ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – ВАРНА

ФАКУЛТЕТ ПО ИЗЧИСЛИТЕЛНА ТЕХНИКА И АВТОМАТИЗАЦИЯ

Катедра „Софтуерни и интернет технологии“

ДИПЛОМНА РАБОТА

На Тема:

Проектиране и разработване на система за управление на Айкидо клуб

Изготвил: Мила Веселинова Ботева

Факултетен номер: 18621708

Специалност: Софтуерни и интернет технологии

Ръководител: хон. ас. Павлина Линова

ТУ Варна, 2022 г.

Съдържание

I. ГЛАВА. Увод

I.1. Постановка на дипломно задание

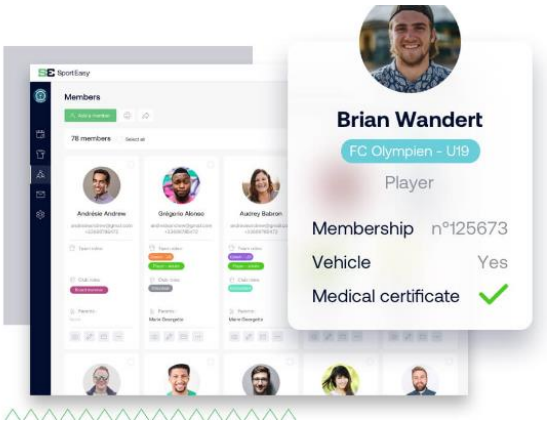
Проектиране и разработка на система за управление на Айкидо Клуб

Да се реализира десктоп приложение, което позволява възможност за вход, съхранение и управление на данните в системата. Да позволява на потребителя да въвежда и обработва максимално много данни за участниците в клуба и за самият клуб. Интерфейсът на приложението трябва да е лесен за използване и да позволява бърза обработка на информацията. Приложението да е сигурно и да защитава личните данни на трениращите.

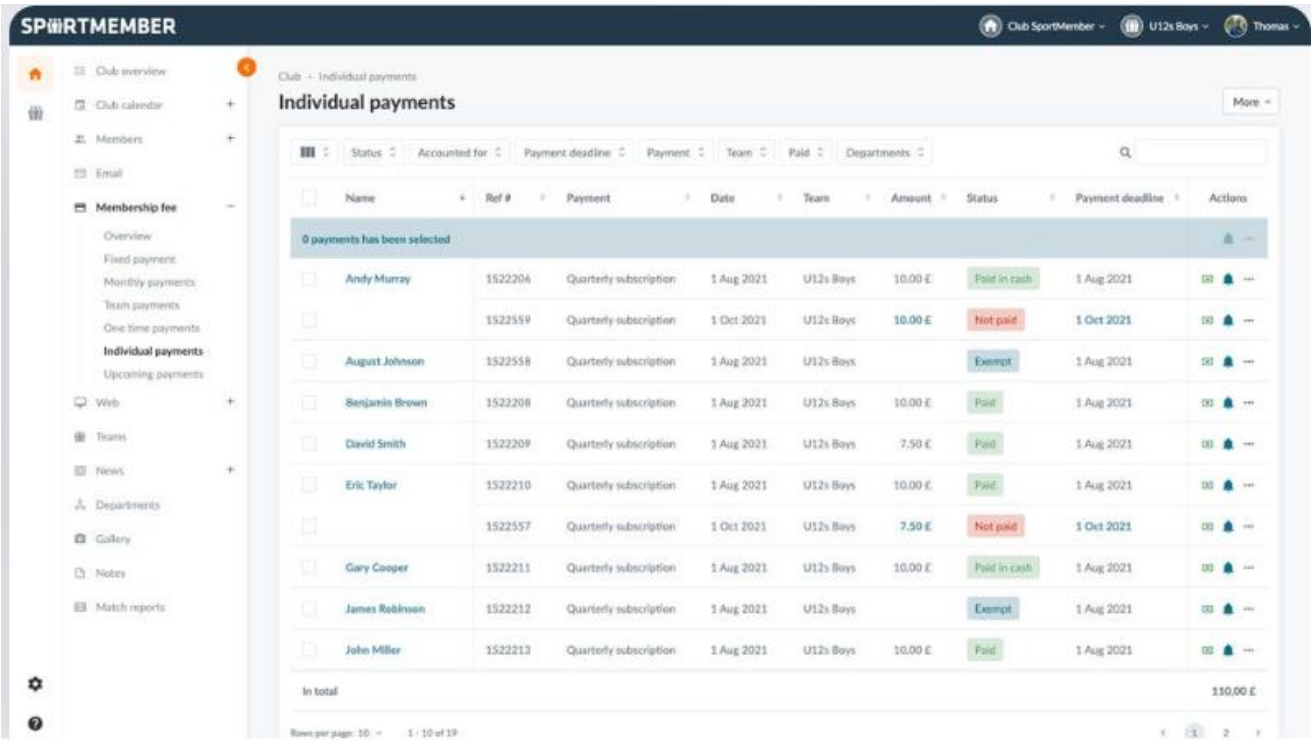
I.2. Проучване за съществуващи решения

В областта на бойните изкуства е трудно да се намери система, с която адекватно да бъдат управлявани клубовете. Платформите , които се предлагат на потребителите, нямат нужните функционалности и възможностти, за да задоволят нуждите на един клуб по бойни изкуства.

При направено проучване на пазара за такъв софтуер са намерени приложенията „www.sportmember.co.uk “ и „sporteasy “. Екранни снимки от платформите са изобразени на фигури 1 и 2.



Фигура 1-Показва снимка от sporteasy



Фигура 2-показва снимка от sportmember

Наблюденията сочат, че

1. Софтуерите предлагат голяма част от нужните функционалности, но липсват специфични елементи като защитени степени на всеки картотекиран.
2. Има изобилие от продукти обслужващи продукти, които могат даа бъдат използвани са повечето спортни клубове, но би било трудно да бъдат използвани за управление на клуб по бойни изкуства.

3. Предимство на повечето платформи е разработката в тип уеб приложение, което предоставя удобството да се използва системата от разлини точки.
4. Недостатък на предлаганите софтуери е липсата на български език. Булшинството от населението знаят чужди езици, но липсата на език може е предпоставка за недосатъчно удовлетворение при използване на продукта.

I.3. Цели на разработката

Целта на разработка е да бъдат подпомогнати собствениците на клубове по бойни изкуства. Програмата трябва да улесни управлението на клубовете и да предостави добър начин са проследяване на дейността в клуба.

- Трябва да се изпълнява бързо.
- Трябва дизайнът да бъде ефкасен и лесен за използване.
- Трябва да поддържа запазване на специфични данни като:
    - Степени на тренiращите.
    - Картотеки.
    - Проведени семинари и участие на тренiращите.
- Програмата трябва да има възможност да подържа акаунти за различен достъп.
- Поради наличието на лични данни, тя трябва да е локална, за да се ограничи риска от изтичане на информация от системата.
- Програмата трябва да създава график на треньорите.
- Програмата трябва да организира залите и групите.

II. ГЛАВА. Анализ на проблема

II.1. Проблемът

Бойните изкуства са специфични и имат много различни детайли от всички други спортове. Работата на един треньор изисква много компетентности и работа с много информация. Всеки клуб има комплексни дейности и е нужно те да бъдат проследявани за да има добра отчестно клуба към по големите организации. Проблемът е обсъждан между много собственици на клубове по бойни изкуства.

Такъв софтуер би бил много полезен и ще олесни упралението на клубовете в България. По този начин ще бъде намалено времето за обработка на данните и треньорите ще могат да се фокусират в по активната част на спорта.

Притеснението на всеки треньор е ,че може да изтекат данни и се страхуват от модерните технологии за отдалечен достъп. Повечето платформи работят като уеб приложения и точно поради тази причина те не са желани от ръководителите на клубовете по бойни изкуства.

Локално приложение, което да бъде инсталирано и фиксирано на една единствена компютърна система, ще бъде ефективна и ще предостави спокойствие и сигурност на треньорите.

II.2. Решение

Решение на проблема е софтуер с локално съхранение на данните, което да дава възможност за пълно управление на всички данни (Например:

създаване на график на треньорите в клуба, запазване на данни свързани с картотекираните в клуба, участия в семинари, защитени степени и др).

Важно е да се бъде взето под внимание, че такъв софтуер повишава ефективността на клуба и неговото развитие.

III. ГЛАВА. Избор на технологии

Фигура 3 - .NET Framework

За реализирането на софтуерното приложение ще бъде използвана библиотеката Windows Forms, заложена в .NET Framework (Фиг. 3). Тя е с отворен код и позволява сравнително лесна работа с потребителския интерфейс, както и лесно се интегрира с база данни. Широко е използвана за създаване на десктоп приложения за Windows и затова ресурсите в интернет пространството са почти неограничени в посока за работа с библиотеката. Съвместима е с всички съвременни версии на Windows OS.

III.1. C#

Фигура 4 - C# програмен език

C# (Фиг. 44) е обектно-ориентиран език за програмиране, разработен от Microsoft като част от софтуерната платформа .NET. Стремежът още при създаването му е бил да се създаде прост, модерен, обектно-ориентиран език с общо

предназначение. Изключително лесен за писане откъм getter-и и setter-и на членпроменливи на класовете. Обектно-ориентираните езици позволяват добра подредба, удобство при писане на кода, преизползване на функционалности, лесно отстраняване на грешки и полиморфизъм.

III.2. Microsoft SQL Server

Фигура 5 - Microsoft SQL Server

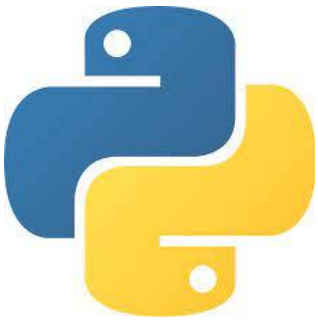
SQL е популярен език за програмиране, предназначен за създаване, видоизменяне, извличане и обработване на данни от релационни системи за управление на бази данни.

Microsoft SQL Server (Фиг. 5) е система за управление на релационни бази данни, която помага за съхраняване и извличане на информация според търсенето. Тя работи с SQL. Разработена е от Microsoft и е един от най-добрите дизайни, които могат да се конкурират с Oracle Database и MySQL. Интегрира се лесно с .NET Framework, което я прави идеалният избор на система за база данни за едно Windows Forms десктоп приложение.

В приложението е използвана олекотената версия на Express, която има всички негови функции за програмиране, работи в потребителски режим и има бърза инсталация без конфигуриране. Казва се „SQL Server Express LocalDB“. Тя е напълно подходяща за вграждане в по-малки приложения.

Базата данни е интегрирана в програмата под формата на MDF (SQL Server Database File) файл.

III.3. Python



Фигура 6 - Microsoft SQL Server

Python се нарежда сред най-популярните и бързо развиващите се езици за програмиране в днешно време. Той е с общо предназначение, обектно-ориентиран и изключително гъвкав, като намира приложение в много области (анализ на данни, машинно обучение, разработка на приложения и др.) и е използван от широк кръг специалисти (програмисти, бизнес анализатори, математици и др.).

В приложенинието езикът намира приложение чрез скрипт за създаване на резервни копия на базата от данни.

III.4. Git и GitHub

Фигура 7 – Git *Фигура 8 - GitHub*

При разработката на софтуерния продукт беше използвана системата за контрол на версиите Git (Фиг. 7). Тя е децентрализирана, съвременна и широко разпространена в бранша за разработка на софтуерни приложения.

За хранилище се използва GitHub (Фиг. 8). Системата дава възможност да бъдат проследени промените в разработката и изграждането на приложението. По този начин разработката е улеснена и ефективна.

IV. ГЛАВА. Проектиране на системата

IV.1. Описание на приложението

Приложението е предназначено клубове по бойни изкуства. Втози случай е създадена версия за конкретен айкидо клуб и присъстват имената и логото на клуба. Реализиран е вход с акаунт.

Работа с приложението могат да имат админ и главен треньор или ако глвния треньор прецени да бъде създаден акаунт и за помощник треньори.

Интерфейсът на програмата се състои от:

1. Първоначален прозорец за вход в програмата.
2. Главен прозорец с меню списък в ляво, в което се достъпват основните раздели:
    a. Картотеки
    b. Членски внос
    c. График
    d. Групи
    e. Семинари
    f. Степени на клуба
    g. Админ панел
3. Второстепенни прозорци за въвеждане, редакция и изтриване на данни

Всеки бутон от страничната лента зарежда при кликане, списъците за конкретния сектор и присъстват бутони за управление на данните(добави, изтрий или редактирай).

Админ панелът съдържа в себе си четири бутона, които препращат към форми за управление на данните в програмата за типовете членски такси, акаунтите, залите на клуба и степените, които могат да бъдат придобити. Акаунтите разграничават достъпа до управлението на тези данни. Профилите на треньорите нямат достъп до тези данни, а само до останалите основни менюта и второстепенните към тях.

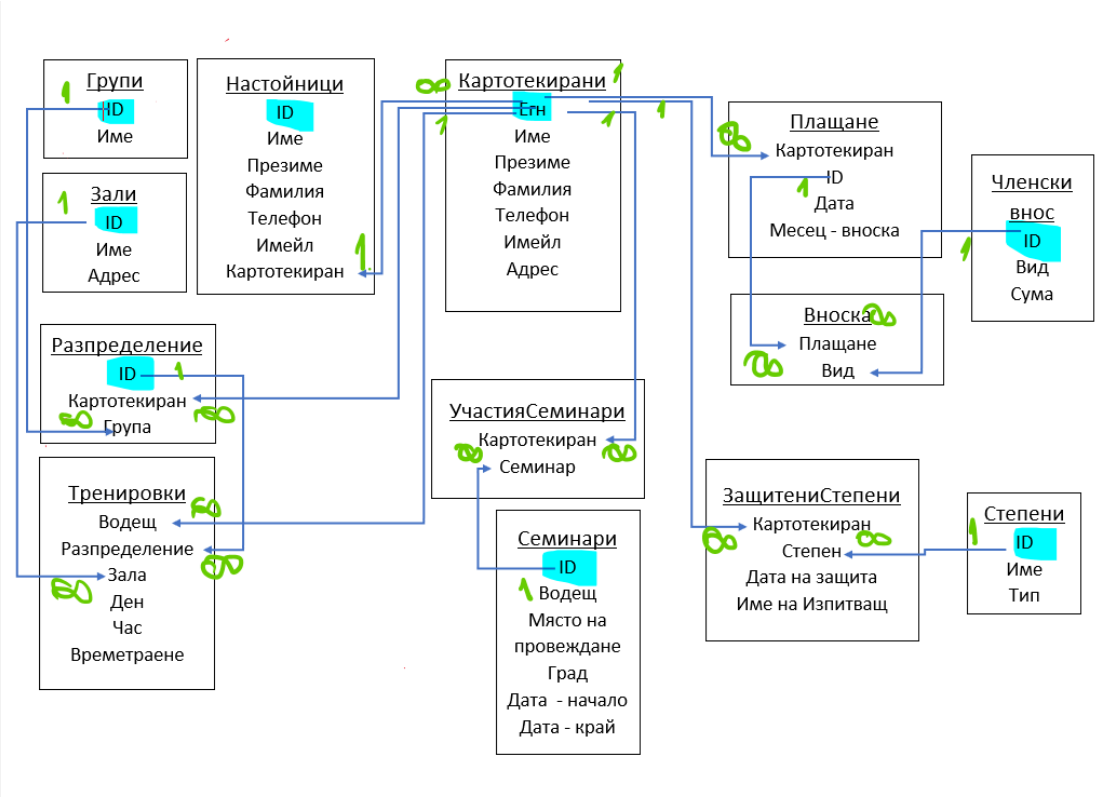
Акаунтите се обработват като лични данни, следователно паролите се пазят като хеш код в базата данни.

Приложението има за цел да организира целия процес в един Айкидо клуб. Дейностите са изразени във обработка на данни на картотекираните трениращи, обработка на данни на членските вноски, създаване на графици с треньори в клуба и разпределение на картотекираните по групи и зали, въвеждане на проведените семинари и участвлите от клуба в тях, провеждане на изпити , въвеждане на степени.

IV.2. Проектиране на системата

IV.2.1. Прототип

Поради краткото време за изработка, приложението не е преминавало през етап прототип, но е важно как ще бъде планирано проектирането. На фигура 9 може да се види работният модел на базата данни. Тя петърпя множество промени преди да бъде започната работа по приложението.



Фигура 9 -работен модел на базата данни

IV.2.2. Спецификации

Минимални изисквания за програмата:

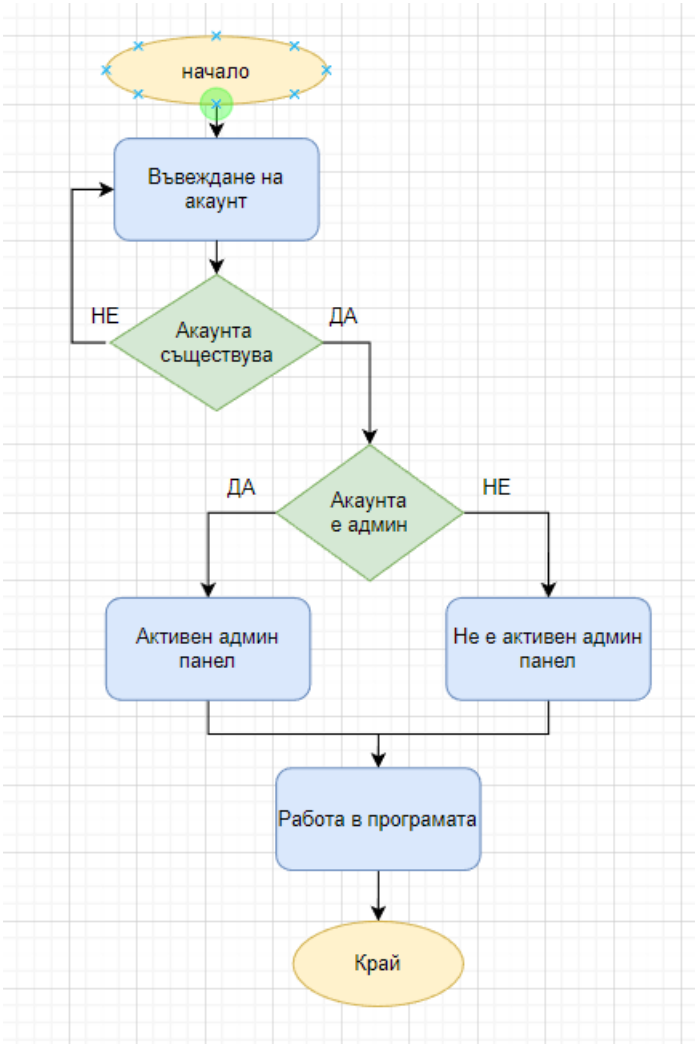
- Минимална честота на часовника на процесора:
  - 1 GHz
- Минимално количество RAM:
  - 4GB
- .NET Framework (включен е в инсталатора):
  - 4.8 или по-нова версия

- Операционна система: ○ Windows 8.1 (32/64 bit) ○ Windows 10 (32/64 bit) ○ Windows 11 (64 bit)

IV.3. Описание на програмните компоненти

Програмата е десктоп приложение, което се състои от един главен прозорец, прозорец за вход и осем второстепенни форми.

Модулите на системата са три:



Фигура 10 – Блок схема на акаунт модула

- Модул на вход с акаунт фир.10
- Модул за управление на данните за графика, семинарите, проведените изпити и картотекираните в клуба
- Модул на администратор

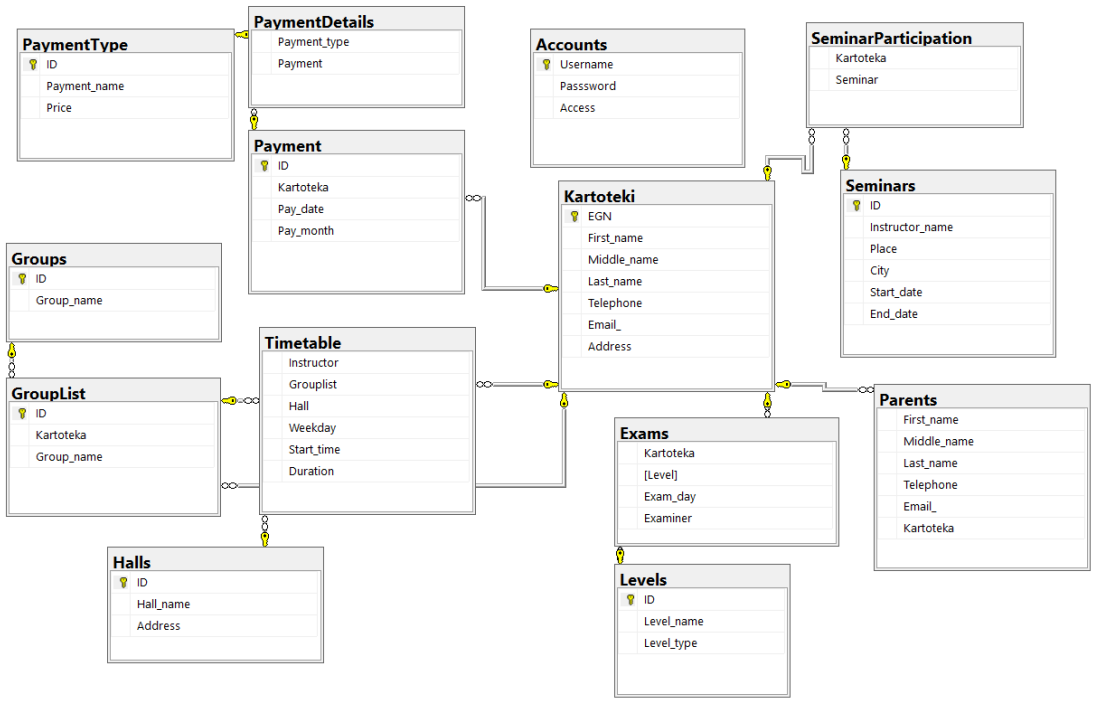
Програмата интегрира бази данни (.mdf файл). Тя се достъпва при вход с акаунт, записване , редактира или изтриване на данни от нея, както и при визуализация на данни в прозорците за справки.

Всеки админ или треньор има възможност да въвежда, редактира или изтрива данни за:

- Картотекираните членове на клуба, за проведените изпити(кой картотекиран се явил,кога се е явил и каква степен е защитил)
- Членските вноски в клуба (Кой картотекиран е заплатил вноска и за кой месец я заплаща)
- График на треньорите в клуба. Треньора може да управлява чрез приложението разпределението на групите по зали и водещият им треньор.
- Групите в клуба и нейните членове.
- Проведени семинари и участниците от клуба.
- Проведените изпити(Кои картотекирани, на коя дата и коя степен са защитили)

IV.3.1. База данни

Базата данни сдържа 13 таблици за данните на програмата и 1 за акаунтите



Фигура 11 - Релационен модел на базата данни

На релационната схема (Фиг. 11) са изобразени таблиците на цялата база данни.

Таблица Картотекирани(Kartoteki) – таблицата отговаря за данните на картотеките (Егн ,Име, Презиме, Фамилия, Телефон, Имейл, Адрес)

```
CREATE TABLE [dbo].[Kartoteki] (
    [EGN]         CHAR (10)      NOT NULL,
    [First_name]  VARCHAR (55)   NOT NULL,
    [Middle_name] VARCHAR (55)   NULL,
    [Last_name]   VARCHAR (55)   NOT NULL,
    [Telephone]   VARCHAR (13)   NULL,
    [Email_]      VARCHAR (100)  NULL,
    [Address]     VARCHAR (200)  NULL,
    PRIMARY KEY CLUSTERED ([EGN] ASC)
);
```

Таблица Настойници(Parents) – таблицата отговаря за данните на родителите на картотеките (Име, Презиме, Фамилия, Телефон, Имейл, Картотекиран)

```
CREATE TABLE [dbo].[Parents] (
    [First_name]  VARCHAR (55)  NOT NULL,
    [Middle_name] VARCHAR (55)  NULL,
    [Last_name]   VARCHAR (55)  NOT NULL,
    [Telephone]   VARCHAR (13)  NULL,
    [Email_]      VARCHAR (100) NULL,
    [Kartoteka]   CHAR (10)     NULL,
    CONSTRAINT [FK_Parents] FOREIGN KEY ([Kartoteka]) REFERENCES
[dbo].[Kartoteki] ([EGN]) ON DELETE CASCADE ON UPDATE CASCADE
);
```

Таблица Групи(Groups) – таблицата отговаря за данните групите(Номер,име на групата)

```
CREATE TABLE [dbo].[Groups] (
    [ID]         INT          IDENTITY (1, 1) NOT NULL,
    [Group_name] VARCHAR (35) NOT NULL,
    PRIMARY KEY CLUSTERED ([ID] ASC)
);
```

Таблица Зали(Halls) – таблицата отговаря за данните на залите (номер, име на залата, адрес)

```
CREATE TABLE [dbo].[Halls] (
    [ID]        INT           IDENTITY (1, 1) NOT NULL,
    [Hall_name] VARCHAR (35)  NOT NULL,
    [Address]   VARCHAR (200) NULL,
    PRIMARY KEY CLUSTERED ([ID] ASC)
);
```

Таблица Степени(Levels) – таблицата отговаря за данните степените (номер, име на степента, вид на степента)

```
CREATE TABLE [dbo].[Levels] (
    [ID]         INT          IDENTITY (1, 1) NOT NULL,
    [Level_name] VARCHAR (30) NOT NULL,
    [Level_type] VARCHAR (40) NOT NULL,
    PRIMARY KEY CLUSTERED ([ID] ASC)
);
```

Таблица Сминари(Seminars) – таблицата отговаря за данните на проведените семинари (номер,име на инструктор, място на провеждане, град, начална дата, крайна дата)

```
CREATE TABLE [dbo].[Seminars] (
    [ID]  INT IDENTITY (1, 1) NOT NULL,
    [Instructor_name] VARCHAR (100) NOT NULL,
    [Place]           VARCHAR (65)  NULL,
    [City]            VARCHAR (60)  NOT NULL,
    [Start_date]      DATE          NOT NULL,
    [End_date]        DATE          NOT NULL,
    PRIMARY KEY CLUSTERED ([ID] ASC)
);
```

Таблица Видове вноски(PaymentType) – таблицата отговаря за данните на видовете месечни такси (номер, име на типа, цена)

```
CREATE TABLE [dbo].[PaymentType] (
    [ID]           INT          IDENTITY (1, 1) NOT NULL,
    [Payment_name] VARCHAR (35) NOT NULL,
    [Price]        INT          NOT NULL,
    PRIMARY KEY CLUSTERED ([ID] ASC)
);
```

Таблица Картотекирани в група(GroupList) – таблицата отговаря за данните на картотекираните в една група (номер, група, картотекиран)

```
CREATE TABLE [dbo].[GroupList] (
    [ID]         INT       NOT NULL,
    [Kartoteka]  CHAR (10) NOT NULL,
    [Group_name] INT       NOT NULL,
    PRIMARY KEY CLUSTERED ([ID] ASC),
    CONSTRAINT [FK_Group_Kartoteka] FOREIGN KEY ([Kartoteka]) REFERENCES
[dbo].[Kartoteki] ([EGN]) ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT [FK_Group_List] FOREIGN KEY ([Group_name]) REFERENCES
[dbo].[Groups] ([ID]) ON DELETE CASCADE ON UPDATE CASCADE
);
```

Таблица График(Timetable) – таблицата отговаря за на графиците (Треньор, Списък на група, Зала, ден от седмицата, час на започване на тренирвоката, продължителност на тренировката)

```
CREATE TABLE [dbo].[Timetable] (
    [Instructor] CHAR (10)    NOT NULL,
    [Grouplist]  INT          NOT NULL,
    [Hall]       INT          NOT NULL,
    [Weekday]    VARCHAR (35) NOT NULL,
    [Start_time] TIME (7)     NOT NULL,
    [Duration]   INT          NOT NULL,
    CONSTRAINT [FK_Timetable_Instructor] FOREIGN KEY ([Instructor])
REFERENCES [dbo].[Kartoteki] ([EGN]) ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT [FK_Timetable_Hall] FOREIGN KEY ([Hall]) REFERENCES
[dbo].[Halls] ([ID]) ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT [FK_Timetable_Group] FOREIGN KEY ([Grouplist]) REFERENCES
[dbo].[GroupList] ([ID])
);
```

Таблица Плащане(Payment) – таблицата отговаря за данните на плащанията (Номер, Картотекиран, Дата на плащане, Месец на заплатената такса)

```
CREATE TABLE [dbo].[Payment] (
    [ID]        INT          IDENTITY (1, 1) NOT NULL,
    [Kartoteka] CHAR (10)    NOT NULL,
    [Pay_date]  DATE         NULL,
    [Pay_month] VARCHAR (30) NOT NULL,
    PRIMARY KEY CLUSTERED ([ID] ASC),
    CONSTRAINT [FK_Payment_kartoteka] FOREIGN KEY ([Kartoteka]) REFERENCES
[dbo].[Kartoteki] ([EGN]) ON DELETE CASCADE ON UPDATE CASCADE
);
```

Таблица Детайли на плащането(PaymentDetails) – таблицата отговаря за данните на вида на извършените плащания (Плащане, Тип на плащане)

```
CREATE TABLE [dbo].[PaymentDetails] (
    [Payment_type] INT NOT NULL,
    [Payment]      INT NOT NULL,
    CONSTRAINT [FK_Payment_type] FOREIGN KEY ([Payment_type]) REFERENCES
[dbo].[PaymentType] ([ID]) ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT [FK_Payment_details] FOREIGN KEY ([Payment]) REFERENCES
[dbo].[Payment] ([ID]) ON DELETE CASCADE ON UPDATE CASCADE
);
```

Таблица Изпити(Exams) – таблицата отговаря за данните на проведените изпити (Картотекиран, Степен, Ден на изпити, име на изпитващ)

```
CREATE TABLE [dbo].[Exams] (
    [Kartoteka] CHAR (10) NOT NULL,
    [Level]     INT       NOT NULL,
    [Exam_day]  TEXT      NULL,
    [Examiner]  TEXT      NOT NULL,
    CONSTRAINT [FK_Exams_levels] FOREIGN KEY ([Level]) REFERENCES
[dbo].[Levels] ([ID]) ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT [FK_Exams_Kartotekas] FOREIGN KEY ([Kartoteka]) REFERENCES
[dbo].[Kartoteki] ([EGN]) ON DELETE CASCADE ON UPDATE CASCADE
);
```

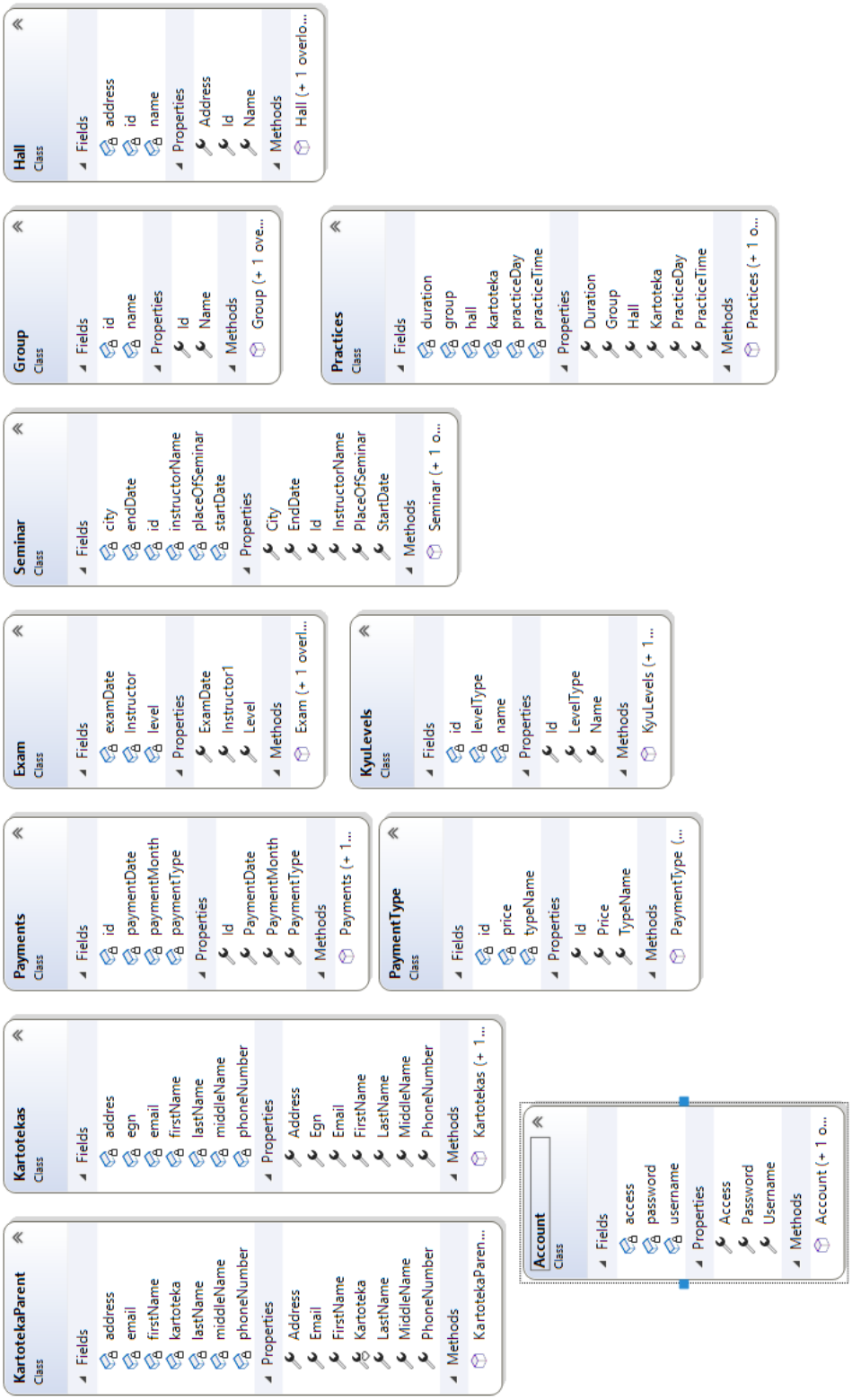
Таблица Участие в семинари(SeminarsPartition) – таблицата отговаря за данните на участията в семинари (Картотекиран,Семинар)

```
CREATE TABLE [dbo].[SeminarParticipation] (
    [Kartoteka] CHAR (10) NOT NULL,
    [Seminar]   INT       NOT NULL,
    CONSTRAINT [FK_Participation_Kartoteka] FOREIGN KEY ([Kartoteka])
REFERENCES [dbo].[Kartoteki] ([EGN]) ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT [FK_Participation_Seminars] FOREIGN KEY ([Seminar]) REFERENCES
[dbo].[Seminars] ([ID]) ON DELETE CASCADE ON UPDATE CASCADE
);
```

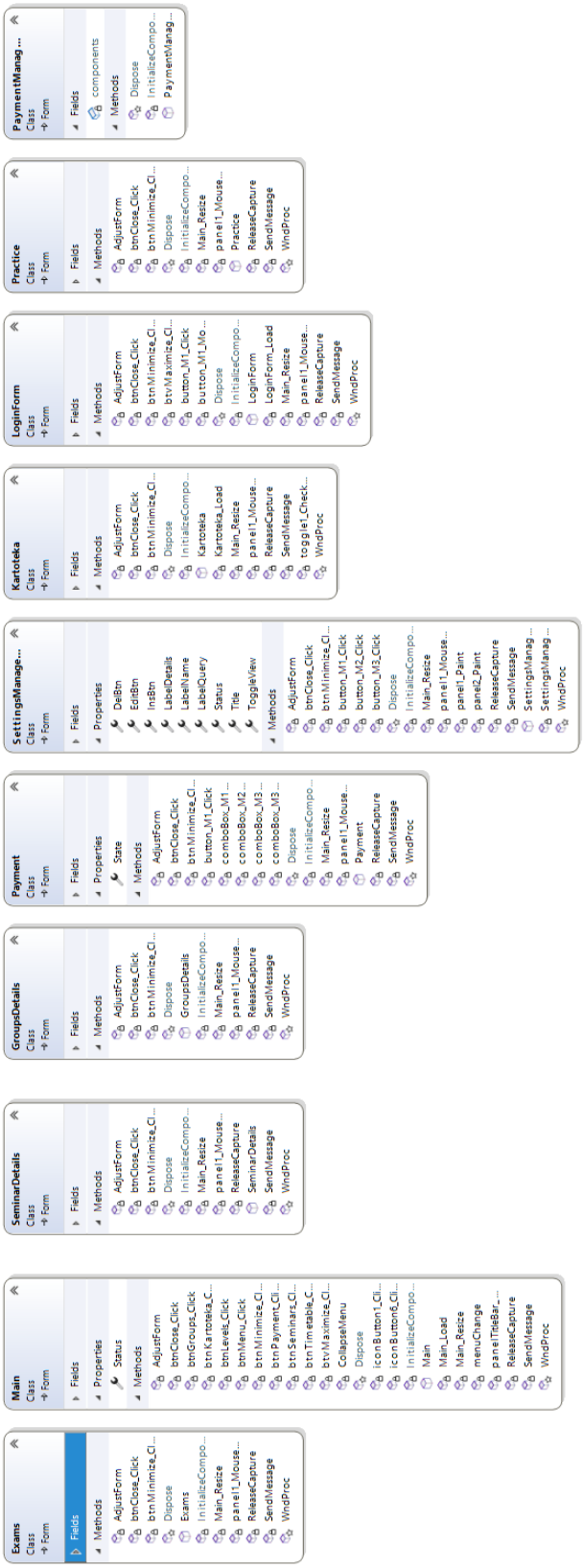
Таблица Акаунти(Accounts) – таблицата отговаря за данните на акаунтите (Потребителско име, парола, ниво на достъп)

```
CREATE TABLE [dbo].[Accounts] (
    [Username]  VARCHAR (256) NOT NULL,
    [Passsword] VARCHAR (256) NOT NULL,
    [Access]    VARCHAR (50)  NOT NULL,
    PRIMARY KEY CLUSTERED ([Username] ASC)
);
```

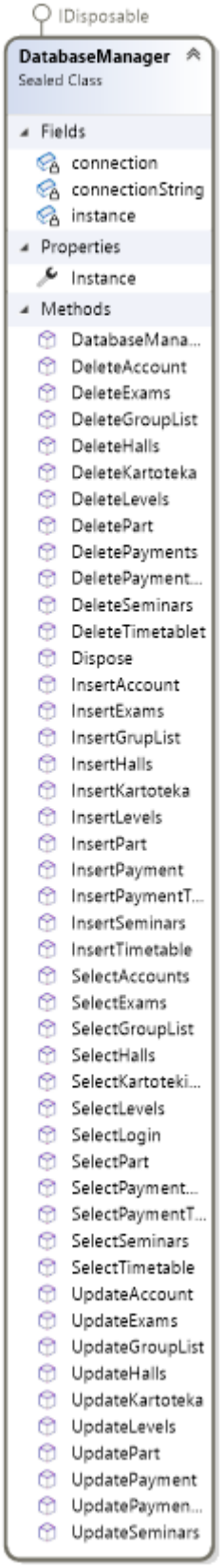
IV.3.2. Описание на класове



Фигура 12 - Клас диаграма на създадените обекти

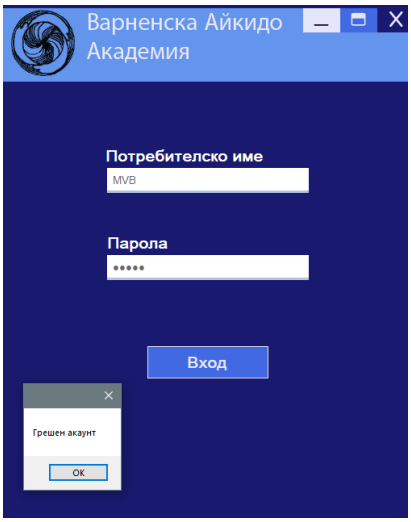


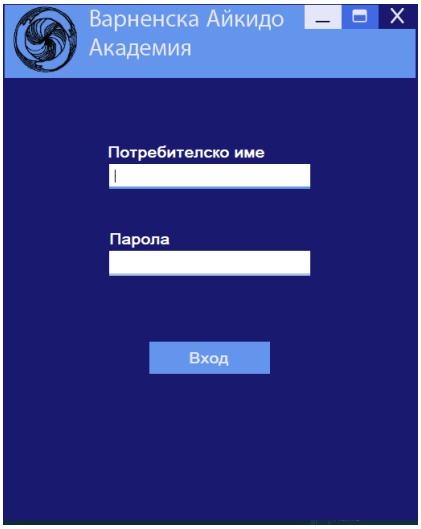
Фигура 13 – Класове на формите



Фигура 14– Клас за обслужване на базата

IV.3.2.1. Описание на клас LoginForm

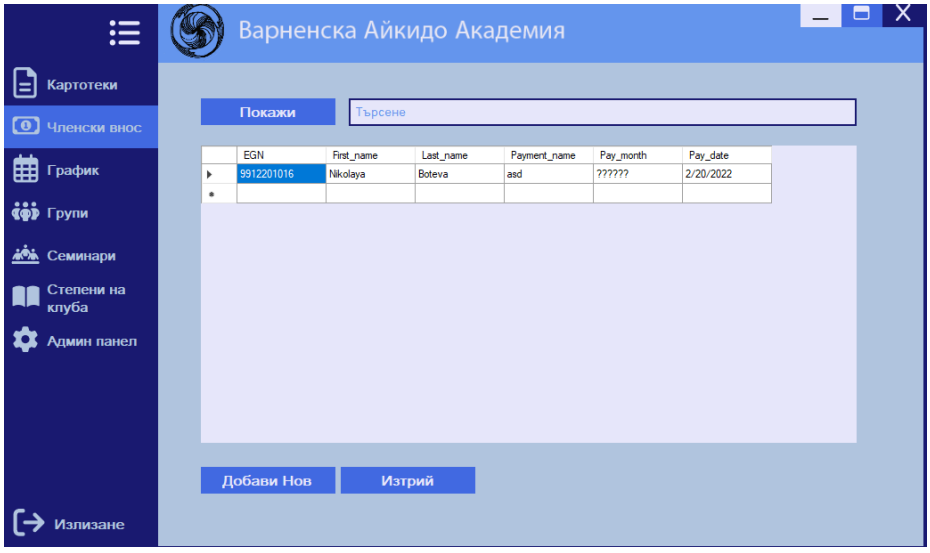




Фигура 15 - Login формуляр за съществуващ потребител *Фигура 16 - Login формуляр*

Класът представлява първия прозорец, който вижда потребителя. При получаване на програмата потребителя получава администрски профил , с който да достъпи настройките на приложението. Формата съдържа две текстови полета за потребителско име и паролаи бутон за вход. В кордния десен ъгъл са бутоните на прозореца.

IV.3.2.2. Описание на клас Main.cs

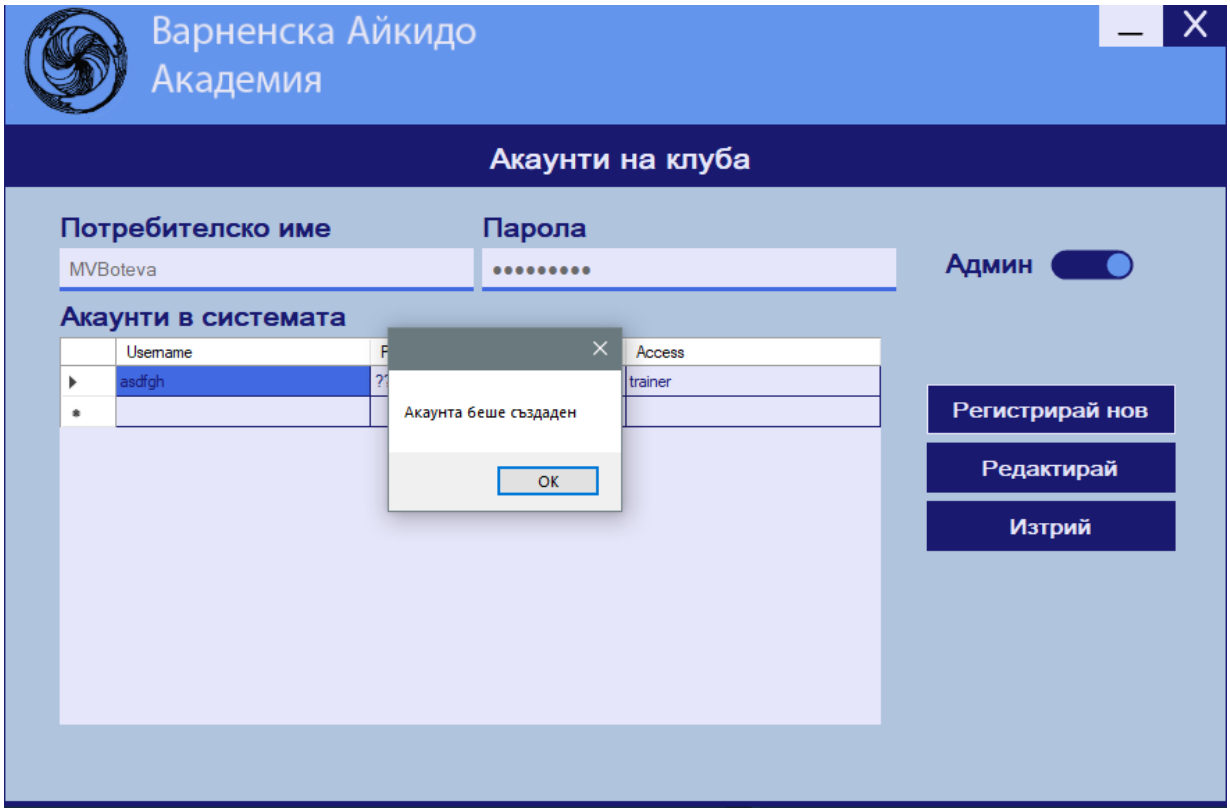


Фигура 17 – основна форма на програмата

Основната форма е частта от програмата, през която минават всички действия.

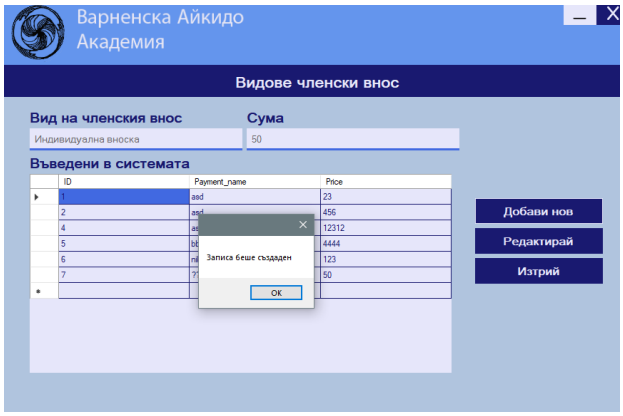
Този клас е „навигиращия“ за цялата програма. Потребителя ще преминава през формата за да достигне всички менюта и функции на приложението.

IV.3.2.3. Описание на клас SettingsManagement.cs

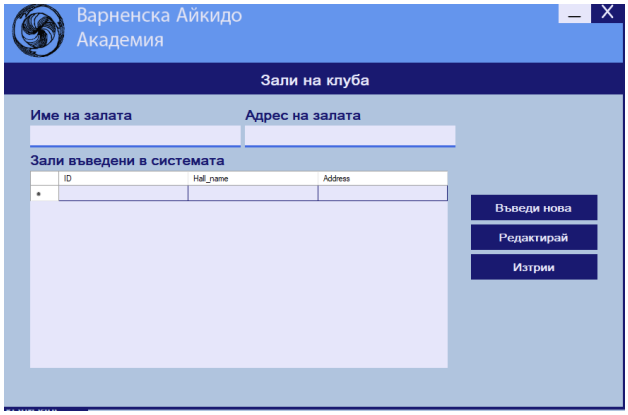


Фигура 18 – Форма за администриране на акаунти

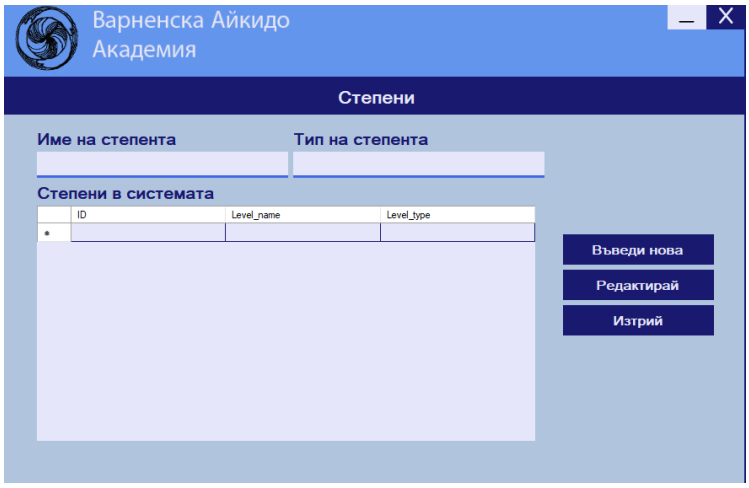
Този клас обслужва формата администките възможстти. В зависимост от избраните типове данни за управление формата зарежда нужните данни. На фиг 19, 20, 21 са представени другите форми.



Фигура 19 – Форма за администриране на членски вноски



Фигура 20 – Форма за администриране на залите

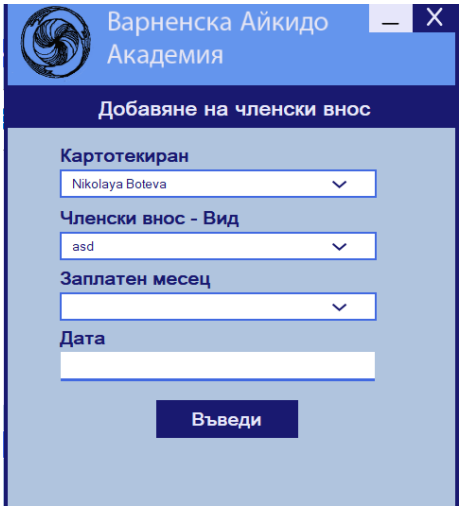


Фигура 21 – Форма за администриране на степените

При създаване на акаунти паролите се хешират и за запасват в хеширан вид в базата данни. За хеширане програмата използва пакет System.IO.Hashing. и по конкретно метода XxHash64 от пакета.

IV.3.2.4. Описание на клас Payment

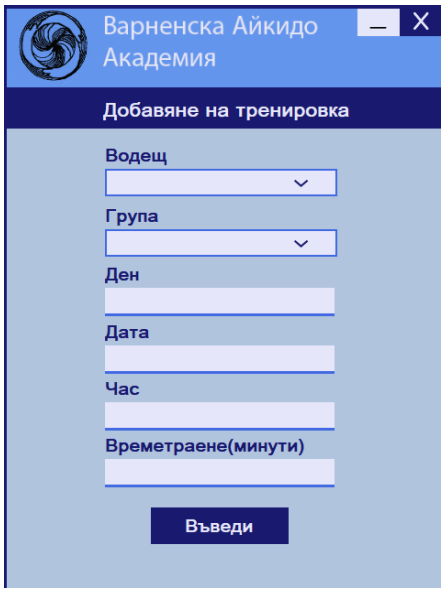
На фигура 22 се вижда формата от класа Payment. Тя се визуализира при натискане на бутона „добави“ в менюто „Членски внос“ за да се извършат операциите за добявяне на нови плащания в системата.



Фигура 22 – Форма за добавяне на месечна вноска

IV.3.2.5. Описание на клас Practice

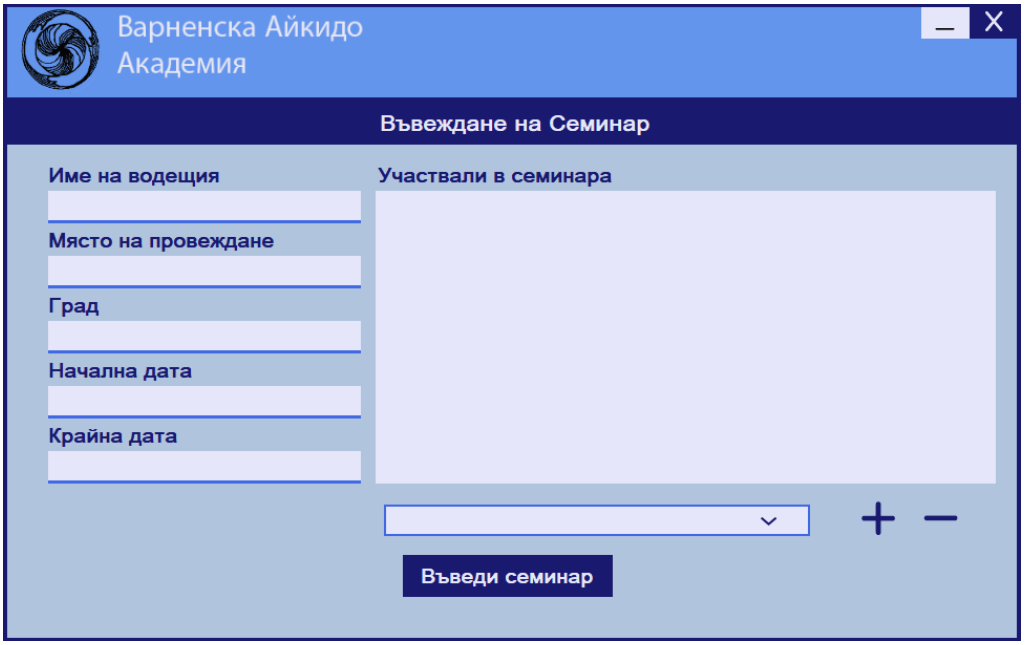
Този клас генерира формата за добавяне на тренировки в графика. Аналогично тя се визуализира при натискане на бутон „добави“ от основното меню за управление на графиците.



Фигура 23 – Форма за добавяне на тренировки

IV.3.2.6. Описание на клас SeminarDetails

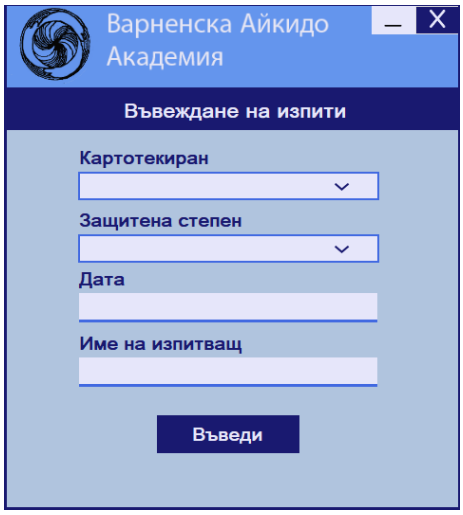
Това е класът, който създава форма за администриране на семинарите и техните участници от клуба. Изобразена е формата на фиг. 24.



Фигура 24 – Форма за добавяне на семинари и участници

IV.3.2.7. Описание на клас Exams

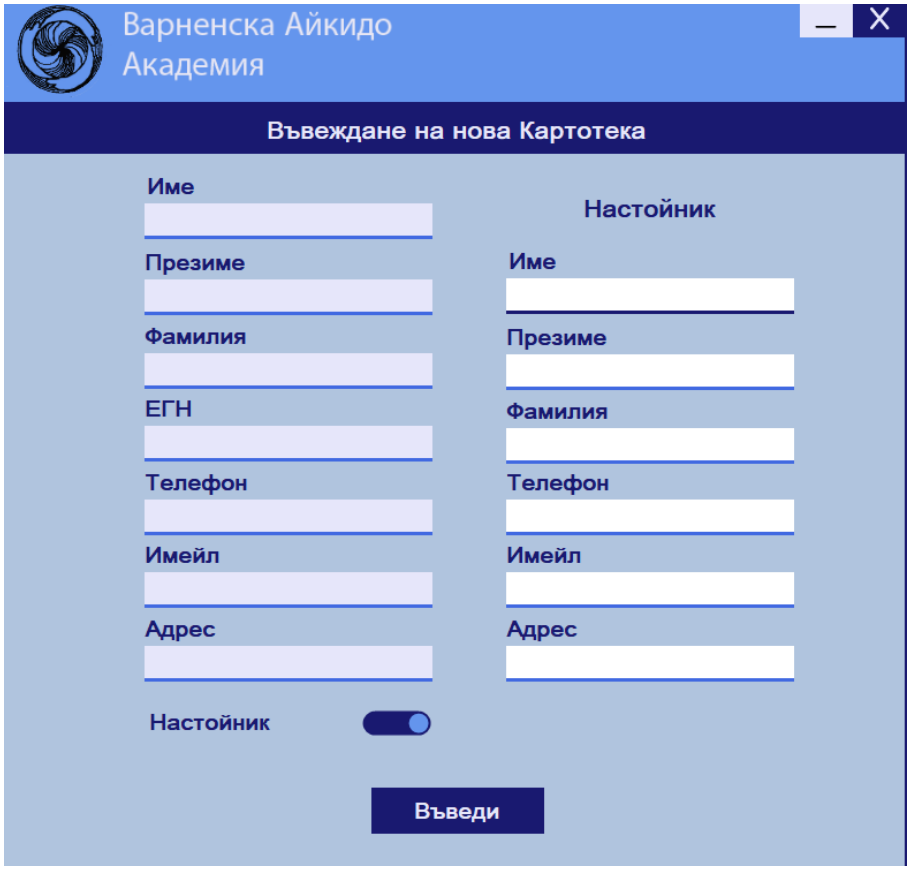
Класът Exams създава форма за въвеждане на данни за взети изпити съответно, кога и кой е защитил степентта.



Фигура 25 – Форма за добавяне на изпити

IV.3.2.8. Описание на клас Kartoteka

Класът отговаря за една от потенциално най - използваните форми, а именно администрирането на картотеките. Операциите могат добавяне и редакция в зависимост от избора в главното меню.



Фигура 26 – Форма за добавяне на семинари и участници

IV.3.2.9. Описание на класовете Objects(Exam, Hall, Group, KartotekaParent, Kartotekas, KyuLevels, LevelType, Payment, PaymentType, Practices, Seminar)

Това са помощни класове за слоя на бизнес логиката. Те са аналогични на класовете за интерфейса. Подпомагат лесната организация на данните във самата програма и лесното им предаване към фрагментите от кода за работа с базите данни. Класовете могат да се ползват във всяка част на програмата.

IV.3.2.10. Описание на клас DatabaseManager

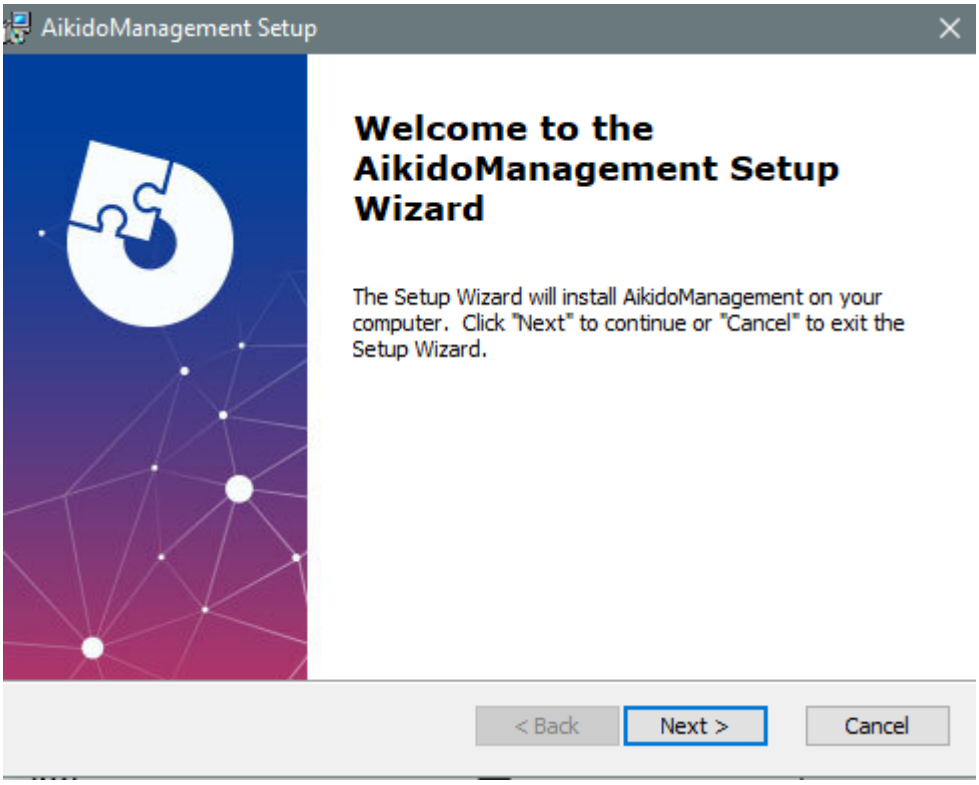
Класът DatabaseManager има много важна роля в програмата. Това е класът, който отговаря за всички действия към и със базата данни. Класът съдържа инстанция, която се иницилизира при нужда за работа в базата. Всички SQL заявки, които се подават към базата данни са записани там и са оформени в методи. Операциите INSERT, UPDATE, DELETE са заложени в методи с връщан тип „bool“ за да се предотврати блокиране или спиране на програмата по време на работа.

IV.3.2.11. Описание на класове от UI

Част от контролите предлагани от .NET framework са надградени. Получават допълнителни настройки и по – модерна и изчистена визия. Също така са изработени и класове тип User_control, който се зареждат в главната форма за да бъде олекотен дизайна и да бъде намалено количеството на прозорците програмата.

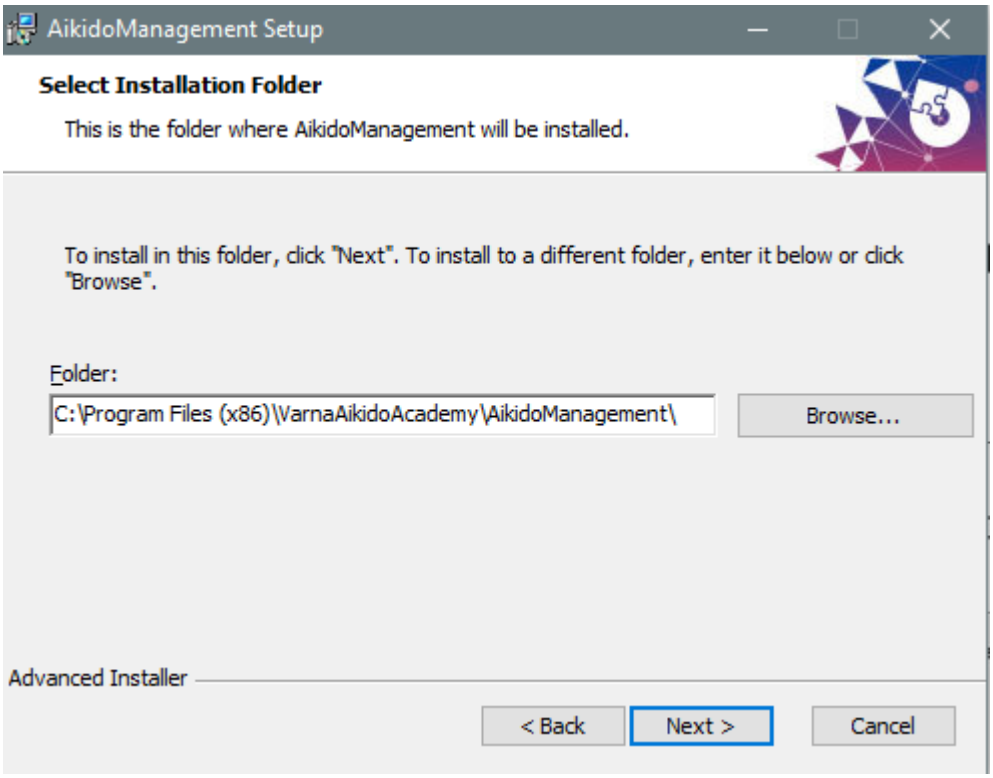
V. ГЛАВА. Реализиране на системата

V.1. Инсталиране и изпълнение



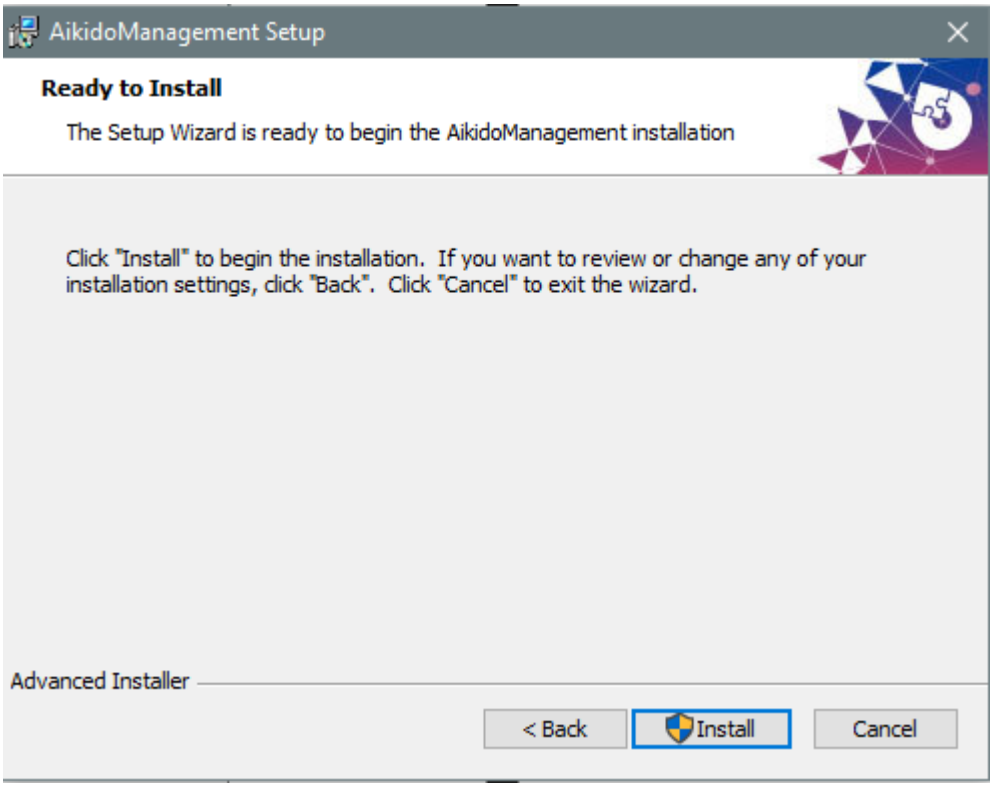
Фигура 27 - Първа страница от помощника за инсталиране

Ако не желаете никакви предпочитания, просто избирате „Next“ бутона всеки път и накрая „Close“. На втората страница (Фиг. 28) може да се избере инсталационната директория и дали програмата да бъде инсталирана само за текущия потребител или за всички. На последната трета страница (Фиг. 29) е бутона за инсталиране.



Фигура 28 - Втора страница от помощника за инсталиране

Фигура 29 - Трета страница от инсталатора



V.2. Деинсталиране

Може да се деинсталира както през настройките:

Settings > Apps > Apps & features > Make a dance > Uninstall

Така и през контролния панел:

Control Panel > Programs > Programs and Features > Make a dance > Uninstall

V.3. Ръководство за потребителя

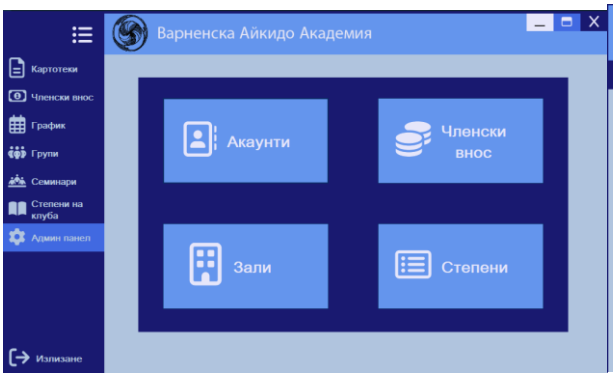
При стартиране на програмата се появява логин формата. При получаване на приложението всеки потребител получава първи администрски акаунт. Влиза се с този акаунт. Администрския панел е активен единствено и само за админ акаунтите. Всички форми притежават бутони за скриване на прозореца, увеличаване и затваряне.

Менюто в главната форма може да бъде свито с бутона над самото меню фиг.26.

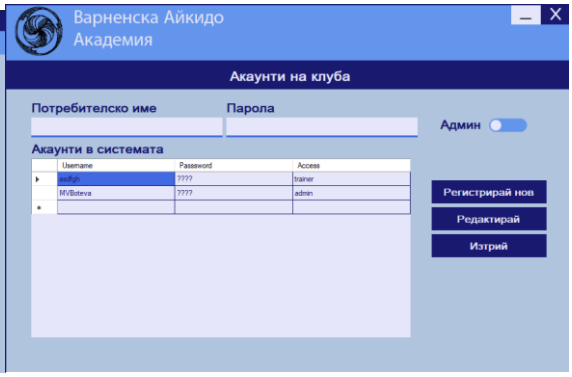

игура 30– бутон за свиване на менюто

След приключване а работа може програмата да бъде затворена от бутона „X“ или да се излезе от акаунта и след това да бъде затворена.

V.3.1. Въвеждане на акаунти, членски внос, зали, степени



Фигура 31 – главно меню



Фигура 32 – прозорец за въвеждане

От страничното меню (фиг. 26) се избира „Админ Панел“. От там се второстепенно меню за избор на на акаунти, членски внос, зали или степени. При кликане, на някой от четирите бутона, се показва прозорец със списък и няколко полета и бутони (фиг. 27). В прозореца за акаунти има бутон за отбелязване дали акаунта да е админ или просто треньор. Всички акаунти в системата, които не са админи се записват като треньори. При натискане на бутон „Регистрирай нов“ за акаунтите и „Въведи“ за останалите форми, се отваря прозорец, в който да въведете съответните данни.

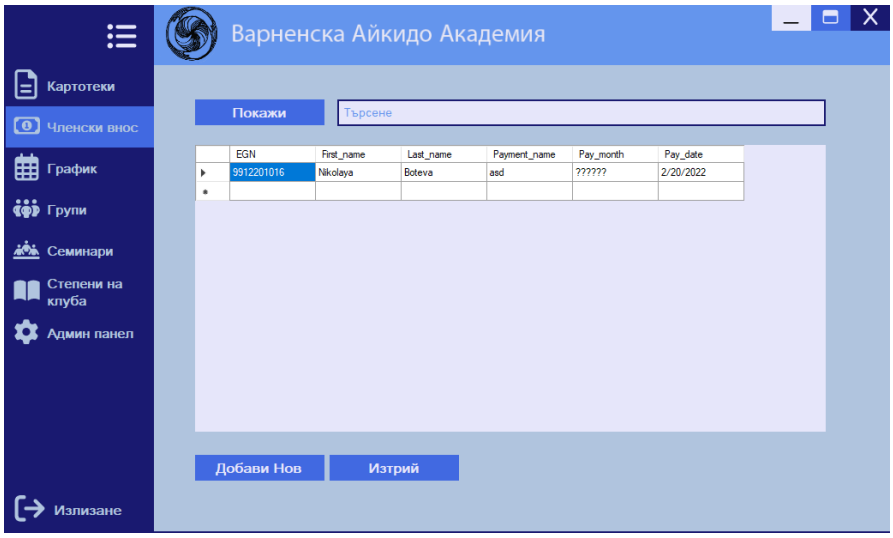
V.3.2. Редактиране на акаунти, членски внос, зали, степени

Аналогично на въвеждането се изълнява и редактирането. Когато се редактират данни , първа стъпка е да се селектира записа от изведените в таблицата. След това се натиска бутона „Редактирай“ и ще се появи същата форма, с която се въвеждат данни, но този път се въвеждат данните за промяна.

V.3.3. Изтриване на акаунти, членски внос, зали, степени

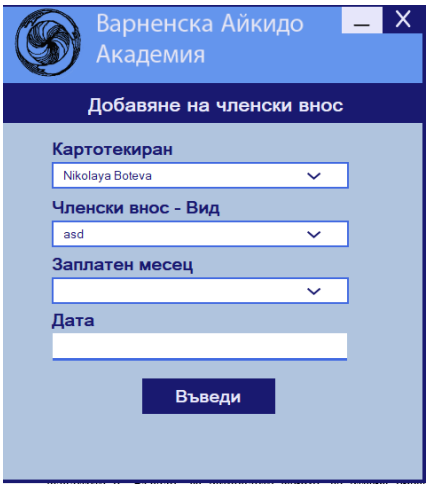
Аналогично на редактирането се изълнява и изтриване. Маркира се първо запис от таблицата, който ще се трие, след това се натиска бутон „Изтрий“.

V.3.4. Въвеждане, редактиране и изтриване на друг тип данни от главното меню.



Фигура 33 – главно меню – ЧЛЕНСКИ ВНОС

Във всички останали менюта потребителя трябва да следва следните стъпки.
Избира се от менюто в ляво секцията от данни, с която ще се работи. Пример фиг.30 Членски внос. Всяко основна функция от менюто притежава таблица,търсачка и бутони за въвеждане, изтриване или редактиране.



Фигура 34

Аналогично на администкия панел, за добавяне на нови данни се клика бутон „Добави“ и се отваря нов прозорец където да бъдат въведени данните фиг.34. Попълват се данните се натиска бутон „Въведи“.

При редакция се избира запис от таблицата, натиска се бутона за редакция и се въвеждат новите данни във формата.
За изтриване само се избира записът и се натиска бутона.По този начин работят всички менюта в прогрмата.

VI. ГЛАВА. Тестове и резултати

VI.1. Тестове на интерфейса

Приложението само по себе си изисква да бъде лесно за работа. Програмата претърпя тестове от тренцьори за удобство и лекота на използване на продукта.

VI.2. Интеграционни тестове

Редица ръчни тестове бяха проведени при интеграцията на следните компоненти:

- Админ панел
- База данни
- Логин платформа
- Справки в приложението

При проверката на работоспособността на горните компоненти бяха срещнати редица проблеми.

Първоначално при създаване на логин формата, алгоритъма за хеширане създаде конфликт при конвертирането. Тъй като метода за хешироването работи с масив от тип байт, наложи да бъдат тествани няколко вина конвертирания.

Базите данни претърпяха тестване при извиличането на данни. От първоначалния си вид базата имаше проблем с едната релация тъй като, не бяха предвидени добре връзките между таблиците в модула за членски внос.

VII. ГЛАВА. Проблеми и решения

VII.1. Работа с лични данни

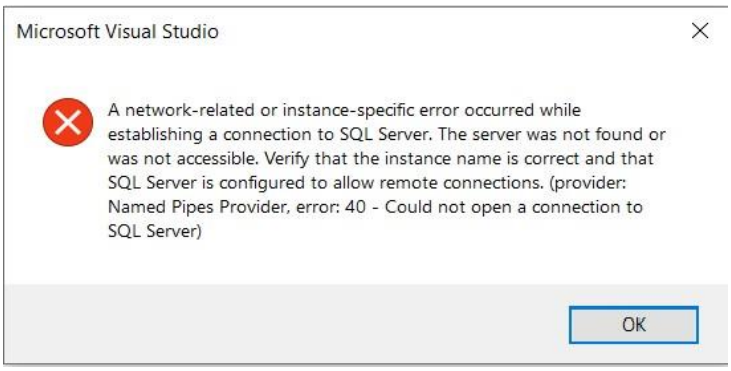
Поради наличието на лични данни , приложението само по себе си изисва добра защита. Комбинацията от риск за данните и кратко време за изработка, реализацията на приложение в уеб пространството с добра защита не беше реалистичен план.

За да бъде постигната максимална защита и готовност на продукта, решението на проблема беше приложението да бъде направено за локално ползване и да работи само на една компютърна система.

VII.2. База данни

При интеграцията с базата данни нещата вървяха добре първоначално. Когато се стигна до момента, че приложението трябваше да работи самостоятелно (без нужда от Visual Studio инсталация), излезе следният проблем:

Базата данни представлява MDF файл. За да може да се чете, трябва на целевия компютър да има инсталация на Microsoft SQL Server или поне негова олекотена версия, която предоставя само API. Пример за такава версия е LocalDB. Грешката,
която излизаше (Фиг. 35):



Фигура 35 – Грешка при липса на инсталация на SQL Server

Проблемът бе решен, чрез създаване на инсталатор, в който са включени всички библиотеки, от които зависи програмата ми, включително LocalDB. Част от тези библиотеки беше .NET Framework. Без нея не тръгваше програмата на друг компютър, на който няма инсталиран този Framework.

VIII. ГЛАВА. Заключение

VIII.1. Заключение

Десктоп приложението за упраление на Айкидо Клуб е реалистично и полезно решение за администриране на данните циркулиращи при треньорите. Повечето ръководители на клубове използват достъпни програми като MS Excel, MS Word и други подобни без специализирано предназначение.

Предназначените софтуери за такава дейност ще повишат качественото обслужване и управление на клубовете. Ще се намали работата на треньорите пред компютъра и ще имат възможност да увеличат своето активно пребиваване край трениращите.

VIII.2. Предложения за бъдещо развитие

Бойните изкуства са развити в световен мащаб. Организациите се образуват от хилядя клубове, които членуват. Приложението има потенциал да бъде развито и конкуретноспособно но ако бъде подобрено. Предложения за развитие:

- Създаване на уеб приложение, което да работи с базата.
- Съответно изнасяне на базата с данни на сървър
- Създаване на по добра защита на данните
- Добавяне на функция за принтиране на справки, графици, документи
- Създаване на анфроид приложение

- Изготвяне на “dashboard” в приложението(във всичките му варианти) за да се улесни анализа на клубовете и да има по-добра отчетност към институциите.
- Подобряване на базата данни, оптимизиране на модела
- Рефакториране на сорс кода и подобряване на скоростта на програмата
- Подобряване на дизайна
- Разширяване на администраторския панел
  - Възможност за пълна промяна на дизайна според желанието на траньора
  - Настройки за съобщения и изпращане на имейли към картотекираните

IX. ГЛАВА. Речник

- Семинари – Тренировъчна дейност, която се извършва от трениращите. Имат гост инструктури.
- Айкидо – японско бойно изкуство

X.ГЛАВА. Източници на информация

- https://docs.microsoft.com/en-us/dotnet/desktop/winforms/
- https://docs.microsoft.com/en-us/dotnet/csharp/
- https://docs.microsoft.com/en-us/visualstudio/data-tools
- https://stackoverflow.com/
- https://github.com/
- https://docs.microsoft.com/en-us/dotnet/api/system.io.hashing.xxhash64?view=dotnet-plat-ext-7.0

XI. Изходен код на програмата

Main.cs

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

namespace AikidoSystem
{
   public partial class Main : Form
   {
      private int borderSize = 4;
      private string status;

      public string Status { get => status; set => status = value; }

      public Main()
      {
         InitializeComponent();
         this.Padding = new Padding(borderSize);
         this.BackColor = Color.MidnightBlue;
      }

      [DllImport("user32.DLL", EntryPoint = "ReleaseCapture")]
      private extern static void ReleaseCapture();
      [DllImport("user32.DLL", EntryPoint = "SendMessage")]
      private extern static void SendMessage(System.IntPtr hWnd, int wMsg, int wParam, int lParam);

      private void panelTitleBar_MouseDown(object sender, MouseEventArgs e)
      {
         ReleaseCapture();
         SendMessage(this.Handle, 0x112,0xf012,0);
      }
      protected override void WndProc(ref Message m)
      {
         const int WM_NCCALCSIZE = 0x0083;//Standar Title Bar - Snap Window
         const int WM_NCHITTEST = 0x0084;//Win32, Mouse Input Notification: Determine what part of the window
corresponds to a point, allows to resize the form.
         const int resizeAreaSize = 10;
```

```
const int HTCLIENT = 1; //Represents the client area of the window
const int HTLEFT = 10;  //Left border of a window, allows resize horizontally to the left
const int HTRIGHT = 11; //Right border of a window, allows resize horizontally to the right
const int HTTOP = 12;   //Upper-horizontal border of a window, allows resize vertically up
const int HTTOPLEFT = 13;//Upper-left corner of a window border, allows resize diagonally to the left
const int HTTOPRIGHT = 14;//Upper-right corner of a window border, allows resize diagonally to the right
const int HTBOTTOM = 15; //Lower-horizontal border of a window, allows resize vertically down
const int HTBOTTOMLEFT = 16;//Lower-left corner of a window border, allows resize diagonally to the left
const int HTBOTTOMRIGHT = 17;//Lower-right corner of a window border, allows resize diagonally to the right
if (m.Msg == WM_NCHITTEST)
{
   base.WndProc(ref m);
   if (this.WindowState == FormWindowState.Normal)
   {
      if ((int)m.Result == HTCLIENT)
      {
         Point screenPoint = new Point(m.LParam.ToInt32());
         Point clientPoint = this.PointToClient(screenPoint);
         if (clientPoint.Y <= resizeAreaSize)
         {
            if (clientPoint.X <= resizeAreaSize)
               m.Result = (IntPtr)HTTOPLEFT;
            else if (clientPoint.X < (this.Size.Width - resizeAreaSize))
               m.Result = (IntPtr)HTTOP;
            else
               m.Result = (IntPtr)HTTOPRIGHT;
         }
         else if (clientPoint.Y <= (this.Size.Height - resizeAreaSize))
         {
            if (clientPoint.X <= resizeAreaSize)
               m.Result = (IntPtr)HTLEFT;
            else if (clientPoint.X > (this.Width - resizeAreaSize))
               m.Result = (IntPtr)HTRIGHT;
         }
         else
         {
            if (clientPoint.X <= resizeAreaSize)
               m.Result = (IntPtr)HTBOTTOMLEFT;
            else if (clientPoint.X < (this.Size.Width - resizeAreaSize))
               m.Result = (IntPtr)HTBOTTOM;
            else
               m.Result = (IntPtr)HTBOTTOMRIGHT;
         }
      }
   }
   return;
}

if (m.Msg == WM_NCCALCSIZE && m.WParam.ToInt32() == 1)
{
   return;
```

```
    }
    base.WndProc(ref m);

}

private void Main_Resize(object sender, EventArgs e)
{
    AdjustForm();
}

private void AdjustForm()
{
    switch (this.WindowState)
    {
        case FormWindowState.Maximized:
            this.Padding = new Padding(0,8,8,0);
            break;
        case FormWindowState.Normal:
            if(this.Padding.Top != borderSize)
                this.Padding = new Padding(borderSize);
            break;

    }

}
private void CollapseMenu()
{
    if (this.panelMenu.Width > 150) //Collapse menu
    {
        panelMenu.Width = 100;
        //pictureBox1.Visible = false;
        btnMenu.Dock = DockStyle.Top;
        foreach (Button menuButton in panelMenu.Controls.OfType<Button>())
        {
            menuButton.Text = "";
            menuButton.ImageAlign = ContentAlignment.MiddleCenter;
            menuButton.Padding = new Padding(0);
        }
    }
    else
    { //Expand menu
        panelMenu.Width = 173;
        //pictureBox1.Visible = true;
        btnMenu.Dock = DockStyle.None;
        foreach (Button menuButton in panelMenu.Controls.OfType<Button>())
        {
            menuButton.Text =menuButton.Tag.ToString();
            menuButton.ImageAlign = ContentAlignment.MiddleLeft;
            menuButton.Padding = new Padding(10, 0, 0, 0);
        }
    }
}
```

```
private void btnMinimize_Click(object sender, EventArgs e)
{
   this.WindowState = FormWindowState.Minimized;
}

private void btvMaximize_Click(object sender, EventArgs e)
{
   if (this.WindowState == FormWindowState.Normal)
      this.WindowState = FormWindowState.Maximized;
   else this.WindowState = FormWindowState.Normal;
}

private void btnClose_Click(object sender, EventArgs e)
{
   Application.Exit();

}

private void btnMenu_Click(object sender, EventArgs e)
{
   CollapseMenu();
}

private void iconButton6_Click(object sender, EventArgs e)
{
   this.Close();
   LoginForm.loginForm.ShowDialog();
}


private void btnKartoteka_Click(object sender, EventArgs e)
{
   if (!panelDesktop.Controls.Contains(Kartoteka_UserControl.Instance))
   {
      panelDesktop.Controls.Add(Kartoteka_UserControl.Instance);
      Kartoteka_UserControl.Instance.Dock = DockStyle.Fill;
   }

   Kartoteka_UserControl.Instance.BringToFront();
   Kartoteka_UserControl.Instance.Focus();
   menuChange(btnKartoteka);
   btnKartoteka.BackColor = Color.RoyalBlue;

}

private void btnPayment_Click(object sender, EventArgs e)
{
   if (!panelDesktop.Controls.Contains(Payment_UserControl.Instance))
   {
      panelDesktop.Controls.Add(Payment_UserControl.Instance);
      Payment_UserControl.Instance.Dock = DockStyle.Fill;
```

```
    }

    Payment_UserControl.Instance.BringToFront();
    Payment_UserControl.Instance.Focus();
    menuChange(btnPayment);
    btnPayment.BackColor = Color.RoyalBlue;

}

private void btnTimetable_Click(object sender, EventArgs e)
{
    if (!panelDesktop.Controls.Contains(Timetable_UserControl.Instance))
    {
        panelDesktop.Controls.Add(Timetable_UserControl.Instance);
        Timetable_UserControl.Instance.Dock = DockStyle.Fill;
    }

    Timetable_UserControl.Instance.BringToFront();
    Timetable_UserControl.Instance.Focus();
    menuChange(btnTimetable);
    btnTimetable.BackColor = Color.RoyalBlue;
}

private void btnGroups_Click(object sender, EventArgs e)
{
    if (!panelDesktop.Controls.Contains(Groups_UserControl.Instance))
    {
        panelDesktop.Controls.Add(Groups_UserControl.Instance);
        Groups_UserControl.Instance.Dock = DockStyle.Fill;
    }

    Groups_UserControl.Instance.BringToFront();
    Groups_UserControl.Instance.Focus();
    menuChange(btnGroups);
    btnGroups.BackColor = Color.RoyalBlue;

}


private void btnSeminars_Click(object sender, EventArgs e)
{
    if (!panelDesktop.Controls.Contains(Seminars_UserControl.Instance))
    {
        panelDesktop.Controls.Add(Seminars_UserControl.Instance);
        Seminars_UserControl.Instance.Dock = DockStyle.Fill;
    }

    Seminars_UserControl.Instance.BringToFront();
    Seminars_UserControl.Instance.Focus();
    menuChange(btnSeminars);
    btnSeminars.BackColor = Color.RoyalBlue;
}
```

```
        private void btnLevels_Click(object sender, EventArgs e)
        {
            if (!panelDesktop.Controls.Contains(Levels_UserControl.Instance))
            {
                panelDesktop.Controls.Add(Levels_UserControl.Instance);
                Levels_UserControl.Instance.Dock = DockStyle.Fill;
            }

            Levels_UserControl.Instance.BringToFront();
            Levels_UserControl.Instance.Focus();
            menuChange(btnLevels);
            btnLevels.BackColor = Color.RoyalBlue;
        }
        private void iconButton1_Click(object sender, EventArgs e)
        {
            if (!panelDesktop.Controls.Contains(Admin_UserControl.Instance))
            {
                panelDesktop.Controls.Add(Admin_UserControl.Instance);
                Admin_UserControl.Instance.Dock = DockStyle.Fill;
            }

            Admin_UserControl.Instance.BringToFront();
            Admin_UserControl.Instance.Focus();
            menuChange(iconButton1);
            iconButton1.BackColor = Color.RoyalBlue;
        }
        private void menuChange(Button btn)
        {
            foreach (Button menuButton in panelMenu.Controls.OfType<Button>())
            {
                if(menuButton!= btn)
                menuButton.BackColor = Color.MidnightBlue;
            }


            System.GC.Collect();
        }

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

LoginForm.cs

```
using AikidoSystem.Objects;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.IO.Hashing;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace AikidoSystem
{
    public partial class LoginForm : Form
    {
        private DatabaseManager databasemanager = new DatabaseManager();

        private void button_M1_MouseHover(object sender, EventArgs e)
        {
            button_M1.BackColor = Color.RoyalBlue;
        }

        private int borderSize = 4;
        public LoginForm()
        {
            InitializeComponent();
            this.Padding = new Padding(borderSize);
            this.BackColor = Color.MidnightBlue;
        }

        [DllImport("user32.DLL", EntryPoint = "ReleaseCapture")]
        private extern static void ReleaseCapture();
        [DllImport("user32.DLL", EntryPoint = "SendMessage")]
        private extern static void SendMessage(System.IntPtr hWnd, int wMsg, int wParam, int
lParam);


        protected override void WndProc(ref Message m)
        {
            const int WM_NCCALCSIZE = 0x0083;//Standar Title Bar - Snap Window
            const int WM_NCHITTEST = 0x0084;//Win32, Mouse Input Notification: Determine what
part of the window corresponds to a point, allows to resize the form.
            const int resizeAreaSize = 10;


            const int HTCLIENT = 1; //Represents the client area of the window
            const int HTLEFT = 10;  //Left border of a window, allows resize horizontally to
the left
            const int HTRIGHT = 11; //Right border of a window, allows resize horizontally to
the right
            const int HTTOP = 12;   //Upper-horizontal border of a window, allows resize
vertically up
            const int HTTOPLEFT = 13;//Upper-left corner of a window border, allows resize
diagonally to the left
```

```
            const int HTTOPRIGHT = 14;//Upper-right corner of a window border, allows resize
diagonally to the right
            const int HTBOTTOM = 15; //Lower-horizontal border of a window, allows resize
vertically down
            const int HTBOTTOMLEFT = 16;//Lower-left corner of a window border, allows resize
diagonally to the left
            const int HTBOTTOMRIGHT = 17;//Lower-right corner of a window border, allows
resize diagonally to the right
            if (m.Msg == WM_NCHITTEST)
            {
                base.WndProc(ref m);
                if (this.WindowState == FormWindowState.Normal)
                {
                    if ((int)m.Result == HTCLIENT)
                    {
                        Point screenPoint = new Point(m.LParam.ToInt32());
                        Point clientPoint = this.PointToClient(screenPoint);
                        if (clientPoint.Y <= resizeAreaSize)
                        {
                            if (clientPoint.X <= resizeAreaSize)
                                m.Result = (IntPtr)HTTOPLEFT;
                            else if (clientPoint.X < (this.Size.Width - resizeAreaSize))
                                m.Result = (IntPtr)HTTOP;
                            else
                                m.Result = (IntPtr)HTTOPRIGHT;
                        }
                        else if (clientPoint.Y <= (this.Size.Height - resizeAreaSize))
                        {
                            if (clientPoint.X <= resizeAreaSize)
                                m.Result = (IntPtr)HTLEFT;
                            else if (clientPoint.X > (this.Width - resizeAreaSize))
                                m.Result = (IntPtr)HTRIGHT;
                        }
                        else
                        {
                            if (clientPoint.X <= resizeAreaSize)
                                m.Result = (IntPtr)HTBOTTOMLEFT;
                            else if (clientPoint.X < (this.Size.Width - resizeAreaSize))
                                m.Result = (IntPtr)HTBOTTOM;
                            else
                                m.Result = (IntPtr)HTBOTTOMRIGHT;
                        }
                    }
                }
                return;
            }

            if (m.Msg == WM_NCCALCSIZE && m.WParam.ToInt32() == 1)
            {
                return;
            }
            base.WndProc(ref m);

        }

        private void Main_Resize(object sender, EventArgs e)
        {
            AdjustForm();
        }

        private void AdjustForm()
        {
            switch (this.WindowState)
```

```
            {
                case FormWindowState.Maximized:
                    this.Padding = new Padding(0, 8, 8, 0);
                    break;
                case FormWindowState.Normal:
                    if (this.Padding.Top != borderSize)
                        this.Padding = new Padding(borderSize);
                    break;

            }

        }

        private void btnClose_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }

        private void btvMaximize_Click(object sender, EventArgs e)
        {
            if (this.WindowState == FormWindowState.Normal)
                this.WindowState = FormWindowState.Maximized;
            else this.WindowState = FormWindowState.Normal;
        }

        private void btnMinimize_Click(object sender, EventArgs e)
        {
            this.WindowState = FormWindowState.Minimized;
        }

        private void panel1_MouseDown(object sender, MouseEventArgs e)
        {
            ReleaseCapture();
            SendMessage(this.Handle, 0x112, 0xf012, 0);
        }
        public static LoginForm loginForm;
        private void button_M1_Click(object sender, EventArgs e)
        {
            Account account = new Account();

            account.Username = textBox_M1.Texts;
            //byte[] str = Encoding.Unicode.GetBytes(textBox_M2.Texts);
            account.Password =
Encoding.Unicode.GetString(XxHash64.Hash(Encoding.Unicode.GetBytes(textBox_M2.Texts)));
            account.Access = databasemanager.SelectLogin(account);
            //account.Password = XxHash64.Hash(str).ToString();
            if (account.Access != null||(textBox_M1.Texts=="MVB"&&textBox_M2.Texts=="123"))
            {

                loginForm = this;
                Main frm = new Main();
                if (textBox_M1.Texts == "MVB" && textBox_M2.Texts == "123")
                    frm.Status = "admin";
                else frm.Status = account.Access;
                frm.Show();
                databasemanager.Dispose();
                this.Hide();
            }
            else
                MessageBox.Show("Грешен акаунт");

        }
```

```
        private void LoginForm_Load(object sender, EventArgs e)
        {
            databasemanager = databasemanager.Instance;
        }
    }
}
```

Kartoteka.cs

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

namespace AikidoSystem
{
    public partial class Kartoteka : Form
    {
        private int borderSize = 4;

        public Kartoteka(string labelText)
        {
            InitializeComponent();
            this.Padding = new Padding(borderSize);
            this.BackColor = Color.MidnightBlue;
            label15.Text = labelText;
            label15.Left = (this.Width - label15.Width) / 2;
        }

        [DllImport("user32.DLL", EntryPoint = "ReleaseCapture")]
        private extern static void ReleaseCapture();
        [DllImport("user32.DLL", EntryPoint = "SendMessage")]
        private extern static void SendMessage(System.IntPtr hWnd, int wMsg, int wParam, int
lParam);


        protected override void WndProc(ref Message m)
        {
            const int WM_NCCALCSIZE = 0x0083;//Standar Title Bar - Snap Window
            const int WM_NCHITTEST = 0x0084;//Win32, Mouse Input Notification: Determine what
part of the window corresponds to a point, allows to resize the form.
            const int resizeAreaSize = 10;


            const int HTCLIENT = 1; //Represents the client area of the window
            const int HTLEFT = 10;  //Left border of a window, allows resize horizontally to
the left
            const int HTRIGHT = 11; //Right border of a window, allows resize horizontally to
the right
            const int HTTOP = 12;   //Upper-horizontal border of a window, allows resize
vertically up
            const int HTTOPLEFT = 13;//Upper-left corner of a window border, allows resize
diagonally to the left
```

```
            const int HTTOPRIGHT = 14;//Upper-right corner of a window border, allows resize
diagonally to the right
            const int HTBOTTOM = 15; //Lower-horizontal border of a window, allows resize
vertically down
            const int HTBOTTOMLEFT = 16;//Lower-left corner of a window border, allows resize
diagonally to the left
            const int HTBOTTOMRIGHT = 17;//Lower-right corner of a window border, allows
resize diagonally to the right
            if (m.Msg == WM_NCHITTEST)
            {
                base.WndProc(ref m);
                if (this.WindowState == FormWindowState.Normal)
                {
                    if ((int)m.Result == HTCLIENT)
                    {
                        Point screenPoint = new Point(m.LParam.ToInt32());
                        Point clientPoint = this.PointToClient(screenPoint);
                        if (clientPoint.Y <= resizeAreaSize)
                        {
                            if (clientPoint.X <= resizeAreaSize)
                                m.Result = (IntPtr)HTTOPLEFT;
                            else if (clientPoint.X < (this.Size.Width - resizeAreaSize))
                                m.Result = (IntPtr)HTTOP;
                            else
                                m.Result = (IntPtr)HTTOPRIGHT;
                        }
                        else if (clientPoint.Y <= (this.Size.Height - resizeAreaSize))
                        {
                            if (clientPoint.X <= resizeAreaSize)
                                m.Result = (IntPtr)HTLEFT;
                            else if (clientPoint.X > (this.Width - resizeAreaSize))
                                m.Result = (IntPtr)HTRIGHT;
                        }
                        else
                        {
                            if (clientPoint.X <= resizeAreaSize)
                                m.Result = (IntPtr)HTBOTTOMLEFT;
                            else if (clientPoint.X < (this.Size.Width - resizeAreaSize))
                                m.Result = (IntPtr)HTBOTTOM;
                            else
                                m.Result = (IntPtr)HTBOTTOMRIGHT;
                        }
                    }
                }
                return;
            }

            if (m.Msg == WM_NCCALCSIZE && m.WParam.ToInt32() == 1)
            {
                return;
            }
            base.WndProc(ref m);

        }

        private void Main_Resize(object sender, EventArgs e)
        {
            AdjustForm();
        }

        private void AdjustForm()
        {
            switch (this.WindowState)
```

```
    {
        case FormWindowState.Maximized:
            this.Padding = new Padding(0, 8, 8, 0);
            break;
        case FormWindowState.Normal:
            if (this.Padding.Top != borderSize)
                this.Padding = new Padding(borderSize);
            break;

    }

}

private void panel1_MouseDown(object sender, MouseEventArgs e)
{
    ReleaseCapture();
    SendMessage(this.Handle, 0x112, 0xf012, 0);
}

private void Kartoteka_Load(object sender, EventArgs e)
{

}

private void btnMinimize_Click(object sender, EventArgs e)
{
    this.WindowState = FormWindowState.Minimized;
}


private void btnClose_Click(object sender, EventArgs e)
{
    panelParent.Controls.Clear();
    panelParent.Width = 1;
    this.Width -= KartotekaParent.Instance.Width;
    button_M1.Left = (this.Width - button_M1.Width) / 2;
    label15.Left = (this.Width - label15.Width) / 2;
    this.Close();
    this.Dispose();
}

private void toggle1_CheckedChanged(object sender, EventArgs e)
{
    if (toggle1.Checked)
    {
        this.Width += KartotekaParent.Instance.Width;
        panelParent.Width = KartotekaParent.Instance.Width;
        if (!panelParent.Controls.Contains(KartotekaParent.Instance))
        {
            panelParent.Controls.Add(KartotekaParent.Instance);
            KartotekaParent.Instance.Dock = DockStyle.Fill;
        }

        KartotekaParent.Instance.BringToFront();
        KartotekaParent.Instance.Focus();

        button_M1.Left = (this.Width - button_M1.Width) / 2;
        label15.Left = (this.Width - label15.Width) / 2;


    }
    else
    {
```

```
                panelParent.Controls.Clear();
                panelParent.Width = 1;
                this.Width -= KartotekaParent.Instance.Width;
                button_M1.Left = (this.Width - button_M1.Width) / 2;
                label15.Left = (this.Width - label15.Width) / 2;
            }
        }
    }
}
```

Payment.cs

```
using AikidoSystem.Objects;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace AikidoSystem
{
    public partial class Payment : Form
    {
        private int borderSize = 4;
        DatabaseManager databaseManager = new DatabaseManager();
        Payments payments = new Payments();
        public string kartoteka;
        string state;

        public string State { get => state; set => state = value; }

        public Payment(string labelText)
        {
            InitializeComponent();
            this.Padding = new Padding(borderSize);
            this.BackColor = Color.MidnightBlue;
            label15.Text = labelText;
            label15.Left = (this.Width - label15.Width) / 2;
        }

        [DllImport("user32.DLL", EntryPoint = "ReleaseCapture")]
        private extern static void ReleaseCapture();
        [DllImport("user32.DLL", EntryPoint = "SendMessage")]
        private extern static void SendMessage(System.IntPtr hWnd, int wMsg, int wParam, int
lParam);


        protected override void WndProc(ref Message m)
        {
            const int WM_NCCALCSIZE = 0x0083;//Standar Title Bar - Snap Window
            const int WM_NCHITTEST = 0x0084;//Win32, Mouse Input Notification: Determine what
part of the window corresponds to a point, allows to resize the form.
            const int resizeAreaSize = 10;
```

```
            const int HTCLIENT = 1; //Represents the client area of the window
            const int HTLEFT = 10;  //Left border of a window, allows resize horizontally to
the left
            const int HTRIGHT = 11; //Right border of a window, allows resize horizontally to
the right
            const int HTTOP = 12;   //Upper-horizontal border of a window, allows resize
vertically up
            const int HTTOPLEFT = 13;//Upper-left corner of a window border, allows resize
diagonally to the left
            const int HTTOPRIGHT = 14;//Upper-right corner of a window border, allows resize
diagonally to the right
            const int HTBOTTOM = 15; //Lower-horizontal border of a window, allows resize
vertically down
            const int HTBOTTOMLEFT = 16;//Lower-left corner of a window border, allows resize
diagonally to the left
            const int HTBOTTOMRIGHT = 17;//Lower-right corner of a window border, allows
resize diagonally to the right
            if (m.Msg == WM_NCHITTEST)
            {
                base.WndProc(ref m);
                if (this.WindowState == FormWindowState.Normal)
                {
                    if ((int)m.Result == HTCLIENT)
                    {
                        Point screenPoint = new Point(m.LParam.ToInt32());
                        Point clientPoint = this.PointToClient(screenPoint);
                        if (clientPoint.Y <= resizeAreaSize)
                        {
                            if (clientPoint.X <= resizeAreaSize)
                                m.Result = (IntPtr)HTTOPLEFT;
                            else if (clientPoint.X < (this.Size.Width - resizeAreaSize))
                                m.Result = (IntPtr)HTTOP;
                            else
                                m.Result = (IntPtr)HTTOPRIGHT;
                        }
                        else if (clientPoint.Y <= (this.Size.Height - resizeAreaSize))
                        {
                            if (clientPoint.X <= resizeAreaSize)
                                m.Result = (IntPtr)HTLEFT;
                            else if (clientPoint.X > (this.Width - resizeAreaSize))
                                m.Result = (IntPtr)HTRIGHT;
                        }
                        else
                        {
                            if (clientPoint.X <= resizeAreaSize)
                                m.Result = (IntPtr)HTBOTTOMLEFT;
                            else if (clientPoint.X < (this.Size.Width - resizeAreaSize))
                                m.Result = (IntPtr)HTBOTTOM;
                            else
                                m.Result = (IntPtr)HTBOTTOMRIGHT;
                        }
                    }
                }
                return;
            }

            if (m.Msg == WM_NCCALCSIZE && m.WParam.ToInt32() == 1)
            {
                return;
            }
            base.WndProc(ref m);
```

```
}

private void Main_Resize(object sender, EventArgs e)
{
    AdjustForm();
}

private void AdjustForm()
{
    switch (this.WindowState)
    {
        case FormWindowState.Maximized:
            this.Padding = new Padding(0, 8, 8, 0);
            break;
        case FormWindowState.Normal:
            if (this.Padding.Top != borderSize)
                this.Padding = new Padding(borderSize);
            break;

    }

}

private void panel1_MouseDown(object sender, MouseEventArgs e)
{
    ReleaseCapture();
    SendMessage(this.Handle, 0x112, 0xf012, 0);
}

private void btnClose_Click(object sender, EventArgs e)
{
    this.Close();
    this.Dispose();
}

private void btnMinimize_Click(object sender, EventArgs e)
{
    this.WindowState = FormWindowState.Minimized;
}

private void comboBox_M3_OnSelectedIndexChanged(object sender, EventArgs e)
{

}

private void comboBox_M3_Load(object sender, EventArgs e)
{

}

private void comboBox_M2_Load(object sender, EventArgs e)
{
    databaseManager = databaseManager.Instance;
    DataTable dt = new DataTable();

    dt = databaseManager.SelectPaymentType();
    comboBox_M2.DataSource = dt;
    comboBox_M2.DisplayMember = "Payment_name";
    comboBox_M2.ValueMember = "ID";
}

private void comboBox_M1_Load(object sender, EventArgs e)
{
```

```
            databaseManager = databaseManager.Instance;
            DataTable dt = new DataTable();

            dt = databaseManager.SelectKartoteki();
            comboBox_M1.DataSource = dt;
            comboBox_M1.DisplayMember = "FULLNAME";
            comboBox_M1.ValueMember = "EGN";
        }

        private void button_M1_Click(object sender, EventArgs e)
        {
            if (state == "add") {
                payments.PaymentMonth = comboBox_M3.SelectedItem.ToString();
                payments.PaymentType = int.Parse(comboBox_M2.SelectedValue.ToString());
                payments.PaymentDate = textBox_M4.Texts;
                kartoteka = comboBox_M1.SelectedValue.ToString();
                if (databaseManager.InsertPayment(payments, kartoteka))
                    MessageBox.Show("Въведено");
                else MessageBox.Show(" Не е Въведено");
            }



        }
    }
}
```

SeminarDetails.cs

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

namespace AikidoSystem
{
    public partial class SeminarDetails : Form
    {
        private int borderSize = 4;
        public SeminarDetails(string labelText)
        {
            InitializeComponent();
            this.Padding = new Padding(borderSize);
            this.BackColor = Color.MidnightBlue;
            label15.Text = labelText;
            label15.Left = (this.Width - label15.Width) / 2;
        }

        [DllImport("user32.DLL", EntryPoint = "ReleaseCapture")]
        private extern static void ReleaseCapture();
        [DllImport("user32.DLL", EntryPoint = "SendMessage")]
```

```
        private extern static void SendMessage(System.IntPtr hWnd, int wMsg, int wParam, int
lParam);


        protected override void WndProc(ref Message m)
        {
            const int WM_NCCALCSIZE = 0x0083;//Standar Title Bar - Snap Window
            const int WM_NCHITTEST = 0x0084;//Win32, Mouse Input Notification: Determine what
part of the window corresponds to a point, allows to resize the form.
            const int resizeAreaSize = 10;


            const int HTCLIENT = 1; //Represents the client area of the window
            const int HTLEFT = 10;  //Left border of a window, allows resize horizontally to
the left
            const int HTRIGHT = 11; //Right border of a window, allows resize horizontally to
the right
            const int HTTOP = 12;   //Upper-horizontal border of a window, allows resize
vertically up
            const int HTTOPLEFT = 13;//Upper-left corner of a window border, allows resize
diagonally to the left
            const int HTTOPRIGHT = 14;//Upper-right corner of a window border, allows resize
diagonally to the right
            const int HTBOTTOM = 15; //Lower-horizontal border of a window, allows resize
vertically down
            const int HTBOTTOMLEFT = 16;//Lower-left corner of a window border, allows resize
diagonally to the left
            const int HTBOTTOMRIGHT = 17;//Lower-right corner of a window border, allows
resize diagonally to the right
            if (m.Msg == WM_NCHITTEST)
            {
                base.WndProc(ref m);
                if (this.WindowState == FormWindowState.Normal)
                {
                    if ((int)m.Result == HTCLIENT)
                    {
                        Point screenPoint = new Point(m.LParam.ToInt32());
                        Point clientPoint = this.PointToClient(screenPoint);
                        if (clientPoint.Y <= resizeAreaSize)
                        {
                            if (clientPoint.X <= resizeAreaSize)
                                m.Result = (IntPtr)HTTOPLEFT;
                            else if (clientPoint.X < (this.Size.Width - resizeAreaSize))
                                m.Result = (IntPtr)HTTOP;
                            else
                                m.Result = (IntPtr)HTTOPRIGHT;
                        }
                        else if (clientPoint.Y <= (this.Size.Height - resizeAreaSize))
                        {
                            if (clientPoint.X <= resizeAreaSize)
                                m.Result = (IntPtr)HTLEFT;
                            else if (clientPoint.X > (this.Width - resizeAreaSize))
                                m.Result = (IntPtr)HTRIGHT;
                        }
                        else
                        {
                            if (clientPoint.X <= resizeAreaSize)
                                m.Result = (IntPtr)HTBOTTOMLEFT;
                            else if (clientPoint.X < (this.Size.Width - resizeAreaSize))
                                m.Result = (IntPtr)HTBOTTOM;
                            else
                                m.Result = (IntPtr)HTBOTTOMRIGHT;
                        }
```

```
                }
            }
            return;
        }

        if (m.Msg == WM_NCCALCSIZE && m.WParam.ToInt32() == 1)
        {
            return;
        }
        base.WndProc(ref m);

    }

    private void Main_Resize(object sender, EventArgs e)
    {
        AdjustForm();
    }

    private void AdjustForm()
    {
        switch (this.WindowState)
        {
            case FormWindowState.Maximized:
                this.Padding = new Padding(0, 8, 8, 0);
                break;
            case FormWindowState.Normal:
                if (this.Padding.Top != borderSize)
                    this.Padding = new Padding(borderSize);
                break;

        }

    }
    private void btnMinimize_Click(object sender, EventArgs e)
    {
        this.WindowState = FormWindowState.Minimized;
    }

    private void btnClose_Click(object sender, EventArgs e)
    {
        this.Close();
    }

    private void panel1_MouseDown(object sender, MouseEventArgs e)
    {
        ReleaseCapture();
        SendMessage(this.Handle, 0x112, 0xf012, 0);
    }
}
}
```

Exams.cs

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

namespace AikidoSystem
{
    public partial class Exams : Form
    {
        private int borderSize = 4;
        public Exams(string labelText)
        {
            InitializeComponent();
            this.Padding = new Padding(borderSize);
            this.BackColor = Color.MidnightBlue;
            label15.Text = labelText;
            label15.Left = (this.Width - label15.Width) / 2;
        }

        [DllImport("user32.DLL", EntryPoint = "ReleaseCapture")]
        private extern static void ReleaseCapture();
        [DllImport("user32.DLL", EntryPoint = "SendMessage")]
        private extern static void SendMessage(System.IntPtr hWnd, int wMsg, int wParam, int
lParam);


        protected override void WndProc(ref Message m)
        {
            const int WM_NCCALCSIZE = 0x0083;//Standar Title Bar - Snap Window
            const int WM_NCHITTEST = 0x0084;//Win32, Mouse Input Notification: Determine what
part of the window corresponds to a point, allows to resize the form.
            const int resizeAreaSize = 10;


            const int HTCLIENT = 1; //Represents the client area of the window
            const int HTLEFT = 10;  //Left border of a window, allows resize horizontally to
the left
            const int HTRIGHT = 11; //Right border of a window, allows resize horizontally to
the right
            const int HTTOP = 12;   //Upper-horizontal border of a window, allows resize
vertically up
            const int HTTOPLEFT = 13;//Upper-left corner of a window border, allows resize
diagonally to the left
            const int HTTOPRIGHT = 14;//Upper-right corner of a window border, allows resize
diagonally to the right
            const int HTBOTTOM = 15; //Lower-horizontal border of a window, allows resize
vertically down
            const int HTBOTTOMLEFT = 16;//Lower-left corner of a window border, allows resize
diagonally to the left
            const int HTBOTTOMRIGHT = 17;//Lower-right corner of a window border, allows
resize diagonally to the right
            if (m.Msg == WM_NCHITTEST)
```

```
    {
        base.WndProc(ref m);
        if (this.WindowState == FormWindowState.Normal)
        {
            if ((int)m.Result == HTCLIENT)
            {
                Point screenPoint = new Point(m.LParam.ToInt32());
                Point clientPoint = this.PointToClient(screenPoint);
                if (clientPoint.Y <= resizeAreaSize)
                {
                    if (clientPoint.X <= resizeAreaSize)
                        m.Result = (IntPtr)HTTOPLEFT;
                    else if (clientPoint.X < (this.Size.Width - resizeAreaSize))
                        m.Result = (IntPtr)HTTOP;
                    else
                        m.Result = (IntPtr)HTTOPRIGHT;
                }
                else if (clientPoint.Y <= (this.Size.Height - resizeAreaSize))
                {
                    if (clientPoint.X <= resizeAreaSize)
                        m.Result = (IntPtr)HTLEFT;
                    else if (clientPoint.X > (this.Width - resizeAreaSize))
                        m.Result = (IntPtr)HTRIGHT;
                }
                else
                {
                    if (clientPoint.X <= resizeAreaSize)
                        m.Result = (IntPtr)HTBOTTOMLEFT;
                    else if (clientPoint.X < (this.Size.Width - resizeAreaSize))
                        m.Result = (IntPtr)HTBOTTOM;
                    else
                        m.Result = (IntPtr)HTBOTTOMRIGHT;
                }
            }
        }
        return;
    }

    if (m.Msg == WM_NCCALCSIZE && m.WParam.ToInt32() == 1)
    {
        return;
    }
    base.WndProc(ref m);

}

private void Main_Resize(object sender, EventArgs e)
{
    AdjustForm();
}

private void AdjustForm()
{
    switch (this.WindowState)
    {
        case FormWindowState.Maximized:
            this.Padding = new Padding(0, 8, 8, 0);
            break;
        case FormWindowState.Normal:
            if (this.Padding.Top != borderSize)
                this.Padding = new Padding(borderSize);
            break;
```

```
            }

        }

        private void btnClose_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void btnMinimize_Click(object sender, EventArgs e)
        {
            this.WindowState = FormWindowState.Minimized;
        }

        private void panel1_MouseDown(object sender, MouseEventArgs e)
        {
            ReleaseCapture();
            SendMessage(this.Handle, 0x112, 0xf012, 0);
        }
    }
}
```

GroupDetails.cs

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

namespace AikidoSystem
{

    public partial class GroupsDetails : Form
    {

        private int borderSize = 4;
        public GroupsDetails(string labelText)
        {
            InitializeComponent();
            this.Padding = new Padding(borderSize);
            this.BackColor = Color.MidnightBlue;
            label15.Text = labelText;
            label15.Left = (this.Width - label15.Width) / 2;
        }

        [DllImport("user32.DLL", EntryPoint = "ReleaseCapture")]
        private extern static void ReleaseCapture();
        [DllImport("user32.DLL", EntryPoint = "SendMessage")]
        private extern static void SendMessage(System.IntPtr hWnd, int wMsg, int wParam, int
lParam);
```

```
        protected override void WndProc(ref Message m)
        {
            const int WM_NCCALCSIZE = 0x0083;//Standar Title Bar - Snap Window
            const int WM_NCHITTEST = 0x0084;//Win32, Mouse Input Notification: Determine what
part of the window corresponds to a point, allows to resize the form.
            const int resizeAreaSize = 10;


            const int HTCLIENT = 1; //Represents the client area of the window
            const int HTLEFT = 10;  //Left border of a window, allows resize horizontally to
the left
            const int HTRIGHT = 11; //Right border of a window, allows resize horizontally to
the right
            const int HTTOP = 12;   //Upper-horizontal border of a window, allows resize
vertically up
            const int HTTOPLEFT = 13;//Upper-left corner of a window border, allows resize
diagonally to the left
            const int HTTOPRIGHT = 14;//Upper-right corner of a window border, allows resize
diagonally to the right
            const int HTBOTTOM = 15; //Lower-horizontal border of a window, allows resize
vertically down
            const int HTBOTTOMLEFT = 16;//Lower-left corner of a window border, allows resize
diagonally to the left
            const int HTBOTTOMRIGHT = 17;//Lower-right corner of a window border, allows
resize diagonally to the right
            if (m.Msg == WM_NCHITTEST)
            {
                base.WndProc(ref m);
                if (this.WindowState == FormWindowState.Normal)
                {
                    if ((int)m.Result == HTCLIENT)
                    {
                        Point screenPoint = new Point(m.LParam.ToInt32());
                        Point clientPoint = this.PointToClient(screenPoint);
                        if (clientPoint.Y <= resizeAreaSize)
                        {
                            if (clientPoint.X <= resizeAreaSize)
                                m.Result = (IntPtr)HTTOPLEFT;
                            else if (clientPoint.X < (this.Size.Width - resizeAreaSize))
                                m.Result = (IntPtr)HTTOP;
                            else
                                m.Result = (IntPtr)HTTOPRIGHT;
                        }
                        else if (clientPoint.Y <= (this.Size.Height - resizeAreaSize))
                        {
                            if (clientPoint.X <= resizeAreaSize)
                                m.Result = (IntPtr)HTLEFT;
                            else if (clientPoint.X > (this.Width - resizeAreaSize))
                                m.Result = (IntPtr)HTRIGHT;
                        }
                        else
                        {
                            if (clientPoint.X <= resizeAreaSize)
                                m.Result = (IntPtr)HTBOTTOMLEFT;
                            else if (clientPoint.X < (this.Size.Width - resizeAreaSize))
                                m.Result = (IntPtr)HTBOTTOM;
                            else
                                m.Result = (IntPtr)HTBOTTOMRIGHT;
                        }
                    }
                }
                return;
```

```
            }

            if (m.Msg == WM_NCCALCSIZE && m.WParam.ToInt32() == 1)
            {
                return;
            }
            base.WndProc(ref m);

        }

        private void Main_Resize(object sender, EventArgs e)
        {
            AdjustForm();
        }

        private void AdjustForm()
        {
            switch (this.WindowState)
            {
                case FormWindowState.Maximized:
                    this.Padding = new Padding(0, 8, 8, 0);
                    break;
                case FormWindowState.Normal:
                    if (this.Padding.Top != borderSize)
                        this.Padding = new Padding(borderSize);
                    break;

            }

        }

        private void btnMinimize_Click(object sender, EventArgs e)
        {
            this.WindowState = FormWindowState.Minimized;
        }

        private void btnClose_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void panel1_MouseDown(object sender, MouseEventArgs e)
        {
            ReleaseCapture();
            SendMessage(this.Handle, 0x112, 0xf012, 0);
        }
    }
}
```

SettingManagement.cs

```
using AikidoSystem.Objects;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO.Hashing;

namespace AikidoSystem
{
   public partial class SettingsManagement : Form
   {
      private DatabaseManager databaseManager = new DatabaseManager();
      private int borderSize = 4;

      private string title;
      private string insBtn;
      private string editBtn;
      private string delBtn;
      private string labelName;
      private string labelDetails;
      private string labelQuery;
      private bool toggleView;
      private string status;

      public SettingsManagement()
      {
         InitializeComponent();
         this.Padding = new Padding(borderSize);
         this.BackColor = Color.MidnightBlue;
         databaseManager = databaseManager.Instance;

      }

      public string Title { get => title; set => title = value; }
      public string InsBtn { get => insBtn; set => insBtn = value; }
      public string EditBtn { get => editBtn; set => editBtn = value; }
      public string DelBtn { get => delBtn; set => delBtn = value; }
      public string LabelName { get => labelName; set => labelName = value; }
      public string LabelDetails { get => labelDetails; set => labelDetails = value; }
      public string LabelQuery { get => labelQuery; set => labelQuery = value; }
```

```
public bool ToggleView { get => toggleView; set => toggleView = value; }
public string Status { get => status; set => status = value; }

private void panel2_Paint(object sender, PaintEventArgs e)
{

}

private void SettingsManagement_Load(object sender, EventArgs e)
{
   dataGridView1.DefaultCellStyle.BackColor = Color.Lavender;
   dataGridView1.DefaultCellStyle.ForeColor = Color.MidnightBlue;
   dataGridView1.ColumnHeadersDefaultCellStyle.ForeColor = Color.MidnightBlue;
   dataGridView1.ColumnHeadersDefaultCellStyle.BackColor = Color.CornflowerBlue;
   dataGridView1.RowHeadersDefaultCellStyle.BackColor = Color.CornflowerBlue;
   dataGridView1.DefaultCellStyle.SelectionBackColor = Color.RoyalBlue;
   dataGridView1.DefaultCellStyle.SelectionForeColor = Color.MidnightBlue;

   label15.Text = title;
   label1.Text = labelName;
   label3.Text = LabelDetails;
   label2.Text = LabelQuery;
   button_M3.Text = insBtn;
   button_M1.Text = editBtn;
   button_M2.Text = delBtn;
   if (toggleView) panel5.Visible = true;
   else panel5.Visible = false;
   label15.Left = (this.Width - label15.Width) / 2;
   switch (status)
   {
      case "account":

         DataTable dt = new DataTable();
         dt = databaseManager.SelectAccounts();
         dataGridView1.DataSource = dt;
         textBox_M2.PasswordChar = true;
         break;
      case "payment":

         dt = databaseManager.SelectPaymentType();
         dataGridView1.DataSource = dt;
         break;
      case "hall":

         dt = databaseManager.SelectHalls();
         dataGridView1.DataSource = dt;
         break;
      case "levels":
```

```
            dt = databaseManager.SelectLevels();
            dataGridView1.DataSource = dt;
            break;
        }
    }


    [DllImport("user32.DLL", EntryPoint = "ReleaseCapture")]
    private extern static void ReleaseCapture();
    [DllImport("user32.DLL", EntryPoint = "SendMessage")]
    private extern static void SendMessage(System.IntPtr hWnd, int wMsg, int wParam, int lParam);


    protected override void WndProc(ref Message m)
    {
        const int WM_NCCALCSIZE = 0x0083;//Standar Title Bar - Snap Window
        const int WM_NCHITTEST = 0x0084;//Win32, Mouse Input Notification: Determine what part of the window corresponds to a point, allows to resize the form.
        const int resizeAreaSize = 10;


        const int HTCLIENT = 1; //Represents the client area of the window
        const int HTLEFT = 10;  //Left border of a window, allows resize horizontally to the left
        const int HTRIGHT = 11; //Right border of a window, allows resize horizontally to the right
        const int HTTOP = 12;   //Upper-horizontal border of a window, allows resize vertically up
        const int HTTOPLEFT = 13;//Upper-left corner of a window border, allows resize diagonally to the left
        const int HTTOPRIGHT = 14;//Upper-right corner of a window border, allows resize diagonally to the right
        const int HTBOTTOM = 15; //Lower-horizontal border of a window, allows resize vertically down
        const int HTBOTTOMLEFT = 16;//Lower-left corner of a window border, allows resize diagonally to the left
        const int HTBOTTOMRIGHT = 17;//Lower-right corner of a window border, allows resize diagonally to the right
        if (m.Msg == WM_NCHITTEST)
        {
            base.WndProc(ref m);
            if (this.WindowState == FormWindowState.Normal)
            {
                if ((int)m.Result == HTCLIENT)
                {
                    Point screenPoint = new Point(m.LParam.ToInt32());
                    Point clientPoint = this.PointToClient(screenPoint);
                    if (clientPoint.Y <= resizeAreaSize)
                    {
                        if (clientPoint.X <= resizeAreaSize)
                            m.Result = (IntPtr)HTTOPLEFT;
                        else if (clientPoint.X < (this.Size.Width - resizeAreaSize))
                            m.Result = (IntPtr)HTTOP;
                        else
                            m.Result = (IntPtr)HTTOPRIGHT;
```

```
            }
            else if (clientPoint.Y <= (this.Size.Height - resizeAreaSize))
            {
               if (clientPoint.X <= resizeAreaSize)
                  m.Result = (IntPtr)HTLEFT;
               else if (clientPoint.X > (this.Width - resizeAreaSize))
                  m.Result = (IntPtr)HTRIGHT;
            }
            else
            {
               if (clientPoint.X <= resizeAreaSize)
                  m.Result = (IntPtr)HTBOTTOMLEFT;
               else if (clientPoint.X < (this.Size.Width - resizeAreaSize))
                  m.Result = (IntPtr)HTBOTTOM;
               else
                  m.Result = (IntPtr)HTBOTTOMRIGHT;
            }
         }
      }
      return;
   }

   if (m.Msg == WM_NCCALCSIZE && m.WParam.ToInt32() == 1)
   {
      return;
   }
   base.WndProc(ref m);

}

private void Main_Resize(object sender, EventArgs e)
{
   AdjustForm();
}

private void AdjustForm()
{
   switch (this.WindowState)
   {
      case FormWindowState.Maximized:
         this.Padding = new Padding(0, 8, 8, 0);
         break;
      case FormWindowState.Normal:
         if (this.Padding.Top != borderSize)
            this.Padding = new Padding(borderSize);
         break;

   }
```

```
    }

    private void panel1_MouseDown(object sender, MouseEventArgs e)
    {
       ReleaseCapture();
       SendMessage(this.Handle, 0x112, 0xf012, 0);
    }

    private void btnMinimize_Click(object sender, EventArgs e)
    {
       this.WindowState = FormWindowState.Minimized;
    }

    private void btnClose_Click(object sender, EventArgs e)
    {
       databaseManager.Dispose();
       this.Close();
    }

    private void panel1_Paint(object sender, PaintEventArgs e)
    {

    }


    private void button_M3_Click(object sender, EventArgs e)
    {

       switch (status)
       {

          case "account":
             DataTable dt = new DataTable();
             bool flag = false;
             Account account = new Account();
              flag = true;

             account.Username = textBox_M1.Texts;
             //byte[] str = Encoding.Unicode.GetBytes(textBox_M2.Texts);
             account.Password =
Encoding.Unicode.GetString(XxHash64.Hash(Encoding.Unicode.GetBytes(textBox_M2.Texts)));
             if (toggle1.Checked)
                account.Access = "admin";
             else account.Access = "trainer";
             while (flag)
             {
                if (databaseManager.InsertAccount(account))
```

```
        {
            flag = false;
            MessageBox.Show("Акаунта беше създаден");
            dt = new DataTable();
            dt = databaseManager.SelectAccounts();
            dataGridView1.DataSource = dt;
        }
        else
        {
            MessageBox.Show("Акаунта НЕ беше създаден");
            break;

        }
    }


    break;

case "payment":

    PaymentType paymentType = new PaymentType();
    flag = true;

    paymentType.TypeName = textBox_M1.Texts;

    paymentType.Price = double.Parse(textBox_M2.Texts);

    while (flag)
    {
        if (databaseManager.InsertPaymentType(paymentType))
        {
            flag = false;
            MessageBox.Show("Записа беше създаден");
            dt = new DataTable();
            dt = databaseManager.SelectPaymentType();
            dataGridView1.DataSource = dt;
        }
        else
        {
            MessageBox.Show("Записа НЕ беше създаден");
            break;

        }
    }


    break;
case "hall":
```

```
    Hall hall = new Hall();
    flag = true;

    hall.Name = textBox_M1.Texts;
    hall.Address = textBox_M2.Texts;


    while (flag)
    {
        if (databaseManager.InsertHalls(hall))
        {
            flag = false;
            MessageBox.Show("Записа беше създаден");
            dt = new DataTable();
            dt = databaseManager.SelectHalls();
            dataGridView1.DataSource = dt;
        }
        else
        {
            MessageBox.Show("Записа НЕ беше създаден");
            break;

        }
    }

    break;
case "levels":
    KyuLevels level = new KyuLevels();
    flag = true;

    level.Name = textBox_M1.Texts;
    level.LevelType = textBox_M2.Texts;

    while (flag)
    {
        if (databaseManager.InsertLevels(level))
        {
            flag = false;
            MessageBox.Show("Записа беше създаден");
            dt = new DataTable();
             dt = databaseManager.SelectLevels();
            dataGridView1.DataSource = dt;
        }
        else
        {
            MessageBox.Show("Записа НЕ беше създаден");
            break;
```

```
            }
        }


        break;



    }
}

private void button_M1_Click(object sender, EventArgs e)
{
    DataTable dt = new DataTable();
    bool flag = false;
    switch (status)
    {

        case "account":
            Account account1 = new Account();
            Account account2 = new Account();
            flag = true;

            account1.Username = textBox_M1.Texts;
            byte[] str = Encoding.Unicode.GetBytes(textBox_M2.Texts);
            account1.Password = XxHash64.Hash(str).ToString();
            if (toggle1.Checked)
                account1.Access = "admin";
            else account1.Access = "trainer";
            account2.Username = dataGridView1.CurrentRow.Cells[0].Value.ToString();

            while (flag)
            {
                if (databaseManager.UpdateAccount(account1,account2))
                {
                    flag = false;
                    MessageBox.Show("Записа беше обновен");
                }
                else
                {
                    MessageBox.Show("Записа НЕ беше обновен");
                    break;

                }
            }
            dt = databaseManager.SelectAccounts();
            dataGridView1.DataSource = dt;
            break;
```

```
case "payment":
   PaymentType paymentType = new PaymentType();
   PaymentType pay = new PaymentType();
   flag = true;

   paymentType.TypeName = textBox_M1.Texts;
   paymentType.Price = double.Parse(textBox_M2.Texts);

   while (flag)
   {
      if (databaseManager.UpdatePaymentType(paymentType, pay))
      {
         flag = false;
         MessageBox.Show("Записа беше обновен");
      }
      else
      {
         MessageBox.Show("Записа НЕ беше обновен");
         break;

      }
   }
   dt = databaseManager.SelectPaymentType();
   dataGridView1.DataSource = dt;
   break;
case "hall":
   Hall hall = new Hall();
   Hall h = new Hall();
   flag = true;

   hall.Name = textBox_M1.Texts;
   hall.Address = textBox_M2.Texts;
   h.Name = dataGridView1.CurrentRow.Cells[0].Value.ToString();

   while (flag)
   {
      if (databaseManager.UpdateHalls(hall, h))
      {
         flag = false;
         MessageBox.Show("Записа беше обновен");
      }
      else
      {
         MessageBox.Show("Записа НЕ беше обновен");
         break;

      }
   }
```

```
            dt = databaseManager.SelectHalls();
            dataGridView1.DataSource = dt;
            break;
        case "levels":
            KyuLevels level = new KyuLevels();
            KyuLevels l = new KyuLevels();
            flag = true;

            level.Name = textBox_M1.Texts;
            level.LevelType = textBox_M2.Texts;
            l.Name = dataGridView1.CurrentRow.Cells[0].Value.ToString();

            while (flag)
            {
                if (databaseManager.UpdateLevels(level, l))
                {
                    flag = false;
                    MessageBox.Show("Записа беше обновен");
                }
                else
                {
                    MessageBox.Show("Записа НЕ беше обновен");
                    break;

                }
            }
            dt = databaseManager.SelectLevels();
            dataGridView1.DataSource = dt;
            break;



    }
}

private void button_M2_Click(object sender, EventArgs e)
{
    switch (status)
    {
        case "account":
            Account account = new Account();

            account.Username = dataGridView1.CurrentRow.Cells[0].Value.ToString();
            //byte[] str = Encoding.Unicode.GetBytes(dataGridView1.CurrentRow.Cells[1].Value.ToString());
            //account.Password = XxHash64.Hash(str).ToString();

            if (databaseManager.DeleteAccount(account))
            {
```

```
            MessageBox.Show("Акаунтът е изтрит!");
            DataTable dt = new DataTable();
            dt = databaseManager.SelectAccounts();
            dataGridView1.DataSource = dt;

        }
        else MessageBox.Show("Акаунтът не е изтрит!");


        break;
    case "payment":
        PaymentType payType = new PaymentType();

        payType.Id = int.Parse(dataGridView1.CurrentRow.Cells[0].Value.ToString());


        if (databaseManager.DeletePaymentType(payType))
        {
            MessageBox.Show("Записът е изтрит!");
            DataTable dt = new DataTable();
            dt = databaseManager.SelectPaymentType();
            dataGridView1.DataSource = dt;

        }
        else MessageBox.Show("Записът не е изтрит!");

        break;
    case "hall":
        Hall hall = new Hall();

        hall.Id = int.Parse(dataGridView1.CurrentRow.Cells[0].Value.ToString());


        if (databaseManager.DeleteHalls(hall))
        {
            MessageBox.Show("Записът е изтрит!");
            DataTable dt = new DataTable();
            dt = databaseManager.SelectHalls();
            dataGridView1.DataSource = dt;
        }
        else MessageBox.Show("Записът не е изтрит!");
        break;
    case "levels":
        KyuLevels level = new KyuLevels();

        level.Id = int.Parse(dataGridView1.CurrentRow.Cells[0].Value.ToString());
```

```
            if (databaseManager.DeleteLevels(level))
            {
               MessageBox.Show("Записът е изтрит!");
               DataTable dt = new DataTable();
               dt = databaseManager.SelectLevels();
               dataGridView1.DataSource = dt;
            }
            else MessageBox.Show("Записът не е изтрит!");
            break;



        }
      }
   }
}
```