetDB : IAsset

{

private static AssetDB instance = null;

private AssetDB() { }

public static AssetDB getInstance()

{

if (instance == null)

instance = new AssetDB();

return instance;

}

public bool Create(Asset asset)

{

DataBase.openConnection();

string query = $"INSERT INTO Assets values('{asset.getName()}', {asset.getWorth()})";

SqlCommand cmd = new SqlCommand(query, DataBase.connection);

cmd.ExecuteNonQuery();

DataBase.connection.Close();

return true;

}

public List<Asset> ReadAll()

{

DataBase.openConnection();

List<Asset> assets = new List<Asset>();

string query = "SELECT \* FROM Assets";

SqlCommand cmd = new SqlCommand(query, DataBase.connection);

SqlDataReader reader = cmd.ExecuteReader();

// read all assets from db and add them to the list

while (reader.Read())

{

string name = reader["name"].ToString();

int worth = Convert.ToInt32(reader["worth"]);

assets.Add(new Asset(name, worth));

}

DataBase.connection.Close();

return assets;

}

}