Технически университет- Варна
Факултет ФИТА
Специалност: “Софтуерни и интернет технологии”

КУРСОВ ПРОЕКТ

по дисциплината: “Технология на софтуерното производство”

Тема: Система за автоматизация на библиотека

Разработващ екип:

№	Име, презиме и фамилия	Фак. №	Роля
1.	Радослав Павлинов Колев	18621786	Ръководител на проекта и тестер
2.	Николай Светославов Николаев	18621695	Софтуерен разработчик и архитект

Срок за представяне: 25 май, 2021г. Водещ преподавател:/……../
доц. В.Божикова

Варна

2021

Съдържание

1. Задание за разработка

1.1 Условие на проекта

В настоящата курсова работа е разработена информационна система, предоставяща услугата - библиотека. В нея се осъществява регистрация на всички читатели и се поддържа каталог на всички постъпващи в библиотеката книги, освен това се фиксира информация за това, какви книги в кой читател се намират в дадения момент. Системата позволява множествен достъп.

Системата поддържа два вида потребители: администратор и читатели, които имат различни роли за достъп до функционалностите на системата.

Приложението дава възможност за:

- добавяне/отстраняване на читател в/от базата данни;
- добавяне/отстраняване на книги в каталога;
- регистрация на взетите и върнатите от читатели книги;
- дава полезна справочна информация, например за наличието на дадена книга в даден момент.
- изготвяне на каталог;
- и др.

Системата поддържа справки за:

- Читатели;
- Книги;
- Заети книги;
- Жанрове

1.2 Анализ на проблема

1.2.1 Функционални изисквания

Приложението позволява на даден читател да се регистрира в системата и да управлява информацията за своя акаунт, както и за промяна на името и паролата на акаунта при необходимост.

Системата позволява на администратора да следи по-лесно читателските акаунти и намиращите се в тях книги и при необходимост да добавя или изтрива данни.

Системата позволява на даден читател да провери за наличието на дадена книга в библиотеката, да я вземе или да върне книга ,която вече е взел.

Приложението трябва да съхранява данни за:

- Читатели
- Книги
- Жанрове
- Заети книги

Позволява на администраторския профил да работи с информацията от таблиците в базата данни в т.ч. да добавя , да нанася корекции и да изтрива данни от тях.

1.2.2 Нефункционални изисквания

Използва се унифицирана форма за логване и регистрация в системата от всички потребители (достъпът до системата се осъществява след въведено потребителско име и парола).

За възвръщане на акаунт, системата проверява за наличието на имейл адреса на читателя в БД. Същото правило важи и за промяна на името и паролата на акаунта.

За всяка от таблиците в БД се използва унифицирана форма за реализиране на стандартните операции: въвеждане, корекция, изтриване и търсене на данни.

Генериране на необходимите справки: както на екран, така и на печат.

1.3 Структура на проекта

Често изборът на правилната структура прави програмата по-ефективна, тъй като се спестява памет и време за изпълнение. Данните биват групирани по определен начин, за да се улесни достъпът до тях, както и тяхното управление. За различни задачи са подходящи различни структури.

За по-сложни случаи могат да се създадат много по-сложни структури. При избор на подходяща такава е възможно по-бързо обработване на информацията с ползване на минимум ресурси.

Курсовата работа има следната структура, която е съставена от 4 части:

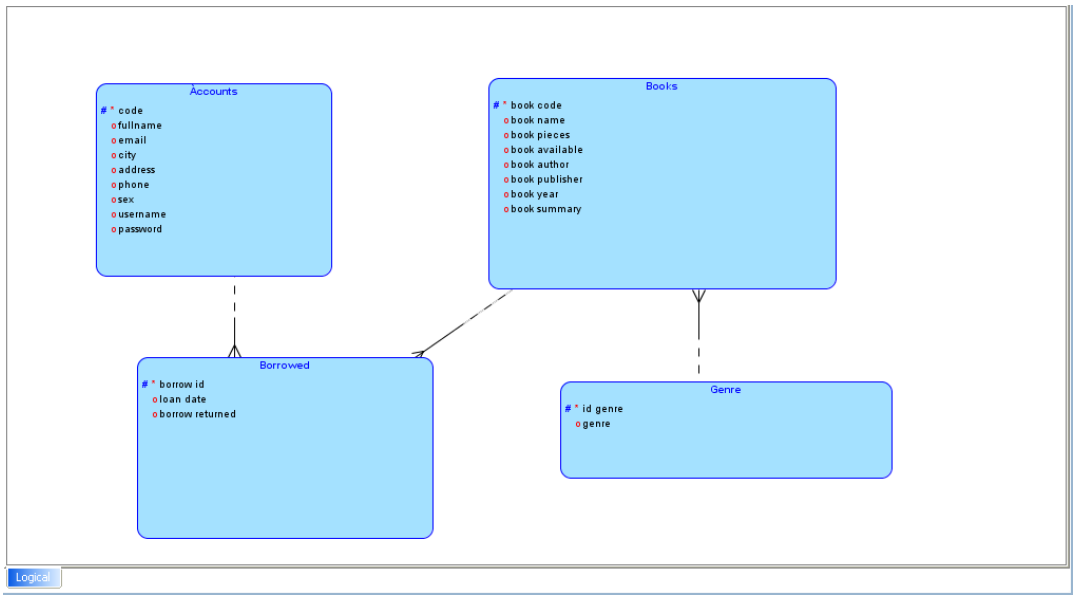
- **Част първа** „Анализ на проблема” – Кратко описание на системата и нейните функционални изисквания.
- **Част втора** „Проектиране на системата” – Разработване на базата от данни, потребителския интерфейс, UML диаграми, ERD диаграма на базата от данни.
- **Част трета** „Реализация на системата” – Реализация на базата от данни с Microsoft SQL server и описание на таблиците. Реализация на слоя за работа с базата от данни, на бизнес логиката и графичния интерфейс.
- **Част четвърта** „Тестови резултати” – Разработване на тестове за проверка на функционалността на системата.

2. Реализация на проекта

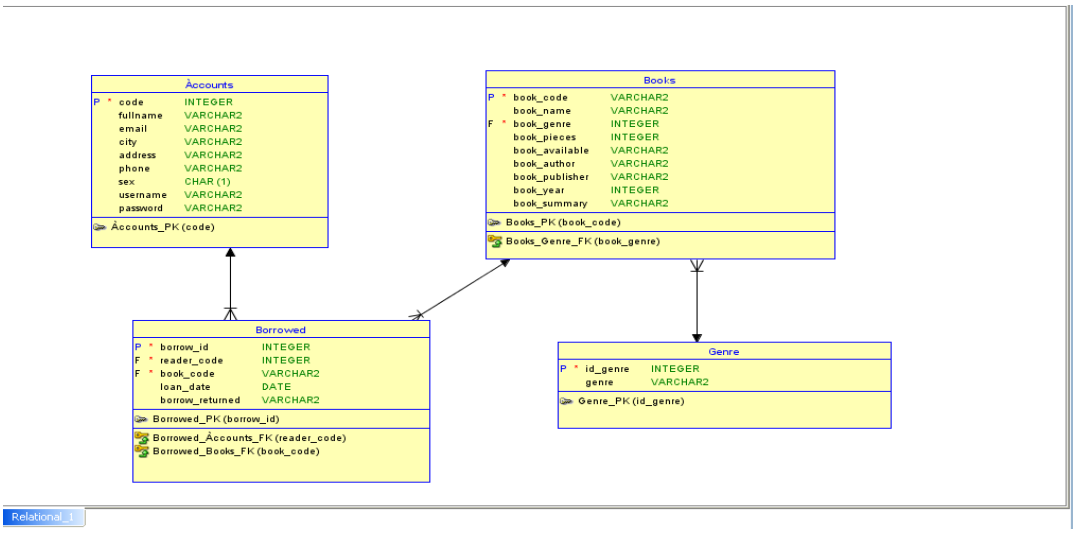
2.1 Проектиране на системата

2.1.1 База от данни

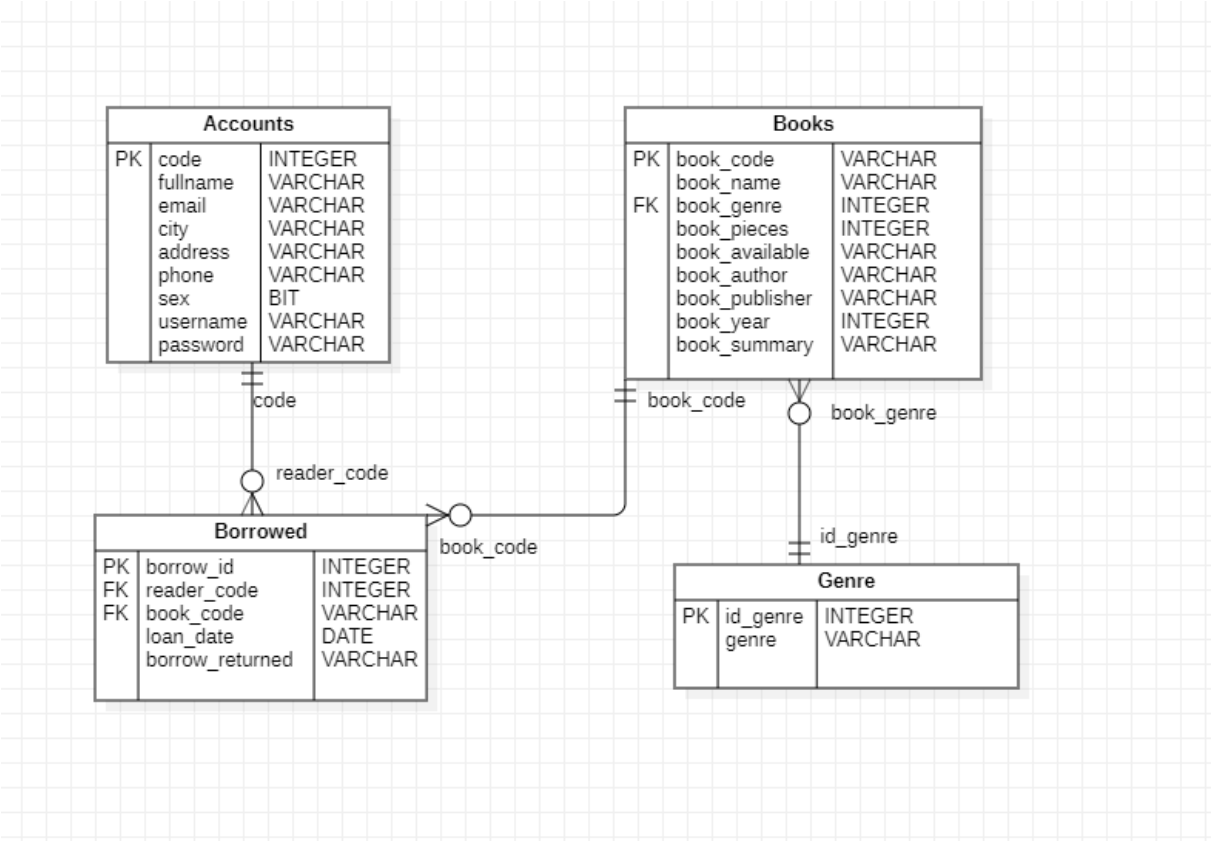
Логически модел:



Релационен модел:



ERD:



За хранилище на данните на приложението бе планирано използването на вградената услуга в Microsoft Visual Studio – Service Based Database, поради факта, че етапът по програмиране ще бъде реализиран по-бързо. Една от основните причини е заради предишния опит на екипа с технологията, както и лесното боравене с кода (SQL заявките се пишат по-лесно отколкото с използването на Access например).

Както логическият, така и релационният модел на базата от данни са проектирани чрез Oracle Data Modeler, а ERD е съставена с помощта на StarUML.

При етапа на планиране, екипът реши да проектира 4 таблици за съхранение на данни:

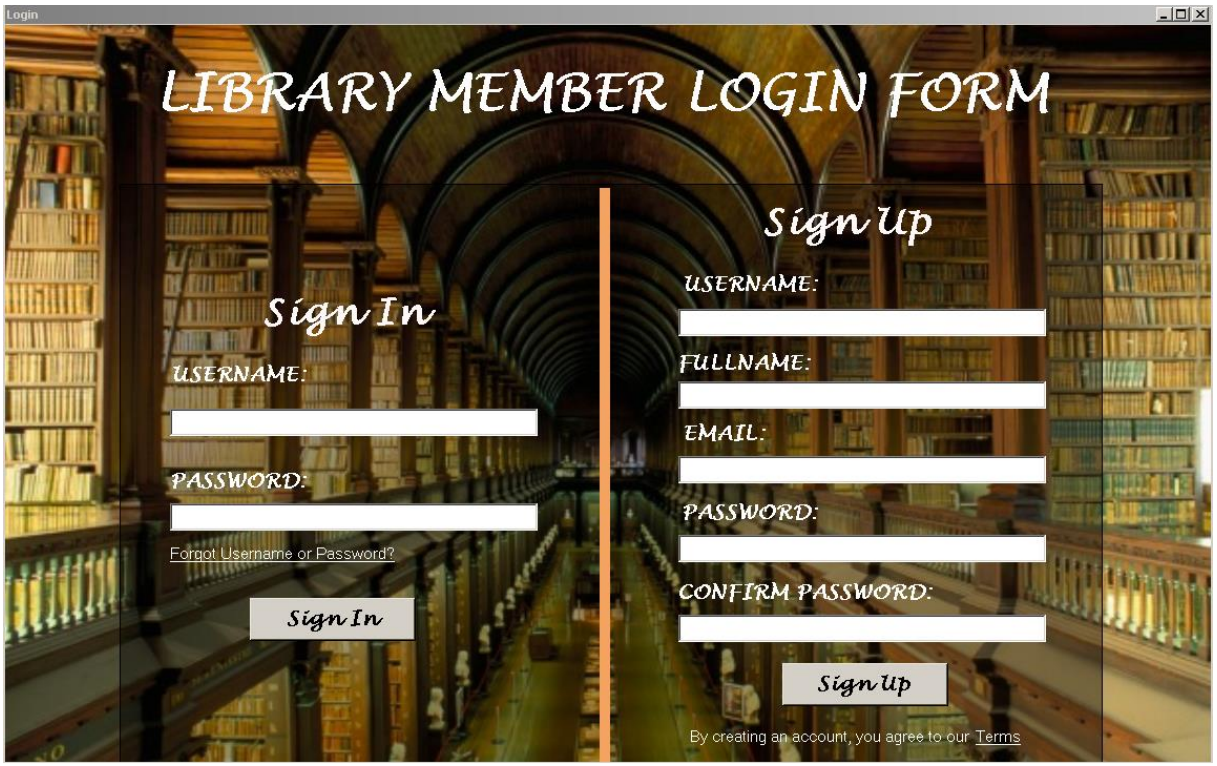
- таблица за акаунти на потребителите;
- таблица за книги и тяхната информация;
- таблица за заети книги от читатели;
- таблица за литературни жанрове.

2.1.2 Потребителски интерфейс (GUI)

За визуализация на потребителския интерфейс бе планирано използването на вградената услуга в Microsoft Visual Studio, при работа с Windows Forms Application, а именно – Windows Forms. При използването му, ръчно се добавя всеки контролер и се разработва всяка негова функционалност. Причина за избора бе лесното и удобното му използване, по-лесното разбиране на кода и най-вече – добрите познания на екипа за работата с тази технология.

Ето и илюстрация на някои от планираните основните форми в приложението:

- **Логин форма** (администратор или читател):



- **Основна (main) форма за читател**:

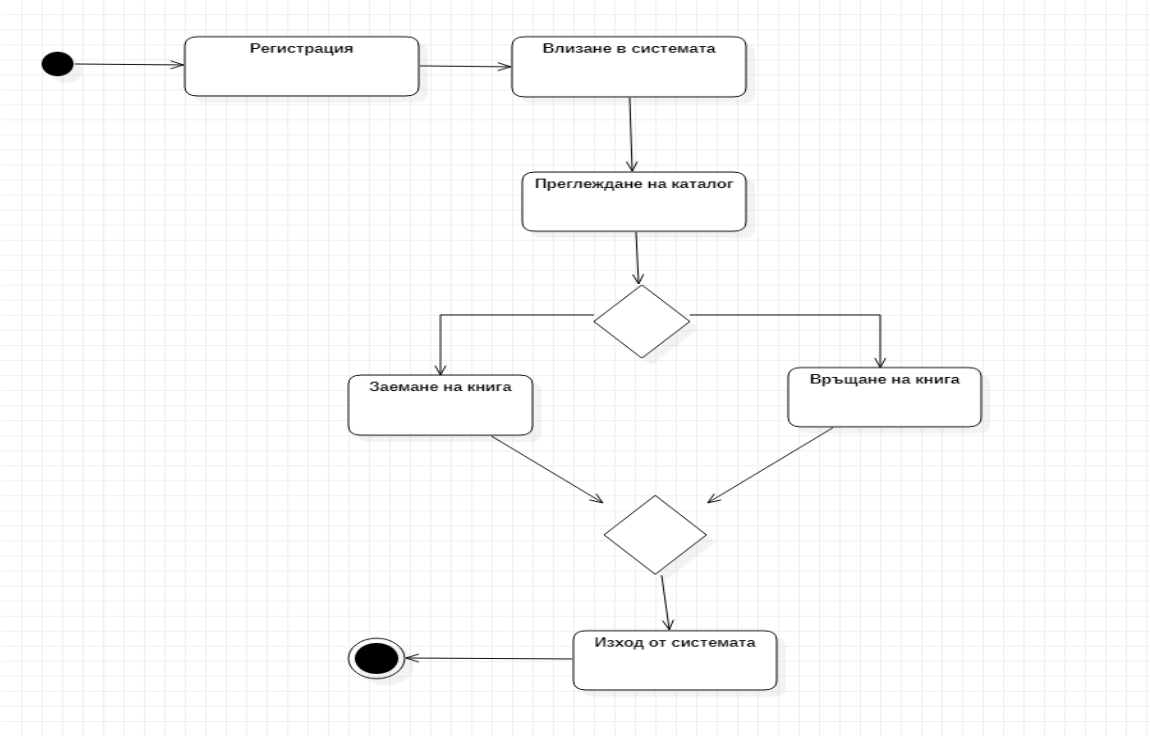
- **Основна (main) форма за администратор**:

2.1.3 UML диаграми

Диаграмите в настоящата курсова работа са проектирани с помощта на приложението **StarUML**.

2.1.3.1 Activity диаграма (проектиране на бизнес логика)

Диаграмата на дейностите показва как действа приложението за обикновения потребител и основните стъпки при работа с него:



Предвидено е при влизане в приложението, читателят да се регистрира в системата. При коректно въведени данни, ще може да въведе своето име и парола и да получи достъп до функционалностите на системата. Съответно той може да преглежда заетите от него книги или наличния каталог на библиотеката, след което при необходимост да заеме дадена книга или да върне вече заета такава. Системата ще му позволява да се върне обратно в екрана за вход и регистрация, както и за спиране на приложението.

2.1.3.2 Use Case диаграми

Планират се два вида потребители на системата:

• администратор
• читател

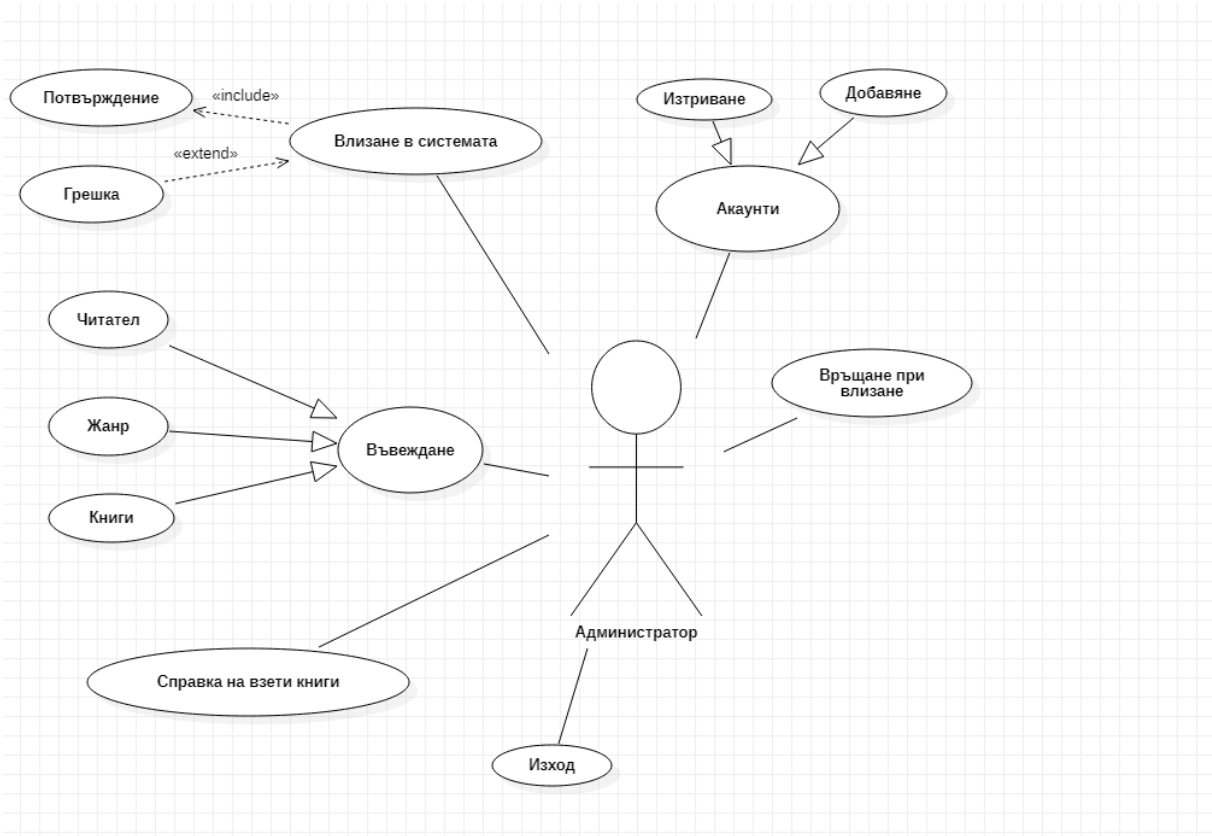
Администраторът на системата ще може да манипулира информацията в БД, т.е. да въвежда, коригира, изтрива или да търси определени данни във всички нейни таблици. Ще може да изготвя каталога на системата с наличните книги в библиотеката, както и тяхната бройка и освен това – да прави необходимите справки, които могат да бъдат:

- за взети книги;
- акаунти;
- книги

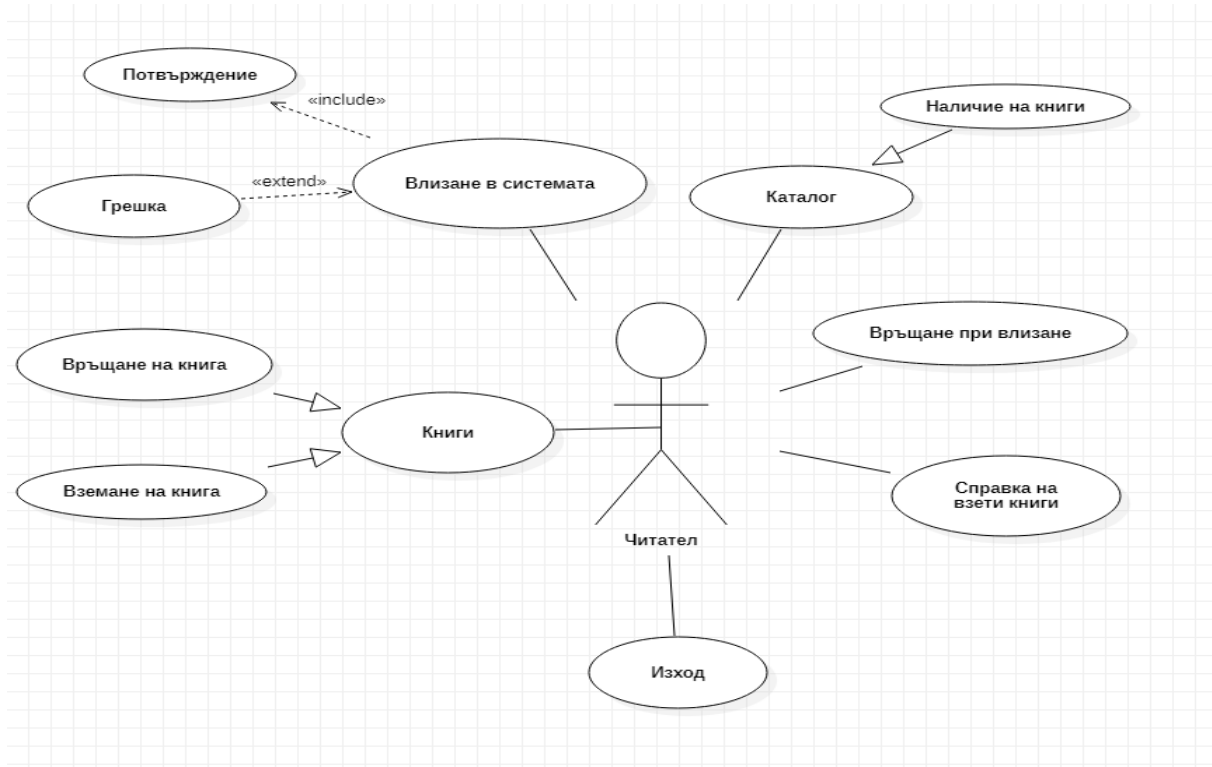
Обикновеният потребител ще може да преглежда своите заети книги от библиотеката, както и наличният каталог, изготвен от администратора на системата. Освен това, той ще може да заеме определена книга. Това става при въвеждане на нейния код в приложението, като то първо ще провери дали съществува такъв, дали има поне 1 бройка и след като всичко е проверено, книгата ще бъде записана в заетите книги на читателя и нейната бройка в системата ще се намали с 1. Обратното важи и за връщането на дадена книга – въвежда се нейният код, системата отново проверява за коректни данни и дали книгата присъства при съответния потребител, след което отстранява книгата от заетите от него книги и увеличава бройката ѝ в системата с 1.

Изготвени са следните UML Use Case диаграми:

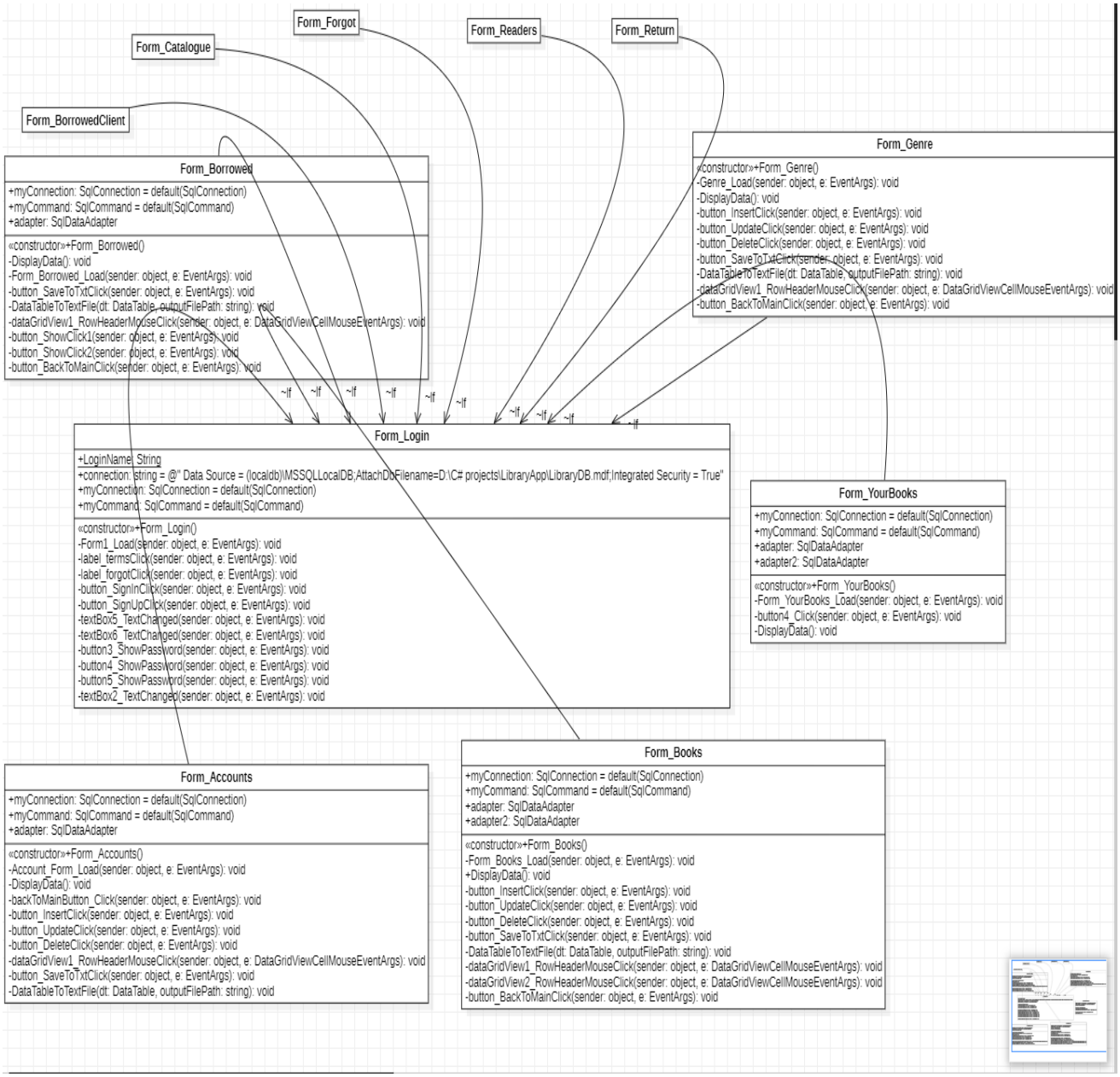
За администратор:



За читател:

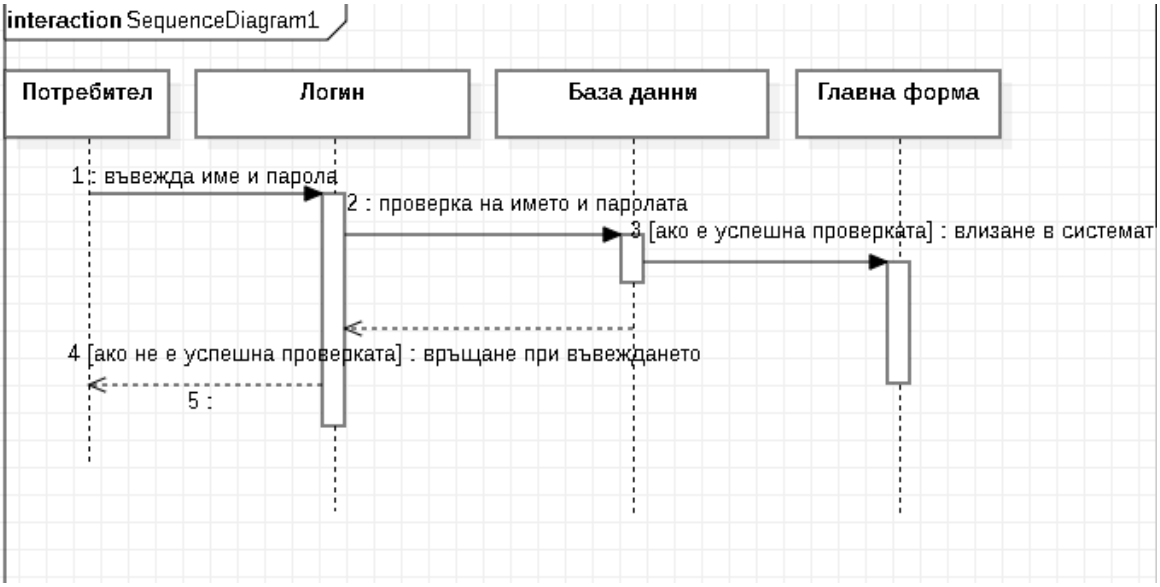


2.1.3.3 Class диаграма



Class диаграмата е автоматично генерирана от завършения код на приложението.

2.1.3.4 Sequence диаграма



2.2 Програмиране

След приключване на етапа по планиране, се започва с разработката на приложението, като първа стъпка е създаването и въвеждането на данни в БД.

2.2.1 Реализация на базата от данни

В приложението се добавя вградената функционалност в Microsoft Visual Studio – Service-based Database, като там са създадени планираните в ERD модела 4 таблици и са въведени примерни данни в тях.

- Създаване на таблица **Accounts(Акаунти)**, която съдържа следните данни**:**

code	Идентификационен номер на акаунт
fullname	Име на читател/администратор
email	Имейл адрес
city	Град
address	Адрес
phone	Телефон за връзка
sex	Пол
username	Потребителско име
password	Потребителска парола

След създаването ѝ е въведен един запис, който служи за разпознаване на администраторския акаунт, като за потребителско име и парола е въведена думата admin. Приложението проверява наличието на admin в двете текстови полета за вход и при намирането им в базата от данни се достъпва администраторската основна форма.

- Създаване на таблица **Books (Книги)**, която съдържа следните данни:

book_code	Идентификационен номер на книга
book_name	Име на книга
book_genre	Жанр на книга
book_pieces	Брой книги
book_available	Дали е налична книгата
book_author	Автор на книга
book_publisher	Издател на книгата
book_year	Година на издаване
book_summary	Информация за книгата

В тази таблица са въведени всички книги, с които разполага библиотеката, техните бройки, информацията за тях и др.

- Създаване на таблица **Borrowed (Заети книги)**, която съдържа следните данни:

borrow_id	Идентификационен номер на вземане на книга
reader_code	Идентификационен номер на читател
book_code	Идентификационен номер на книга
loan_date	Дата на взимане на книга
borrow_returned	Дали книгата е върната или не

В тази таблица не са въведени никакви данни. Тя ще почне да се запълва, когато читател използва функцията за заемане на книга.

- Създаване на таблица **Genre (Жанр)**, която съдържа следните данни:

id_genre	Идентификационен номер на жанр
genre	Име на жанра

В тази таблица са въведени всички литературни жанрове, като самата таблица служи за помощна на тази за филмите, като не позволява дублиране на данните в нея.

2.2.2 Реализация на бизнес логика

Приложението има за цел да помогне на регистрираните потребители в библиотеката да преглеждат, заемат и връщат книги, без да е необходима намесата на служебно лице. Също така помага и на персонала на предприятието да следи движението на книгите, коя книга къде се намира в даден момент, колко бройки от нея има в наличност и освен това да управлява читателските акаунти – при необходимост да добави, редактира или изтрие даден акаунт.

За достъп до слоя за работа с базата от данни се използва библиотеката „System.Data.SqlClient”, която позволява на програмиста да използва необходимите й фунцкии и обекти и от тип SqlConnection, SqlCommand, SqlParameter, SqlAdapter и др.

Връзката с базата от данни се записва в публична стрингова променлива, след което всяка функция (в същия или друг клас) я достъпва поотделно чрез обект от тип SqlConnection. SQL заявките се изпълняват, благодарение на обекта от тип SqlCommand, като си взаимодейства с обектите от SqlParameter – взимат се въведените данни в текст боксовете на приложението и се добавят в SQL заявката, там където има символ „@”. Ето пример за достъп до БД:

```
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 LibraryApp
{
    30 references
    public partial class Form_Login : Form
    {
        2 references
        public static String LoginName { get; set; }
        13 references
        public Form_Login()
        {
            InitializeComponent();
        }

        //public string connection = @" Data Source = (localdb)\MSSQLLocalDB;AttachDbFilename=D:\C# projects\LibraryApp\LibraryDB.mdf;Integrated Security = True";
        public string connection = @"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=D:\ТУ Варна\Семестър 6\ТСП - проект\LibraryApp\LibraryDB.mdf;Integrated Security=True
        public SqlConnection myConnection = default(SqlConnection);
        public SqlCommand myCommand = default(SqlCommand);
```

```
private void button_SignInClick(object sender, EventArgs e)
{
    try
    {
        myConnection = new SqlConnection(connection);
        myCommand = new SqlCommand("SELECT username, password FROM Accounts WHERE username = @username AND password = @password", myConnection);
        SqlParameter username = new SqlParameter("@username", SqlDbType.VarChar);
        SqlParameter password = new SqlParameter("@password", SqlDbType.VarChar);

        username.Value = textBox1.Text;
        password.Value = textBox2.Text;

        myCommand.Parameters.Add(username);
        myCommand.Parameters.Add(password);

        myCommand.Connection.Open();
        SqlDataReader myReader = myCommand.ExecuteReader(CommandBehavior.CloseConnection);

        if (myReader.Read() == true)
        {
            if(textBox1.Text == "admin" && textBox2.Text == "admin")
            {
                MessageBox.Show("Admin Login Successfull!", "Login Success");
                Form_MainAdmin admin = new Form_MainAdmin();
                admin.Show();
                this.Hide();
            }
            else
            {
                MessageBox.Show("Login Successfull!", "Login Success");
                Form_Login.LoginName = textBox1.Text;
                Form_MainUser user = new Form_MainUser();
                user.Show();
                this.Hide();
            }
        }
        else
        {
            if (textBox1.Text == "" && textBox2.Text == "")
                MessageBox.Show("Username and Password cannot be empty!", "Login Denied", MessageBoxButtons.OK, MessageBoxIcon.Error);
            else if (textBox1.Text == "")
                MessageBox.Show("Username cannot be empty!", "Login Denied", MessageBoxButtons.OK, MessageBoxIcon.Error);
            else if (textBox2.Text == "")
                MessageBox.Show("Password cannot be empty!", "Login Denied", MessageBoxButtons.OK, MessageBoxIcon.Error);
            else if(textBox1.Text != username.ToString())
                MessageBox.Show("Username or Password not found!", "Login Denied", MessageBoxButtons.OK, MessageBoxIcon.Error);
            else
                MessageBox.Show("Login failed! Try again!", "Login Denied", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }

        if (myConnection.State == ConnectionState.Open)
            myConnection.Dispose();
```

При стартиране на приложението се показва екран за вход и регистрация, който е разделен на 2 части – лявата страна на формата е за логване в системата, дясната за регистрация.

За регистрацията е необходимо да се зададе:

- Потребителско име;
- Пълното име на читателя;
- Имейл адрес;
- Парола;
- Повторена парола.

Системата проверява за несъответствия на въведените от потребителя данни, например при празни полета или ако липсва символ „@” за имейл полето, системата извежда съобщение за конкретната грешка. При успешно валидирани данни, те се добавят в базата от данни и по-конкретно – в таблица „Accounts”. Допълнителна информация за акаунтите се добавя от администратора, след като потребителят е попълнил карта със своите данни и я предостави на служебно лице.

След направената регистрация, потребителят може да влезе в своя профил. Системата проверява дали потребителят, който се опитва да влезе е администратор или не, като това става с проверка на името и паролата на администратора.

Администраторският профил може да управлява и манипулира данните в приложението. Това включва акаунтите, книгите, наличността на дадена книга, изготвяне на каталог за потребителя и др.

Читателският профил от своя страна може да разглежда направения от администратора каталог, да заеме дадена книга, ако тя е в наличност или да върне вече заета такава, като при заемането системата намаля бройките на съответната книга с 1, а при връщането – увеличава с 1.

Графичният интерфейс е разработен с Windows Forms чрез средата за разработка Microsoft Visual Studio. Всяка отделна форма има снимка за background, съответстваща с формата, прозрачен панел, label-и, текстови полета, бутони и др.

За шрифт на текста е използван Lucida Handwriting, който придава красив античен стил на формите (калиграфски шрифт).

Първата създадена форма е тази за влизане и регистрация в системата. В нея се намира и главната връзка с базата от данни. В останалите форми се създава обект от тази форма и така конекцията се достъпва.

Логин формата включва следните компоненти:

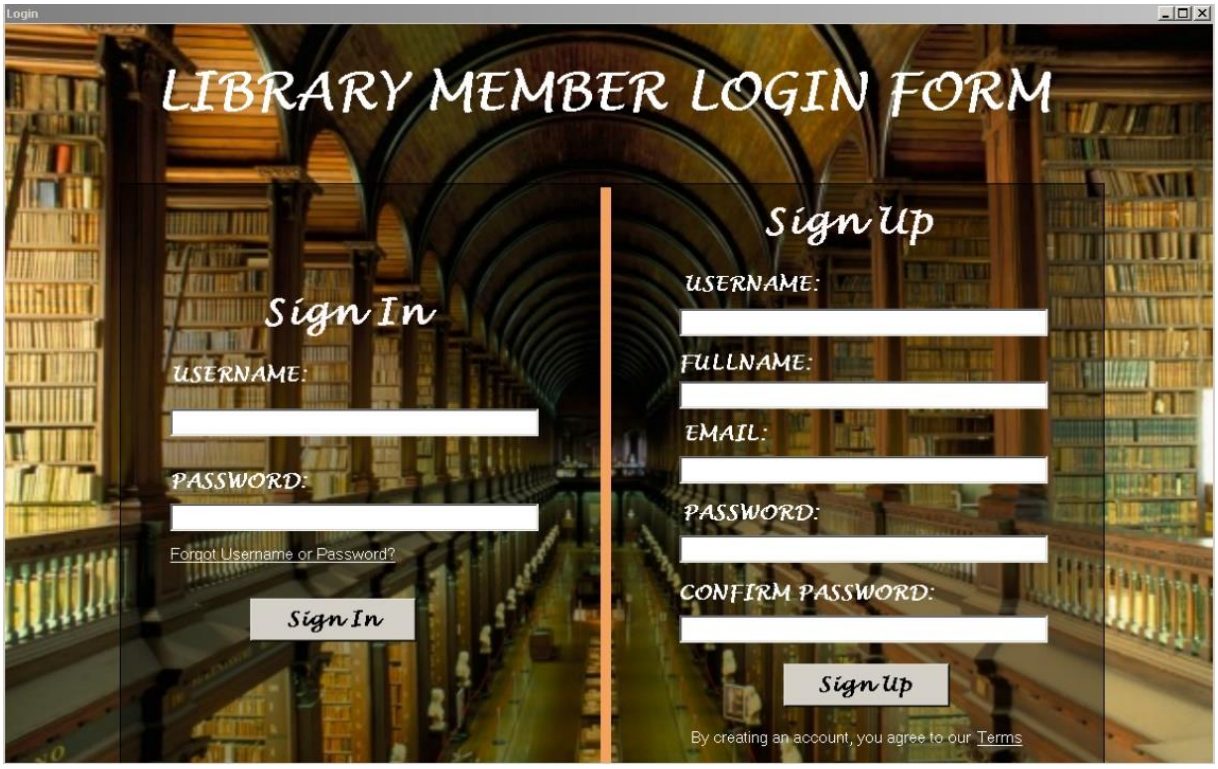
- Label и LabelLink;
- TextBox;
- Button;
- Panel.

Формата позволява да се направи регистрация на потребител, като проверява за вече съществуващи такива данни или за други несъответствия, като празни полета и др. Позволява достъп до основната форма на приложението, като проверява дали въведеното потребителско име и парола съответстват на

администраторския или на отделен читателски профил. При некоректни данни се извежда съобщение за грешка.

Освен това формата дава достъп до друга форма, която е свързана с разкриване на името и паролата на даден потребител, както и тяхната промяна и до Terms of Service.

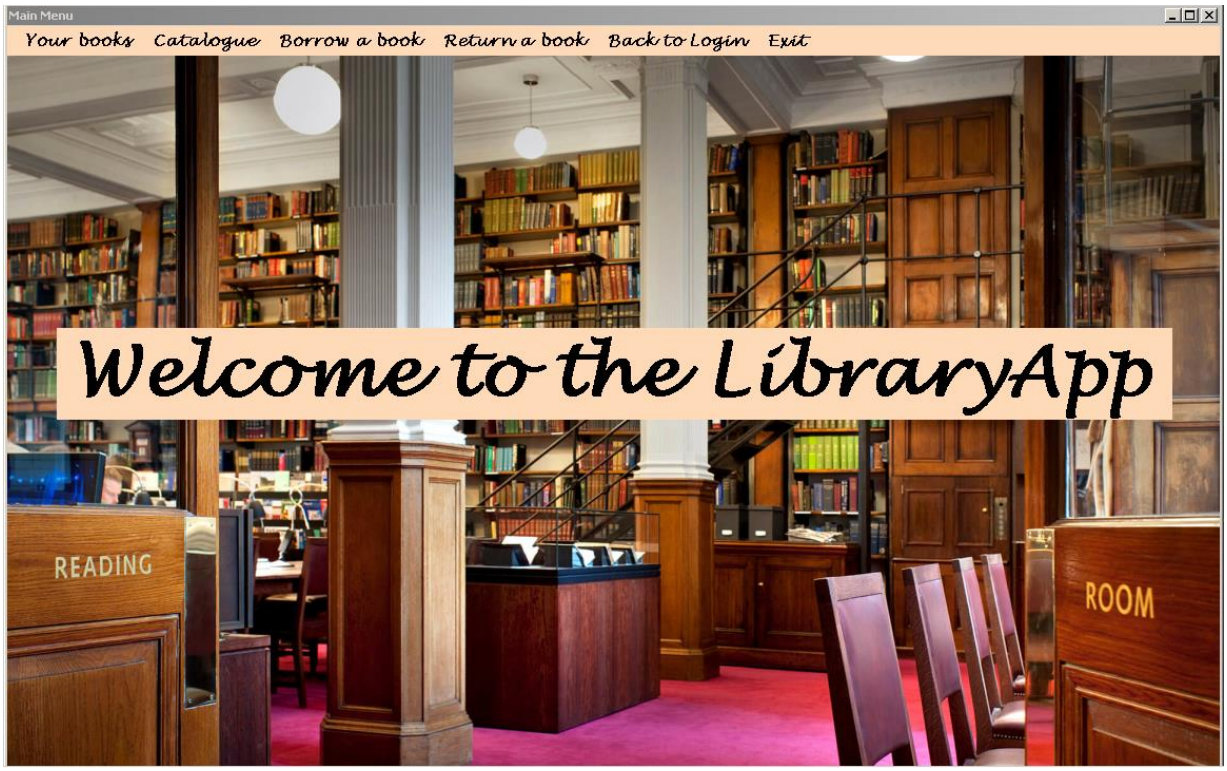
Графично изобразяване на формата за въвеждане:



При коректно въведени данни (ако потребителското име и парола присъстват в БД) се достъпва главната форма, която е разделена на 2 части – администраторска и потребителска. И двете форми съдържат меню за достъп до различните фунцкии, снимка на фона и текст по средата.

Администраторска основна форма:

Потребителска основна форма:



Функция за добавяне на данни в хранилището:

```
1 reference
private void button_InsertClick(object sender, EventArgs e)
{
    try
    {
        string check = "[A-Za-z]";
        if (textBox1.Text == "" || textBox2.Text == "" || textBox3.Text == "" || textBox4.Text == "")
            MessageBox.Show("The first 4 fields cannot be empty", "Empty fields", MessageBoxButtons.OK, MessageBoxIcon.Error);
        else if(Regex.IsMatch(textBox1.Text, check))
            MessageBox.Show("The book code must contains only numbers!", "Code error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        else
        {
            myConnection = new SqlConnection(lf.connection);
            myCommand = new SqlCommand("INSERT INTO Books VALUES(@code, @name, @genre, @pieces, @available, @author, @publisher, @year, @summary)", myConnection);

            myConnection.Open();
            myCommand.Parameters.AddWithValue("@code", textBox1.Text);
            myCommand.Parameters.AddWithValue("@name", textBox2.Text);
            myCommand.Parameters.AddWithValue("@genre", textBox3.Text);
            myCommand.Parameters.AddWithValue("@pieces", textBox4.Text);
            string checkIfAvailable = "";
            if (radioButton1.Checked)
                checkIfAvailable = "Yes";
            else
                checkIfAvailable = "No";
            myCommand.Parameters.AddWithValue("@available", checkIfAvailable);
            myCommand.Parameters.AddWithValue("@author", textBox5.Text);
            myCommand.Parameters.AddWithValue("@publisher", textBox6.Text);
            myCommand.Parameters.AddWithValue("@year", textBox7.Text);
            myCommand.Parameters.AddWithValue("@summary", richTextBox1.Text);

            myCommand.ExecuteNonQuery();
            MessageBox.Show("Book added successfully!");

            myConnection.Close();
            DisplayData();

            if (myConnection.State == ConnectionState.Open)
                myConnection.Dispose();

            textBox1.Clear();
            textBox2.Clear();
            textBox3.Clear();
            textBox4.Clear();
            textBox5.Clear();
            textBox6.Clear();
            textBox7.Clear();
            richTextBox1.Clear();
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Функция за коригиране на данни в хранилището:

```
1 reference
private void button_UpdateClick(object sender, EventArgs e)
{
    try
    {
        myConnection = new SqlConnection(lf.connection);
        myCommand = new SqlCommand("UPDATE Books SET book_pieces = @pieces, book_available = @available WHERE book_code = @code", myConnection);
        SqlCommand checkCode = new SqlCommand("SELECT book_code FROM Books WHERE book_code = @code", myConnection);
        myConnection.Open();
        myCommand.Parameters.AddWithValue("@code", textBox1.Text);
        myCommand.Parameters.AddWithValue("@pieces", textBox4.Text);
        string checkIfAvailable = "";
        if (radioButton1.Checked)
            checkIfAvailable = "Yes";
        else
            checkIfAvailable = "No";
        myCommand.Parameters.AddWithValue("@available", checkIfAvailable);

        checkCode.Parameters.AddWithValue("@code", textBox1.Text);

        SqlDataReader sdr = checkCode.ExecuteReader();

        if (!sdr.HasRows)
            MessageBox.Show("Code not found", "Register Denied", MessageBoxButtons.OK, MessageBoxIcon.Error);
        else
            sdr.Close();

        if (textBox1.Text == "" || textBox4.Text == "")
            MessageBox.Show("The fields cannot be empty!", "Empty fields", MessageBoxButtons.OK, MessageBoxIcon.Error);
        else
        {
            myCommand.ExecuteNonQuery();
            myConnection.Close();

            MessageBox.Show("Book availability updated successfully!");
            DisplayData();
        }

        if (myConnection.State == ConnectionState.Open)
            myConnection.Dispose();

        textBox1.Clear();
        textBox4.Clear();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Функции за изтриване на данни в хранилището и принтиране на справки:

```
1 reference
private void button_DeleteClick(object sender, EventArgs e)
{
    try
    {
        myConnection = new SqlConnection(lf.connection);
        myCommand = new SqlCommand("DELETE Books WHERE book_code = @code", myConnection);
        myConnection.Open();
        myCommand.Parameters.AddWithValue("@code", textBox1.Text);

        myCommand.ExecuteNonQuery();
        myConnection.Close();

        MessageBox.Show("Book deleted successfully!");
        DisplayData();

        if (myConnection.State == ConnectionState.Open)
            myConnection.Dispose();

        textBox1.Clear();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

1 reference
private void button_SaveToTxtClick(object sender, EventArgs e)
{
    string connectionString = null;
    connectionString = lf.connection;

    DataTable dt = new DataTable();
    foreach (DataGridViewTextBoxColumn column in dataGridView1.Columns)
        dt.Columns.Add(column.Name, column.ValueType);

    foreach (DataGridViewRow row in dataGridView1.Rows)
    {
        DataRow dr = dt.NewRow();
        foreach (DataGridViewTextBoxColumn column in dataGridView1.Columns)
            if (row.Cells[column.Name].Value != null)
                dr[column.Name] = row.Cells[column.Name].Value.ToString();
        dt.Rows.Add(dr);
    }

    string filePath = "D:\\ТУ Варна\\Семестър 6\\ТСП - проект\\LibraryApp\\Справки\\Books.txt";
    DataTableToTextFile(dt, filePath);
    MessageBox.Show("Data saved successfully!", "Data saved!");
}
```

```
1 reference
private void DataTableToTextFile(DataTable dt, string outputFilePath)
{
    int[] maxLengths = new int[dt.Columns.Count];
    for (int i = 0; i < dt.Columns.Count; i++)
    {
        maxLengths[i] = dt.Columns[i].ColumnName.Length;
        foreach (DataRow row in dt.Rows)
        {
            if (!row.IsNull(i))
            {
                int length = row[i].ToString().Length;
                if (length > maxLengths[i])
                    maxLengths[i] = length;
            }
        }
    }

    try
    {
        using (StreamWriter sw = new StreamWriter(outputFilePath, false))
        {
            for (int i = 0; i < dt.Columns.Count; i++)
                sw.Write(dt.Columns[i].ColumnName.PadRight(maxLengths[i] + 2));

            sw.WriteLine();
            foreach (DataRow row in dt.Rows)
            {
                for (int i = 0; i < dt.Columns.Count; i++)
                {
                    if (!row.IsNull(i))
                        sw.Write(row[i].ToString().PadRight(maxLengths[i] + 2));
                    else
                        sw.Write(new string(' ', maxLengths[i] + 2));
                }
                sw.WriteLine();
            }
            sw.Close();
        }
    }
    catch { }
}
```

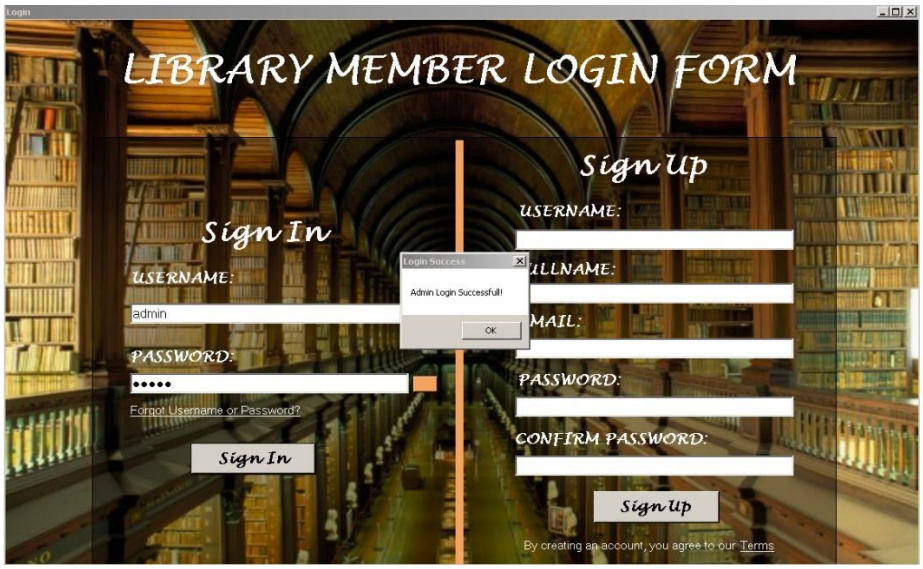
3.Тестване на приложението и възможности за развитие

3.1 Таблица с тестови резултати

След завършване на приложението, следва етапът по цялостното му тестване. Ако даден тест е преминат успешно се записва в тип на таблицата – успешен. Ако даден тест не е преминат се записва – грешка.

Име на тест:	Тип на тест:
Свързване с базата от данни	Успешен
Проверка за потребител и парола	Успешен
Добавяне на данни в базата	Успешен
Редактиране на данни в базата	Успешен
Изтриване на данни в базата	Успешен
Запазване в текстов файл	Успешен
Влизане в системата	Успешен
Създаване на акаунт	Успешен
Промяна на име и парола	Успешен
Преглеждане на общите условия	Успешен
Търсене на данни в базата	Успешен
Валидация на данни	Успешен
Функционални тестове	Успешен
Нефункционални тестове	Успешен
Цялостна проверка	Успешен

Примерен тест за успешен вход на администратор:



3.2 Възможности за бъдещо развитие

Разработеното десктоп приложение е все още ново (Alpha Release) и има доста възможности за бъдещо подобрение и оптимизация, като например:

- Разработване на web и мобилна версия;
- Възможност за закриване на акаунт от читател;
- Смяна на хранилището към MySQL или Oracle;
- Подобряване на потребителския интерфейс;
- Онлайн разглеждане на съдържанието на дадена книга;
- Възстановяване на акаунт чрез имейл;
- Оптимизация на кода;
- И др.

4. Използвана литература

https://stackoverflow.com

https://www.geeksforgeeks.org

https://www.tutorialspoint.com/uml/uml_standard_diagrams.htm

https://docs.microsoft.com/en-us/dotnet/desktop/winforms/overview/?view=netdesktop-5.0

https://www.w3schools.com/cs/

https://www.youtube.com

https://www.introprogramming.info/wp-content/uploads/2013/07/Books/CSharpEn/Fundamentals-of-Computer-Programming-with-CSharp-Nakov-eBook-v2013.pdf

https://www.tutorialsteacher.com/csharp/csharp-tutorials

https://zetcode.com/csharp/mysql/