



# **УНИВЕРСИТЕТ ПО БИБЛИОТЕКОЗНАНИЕ И ИНФОРМАЦИОННИ ТЕХНОЛОГИИ**

Факултет:

Информационни науки

## **КУРСОВА РАБОТА**

По “Визуални среди за програмиране”

Тема

“Управление на банков профил”

Разработил

Георги Георгиев

Факултетен номер: 46156р

Специалност ИКН

## *Съдържание*

1. Идея и представяне на проекта.....	3-8
2. Съдържание на проекта.....	9
3. Използвани библиотеки.....	9
4. Описание на кода на проекта.....	10-25
5. Използвани източници на ресурси.....	26

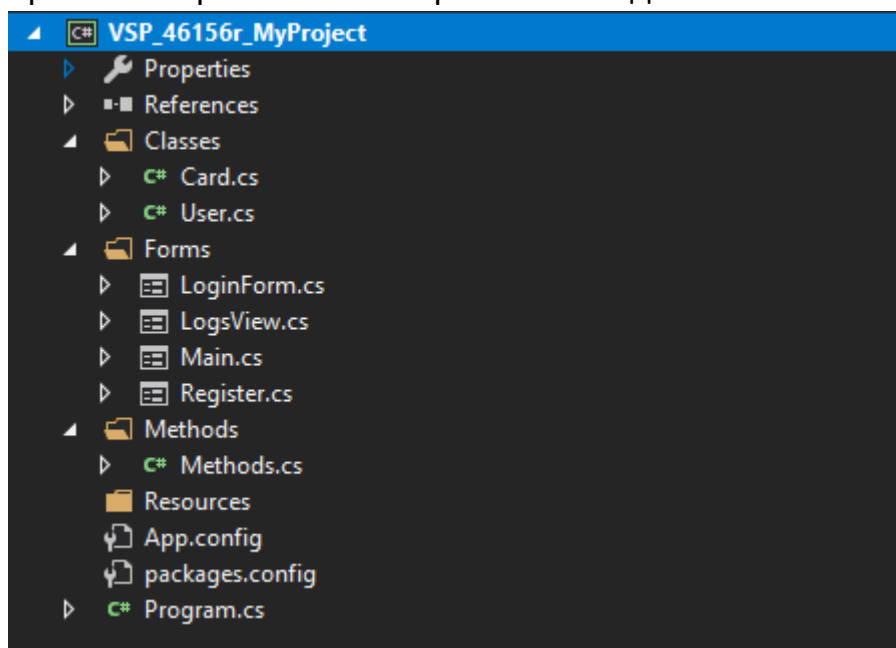
## 1. Идея и представяне на проекта

### А) Идея

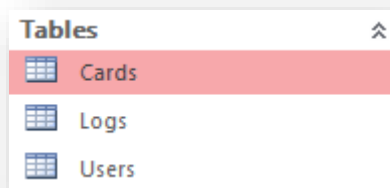
Проектът представлява програма за базово управление на пари по сметка на клиент в банка и извеждане на последни дейности по сметката. Програмата съдържа модул за вход(**LoginForm**), регистрация на нов потребител и създаване на виртуална банкова карта с ПИН код(**Register**), преглед на последни действия по сметка(**LogsView**) и основна страница в която могат да се извършват депозити и тегления на пари по сметка(**Main**). Всеки един от тези модули взаимодейства с база данни, разработена на **MS Access** за по-лесна миграция.

### Б) Представяне

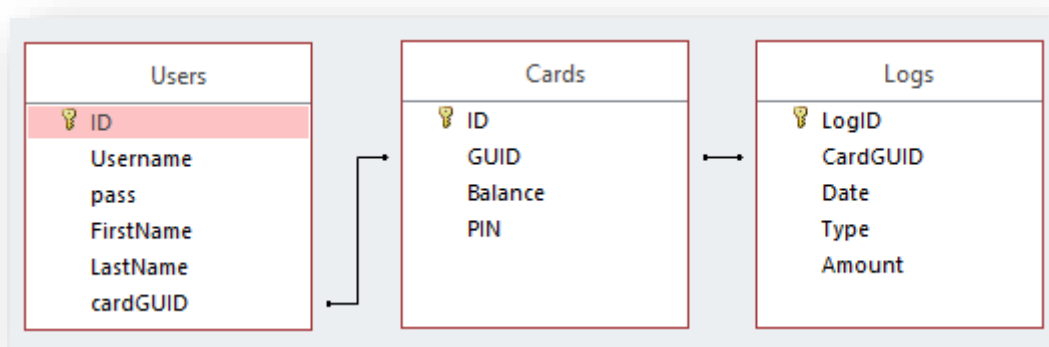
Йерархията на файловете по проекта е следната:



Самата база данни се намира в папката на проекта и съдържа следните таблици:



Таблиците съдържат следните полета и следните връзки:



**“Users”** – Съхранява информация за потребителя:

- **Username** – *String* – Съхранява потребителското име на клиента
- **Pass** – *String* – Съхранява криптираната парола на потребителя (виж описание на кода – използвани библиотеки)
- **FirstName** – *String* – Съхранява първото име на потребителя
- **LastName** – *String* – Съхранява фамилията на потребителя
- **CardGUID** – *String* – Съхранява уникален идентификационен текст на банковата карта на потребителя

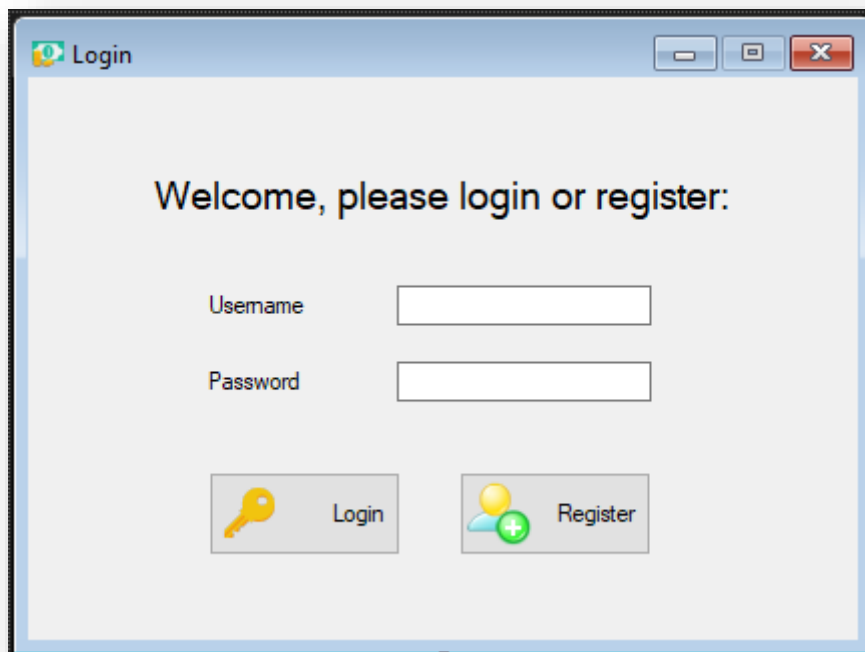
**“Cards”** – Съхранява информация за банковите карти:

- **GUID** – *String* – Съхранява уникален идентификационен текст на картата
- **Balance** – *Number* – Съхранява балансът по сметката
- **PIN** – *Number* – Съхранява ПИН кодът на сметката

**“Logs”** – Съхранява информация за движения по сметките:

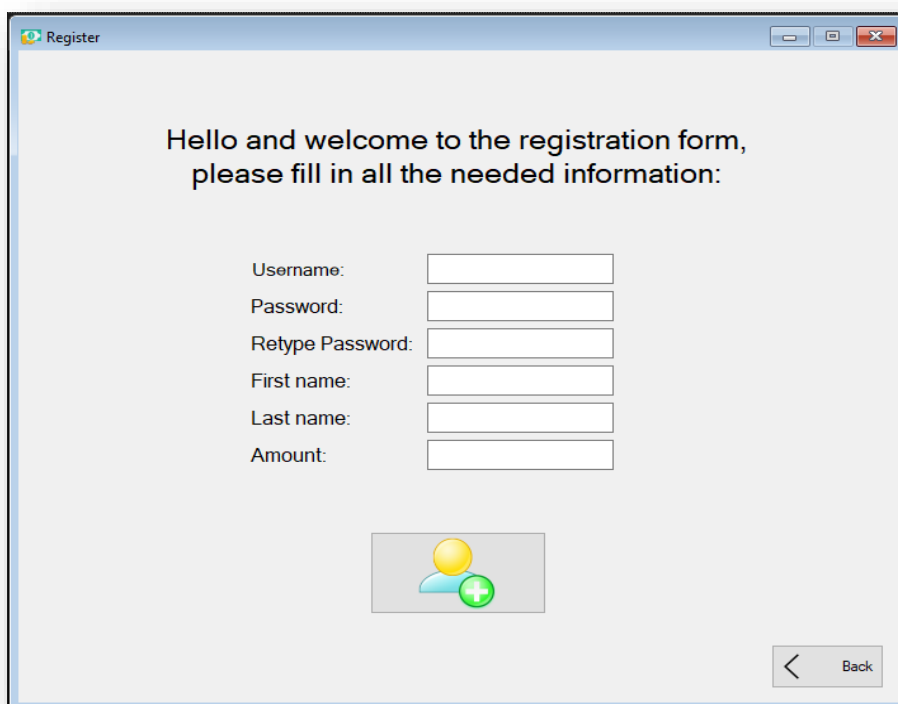
- **CardGUID** – *String* – Съхранява уникален идентификационен текст на картата за която се отнасят последните движения
- **Date** – *String* – Съхранява датата и часът на движението
- **Type** – *String* – Съхранява типа на движение по сметката
- **Amount** – *Number* – Съхранява количеството средства, които са били обект на движения по сметка

При пускане на програмата първото, което се явява на екрана е формата за вход(**LoginForm**):



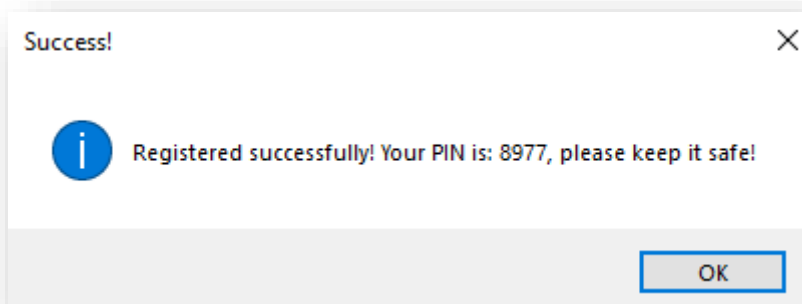
The screenshot shows a window titled "Login" with a standard Windows-style title bar. The main content area has a light gray background. At the top, it says "Welcome, please login or register:". Below this, there are two input fields: "Username" and "Password". At the bottom, there are two buttons: "Login" with a yellow key icon and "Register" with a yellow person icon and a green plus sign.

От тук насетне имаме 2 избора. Единият от които е да използваме вече създаден профил, който фигурира в базата данни и да влезем в системата чрез него, а вторият вариант е да създаден нов профил, натискайки бутона "**Register**", който отваря следващата форма, а именно тази за регистрация(**Register**):

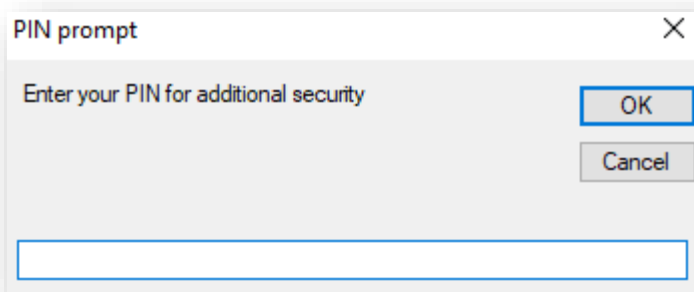


The screenshot shows a window titled "Register" with a standard Windows-style title bar. The main content area has a light gray background. At the top, it says "Hello and welcome to the registration form, please fill in all the needed information:". Below this, there are six input fields: "Username:", "Password:", "Retype Password:", "First name:", "Last name:", and "Amount:". At the bottom center, there is a button with a yellow person icon and a green plus sign. In the bottom right corner, there is a "Back" button with a left arrow icon.

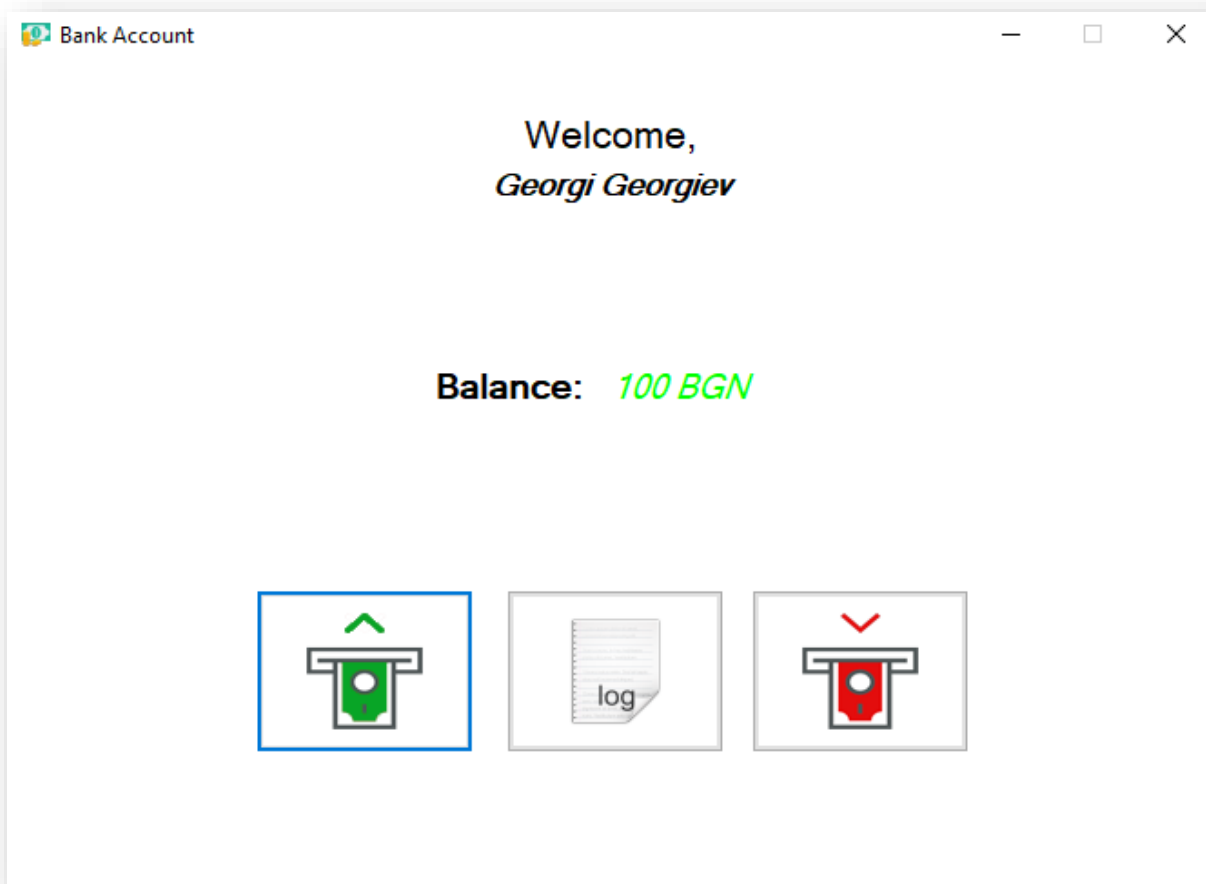
Важно е да се отбележи, че полетата съдържат валидации в реално време посредством събитие **TextChanged** и на потребителя няма да му бъде позволено да натисне бутона за регистрация, ако всички полета не са коректно зададени. При натискане на бутон „Back“, се връщаме обратно при формата за вход. При коректно въведени данни, бутонът за регистрация се активира при натискането му се проверява въведеното потребителско име и ако то не фигурира в базата данни, регистрацията преминава успешно. При успешна регистрация се извежда следната диалогова кутия, която ни дава PIN кодът, който ще ни е нужен за вход в системата(**LoginForm**):



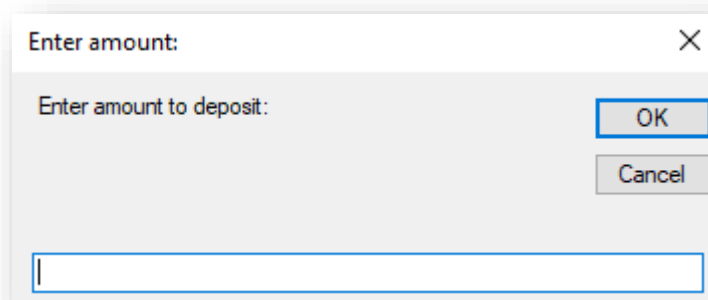
Обратно при формата за вход въвеждаме нужните данни, при грешка или несъществуващ профил се извежда диалогов прозорец с грешка. При успешно въвеждане се показва следният прозорец изискващ нашият ПИН като допълнителна верификация:



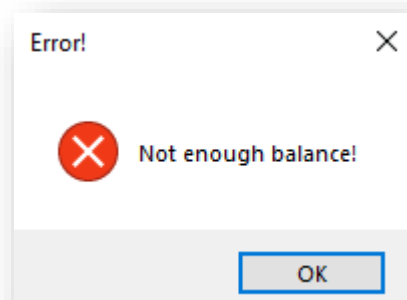
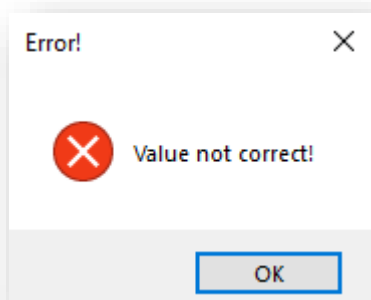
При неуспешно въвеждане на ПИН кодът се извежда диалогов прозорец съобщаващ за грешка с ПИН кдоът. При успешно въвеждане влизаме в главното меню(**Main**):



В това меню виждаме съобщение за добре дошли, като името и балансът се взимат от базада данни. При баланс от 0 BGN цветът е червен. Първият(от ляво надясно) бутон се използва за вкарване на пари по сметка, а последният за изтегляне, като при натискане на който и да е от бутоните се показва следният диалогов прозорец(с единствената разлика, че се текста се променя от “..to deposit” на “..to withdraw”:

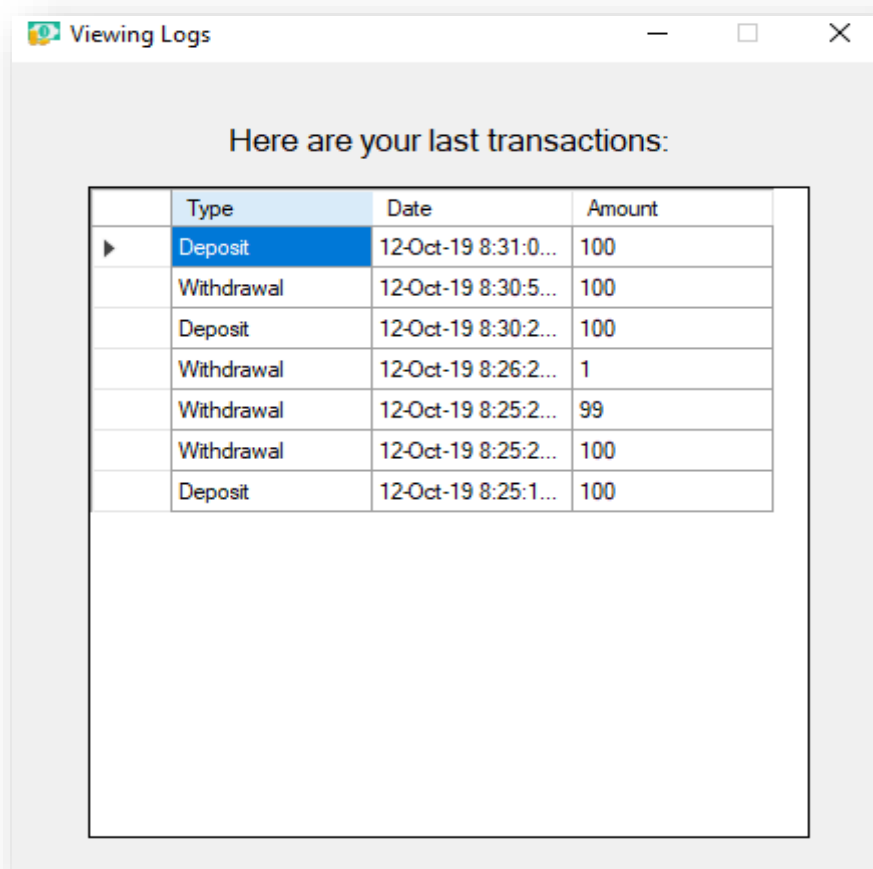


При грешен формат на сумата или недостатъчно баланс по сметката се извеждат следните диалогови прозорци:



При успешно въведени суми, се извежда диалогов прозорец, който потвърждава, че сумата е добавена и това се вижда и в главното меню(**Main**), като балансът е променен.

При натискане на средният бутон за отваряне на последни движения по сметката се извежда съответната форма(**LogsView**), която е във формата на таблица показваща последните движения.





## 2. Съдържание на проекта

Проекта се състои от 4 форми, а именно модулите – **LoginForm**, **Register**, **Main**, **LogsView**. Също така има 2 класа – **User** и **Card** и отделен клас **Methods**, който съдържа всички методи на проекта.

## 3. Използвани библиотеки в проекта

```
using System;
using System.Data.OleDb;
using System.Windows.Forms;
using Bank.Classes;
using static Bank.Methods.Methods;
using BCrypt.Net;
using Microsoft.VisualBasic;
using System.Text.RegularExpressions;
```

**Using System.Data.OleDb** – Използва се за да се осигури връзката на проекта с MS Access база данни;

**Using System.Windows.Forms** – Използва се за да се осигурят базови функционалности на Windows Form Applications;

**Using Bank.Classes** – Използва се за да се виждат класовете на проекта, намиращи се в директория *Classes*;

**Using static Bank.Methods.Methods** – Използва се за да се осигури видимост на всички методи, намиращи се в директория *Methods*, класа *Methods*;

**Using BCrypt.Net** – Използва се за да се осигури бавен алгоритъм за криптиране на парола като се добавя допълнителен текст към паролата или така нареченият *Salt*;

**Using Microsoft.VisualBasic** – Използва се за да се осигури достъп до *Interaction.InputBox* или така наречените кутии за въвеждане;

**Using System.Text.RegularExpressions** – Използва се за да се осигури достъп до използване на регулярни изрази с цел валидация.

## 4. Описание на кода

A) Описание на класове:

**-Cards.cs**

```
public class Card
{
    //Properties
    public string CardGUID { get; set; }
    public decimal Balance { get; set; }
    public int PIN { get; set; }

    //Constructors
    public Card() { }
    public Card(string cardGUID, decimal balance, int pin)
    {
        this.CardGUID = cardGUID;
        this.Balance = balance;
        this.PIN = pin;
    }
}
```

Свойствата отговарят на колоните в таблицата **Cards** на базата данни. Съдържа 2 конструктора, един празен и един изискващ всички свойства за класа.

```
//Methods
//Method that deposits money and then updated the DB
public void Deposit(decimal money)
{
    if (money > 0m)
    {
        this.Balance += money;

        //Updates the balance
        using (OleDbConnection connection = new OleDbConnection(@"Provider=Microsoft.ACE.OLEDB.12.0;
Data Source=../Main.accdb;
Persist Security Info=False;"))
        {
            connection.Open();
            OleDbCommand updateCommand = new OleDbCommand("UPDATE Cards SET Balance=@1 WHERE [GUID]=@2", connection);
            updateCommand.Parameters.AddWithValue("@1", this.Balance);
            updateCommand.Parameters.AddWithValue("@2", this.CardGUID);
            updateCommand.ExecuteNonQuery();

            //Logs every transaction
            LogChanges(this.CardGUID, DateTime.Now.ToString(), "Deposit", money, connection);
        }
    }
}
```

Един от методите е методът **Deposit**, който отговаря за внасяне на пари по сметка, като приема стойност като параметър, който са парите, които искаме да внесем.

```
//Updates the balance
using (OleDbConnection connection = new OleDbConnection(@"Provider=Microsoft.ACE.OLEDB.12.0;
Data Source=../Main.accdb;
Persist Security Info=False;"))
{
    connection.Open();
    OleDbCommand updateCommand = new OleDbCommand("UPDATE Cards SET Balance=@1 WHERE [GUID]=@2", connection);
    updateCommand.Parameters.AddWithValue("@1", this.Balance);
    updateCommand.Parameters.AddWithValue("@2", this.CardGUID);
    updateCommand.ExecuteNonQuery();
}
```

При премината проверка се задава връзката към базата данни, като се задават **Provider** – тип на базата данни, **Data Source** – път към базата и **Persist Security Info** – което при стойност **False** не връща информация за пароли като част от връзката с базата от данни. Отваряме връзката към базата данни чрез **connection.Open()** и създаваме нова заявка, която да промени балансът по сметката в базата данни, като тя е параметеризирана за да се избегне възможност от **SQL Injection** атаки, това се изразява в използването на **@1**, **@2** в самата заявка и добавянето на параметрите отделно. Използваме **ExecuteNonQuery()**, за да извършим операцията върху базата данни.

```
//Logs every transaction
LogChanges(this.CardGUID, DateTime.Now.ToString(), "Deposit", money, connection);
```

Накрая се извиква методът **LogChanges**, който записва преводът в таблицата **Logs** на базата данни.

```
//Method that withdraws money and then updates the DB
public void Withdraw(decimal money)
{
    if (money > 0m && this.Balance >= money)
    {
        this.Balance -= money;

        //Updates the balance
        using (OleDbConnection connection = new OleDbConnection(@"Provider=Microsoft.ACE.OLEDB.12.0;
Data Source=../Main.accdb;
Persist Security Info=False;"))
        {
            connection.Open();
            OleDbCommand updateCommand = new OleDbCommand("UPDATE Cards SET Balance=@1 WHERE [GUID]=@2", connection);
            updateCommand.Parameters.AddWithValue("@1", this.Balance);
            updateCommand.Parameters.AddWithValue("@2", this.CardGUID);
            updateCommand.ExecuteNonQuery();

            //Logs every transaction
            LogChanges(this.CardGUID, DateTime.Now.ToString(), "Withdrawal", money, connection);
        }
    }
}
```

Другият метод на класът **Cards** е **Withdraw**, който се използва за теглене на пари от сметка и работи на същия принцип като **Deposit**, с разлика в проверката и типът на превода, записван в таблицата **Logs**.

### - *User.cs*

```
public class User
{
    //Properties

    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string CardGUID { get; set; }
    public Card DebitCard { get; set; }

    //Constructors
    public User() { }
    public User(string firstName, string lastName, string cardGUID)
    {
        this.FirstName = firstName;
        this.LastName = lastName;
        this.CardGUID = cardGUID;
    }
}
```

- Свойствата отговарят на тези в таблицата Users на базата данни, като **Card DebitCard** показва връзката между двете таблици
- Съдържа 2 конструктора – един празен и един с 3 от 4-те свойства на класа, тъй като картата се добавя от външен метод

### -*Method.cs* – Описание на всички методи, събрани в един клас

```
//Method that logs any deposit/withdrawal from card
public static void LogChanges(string cardGUID, string date, string type, decimal amount, OleDbConnection conn)
{
    OleDbCommand commandLog = new OleDbCommand("INSERT INTO Logs (CardGUID, [Date], Type, Amount) values (@1,@2,@3,@4)", conn);
    commandLog.Parameters.AddWithValue("@1", cardGUID);
    commandLog.Parameters.AddWithValue("@2", date);
    commandLog.Parameters.AddWithValue("@3", type);
    commandLog.Parameters.AddWithValue("@4", amount);
    commandLog.ExecuteNonQuery();
}
```

Това е метод, който записва всяко едно движение на пари в сметката, като приема 4 параметъра, който отговарят на колоните в таблицата **Logs** на базата данни и допълнителен параметър, който предава текущата връзка с базата данни. Принципа на операцията с базата данни е същият като при горните методи.

```
//Method that checks the PIN for additional security
public static bool CheckPIN(int enteredPIN, int dbPIN)
{
    return (enteredPIN == dbPIN);
}
```

Това е метод, който сравнява PIN кодът въведен от потребителя с този от базата данни.

```
//Method that registers the user and its debit card
public static int RegisterUser(string user, string password, string fName, string lName, decimal amount, OleDbConnection connection)
{
    //Creating GUID for the card unique ID
    var newGuid = Guid.NewGuid();
    //Hashing the password using BCrypt
    string hashedPwd = BCrypt.HashPassword(password, BCrypt.GenerateSalt());

    //Query for inserting the new user in the DB
    OleDbCommand CommandRegisterUser = new OleDbCommand("INSERT into Users (Username,pass,FirstName,LastName,CardGUID) values (@1,@2,@3,@4,@5)", connection);
    CommandRegisterUser.Parameters.AddWithValue("@1", user);
    CommandRegisterUser.Parameters.AddWithValue("@2", hashedPwd);
    CommandRegisterUser.Parameters.AddWithValue("@3", fName);
    CommandRegisterUser.Parameters.AddWithValue("@4", lName);
    CommandRegisterUser.Parameters.AddWithValue("@5", newGuid);
    CommandRegisterUser.ExecuteNonQuery();

    //Used in creating PIN
    Random rand = new Random();
    int pin = rand.Next(1000, 9999);

    //Query for creating the new Debit card for the newly created user
    OleDbCommand CommandRegisterCard = new OleDbCommand("INSERT INTO Cards ([GUID],Balance,PIN) VALUES (@guid,@amount,@pin)", connection);
    CommandRegisterCard.Parameters.AddWithValue("@guid", newGuid);
    CommandRegisterCard.Parameters.AddWithValue("@amount", amount);
    CommandRegisterCard.Parameters.AddWithValue("@pin", pin);
    CommandRegisterCard.ExecuteNonQuery();

    return pin;
}
```

Този метод регистрира потребителя, като добавя неговите данни в базата данни. Използва се **GUID** за да се генерира уникален идентификационен текст за дебитната карта на потребителя. За максимална защитеност на паролата се използва методът **HashPassword()** в комбинация с **GenerateSalt()**, които са част от библиотеката **BCrypt.Net**. Извършват се 2 заявки към 2 от таблиците на базата данни, а именно **Users** и **Cards**, като накрая методът връща **PIN**-ът на картата, който е създаден чрез **Random Number Generator** и е в диапазона от 1000 до 9999, за да е четирицифрен.

```
//Method that tries to log in the user and returns reader with data from the DB according to username and password
public static bool CheckLogin(string user, string password, OleDbConnection connection)
{
    OleDbCommand commandUser; // create command for User only
    OleDbDataReader readerUser; //Reader for User only
    commandUser = new OleDbCommand("Select pass from Users where Username=@user", connection); //Query for checking username and password
    commandUser.Parameters.AddWithValue("@user", user);
    readerUser = commandUser.ExecuteReader();
    bool isOkay = false;

    if (readerUser.HasRows)
    {
        while (readerUser.Read())
        {
            //Checks if the stored hashed password is the same as its non-hashed version
            isOkay = BCrypt.Verify(password, readerUser["pass"].ToString());
        }

        return (isOkay == true);
    }
    else
    {
        return false;
    }
}
}
```

Този метод проверява данните въведени от потребителя във формата за вход(**LoginForm**). Като се извършва заявка към базата данни, която връща стойности, които се записват в **OleDbDataReader readerUser**, след това се проверява дали същият четец съдържа някаква информация в себе си и се проверява дали паролата, въведена от потребителя отговаря на тази от базата данни, чрез използване на метода **Verify()**, който също е част от библиотеката **BCrypt.Net**.

```
//Method that returns the card with the associated ID of the user
public static Card SetCardToUser(User user, OleDbConnection connection)
{
    OleDbCommand commandCard; // create command for the Card only
    OleDbDataReader readerCard; //Reader for the Card only

    //The query for finding the card in the DB
    commandCard = new OleDbCommand("Select * from Cards where GUID=@1", connection);
    //Preventing SQL Injection attacks
    commandCard.Parameters.AddWithValue("@1", user.CardGUID);
    readerCard = commandCard.ExecuteReader();

    //Creating new Card Object
    Card currentCard = new Card();

    if (readerCard.HasRows)
    {
        //Reads from the DB
        while (readerCard.Read())
        {
            currentCard = new Card(user.CardGUID, Convert.ToDecimal(readerCard["Balance"]), Convert.ToInt32(readerCard["PIN"]));
        }

        return currentCard;
    }
    else { return null; }
}
}
```

Това е метод, който извършва заявка към базата данни и връща информация за карта на даден потребител посредством параметъра **GUID**. Информацията върната от базата отново се записва в четец, а информацията от него се използва за изграждане на нов обект от тип **Card** и след това този обект се връща от метода, ако не е намерена карта, методът връща **Null**.

## - LoginForm.cs

```
private void btnLogin_Click(object sender, EventArgs e)
{
    //Takes the entered values
    string user = txtUser.Text.Trim();
    string pass = txtPwd.Text.Trim();

    using (OleDbConnection connection = new OleDbConnection(@"Provider=Microsoft.ACE.OLEDB.12.0;
Data Source=../Main.accdb;
Persist Security Info=False;"))
    {
        connection.Open();

        if (CheckLogin(user, pass, connection))
        {
            OleDbCommand commandUser; // create command for User only
            OleDbDataReader readerUser; //Reader for User only
            commandUser = new OleDbCommand("Select * from Users where username=@1", connection); //Query for getting the info about user
            commandUser.Parameters.AddWithValue("@1", user);
            readerUser = commandUser.ExecuteReader();

            //Creating new User object
            User currentUser = new User();
            //Reading from the user Reader
            while (readerUser.Read())
            {
                currentUser = new User(readerUser["FirstName"].ToString(), readerUser["LastName"].ToString(), readerUser["cardGUID"].ToString());
            }

            //Setting the DebitCard property with method that returns the Card from the DB that is with the associated ID
            currentUser.DebitCard = SetCardToUser(currentUser, connection);
            var pin = Interaction.InputBox("Enter your PIN for additional security", "PIN prompt").ToString();

            if (int.TryParse(pin, out int parsedPIN))
            {
                //Checks for PIN for additional security
                if (CheckPIN(parsedPIN, currentUser.DebitCard.PIN))
                {
                    //Closing the current form and opens the main one while passing the current user
                    Main frm = new Main(currentUser);
                    frm.Show();
                    this.Hide();
                    frm.Closed += (s, args) => this.Close();
                    frm.Show();
                }
                else
                {
                    MessageBox.Show("Wrong PIN!", "Error!", MessageBoxButtons.OK, MessageBoxIcon.Error);
                    txtUser.Text = "";
                    txtPwd.Text = "";
                }
            }
            else
            {
                MessageBox.Show("PIN format not correct!", "Error!", MessageBoxButtons.OK, MessageBoxIcon.Error);
                txtUser.Text = "";
                txtPwd.Text = "";
            }
        }
        else
        {
            MessageBox.Show("Wrong username or password!", "Error!", MessageBoxButtons.OK, MessageBoxIcon.Error);
            txtUser.Text = "";
            txtPwd.Text = "";
        }
    }
}
```

Събитието **Click()** върху бутона за **Login** е свързано с използването на метода **CheckLogin()** и при правилно въведени потребителско име и парола се извършва заявка върху базата данни, връщаща цялата информация за потребителя и създаваща нов обект от тип **User**. След това се извиква методът **SetCardToUser()**, който запълва свойството **Card DebitCard** на потребителя с нов обект от тип **Card**. След това се извежда кутия за въвеждане на данни (**InputBox**), която служи за въвеждане на **PIN** кодът. Първо се прави опит въведеното да бъде присвоено на променлива от тип **Int** и при успех се проверява дали въведеният **PIN** съвпада с този от базата данни, чрез методът **CheckPIN()**, при успех се отваря главното меню(**Main**) и се предава текущият потребител, като формата за вход(**LoginForm**) се затваря. Ако някои от проверките не е успешна се извежда диалогов прозорец с грешка.

```
private void btnReg_Click(object sender, EventArgs e)
{
    Register frm = new Register();
    frm.Show();
}
```

Събитието **Click()** на бутона Register отваря формата за регистрация(**Register**).

**-Register.cs**

```
public Register()
{
    InitializeComponent();

    //Disables the register button
    btnRegister.Enabled = false;

    //Changes the Tag property of each control to false
    txtRegAmount.Tag = false;
    txtRegFirstName.Tag = false;
    txtRegLastName.Tag = false;
    txtRegPwd.Tag = false;
    txtRegRetypePwd.Tag = false;
    txtRegUser.Tag = false;
}
```

Бутонът за регистрация се прави неактивен и свойството **Tag** на всяка от текстовите кутии се задава като **false**.



```

//Validation method that checks for empty values and then call ValidationRouter for current textbox
public void txt_Validation(object sender)
{
    btnRegister.Enabled = false;

    TextBox txt = (TextBox)sender;
    bool isOk = false;
    if (txt.Text.Length == 0)
    {
        //Changes the color and the Tag property of the current textbox
        txt.BackColor = Color.Red;
        txt.Tag = false;
    }
    else
    {
        //Excludes the Username textbox
        if (txt.Name != "txtRegUser")
        {
            //Calls the ValidationRouter
            isOk = ValidationRouter(txt);
        }
        else
        {
            isOk = true;
        }
    }

    if (isOk)
    {
        //If everything is okay turns the box in to green, the Tag property to true
        //and attempts to enable the registration button
        txt.BackColor = Color.Lime;
        txt.Tag = true;
        EnableRegister();
    }
    else
    {
        //If the ValidationRouter returned false, then turns the textbox red
        txt.BackColor = Color.Red;
        txt.Tag = false;
    }
}
}

```

Този метод е началният за нашата валидация, првоначално той проверява дали има празно поле във формата, като променя цвета на всяка текстова кутия в която сме се опитали да пишем. При въведен в кутията текст се извиква методът **ValidationRouter()**, който взима като параметър текущата контрола(не се използва при контролата за въвеждане на потребителско име, тъй като за нея има валидация чрез сравнение в базата данни), ако методът **ValidationRouter()** върне **true**, се сменя цвета на полето на **зелен(Lime)**, свойството **Tag** на контролата се променя на **true** и се прави опит за активиране на бутона за регистрация, а ако върне **false**, цвета става **червен** и свойството **Tag** става **false**. Всички методи за валидация са от тип **bool** и връщат **true** или **false**.

```
//Method that calls the appropriate validation based on the name of the control
public bool ValidationRouter(TextBox tb)
{
    if (tb.Name == "txtRegPwd")
    {
        return ValidatePassword(tb);
    }
    else if (tb.Name == "txtRegRetypePwd")
    {
        return ValidatePasswordMatch(tb);
    }
    else if (tb.Name == "txtRegAmount")
    {
        return ValidateAmount(tb);
    }
    else
    {
        return ValidateNames(tb);
    }
}
```

Този метод се използва като разпределител на контроли, спрямо тяхното име, като извикват различни методи за валидация, спрямо различните текстови кутии.

```
//Validation method for password
public bool ValidatePassword(TextBox tb)
{
    //Checking for password strength using regular expressions
    var hasNumber = new Regex(@"[0-9]+");
    var hasUpperChar = new Regex(@"[A-Z]+");
    var hasMinimum8Chars = new Regex(@".{8,}");
    var isPwdOk = hasNumber.IsMatch(tb.Text) && hasUpperChar.IsMatch(tb.Text) && hasMinimum8Chars.IsMatch(tb.Text);

    if (!isPwdOk)
    {
        //If the password doesn't match the regular expression, adds errorProvider error next to the textbox
        errorProvider.SetError(tb, "Password must be atleast 8 characters\nwith atleast 1 capital letter and 1 number");
        return false;
    }
    //Clears the error if everything is okay
    errorProvider.SetError(tb, null);
    return true;
}
```

Това е методът за валидация на паролата, който използва регулярни изрази, които изискват:

- Минимум 8 знака;
- Поне една главна буква;
- Число.

При несъответствие с регулярния израз се използва **errorProvider** за да изведе иконка за грешка до самата контрола и връща **false**, а при съответствие се чисти грешката и се връща **true**.

```
//Validation for the retyped password
public bool ValidatePasswordMatch(TextBox tb)
{
    if (tb.Text != txtRegPwd.Text)
    {
        //Sets errorProvider
        errorProvider.SetError(tb, "Passwords does not match!");
        return false;
    }
    //Clears errorProvider
    errorProvider.SetError(tb, null);
    return true;
}
```

Това е метод за валидиране на повторената парола, като я сравнява с паролата въведена в полето за парола. Отново се използва **errorProvider** за извеждане на иконка с грешка и се връща **false** и съответно при съответствие се чисти грешката и се връща **true**

```
//Validation for the names field
public bool ValidateNames(TextBox tb)
{
    //Regular expression for text only
    if (!Regex.IsMatch(tb.Text, @"^[a-zA-Z]+$"))
    {
        //Sets errorProvider
        errorProvider.SetError(tb, "Only letters allowed!");
        return false;
    }
    //Clears errorProvider
    errorProvider.SetError(tb, null);
    return true;
}
```

Това е общ метод за валидация на имената на потребителя. Проверява дали полето съдържа само букви, отново чрез регулярни изрази и използва **errorProvider** за да изведе икона с грешка и при съответствие чисти грешката.

```
//Validation for username that checks the DB for the username
public bool ValidateUsername(TextBox tb, OleDbConnection connection)
{
    //Query for checking if username exists
    OleDbCommand commandUserName = new OleDbCommand("Select Username from Users where Username=@1", connection);
    commandUserName.Parameters.AddWithValue("@1", tb.Text);
    OleDbDataReader readerUser = commandUserName.ExecuteReader();

    //If the query returned any rows add errorProvider and changes the color
    if (readerUser.HasRows)
    {
        errorProvider.SetError(tb, "Username already in use!");
        tb.BackColor = Color.Red;
        return false;
    }
    //Clears error
    errorProvider.SetError(tb, null);
    return true;
}
```

Това е метод за валидация на потребителското име. Този метод прави заявка към базата данни за даденото потребителско име и при грешка сменя цвета на контролата и използва **errorProvider** за да изведе иконка с грешка до самата контрола. Ако името не фигурира в базата данни грешката се чисти и цветът остава зелен.

```
//Validation for the money amount with regular expression to only accept numbers
//and decimals with 2 decimal places after floating point
public bool ValidateAmount(TextBox tb)
{
    if (!Regex.IsMatch(tb.Text, @"^(?:0|[1-9][0-9]*)?(?:\.[0-9]{1,2})?$$"))
    {
        //Sets error
        errorProvider.SetError(tb, "Amount must be integer or decimal with maximum 2 decimal places\n after floating point");
        return false;
    }
    //Clears error
    errorProvider.SetError(tb, null);
    return true;
}
```

Това е метод за валидация на въведената сума. Отново чрез регулярни изрази се проверява дали въведената сума отговаря на цяло число или десетично число с 2 числа след десетичната запетайка.

```
//Enables the register Button
private void EnableRegister()
{
    this.btnRegister.Enabled = ((bool)txtRegAmount.Tag &&
        (bool)txtRegUser.Tag && (bool)txtRegPwd.Tag &&
        (bool)txtRegRetypePwd.Tag && (bool)txtRegFirstName.Tag &&
        (bool)txtRegLastName.Tag);
}
```

Това е метод, който прави бутона за регистрация активен, ако таговете на всички контроли са **true**.

```

//Calls the validation method for each control onTextChanged event
private void txtRegUser_TextChanged(object sender, EventArgs e)
{
    errorProvider.SetError((TextBox)sender, null);
    txt_Validation(sender);
}

private void txtRegPwd_TextChanged(object sender, EventArgs e)
{
    txt_Validation(sender);
    errorProvider.SetError(txtRegRetypePwd, "Passwords does not match!");
    txt_Validation(txtRegRetypePwd);
}

private void txtRegRetypePwd_TextChanged(object sender, EventArgs e)
{
    txt_Validation(sender);
}

private void txtRegFirstName_TextChanged(object sender, EventArgs e)
{
    txt_Validation(sender);
}

private void txtRegLastName_TextChanged(object sender, EventArgs e)
{
    txt_Validation(sender);
}

private void txtRegAmount_TextChanged(object sender, EventArgs e)
{
    txt_Validation(sender);
}

```

Това са събитията, които извикват методите за валидация.

-*Main.cs*

```
public User PassedUser { get; set; }

public Main(User passedUser)
{
    InitializeComponent();
    this.PassedUser = passedUser;
}
```

Специфичното при тази форма е свойството ***PassedUser***, което съдържа в себе си информация за потребителя спрямо неговото потребителско име, проверено от формата за вход(***LoginForm***), като при зареждане на главната форма(***Main***) това свойство се приравнява на предаденият потребител(***PassedUser***).

```
private void Main_Load(object sender, EventArgs e)
{
    //Changes the values for the labels
    lblName.Text = PassedUser.FirstName + " " + PassedUser.LastName;
    lblBalance.Text = PassedUser.DebitCard.Balance + " BGN";
}
```

Събитието ***Load()*** на главната форма(***Main***) извършва единствено запълване на етикетите за име и баланс по сметка.

```

private void btnDeposit_Click(object sender, EventArgs e)
{
    //Inputbox for the amount
    string value = Interaction.InputBox("Enter amount to deposit:", "Enter amount:");

    //Tries to parse the amount as usable decimal
    if (decimal.TryParse(value, out decimal amount))
    {
        //Deposits upon success
        PassedUser.DebitCard.Deposit(amount);
        MessageBox.Show("Money deposited successfully!", "Success!", MessageBoxButtons.OK, MessageBoxIcon.Information);
        //Updates the label
        lblBalance.Text = PassedUser.DebitCard.Balance.ToString() + " BGN";
        if (PassedUser.DebitCard.Balance == 0m)
        {
            lblBalance.ForeColor = Color.Red;
        }
        else
        {
            lblBalance.ForeColor = Color.Lime;
        }
    }
    else
    {
        MessageBox.Show("Value not correct!", "Error!", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

Събитието **Click()** на бутона **Deposit** извежда диалогов прозорец за въвеждане на данни(**InputBox**). След като нещо е въведено се прави опит то да се запише в променлива от тип **Decimal** и при успешно записване се извиква методът **Deposit()** от клас **Card**, като за параметър се предава въведеното в диалоговия прозорец. Извежда се съобщение, че операцията е успешна и се променя етикетът в главната форма(**Main**) с новият баланс, като се променя и цвета според зависи от стойността на баланса. При грешно въведени данни се извежда диалогов прозорец описващ грешката.

```

private void btnLog_Click(object sender, EventArgs e)
{
    LogsView logsView = new LogsView(PassedUser.DebitCard.CardGUID);
    logsView.Show();
}

```

Събитието **Click()** на бутона **Log** отваря формата за преглеждане на последни движения по сметка(**LogsView**), като се предава **GUID** на картата.

```

private void btnWithdraw_Click(object sender, EventArgs e)
{
    //Inputbox for the amount
    string value = Interaction.InputBox("Enter amount to withdraw:", "Enter amount:");

    //Tries to parse the amount as usable decimal
    if (decimal.TryParse(value, out decimal amount))
    {
        //Checks if the wanted amount for withdrawal is more than the current balance
        if (amount <= PassedUser.DebitCard.Balance)
        {
            //Withdraws upon success
            PassedUser.DebitCard.Withdraw(amount);
            MessageBox.Show("Money withdrawn successfully!", "Success!", MessageBoxButtons.OK, MessageBoxIcon.Information);
            //Updates the label
            lblBalance.Text = PassedUser.DebitCard.Balance.ToString() + " BGN";
            if (PassedUser.DebitCard.Balance == 0m)
            {
                lblBalance.ForeColor = Color.Red;
            }
            else
            {
                lblBalance.ForeColor = Color.Lime;
            }
        }
        else
        {
            MessageBox.Show("Not enough balance!", "Error!", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
    else
    {
        MessageBox.Show("Value not correct!", "Error!", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
}

```

Събитието **Click()** върху бутона **Withdraw** работи на същият принцип като **Deposit.Click()** с единствената разлика, че тук се извиква методът **Withdraw()** на класа **Card**.

### -LogsView.cs

```

public string CardGUID { get; set; }
public LogsView(string cardGUID)
{
    InitializeComponent();
    this.CardGUID = cardGUID;
}

```

Тук отново има свойство на формата, но този път е **CardGUID** и това свойство се приравнява на предаденото свойство от предишната форма.



```

private void LogsView_Load(object sender, EventArgs e)
{
    //Upon load get the logs from the DB and display them in GridView Control
    using (OleDbConnection connection = new OleDbConnection(@"Provider=Microsoft.ACE.OLEDB.12.0;
Data Source=../Main.accdb;
Persist Security Info=False;"))
    {
        connection.Open();
        //Get all the logs for this card and orders them by descending order
        OleDbCommand commandLogs = new OleDbCommand("Select Type,[Date],Amount from Logs where cardGUID=@1 ORDER BY [date] desc",connection);
        commandLogs.Parameters.AddWithValue("@1", this.CardGUID);
        OleDbDataAdapter da = new OleDbDataAdapter(commandLogs);
        DataTable logs = new DataTable();
        da.Fill(logs);
        //Points that the source of data for the dataGridView is the datatable Logs
        dataGridView1.DataSource = logs;
    }
}

```

Събитието **Load()** на формата за преглеждане на последни движения по банкова сметка(**LogsView**) извършва заявка към таблицата **Logs** от базата данни и пълненето на контролата **DataGridView** с информация взета чрез заявката.

## 5. Използвани източници на ресурси

- А) <https://stackoverflow.com/questions/17023861/how-to-connect-access-database-in-c-sharp>
- Б) <https://docs.microsoft.com/en-us/dotnet/api/system.data.oledb?view=netframework-4.8>
- В) <https://stackoverflow.com/questions/5859632/regular-expression-for-password-validation>
- Г) <https://stackoverflow.com/questions/8321871/how-to-make-a-textbox-accept-only-alphabetic-characters>
- Д) <http://www.iconarchive.com/>