BANKING SOLUTIONS

1. Project overview

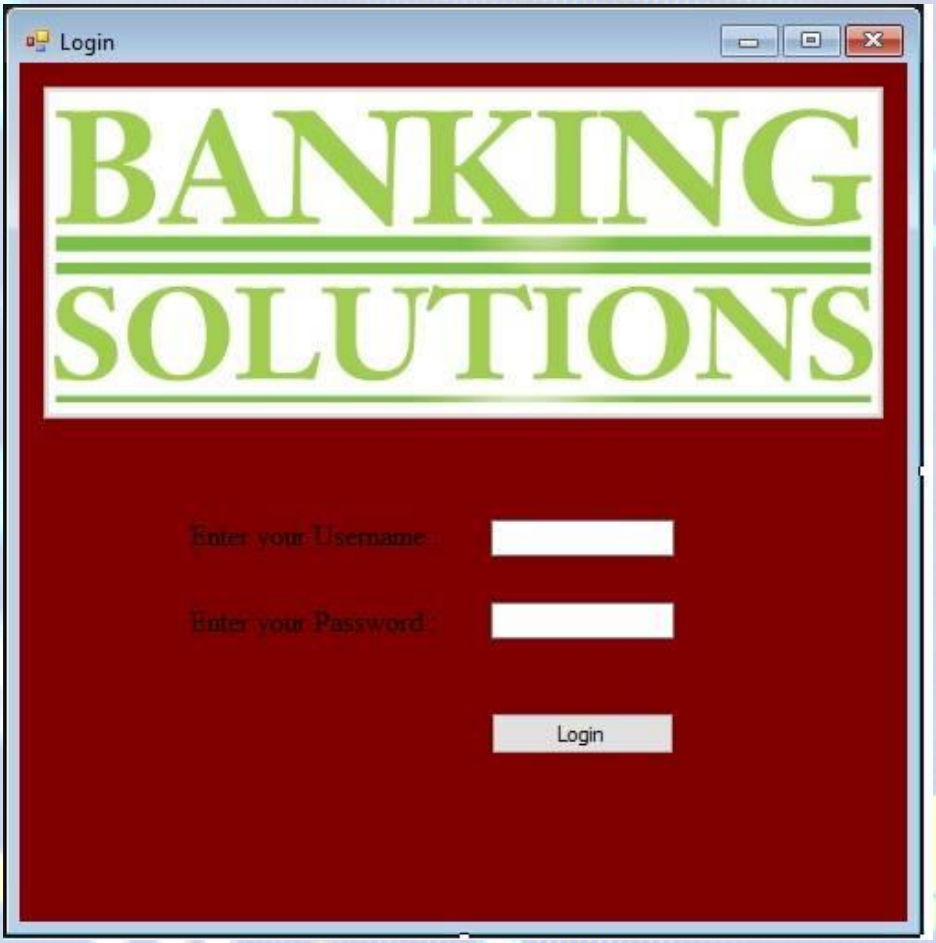
<u>1.1 Project Objective:</u> building a program that offers the following banking operations:

- **Administration of the financial accounts for individuals and legal entities**
- **Loan options**
- **The posibility of performing banking operations such as:**
  - **Deposit**
  - **Withdrawal**
  - **Transfer**
  - **Currency Exchange**
  - **Application for loan**

2. Application design

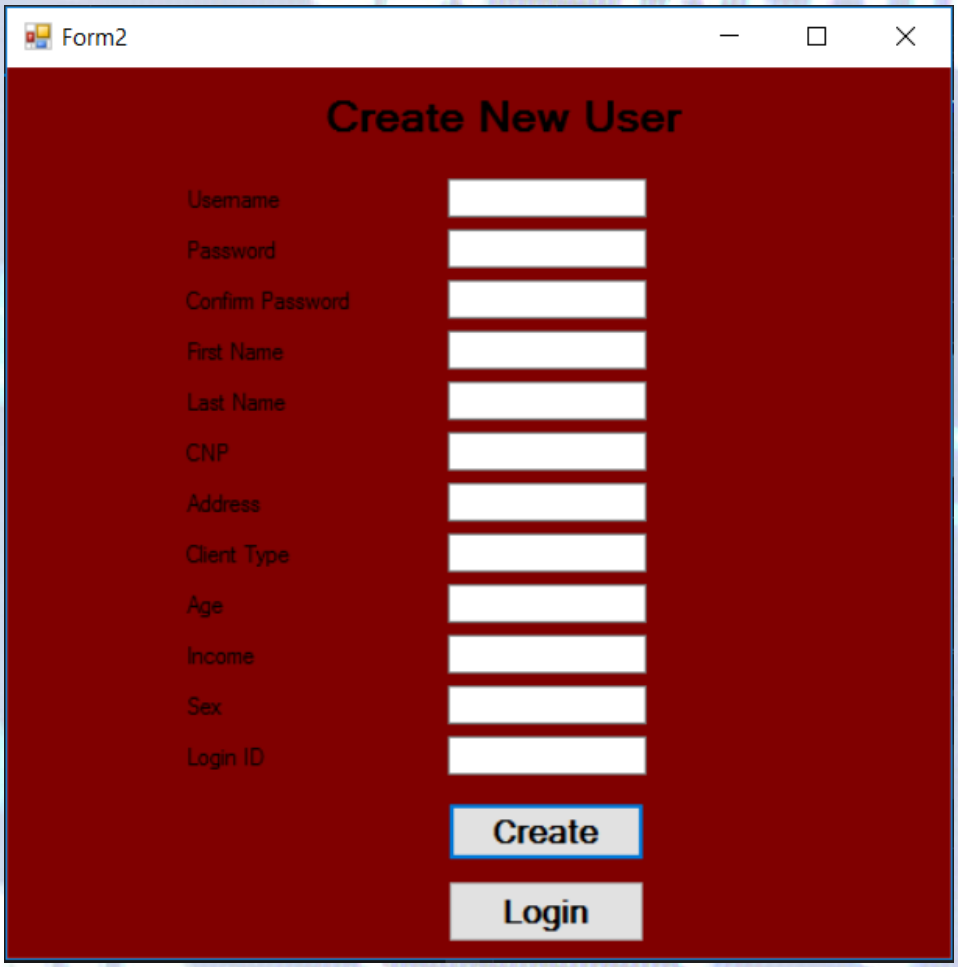
Login Form

The first Form of the application is the Login Form. It gives the user the possibility to log on their page or to create a new account.



Sign Up Form

From the Login Form (shown previously) you have the possibility to create a new user. This procedure is done on the Sign Up Form. Here you have the possibility to fill up all the Text Boxes with personal information that will be stored in the Database.

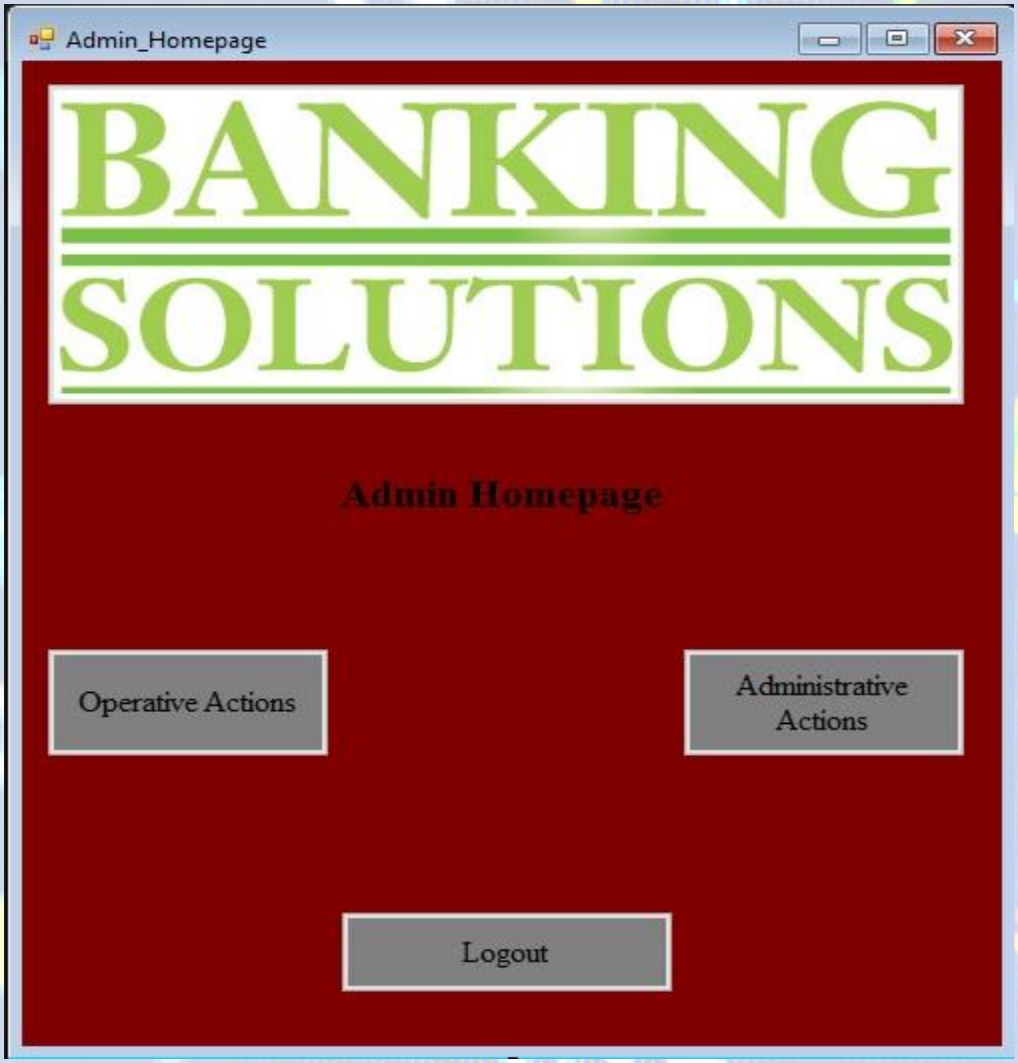


2.1 Types of users and their functions

In this application we have the possibility to login in 2 different ways: as Administrator (admin) and as a client (user).

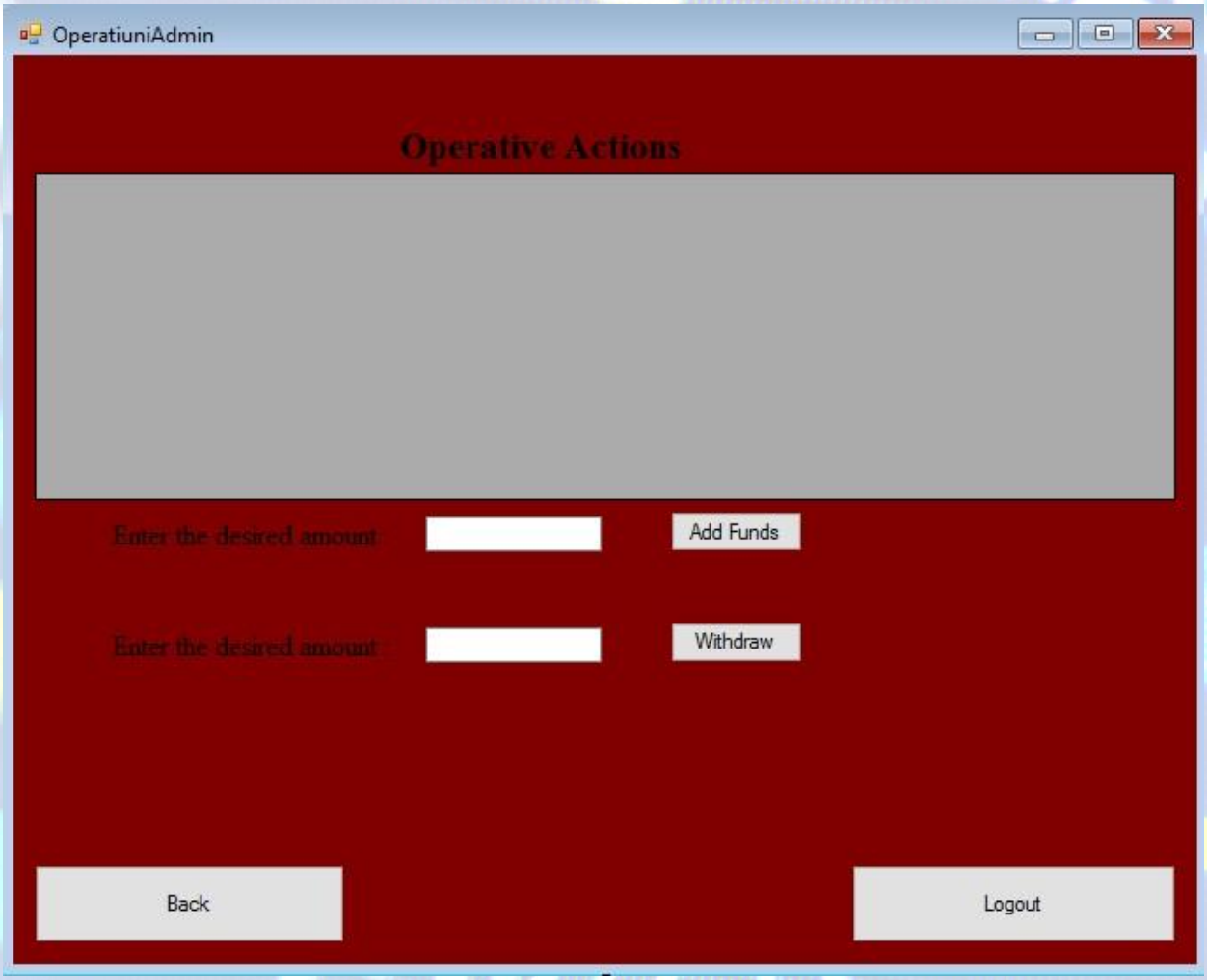
2.1.1 Administrator

Firstly, the Administrator homepage has the Bank logo and the buttons for the different actions you want to take, such as Operative Actions and Administrative Actions, among a Logout button.



As an Administrator you have two options:

1. <u>Operative</u> Actions



By selecting this option you can add or withdraw funds to/from different accounts displayed on the grid. Their account balance will be changed in the database automatically once the according buttons are pushed.

2. Administrative Actions

By selecting this option the administrator can change the password of the selected user as needed and changing it in the database automatically.

2.1.2 User

One of the most important characteristics of a bank application are the actions and the account menu.

To be able to manage the accounts and the information of the current user we created a form with multiple buttons to access all the options.

1. Account Menu

From this page the client has the possibility to access information about his current account, to perform banking operations, to read Frequently Asked Questions or to read the bank's details and history.

He also has the option to logout from his account by pressing the "Logout" button or to access the homepage by pressing the "Home" button.

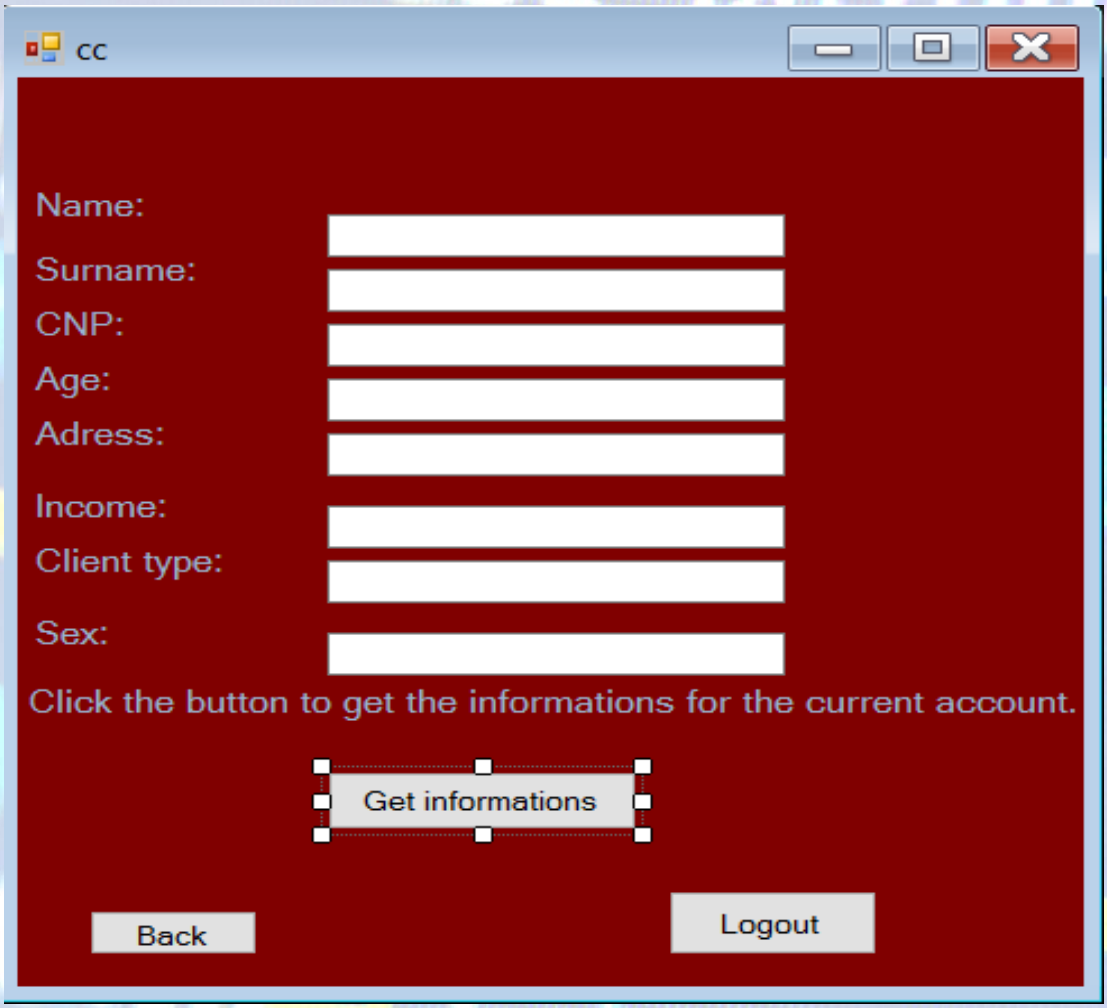
2. Operations

Here we have the Operations menu where the client (user) has the option of choosing from multiple banking operations including:

- **Verifying his balance**
- **Performing a deposit**
- **Performing a transfer**
- **Exchanging currency**

3. Current Account form

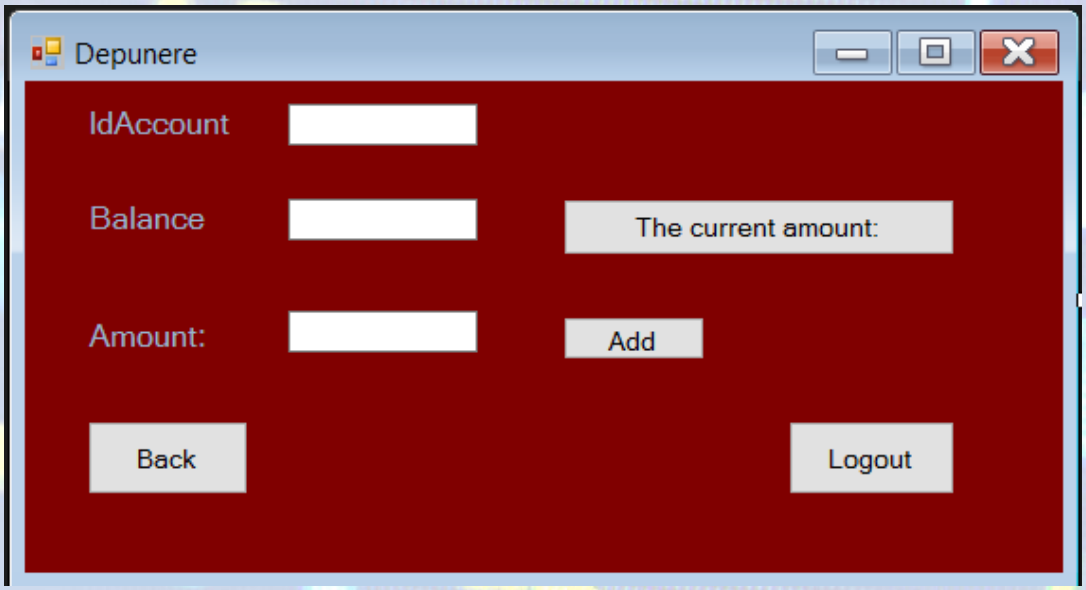
On this page the user has the possibility to verify information about his account by filling in his personal details such as Name, Surname, CNP, Age, Address, Income, Client type and Gender.



4. Deposit form

From this page the client can perform a deposit to his personal account by typing the id of the account and the amount of currency he wants to deposit.

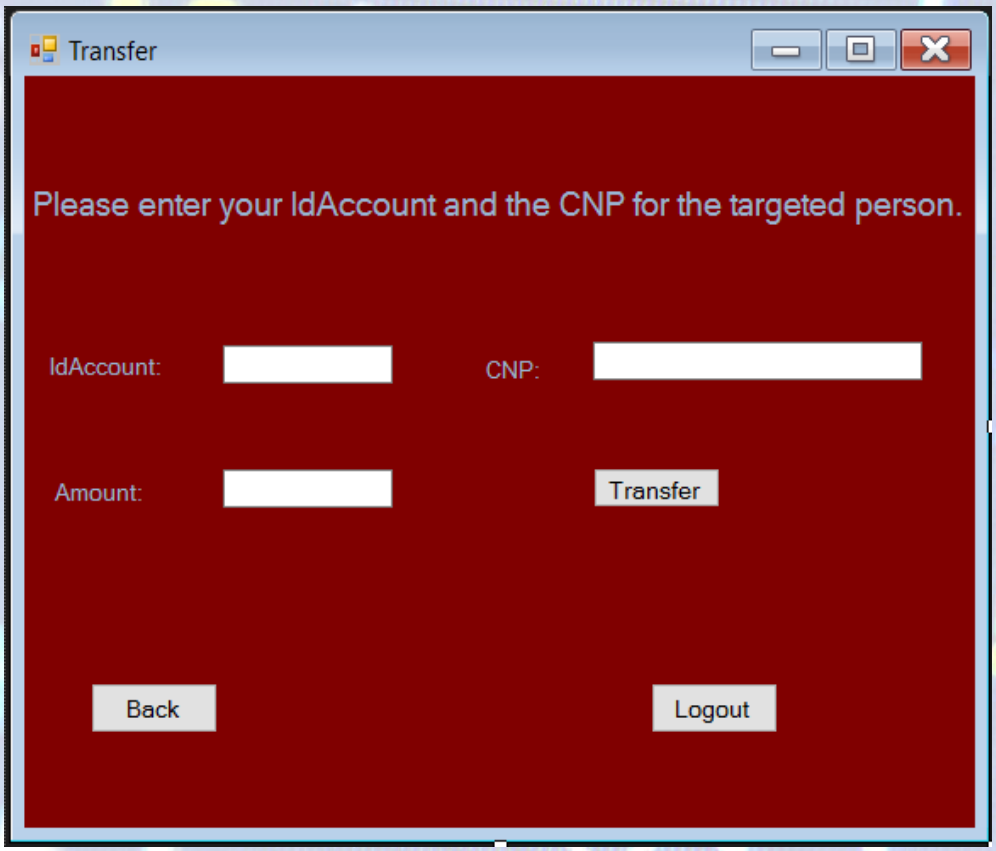
The action is finished by pressing the "Add" button.



5. Transfer form

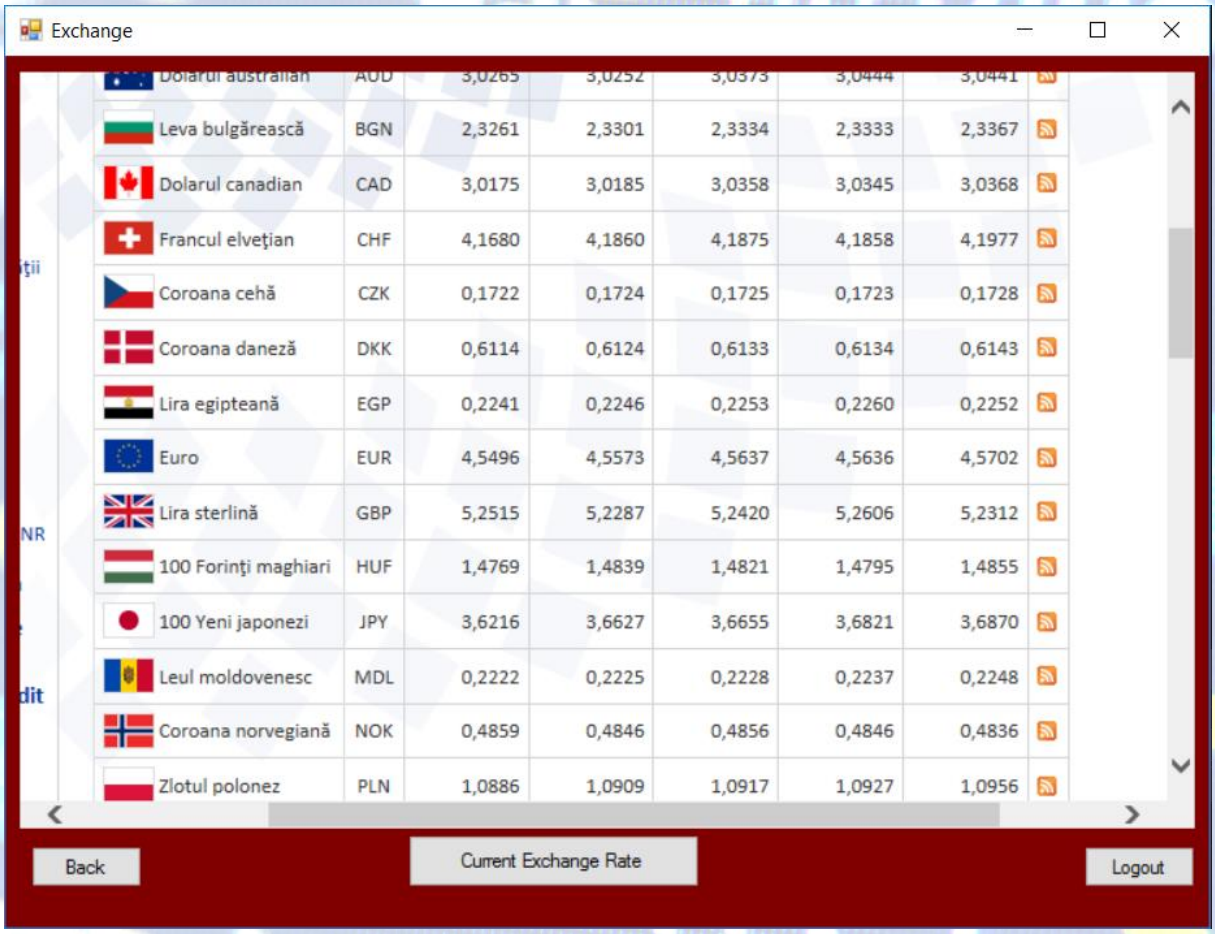
From here the client can transfer currency to a different account by adding the required information such as the id of the account, the CNP and the amount of currency he wishes to transfer.

The operation is completed by pressing the "Transfer" button.



6. Exchange page

From this page the client can check the current exchange rate of different currencies.



Dolarul australian	AUD	3,0265	3,0252	3,0373	3,0444	3,0441
Leva bulgărească	BGN	2,3261	2,3301	2,3334	2,3333	2,3367
Dolarul canadian	CAD	3,0175	3,0185	3,0358	3,0345	3,0368
Francul elveţian	CHF	4,1680	4,1860	4,1875	4,1858	4,1977
Coroana cehă	CZK	0,1722	0,1724	0,1725	0,1723	0,1728
Coroana daneză	DKK	0,6114	0,6124	0,6133	0,6134	0,6143
Lira egipteană	EGP	0,2241	0,2246	0,2253	0,2260	0,2252
Euro	EUR	4,5496	4,5573	4,5637	4,5636	4,5702
Lira sterlină	GBP	5,2515	5,2287	5,2420	5,2606	5,2312
100 Forinţi maghiari	HUF	1,4769	1,4839	1,4821	1,4795	1,4855
100 Yeni japonezi	JPY	3,6216	3,6627	3,6655	3,6821	3,6870
Leul moldovenesc	MDL	0,2222	0,2225	0,2228	0,2237	0,2248
Coroana norvegiană	NOK	0,4859	0,4846	0,4856	0,4846	0,4836
Zlotul polonez	PLN	1,0886	1,0909	1,0917	1,0927	1,0956

7. Credit application

This application is used by the current logged in user to apply for bank credits. In order to get a credit, the user must meet certain conditions.
The first condition is to have the income bigger than the credit rate.
The second condition is not to have more credits of the same credit option.
The third condition implies that the user income must be greater than all credit rates that he currently has.

Credit form class:

This class is where the graphical user interface is created and where the credit is calculated.

Firstly, the user chooses the credit type, after that he chooses the credit option, and then he can set the loan amount and the loan duration according to the credit type chosen.
When the user clicks the button “Apply for credit” a credit object is created and there, the eligibility of the user is set.

2.2 Database design

The first step in the database design was to analyze the data that would be involved into this application and determine how to use the data.

To be able to manage the data and the tables of the database, each table must have at least one field that contain some data. The fields are hidden from the user and are update automatically by the application.

Certain conventions were followed in this documentation. All table names are in **bold**, and the column names are in *italic*

Account table

The Account table was created to store information about a specific account of a bank. The fields of the table are:

Column name	Type	Index Column	Allows Nulls	Description
IdAccount	Unique identifier	Primary key	NO	This field is used to give an unique ID for each account
Currency	VARCHAR(3)		YES	This field represents the type of currency from the account. (USD,EUR,RON)
Balance	int		YES	This field presents the balance of an account
Account Type	Varchar(20)		YES	This field presents what type of account exists in our bank
IDClient	Int	Foreign Key	YES	This field links these records to the ***Client*** table.

Administrator table

This table was created to store information about the administrators of the application.

Column name	Type	Index Column	Allows Nulls	Description
IdAdmin	Int	Primary Key	NO	This field is user to identify each administrator
User1	Varchar(50)		YES	This field contain each username of each administrator
Pass1	Varchar(50)		YES	This field contain the password of each administrator
LoginID	Int		YES	This field is used to see what is the difference between users

Client table

The *Client* table stores all the necessary information about a client within the banking application.

Column name	Type	Index Column	Allows Nulls	Description
IdClient	Int	Primary Key	NO	This field stores an identification number to differentiate between users
Name	Varchar(50)		YES	This field stores the name of the client
Surname	Varchar(50)		YES	This field stores the surname of the client
CNP	Varchar(19)		YES	This field stores the personal identification number of each client
Age	Varchar(2)		YES	This field contain the age of the client
Adress	Varchar(50)		YES	This field has the purpose to store the adress of the client

User1	Varchar(50)		YES	This field contain each username of each client
Pass1	Varchar(50)		YES	This field contain the password of each client
LoginID	Int		YES	This field is used to see what is the difference between users
Income	smallint		YES	This field store the income of the client
Client Type	Varchar(20)		YES	This field store the type of the client (Retail Client, Customer Client)
Gender	Varchar(2)		YES	This field store the gender of the client

Credit table

The *Credit* table stores all the necessary information about the option to take a loan from the bank. A client can apply for a loan and he can get it related to his age, income. The loan has is determinated plan of payment in a certain amount of time (10-15 years)

Column name	Type	Index Column	Allows Nulls	Description
IdCredit	Int	Primary Key	NO	This field stores an identification number to each loan transaction
Amount	Int		YES	This field stores the sum which is requested for a loan
Eligible	Vachar(5)		YES	This field set if the client is eligible for the loan
Interest Rate	Varchar(5)		YES	This field stores the interest rate of the loan
Term	Varchar(10)		YES	This field contain the term when the loan should be fi

Loan Type	Varchar(30)		YES	This field stores the type of the loan (Personal Loan, Car Loan)
Currency	Varchar(10)		YES	This field stores the currency of the loan
Id_Client	Int	Foreign Key	YES	This field links up with the ***Client*** table

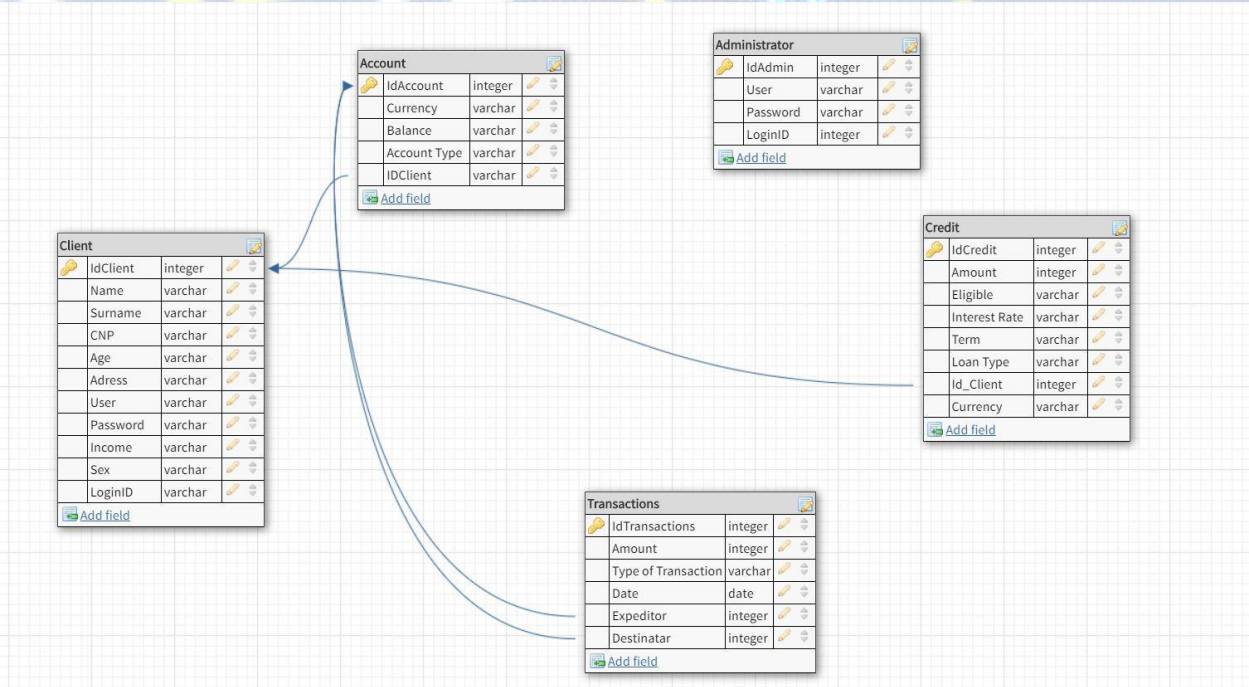
Transactions table

The table *Transactions* contains all the information about the transactions from the bank. There are different types of transactions, one of them can be made by the user, and some of them can pe made by the admin. The transactions are refund, transfer, exchange.

Column name	Type	Index Column	Allows Nulls	Description
IdTransactions	Int	Primary Key	NO	This field stores each transaction from the bank
Amount	Int		YES	This field contain the amount of money which is involved in a transaction
Type of Transaction	Varchar(50)		YES	This field contain the type of transaction which is can be done
Date	Date		YES	This field stores the data when a transaction is made
Expeditor	Int	Foreign Key	YES	This field is linked with ***Client*** table

Destinatar	Int	Foreign Key	YES	This field is linked with ***Client*** table

Database Diagram



3. Application implementation

3.1 Database implementation

SQL Server Management Studio (SSMS) is an integrated environment for managing any SQL infrastructure, from SQL Server to SQL Database. SSMS provides tools to configure, monitor, and administer instances of SQL from wherever you deploy it. SSMS provides tools to deploy, monitor, and upgrade the data-tier components, such as databases and data warehouses used by your applications, and to build queries and scripts.

SQL is a special programming language design for storing, managing, manipulatin and retrieving data from a relational database management system.

The main scope of SQL is to include methods like insert, query, update, delete, schema creation, and data access control.

The database has been implemented using Microsoft SQL Server Management Studio 2014. Below, we will present how we implemented each table.

3.1.1. ***Account*** table implementation

- IdAccount (PK, int, not null)
- Currency (varchar(3), null)
- Balance (int, null)
- Account Type (varchar(20), null)
- IDClient (FK, int, null)

The *Acount* table was created to store specific information about each account from the bank. This table does not have so many fields, because we think that these five fields sums all the information about an account.

```sql
CREATE TABLE [dbo].[Account] (
    [IdAccount]    INT        NOT NULL,
    [Currency]     varchar (3) NULL,
    [Balance]      int NULL,
    [Account Type] varchar (20) NULL,
    [IDClient]     INT        NULL,
    CONSTRAINT [PK_Account] PRIMARY KEY CLUSTERED ([IdAccount] ASC),
    CONSTRAINT [FK_Account_ToTable] FOREIGN KEY ([IDClient]) REFERENCES [dbo].[Client] ([IdClient])
);
```

3.1.2 ***Administrator*** table implementation

- IdAdmin (PK, int, not null)
- User1 (varchar(50), null)
- Pass1 (varchar(50), null)
- LoginID (int, null)

The *Administrator* table consist of having few fields, because these fields reveals all the information that we need to know about an administrator.

```sql
CREATE TABLE [dbo].[Administrator] (
    [IdAdmin] INT          NOT NULL,
    [User]    VARCHAR (50) NULL,
    [Parola]  VARCHAR (50) NULL,
    [LoginID] INT          NULL,
    PRIMARY KEY CLUSTERED ([IdAdmin] ASC)
);
```

3.1.3 ***Client*** table implementation

- IdClient (PK, int, not null)
- Name (varchar(50), null)
- Surname (varchar(50), null)
- CNP (varchar(19), null)
- Age (varchar(2), null)
- Adress (varchar(50), null)
- User1 (varchar(50), null)
- Pass1 (varchar(50), null)
- Income (smallint, null)
- ClientType (varchar(20), null)
- Sex (varchar(2), null)
- LoginID (int, null)

This table represent the main table of our database. We thought that the fields that are presented in figure reveals all the important information about the client. We believe that we need to know the income, the age of a client in order to establish if the client is eligible for a loan.

```sql
CREATE TABLE [dbo].[Client] (
    [IdClient]    INT           NOT NULL,
    [Name]        VARCHAR (50)  NULL,
    [Surname]     VARCHAR (50)  NULL,
    [CNP]         VARCHAR (19)  NULL,
    [Age]         varchar(2) NULL,
    [Adress]      VARCHAR (50)  NULL,
    [User]        VARCHAR (50)  NULL,
    [Password]    VARCHAR (50)  NULL,
    [Income]      SMALLINT  NULL,
    [Client Type] VARCHAR (20)  NULL,
    [Sex]         VARCHAR (2)  NULL,
    [LoginID]     INT           NULL,
    CONSTRAINT [PK_Client] PRIMARY KEY CLUSTERED ([IdClient] ASC)
);
```

3.1.4 ***Credit*** table implementation

- IdCredit (PK, int, not null)
- Amount (int, null)
- Eligible (varchar(5), null)
- Interest Rate (varchar(5), null)
- Term (varchar(10), null)
- Loan Type (varchar(30), null)
- Id_Client (FK, int, null)
- Currency (varchar(10), null)

Credit **table has the fields presented in figure. We think that all the fields sum up the information that we need to know about a loan. Using Id_Client field, we build a link-up with *Client* table.**

```
CREATE TABLE [dbo].[Credit] (
    [IdCredit]      INT          NOT NULL,
    [Amount]        INT          NULL,
    [Eligible]      VARCHAR (5)  NULL,
    [Interest Rate] VARCHAR (5)  NULL,
    [Term]          VARCHAR (10) NULL,
    [Loan Type]     VARCHAR (30) NULL,
    [Id_Client]     INT          NULL,
    [Currency]      varchar (10)   NULL,
    CONSTRAINT [IdCredit] PRIMARY KEY CLUSTERED ([IdCredit] ASC),
    CONSTRAINT [IdAccount] FOREIGN KEY ([Id_Client]) REFERENCES [dbo].[Client] ([IdClient])
);
```

3.1.5 ***<u>Transactions</u>*** table

- IdTransactions (PK, int, not null)
- Amount (int, null)
- Type of Transaction (varchar(50), null)
- Date (date, null)
- Expeditor (FK, int, null)
- Destinatar (FK, int, null)

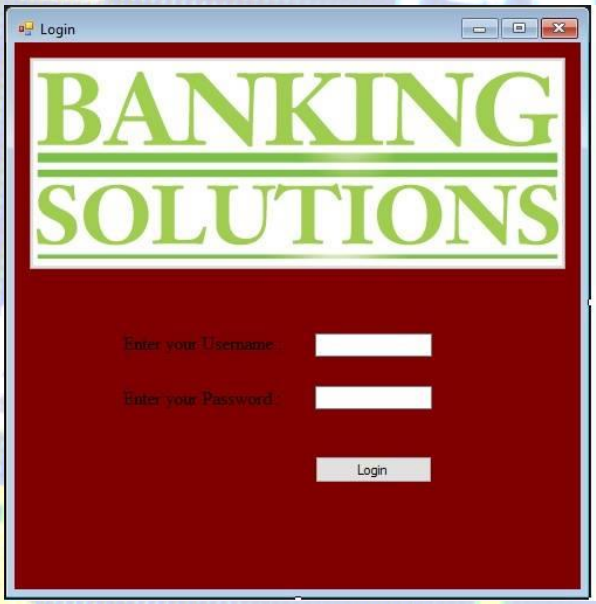
In this table we store the information about every transaction which is made inside the bank. Using *Expeditor* and *Destintar* fields we create a link with the *Account* table, if it is necessary to transfer an amount of money from an account to another.

```sql
CREATE TABLE [dbo].[Transactions] (
    [IdTransactions]      INT        NOT NULL,
    [Amount]              NCHAR (10) NULL,
    [Type of Transaction] NCHAR (10) NULL,
    [Date]                DATE       NULL,
    [Expeditor]           INT        NULL,
    [Destinatar]          INT        NULL,
    PRIMARY KEY CLUSTERED ([IdTransactions] ASC),
    CONSTRAINT [Id.Account] FOREIGN KEY ([Expeditor]) REFERENCES [dbo].[Account] ([IdAccount]),
    CONSTRAINT [FK_Transactions_ToTable] FOREIGN KEY ([Destinatar]) REFERENCES [dbo].[Account] ([IdAccount])
);
```

3.2 Implementation Details

Here we mostly have the source code behind all the forms and operations.

3.2.1 Login Form



Source code:

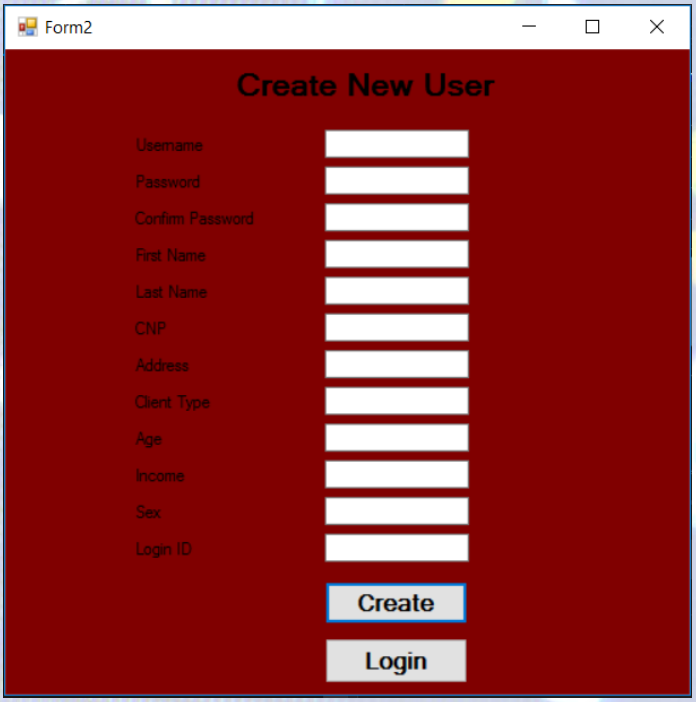
****

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace User_Homepage
{
    public partial class Login : Form
    {

            public Login()
```

```
 {
        InitializeComponent();
    }

    private void Login_Load(object sender, EventArgs e)
    {

    }

        private void LoginFunction()
        {
        SqlConnection abcdata = new SqlConnection(@"Data Source=DESKTOP-QBIQDJ9\SQLEXPRESS;Initial
Catalog=Database;Integrated Security=True");
        SqlDataAdapter asdf = new SqlDataAdapter("select [IdClient] from [Client] where [User] = '" +
textBox1.Text + "' AND     [Password]='" + textBox2.Text + "'", abcdata);
        DataTable ss = new DataTable();

        SqlDataAdapter asdfg = new SqlDataAdapter("select * from [Administrator] where [User] = '" +
textBox1.Text + "' AND     [Parola]='" + textBox2.Text + "'", abcdata);
        DataTable adminDT = new DataTable();

        asdf.Fill(ss);
        asdfg.Fill(adminDT);
        if (ss.Rows.Count != 0)
        {
           //Conturi con = new Proiect.Conturi(int.Parse(ss.Rows[0][0].ToString()));
           //con.Show();

        }
        else if (adminDT.Rows.Count != 0)
        {
           Admin_Homepage ah = new Admin_Homepage();
           this.Hide();
           ah.ShowDialog();
           this.Close();
        }
        else
           MessageBox.Show("Invalid username or password!");
           }
    private void button1_Click(object sender, EventArgs e)
    {
                LoginFunction();
    }
  }
}
*******
```

The function <u>LoginFunction</u> compares the information from Text Boxes with the information from the database and allows or denies access to the application. It also retrieves the ID of the client.

3.2.2 Sign Up Form



Source code:

****

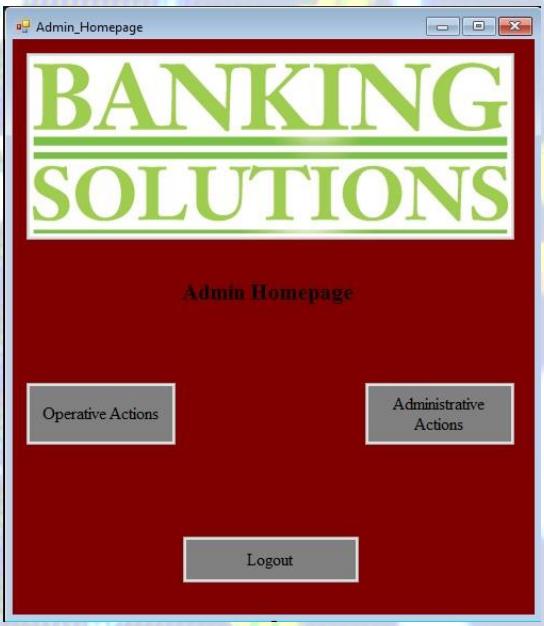
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace IIHoria
{
    public partial class Form2 : Form
    {
        public Form2()
        {
            InitializeComponent();
        }

        private void label2_Click(object sender, EventArgs e)
        {
        }
        private void button2_Click(object sender, EventArgs e)
        {
            closeSignUpForm();
        }
        public void closeSignUpForm()
        {
```

```
        this.Close();
    }
    private void CreateButton_Click(object sender, EventArgs e)
    {
        CreateUser();
    }
    public void CreateUser()
    {
        using (SqlConnection con = new SqlConnection("Data Source=DESKTOP-PJH0QFO\\SQLEXPRESS;Initial
Catalog=Database;Integrated Security=True"))
        {
            con.Open();
SqlCommand cmd = new SqlCommand("INSERT INTO Client (Name, Surname, CNP, Adress, User1, Password1,
Age, Income, Sex, LoginID) VALUES (@Name, @Surname, @CNP, @Adress, @User, @Password, @Age,
@Income, @Sex, @LoginID)", con);
            cmd.Parameters.AddWithValue("@Name", FirstNameTextBox.Text);
            cmd.Parameters.AddWithValue("@Surname", LastNameTextBox.Text);
            cmd.Parameters.AddWithValue("@CNP", CnpTextBox.Text);
            cmd.Parameters.AddWithValue("@Age", AgeTextBox.Text);
            cmd.Parameters.AddWithValue("@Adress", AddressTextBox.Text);
            cmd.Parameters.AddWithValue("@User", UsernameTextBox.Text);
            cmd.Parameters.AddWithValue("@Password", PasswordTextBox.Text);
            cmd.Parameters.AddWithValue("@Income", IncomeTextBox.Text);
            cmd.Parameters.AddWithValue("@Sex", SexTextBox.Text);
            cmd.Parameters.AddWithValue("@LoginID", LoginIDTextBox.Text);
            cmd.Parameters.AddWithValue("@Type", ClientTypeTextBox.Text);
            cmd.ExecuteNonQuery();
            MessageBox.Show("user registered");
        }

        Console.ReadKey();

    }

    private void Form2_Load(object sender, EventArgs e)
    {
    }
  }
}
```

****

The function <u>CreateUser</u> retrieves the information from Text Boxes and stores it in the database. The function is called by clicking on the Create Button.
The LOGIN Button gives the user the possibility to go back to Login Form by closing the current Form.

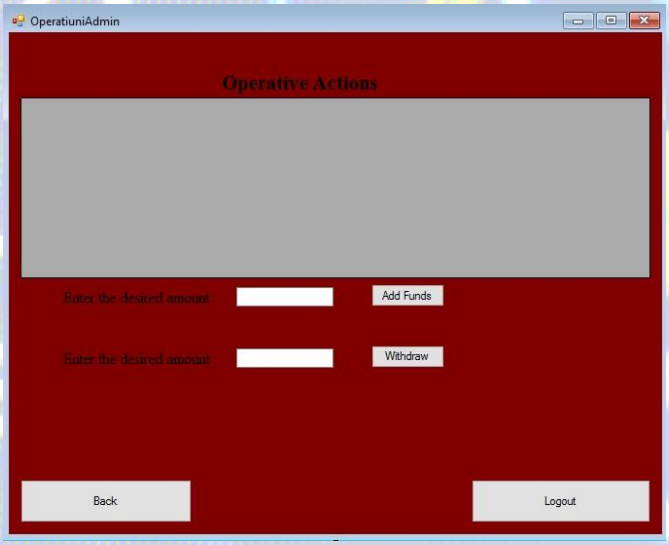
3.2.3 Administrator homepage



Source code:

```
namespace User_Homepage
{
    public partial class Admin_Homepage : Form
    {
        SqlConnection myCon;
        string connectionString;
        public Admin_Homepage()
        {
            InitializeComponent();
            connectionString = @"Data Source=DESKTOP-QBIQDJ9\SQLEXPRESS;Initial Catalog=Database;Integrated Security=True";
            myCon = new SqlConnection(connectionString);
        }


        private void Admin_Homepage_Load(object sender, EventArgs e)
        {

        }

        private void button1_Click(object sender, EventArgs e)
        {

        }


        private void button3_Click(object sender, EventArgs e)      //Operatiuni
        {
            OperatiuniAdmin myForm = new OperatiuniAdmin();
            this.Hide();
            myForm.ShowDialog();
            this.Close();
        }



        private void button6_Click(object sender, EventArgs e)      //Administrare
        {
            Admin_Informatii_Client myForm = new Admin_Informatii_Client();
            this.Hide();
            myForm.ShowDialog();
            this.Close();
        }


        private void button10_Click(object sender, EventArgs e)     //Logout
        {
            Login myform = new Login();
            this.Hide();
            myform.ShowDialog();
            this.Close();
```

3.2.4 Operative Actions of the Administrator



Source code:

```
22
23          private void OperatiuniAdmin_Load(object sender, EventArgs e)
24          {
25              p.ListAccounts(dataGridView1);
26          }
27
28          private void button1_Click(object sender, EventArgs e)
29          {
30              if (idAccount != 0)
31              {
32                  p.AddBalance(idAccount, Int32.Parse(textBox1.Text));
33                  p.ListAccounts(dataGridView1);
34              }
35              textBox1.Text = "";
36              textBox2.Text = "";
37              idAccount = 0;
38          }
39
40          private void dataGridView1_CellContentClick(object sender, DataGridViewCellEventArgs e)
41          {
42              idAccount = Int32.Parse(dataGridView1.CurrentRow.Cells[0].Value.ToString());
43          }
44
45          private void button2_Click(object sender, EventArgs e)
46          {
47              if (idAccount != 0)
48              {
49                  p.RemoveBalance(idAccount, Int32.Parse(textBox2.Text));
50                  p.ListAccounts(dataGridView1);
51              }
52              textBox2.Text = "";
53              textBox1.Text = "";
54              idAccount = 0;
55          }
56
57          private void button3_Click(object sender, EventArgs e)
58          {
59              Admin_Homepage myForm = new Admin_Homepage();
60              this.Hide();
61              myForm.ShowDialog();
62              this.Close();
63          }
64
65          private void button4_Click(object sender, EventArgs e)
66          {
67              Login myForm = new Login();
68              this.Hide();
69              myForm.ShowDialog();
70              this.Close();
71          }
```

3.2.5 Administrative Actions of the Administrator

Source code:

```
public partial class Admin_Informatii_Client : Form
{
    int idClient;
    pocol poc = new pocol();
    public Admin_Informatii_Client()
    {
        InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e)
    {
        Admin_Homepage myForm = new Admin_Homepage();
        this.Hide();
        myForm.ShowDialog();
        this.Close();
    }

    private void Admin_Informatii_Client_Load(object sender, EventArgs e)
    {
        poc.ListClients(dataGridView1);
    }



    private void dataGridView1_CellContentClick(object sender, DataGridViewCellEventArgs e)
    {
        idClient = Int32.Parse(dataGridView1.CurrentRow.Cells[0].Value.ToString());
    }

    private void button2_Click(object sender, EventArgs e)
    {
        if (textBox1.Text != "")
        {
            if (idClient != 0) poc.UpdatePassword(idClient, textBox1.Text);
        }
        else MessageBox.Show("Invalid password");
        poc.ListClients(dataGridView1);
        textBox1.Text = "";
        idClient = 0;
    }

    private void button3_Click(object sender, EventArgs e)
    {
        Admin_Homepage myForm = new Admin_Homepage();
        this.Hide();
        myForm.ShowDialog();
        this.Close();
    }

    private void button4_Click(object sender, EventArgs e)
```

Concerning the methods used in the operative and administrative windows, we created another class containing them besides the connections to the database.

```
class pocol
{

    string connS = @"Data Source=DESKTOP-Q8IQDJ9\SQLEXPRESS;Initial Catalog=Database;Integrated Security=True";
    public void ListAccounts(DataGridView dgv)
    {
        DataTable data = new DataTable();

        data.Columns.Add("IdAccount", typeof(int));
        data.Columns.Add("Name", typeof(string));
        data.Columns.Add("Surname", typeof(string));
        data.Columns.Add("CNP", typeof(string));
        data.Columns.Add("Curency", typeof(string));
        data.Columns.Add("Balance", typeof(int));
        data.Columns.Add("Account Type", typeof(string));

        using (SqlConnection connection = new SqlConnection(connS))
        {
            using (SqlCommand command = new SqlCommand("SELECT a.IdAccount,c.Name,c.Surname,c.CNP,a.Currency,a.Balance,a.[Account Type] FROM Ac
            {
                connection.Open();
                using (var reader = command.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        int id, ba;
                        string na, sn, cnp, cu, at;
                        id = reader.GetInt32(0);
                        na = reader.GetString(1);
                        sn = reader.GetString(2);
                        cnp = reader.GetString(3);
                        cu = reader.GetString(4);
                        ba = reader.GetInt32(5);
                        at = reader.GetString(6);
                        data.Rows.Add(id, na, sn, cnp, cu, ba, at);
                    }
                }
            }
        }

        dgv.DataSource = data;
        dgv.Columns[0].Visible = false;
    }

    public void AddBalance(int id, int sum)
    {
        using (SqlConnection connection = new SqlConnection(connS))
        {
            using (SqlCommand command = new SqlCommand("UPDATE Account SET Balance=Balance+@sum WHERE IdAccount=@id", connection))
            {
                command.Parameters.AddWithValue("@sum", sum);
                command.Parameters.AddWithValue("@id", id);

                connection.Open();
                command.ExecuteNonQuery();
                connection.Close();
            }
        }
    }

    public void RemoveBalance(int id, int summ)
    {
        using (SqlConnection connection = new SqlConnection(connS))
```

These are be the Administrative options presented above.

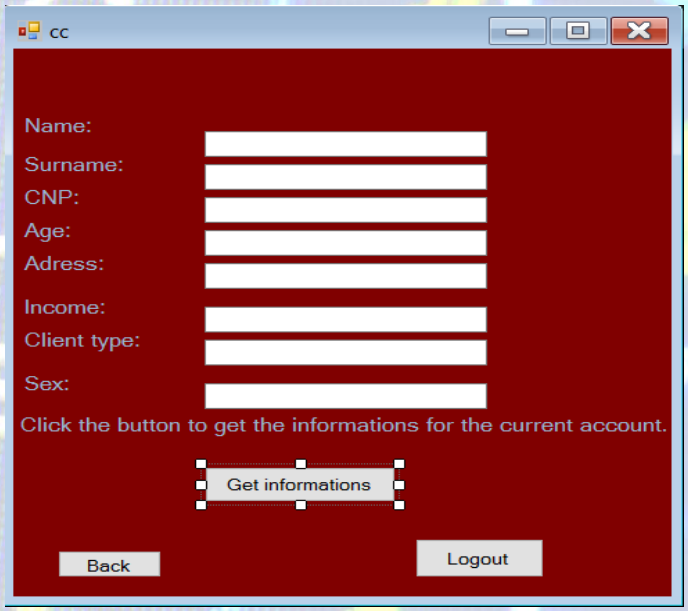
3.2.6 User (client) Account Menu

Source code:

```
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Proiect
{
    public partial class Conturi : Form
    {
        int IdClient;
        public Conturi(int IdClient)
        {
            InitializeComponent();
            this.IdClient = IdClient;

        }

        private void Conturi_Load(object sender, EventArgs e)
        {

        }



        private void button9_Click(object sender, EventArgs e)
        {
            this.Hide();
            Operatiuni op = new Operatiuni(IdClient);
            op.Show();

```

We mainly used the same method to hide the current form and to Show the next one on buttons. And a specific property would be the attribute “IdClient” which we use in our Operations. To generate information for the current user depending on the value from “IdClient”.

3.2.7 User (client) Current Account Form

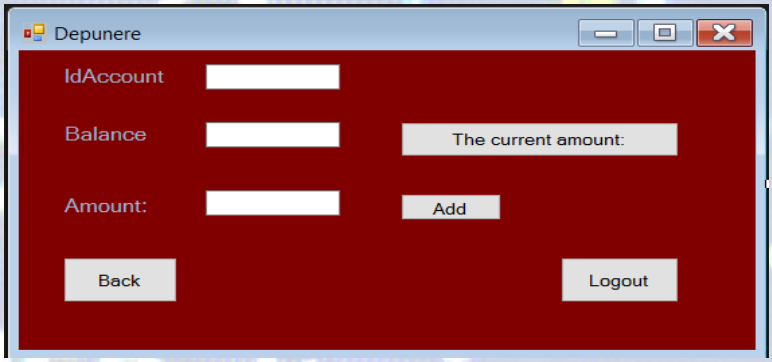


Source code:

```
*******
int IdClient;// we declare the attribute which is inherited
SqlConnection abcdata = new SqlConnection(@"Data Source=DESKTOPU7518PG\SQLEXPRESS;Initial Catalog=Database;Integrated Security=True");//establishing the connection by  defining the path

// in the button we have the next code lines

SqlDataAdapter asdf = new SqlDataAdapter("select * from Client where IdClient=" + IdClient + " ", abcdata);// the query which selects all the information for the client
DataTable ss = new DataTable(); //declaring a data table to fill the text boxes

asdf.Fill(ss);

textBox2.Text = ss.Rows[0][1].ToString(); // fills the text box with the specific information from index 1 of the table

//The next text boxes having the same method of filling.

*******
```

The "Balance" form has the same concept, the single difference is the query which selects only the amount from the balance field.

3.2.8 User (client) Deposit Form



Source code:

****

```
private void button2_Click(object sender, EventArgs e)
{
        try
        {
            SqlConnection con = new SqlConnection(@"Data Source=DESKTOP-
U7518PG\SQLEXPRESS;Initial Catalog=Database;Integrated Security=True");
            con.Open(); //opens the connection
            SqlCommand cmd = new SqlCommand("select Balance from Account where
IdAccount=" + textBox1.Text + "   ", con);
            SqlDataReader dr = cmd.ExecuteReader();

            int a, b, c;
            a = Convert.ToInt32(textBox2.Text); //we parametrize our values from the text box
            b = Convert.ToInt32(textBox3.Text);
            c = a + b; //summing the amounts
            textBox2.Text = c.ToString();
            if (dr.Read())
            {
               SqlCommand cmd2 = new SqlCommand("update Account set Balance=@bal where
IdAccount=@id", con);   //updating the table with the new amount
            cmd2.Parameters.AddWithValue("@bal", int.Parse(textBox2.Text));
            cmd2.Parameters.AddWithValue("@id", int.Parse(textBox1.Text));
               dr.Close();
             cmd2.ExecuteNonQuery();


               MessageBox.Show("Value Updated");

            }
            else
            {
```

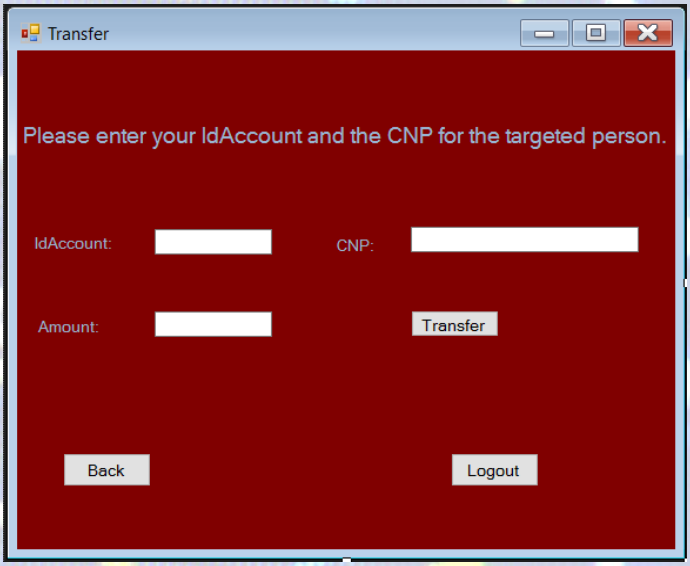
```
            MessageBox.Show("No value entered");

        }
    }
    catch (Exception e1)
    {
        MessageBox.Show("Error:" + e1.Message);
    }
}
```

****

The first button “The current amount” shows us the current Balance in the first textbox with the same method as we saw in our previously forms.

The other one “Add”, sums up the amount from Balance with the amount from the second textbox.

3.2.9 User (client) Transfer Form



Source code:

```
*******

int IdClient;
      public Transfer(int i)
      {
InitializeComponent();
         IdClient = i;
      } // initalizing the attribute we inherited

private void button1_Click(object sender, EventArgs e)
      {
         SqlConnection con = new SqlConnection(@"Data Source=DESKTOP-
U7518PG\SQLEXPRESS;Initial Catalog=Database;Integrated Security=True");
         Try //establishing the connection
         {
            int a, b, c;
         SqlDataAdapter asdf = new SqlDataAdapter("select Balance from Account where
IdAccount='" + textBox1.Text + "' ", con);
         DataTable ss = new DataTable(); // selecting the balance and declaring the datatable

            asdf.Fill(ss);
            a = int.Parse(ss.Rows[0][0].ToString());
            if (textBox2.Text != "" && textBox1.Text != "" && textBox3.Text != "")
            {

               b = Convert.ToInt32(textBox2.Text);
               c = a - b; //substracting the amount we want to transfer from the „BALANCE" field
```

```
            if (c > 0)
            {
                SqlCommand cmd2 = new SqlCommand("begin transaction; update Account set Balance=@bal where IdAccount=@id; update Account set Balance=Balance + @bal2 where IdAccount LIKE(select IDAccount FROM Account where IDClient LIKE (SELECT IdClient FROM Client WHERE CNP =@cnp)); commit;", con);
```

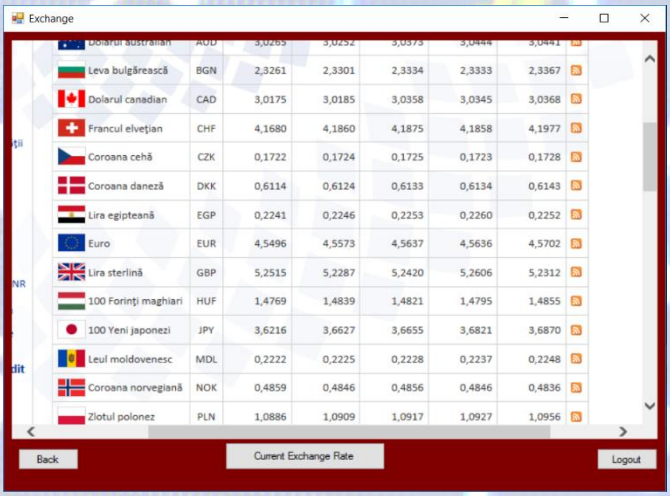
//Updating the table with the new amounts, from the current id and balance to the uniqes targeted person „CNP”

// The COMMIT command is the transactional command used to save changes invoked by a transaction to the database.

```
                cmd2.Parameters.AddWithValue("@bal", c);
                cmd2.Parameters.AddWithValue("@id", int.Parse(textBox1.Text));
                cmd2.Parameters.AddWithValue("@bal2", c);
                cmd2.Parameters.AddWithValue("@cnp", textBox3.Text);
                con.Open();
                cmd2.ExecuteNonQuery();
                con.Close(); //parametrizing the values we get from the text boxes

                MessageBox.Show("Transaction completed!");
            }
            else
                MessageBox.Show("We are sorry, but you don't have enough money!");
        }
        else
        {
            MessageBox.Show("No value entered");
        }
    }
    catch (Exception e1)
    {
        MessageBox.Show("Error:" + e1.Message);
    }
}
```

****

3.2.10 User (client) Exchange Form



Source code:

****

```
private void button3_Click(object sender, EventArgs e)
    {
        this.webBrowser1.Navigate("http://www.bnr.ro/Cursul-de-schimb-524.aspx");
    }
```

For this form we used a „WebBrowser" from the *Toolbox.*

****

By clicking the "Current Exchange Rate" we get the access to BNR's site for the current Exchange rate.

3.2.11 Credit Class

This class contains all the details about the credit created. It contains the preset values for every type of credit. It contains the preset values of every credit type, used to fill the "Credit Form" class. Also, it is used to set the interest according to the age and income of the user.

```
public void getEligiablility()
{
    double incomeInEuro;
    int incomeInLei = 0;

    if (incomeCurrency.Equals(creditCurrency))
    {
        if (income > creditRate)
            isEligible = true;
        if (incomeCurrency.Equals("LEI"))
            incomeInLei = income;
        else if (incomeCurrency.Equals("EURO"))
            incomeInLei = Convert.ToInt32(income * 4.5);
    }
    else if (incomeCurrency.Equals("LEI") && creditCurrency.Equals("EURO"))
    {
        incomeInEuro = income / 4.5;
        if (incomeInEuro > creditRate)
            isEligible = true;
        else isEligible = false;
    }
    else if (incomeCurrency.Equals("EURO") && creditCurrency.Equals("LEI"))
    {
        incomeInLei = Convert.ToInt32(income * 4.5);
        if (incomeInLei > creditRate)
            isEligible = true;
        else isEligible = false;
    }
```

This function sets if the user is eligible to get the credit required.

```
public void setAPRC()
{
    if (age > 45)
        interestRectification -= 0.5;
    else interestRectification += 0.5;

    if (income < 3000)
        interestRectification += 1.5;
    else if (income > 3000 && income < 5000)
        interestRectification += 1;
    else if (income > 5000)
        interestRectification += 0.75;
}
```

This sets the interest type according to the income and the age of the user.

```
public void setDefaultValues()

{
    if (creditType.Equals("Personal Needs") && creditOption.Equals("Guaranteed with mortgage"))
    {
        creditAmount = 1136075;
        creditTerm = 60;
        APRC = 5.22 + interestRectification;
        totalSum = truncateDoubleNumber(creditAmount + APRC / 100 * creditAmount);
        creditRate = truncateDoubleNumber(totalSum / creditTerm);
    }

    if (creditType.Equals("Personal Needs") && creditOption.Equals("With real estate purposes"))
    {
        creditAmount = 90886;
        creditTerm = 120;
        APRC = 7.06 + interestRectification;
        totalSum = truncateDoubleNumber(creditAmount + APRC / 100 * creditAmount);
        creditRate = truncateDoubleNumber(totalSum / creditTerm);
    }
```

This function is used to set the default values according to the credit type chosen by the user.

3.2.12 Database Operation Class

This class is used to make operations like reading and inserting data on the database.

```
public static void addCredit(Credit credit, int Id_client)
{
    string connectionString = @"Data Source=SELAX-PC\SQLEXPRESSSELAX;Initial Catalog=Database;Integrated Security=True";
    SqlConnection myCon = new SqlConnection(connectionString);

    string command = "INSERT INTO Credit(Credit_Type, Credit_Option, Rate, Term, Currency, Amount, Id_Client) VALUES (@Credit_Type,
    using (myCon = new SqlConnection(connectionString))
    using (SqlCommand sqlCommand = new SqlCommand(command, myCon))
    {
        myCon.Open();

        sqlCommand.Parameters.AddWithValue("@Credit_Type", credit.CreditType);
        sqlCommand.Parameters.AddWithValue("@Credit_Option", credit.CreditOption);
        sqlCommand.Parameters.AddWithValue("@Rate", credit.CreditRate);
        sqlCommand.Parameters.AddWithValue("@Term", credit.CreditTerm);
        sqlCommand.Parameters.AddWithValue("@Currency", credit.CreditCurrency);
        sqlCommand.Parameters.AddWithValue("@Amount", credit.CreditAmount);
        sqlCommand.Parameters.AddWithValue("@Id_Client", Id_client);

        sqlCommand.ExecuteNonQuery();
    }
}
```

This function inserts the credit details into the database if the user is eligible.

```
public static bool IfCreditTypeExists(string Credit_Option, int Id_Client)
{
    string connectionString = @"Data Source=SELAX-PC\SQLEXPRESSSELAX;Initial Catalog=Database;Integrated Security=True";
    SqlConnection myCon = new SqlConnection(connectionString);

    int num_of_records = 0;

    string command = "SELECT COUNT(*) FROM Credit WHERE Credit_Option = @Credit_Option AND Id_Client = @Id_Client";

    using (myCon)
    using (SqlCommand comm = new SqlCommand(command, myCon))
    {
        myCon.Open();

        comm.Parameters.AddWithValue("@Credit_Option", Credit_Option);
        comm.Parameters.AddWithValue("@Id_Client", Id_Client);

        num_of_records = (Int32)comm.ExecuteScalar();
    }

    if (num_of_records > 0)
        return true;
    else return false;
}
```

This function finds out if there exists another credit of the current user which has the same credit option.

```
public static int ifCanAfford(int Id_Client, int income)
{
    string connectionString = @"Data Source=SELAX-PC\SQLEXPRESSSELAX;Initial Catalog=Database;Integrated Security=True";
    SqlConnection myCon = new SqlConnection(connectionString);

    SqlDataReader reader;
    int totalCreditAmount = 0;
    string command = "SELECT ALL Currency, Rate FROM Credit WHERE Id_Client = @Id_Client";

    using (myCon)
    using (SqlCommand comm = new SqlCommand(command, myCon))
    {
        myCon.Open();

        comm.Parameters.AddWithValue("@Id_Client", Id_Client);

        reader = comm.ExecuteReader();

        while (reader.Read())
        {
            if (reader["Currency"].ToString().Equals("EURO"))
                totalCreditAmount += Convert.ToInt32(double.Parse(reader["Rate"].ToString()) * 4.5);
            else totalCreditAmount += Convert.ToInt32(double.Parse(reader["Rate"].ToString()));
        }

    }

    return totalCreditAmount;
}
```

This function determines if the user income is greater than all the credit rates that he has.

```
public static UserAccount getUserData(int Id_Client)
{
    UserAccount user;
    string connectionString = @"Data Source=SELAX-PC\SQLEXPRESSSELAX;Initial Catalog=Database;Integrated Security=True";
    SqlConnection myCon = new SqlConnection(connectionString);

    SqlDataReader reader;

    string command = "SELECT Age, Income, Income_Currency FROM Client WHERE IdClient = @Id_Client";

    using (myCon)
    using (SqlCommand comm = new SqlCommand(command, myCon))
    {
        myCon.Open();

        comm.Parameters.AddWithValue("@Id_Client", Id_Client);

        reader = comm.ExecuteReader();

        reader.Read();

        int age = Int32.Parse(reader["Age"].ToString());
        int income = Int32.Parse(reader["Income"].ToString());
        string incomeCurrency = reader["Income_Currency"].ToString();

        user = new UserAccount(Id_Client, age, income, incomeCurrency);
    }

    return user;
}
```

This function gets the information that is needed in order to calculate the credit for the current user.