МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ
И ПРОГРАММНОЙ ИНЖЕНЕРИИ (КАФЕДРА №43)

КУРСОВОЙ ПРОЕКТ
ЗАЩИЩЕН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

Старший преподаватель		Е.О. Шумова
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОМУ ПРОЕКТУ

Разработка приложения для организации взаимодействия объектов при заданных критериях

по дисциплине: ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР.№ 4832		В.В. Поляков
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2020

Задание на курсовое проектирование

В ходе курсового проекта необходимо разработать приложение, позволяющее протестировать взаимодействие объектов классов, спроектированных и реализованных студентом для решения конкретной задачи при заданных критериях.

При разработке программного обеспечения следует использовать шаблоны проектирования. Также предпочтение должно быть отдано графическому приложению.

Вариант задания на курсовое проектирование:

20) Разработка системы классов «Расписание занятий в университете».

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Одной из важнейших проблем качественной организации учебного процесса в высшем учебном заведении является задача создания автоматизированного учебного расписания. Правильно и точно составленное расписание обеспечивает равномерную загрузку студенческих групп и профессорско-преподавательского состава.

В настоящее время использование информационных систем в высших образовательных учреждениях не является редкостью. Спектр их применения широк и варьируется от автоматизации отдельно взятых рабочих мест до полной автоматизации деятельности ВУЗа.

Вне зависимости от объекта автоматизации, будь то преподавательский состав или администрация университета, в образовательном учреждении такие системы внедряют, преследуя конечную цель - повышение качества образования.

Целью курсовой работой является разработка системы классов «Расписание занятий в университете».

1. Постановка задачи

1.1 Анализ предметной области

Расписание занятий в высшем учебном заведении служит для сведения в единую взаимосвязанную систему учащихся, преподавателей и мест проведения занятий (аудиторий).

Оптимизация расписания занятий является одним из основных факторов, способных существенно оптимизировать учебный процесс.

В частности, организация учебного процесса, представляя собой один из важнейших этапов на пути развития и эффективного функционирования вуза, является совокупностью взаимосвязанных задач, решаемых различными подразделениями учебного заведения. Одной из таких задач является составление расписания. От того, насколько хорошо составлено расписание зависит эффективность работы преподавателей, усвоение учебного материала студентами, рациональное использование интеллектуальной и материальной баз вуза.

Методический отдел занимается составлением расписания занятий в университете. Студенты и преподаватели пользуются им. Необходимо разработать систему классов, которая поможет хранить расписание и осуществлять с ним различные операции. Для этого необходимо иметь следующую информацию:

- группы;
- преподаватели;
- аудитории;
- учебный план.

Методический отдел должен иметь возможность редактирования хранящихся данных. При составлении или редактировании расписания важно учесть несколько моментов:

- В одной аудитории, в одно и то же время не могут заниматься разные группы, если это не лекция;

- Одна и та же группа в одно и то же время занимается в одной аудитории, одним предметом, с одним преподавателем;
- Лекции должны проводиться в аудиториях с достаточной вместительностью для потока;
- Нагрузка преподавателя не может быть больше определенного значения;
- Один и тот же преподаватель преподает в одно и то же время в одной и той же аудитории, один и тот же предмет.

Студенты и преподаватели должны иметь возможность просмотра данных. При этом они могут искать расписание по разным критериям и их совокупности:

- преподаватели;
- группы;
- номер пары.

Словарь предметной области:

- Расписание;
- Учебный план;
- Преподаватель;
- Группа;
- Предмет;
- Аудитория;
- Время;
- Нагрузка;
- Вместительность;
- Дисциплина
- Добавить;
- Удалить;
- Найти;

- Просмотреть;
- Сгенерировать.

1.2 Формулировка технического задания

Необходимо разработать и реализовать систему классов «Расписание занятий в университете».

1.2.1 Назначение разработки

Система классов предназначена для обработки данных, необходимых для составления и осуществления различных операций с расписанием занятий в университете.

1.2.2 Требования к функциональным характеристикам

- Хранение информации о расписании;
- Хранение информации о группах;
- Хранение информации о преподавателях;
- Хранение информации об учебном плане;
- Хранение информации об аудиториях;
- Просмотр информации о расписании;
- Просмотр информации о группах;
- Просмотр информации о преподавателях;
- Просмотр информации об учебном плане;
- Просмотр информации об аудиториях;
- Добавление данных в расписание;
- Добавление данных о группах;
- Добавление данных о преподавателях;
- Добавление данных об учебном плане;
- Добавление данных об аудиториях;
- Удаление данных из расписания;

- Удаление данных о группах;
- Удаление данных о преподавателях;
- Удаление данных об учебном плане;
- Удаление данных об аудиториях;
- Генерация расписания;
- Поиск в расписании по группе;
- Поиск в расписании по преподавателю;
- Поиск в расписании по времени;
- Поиск в расписании по группе и времени;
- Поиск в расписании по группе и преподавателю;
- Поиск в расписании по преподавателю и времени.

1.2.3 Требования к графическому интерфейсу

Программа состоит из одной формы – формы главного интерфейса. Она состоит из пяти таблиц, полей ввода, полей выбора элементов, различных кнопок и заголовков:

- Таблицы отображают информацию о расписании, учебном плане, преподавателях, группах и аудиториях.
- Поля ввода предназначены для добавления/удаления элементов в таблицы учебного плана, преподавателей, групп и аудиторий.
- Поля выбора элементов предназначены для добавления/удаления элементов в таблицу расписания, а также для поиска в расписании.
- Кнопки запускают процесс осуществления всех вышеперечисленных функций.

1.2.4 Требования к составу и параметрам технических средств

Система должна работать на совместимых персональных компьютерах.

Минимальная конфигурация:

- тип процессора – Intel Pentium
- объем RAM – 4096 Мб
- операционная система: Windows 7
- установленная версия .NET: 4.7.2

Рекомендуемая конфигурация:

- тип процессора – Intel Core I5
- объем RAM – 8 Гб
- операционная система: Windows 10
- установленная версия .NET: 4.7.2

2. Проектирование классов

2.1 Классы сущностей

Классы-сущности (entity classes) содержат хранимую информацию. Они имеют наибольшее значение для пользователя, и потому в их названиях часто используют термины из предметной области. Обычно для каждого класса-сущности создают таблицу в базе данных.

В представленной системе классами-сущностями являются следующие классы:

Рисунок 1 - Класс «Расписание занятий»

- Расписание занятий – хранит информацию для каждой конкретной единицы единого расписания;

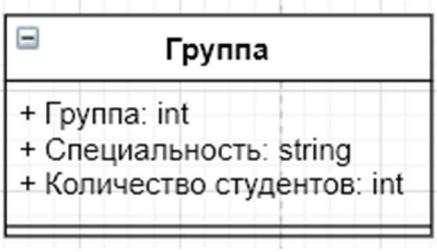


Рисунок 2 - Класс «Группа»

- Группа – хранит информацию о группах;

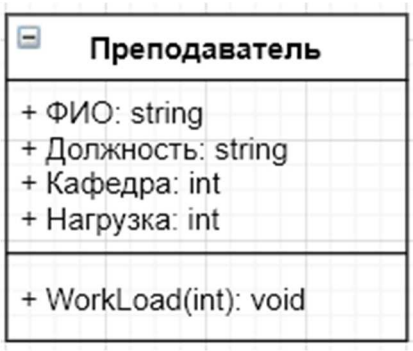


Рисунок 3 - Класс «Преподаватель»

- Преподаватель – хранит информацию о преподавателях;

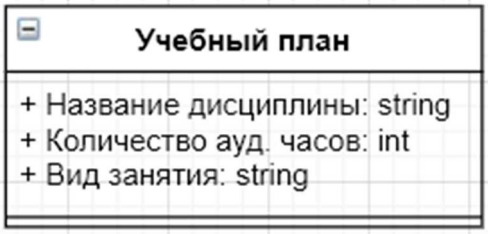


Рисунок 4 - Класс «Учебный план»

- Учебный план – хранит информацию об учебном плане;

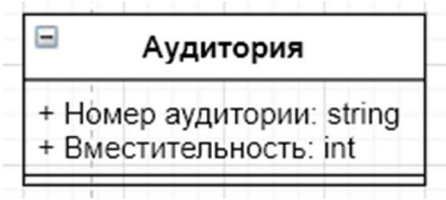


Рисунок 5 - Класс «Аудитория»

- Аудитория – хранит информацию об аудиториях;

2.2 Управляющие классы

Управляющие классы (control classes) отвечают за координацию действий других классов. Обычно у каждого варианта использования имеется один управляющий класс, контролирующий последовательность событий этого варианта использования. Управляющий класс отвечает за координацию, но сам не несет в себе никакой функциональности, так как остальные классы не посылают ему большого количества сообщений. Вместо

этого он сам посылает множество сообщений. Управляющий класс просто делегирует ответственность другим классам, по этой причине его часто называют классом-менеджером.

В представленной системе управляющем классом является класс Пользователь:

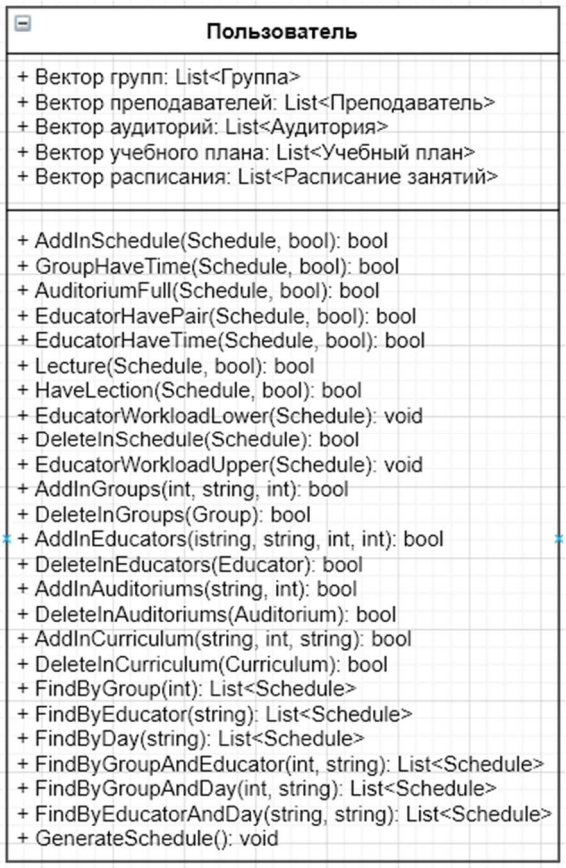


Рисунок 6 - Класс «Пользователь»

Данный класс осуществляет работу с остальными классами и является посредником между классом формы и классами-сущностями.

Данная система спроектирована в форме приложения с экранными формами. Для реализации проектируемой системы используется одна экранная форма для которой предназначен данный класс:

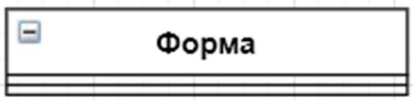


Рисунок 7 - Класс «Форма»

2.3 Общее представление диаграммы классов

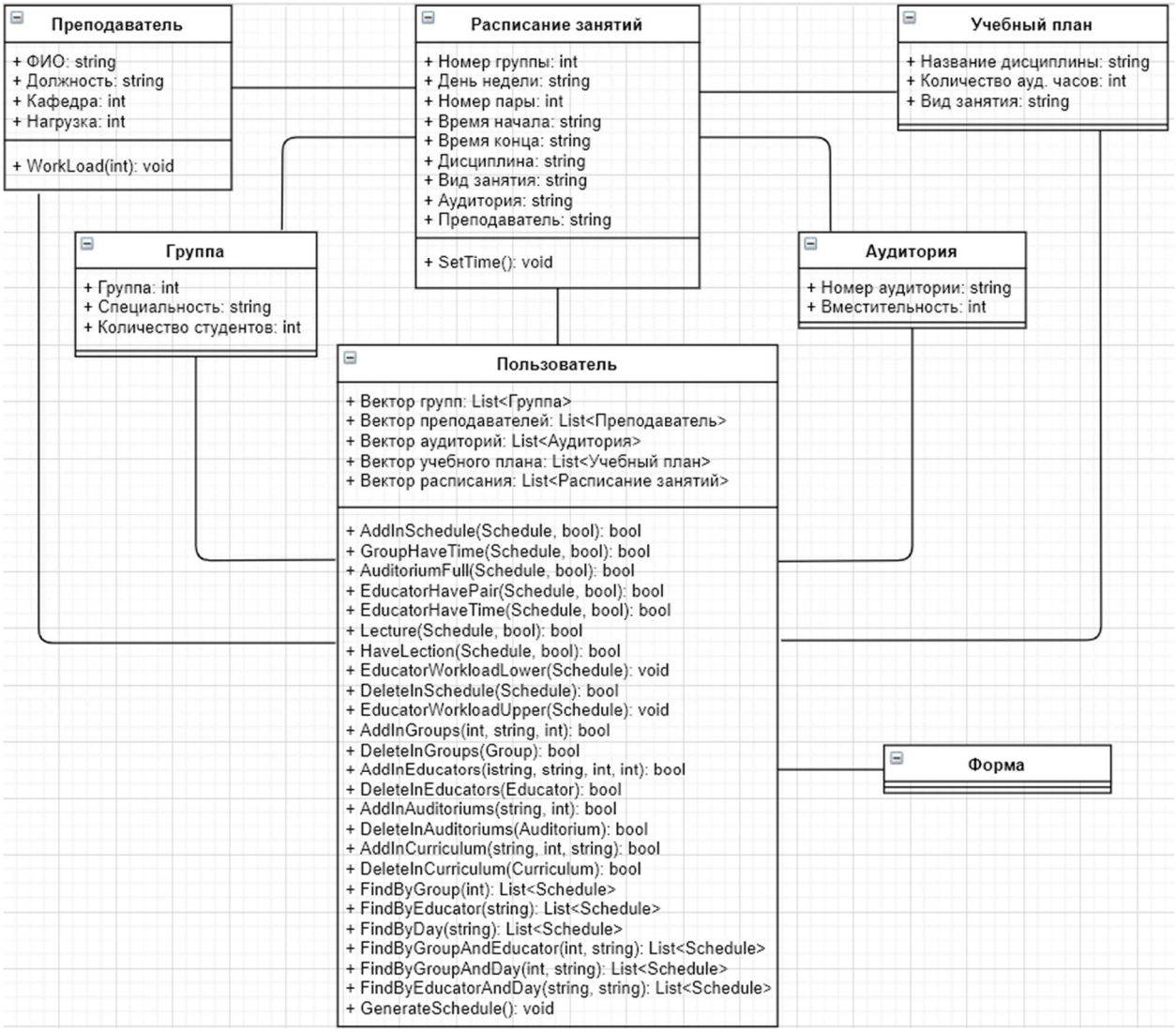


Рисунок 8 - Диаграмма классов «Расписание занятий в университете»

3. Разработка приложения

3.1 Разработка интерфейса приложения

Программа состоит из одной формы – формы главного интерфейса. Она состоит из пяти таблиц, полей ввода, полей выбора элементов, различных кнопок и заголовков:

- Таблицы отображают информацию о расписании, учебном плане, преподавателях, группах и аудиториях.
- Поля ввода предназначены для добавления/удаления элементов в таблицы учебного плана, преподавателей, групп и аудиторий.
- Поля выбора элементов предназначены для добавления/удаления элементов в таблицу расписания, а также для поиска в расписании.

Кнопки запускают процесс осуществления всех вышеперечисленных функций.

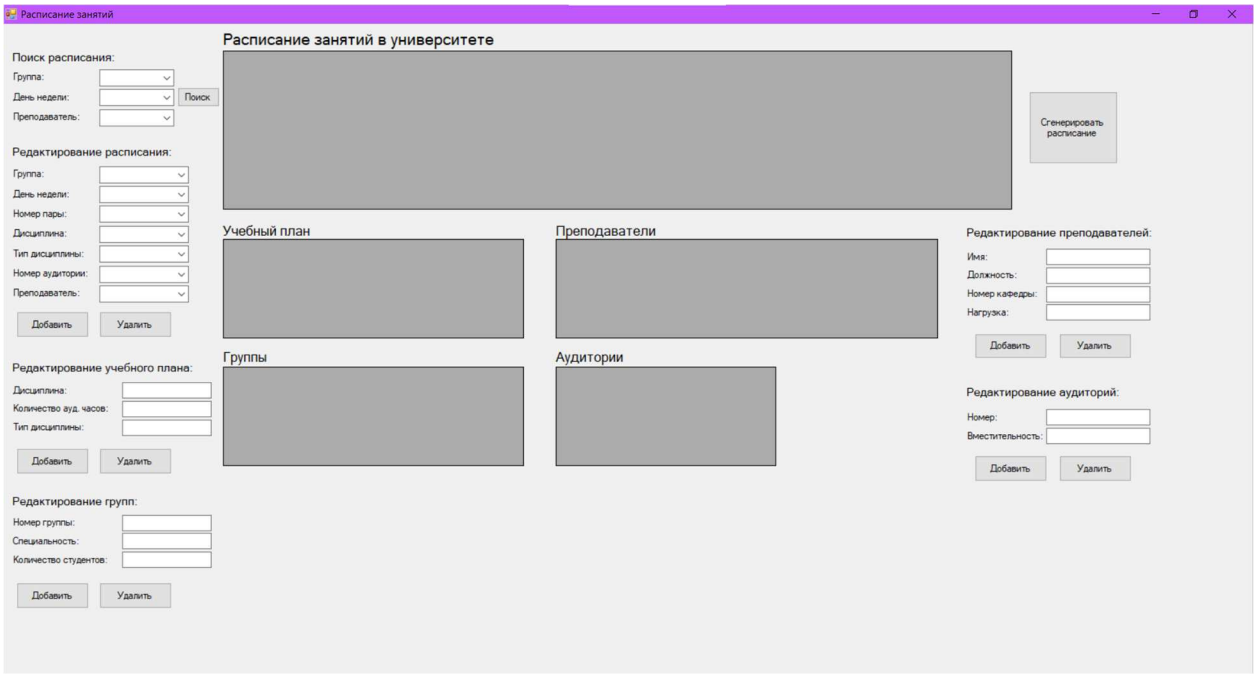


Рисунок 9 - Форма главного интерфейса

3.2 Реализация классов

В программе были реализованы все классы-сущности разрабатываемой системы, в частности классы: *Schedule*, *Group*, *Educator*, *Curriculum* и *Auditorium*. Для класса Schedule определен метод *void SetTime()*, предназначенный для установки времени начала и окончания пар. Для класса Educator определен метод *void WorkLoad(int)*, предназначенный для корректировки оставшегося времени нагрузки преподавателя.

Также был реализован управляющий класс Пользователь (*User*), для которого были определены следующие методы:

- *bool AddInSchedule(Schedule, bool)* – метод добавления новой записи в расписание;
- *bool GroupHaveTime(Schedule, bool)* – метод проверки наличия занятий у группы в данное время;
- *bool AuditoriumFull(Schedule, bool)* – метод проверки занятости в данное время и вместительности аудитории;
- *bool EducatorHavePair(Schedule, bool)* – метод проверки занятости преподавателя в данное время;
- *bool EducatorHaveTime(Schedule, bool)* – метод проверки превышения нагрузки на преподавателя;
- *bool Lecture(Schedule, bool)* – метод проверки лекционных занятий;
- *bool HaveLection(Schedule, bool)* – метод проверки наличия лекционных занятий у группы в данное время;
- *void EducatorWorkloadLower(Schedule)* – метод уменьшения оставшегося времени нагрузки преподавателя;
- *bool DeleteInSchedule(Schedule)* – метод удаления записи из расписания;
- *void EducatorWorkloadUpper(Schedule)* – метод увеличения оставшегося времени нагрузки на преподавателя;
- *bool AddInGroups(int, string, int)* – метод добавления новой группы;

- *bool DeleteInGroups(Group)* – метод удаления информации о группе;
- *bool AddInEducators(istring, string, int, int)* – метод добавления нового преподавателя;
- *bool DeleteInEducators(Educator)* – метод удаления информации о преподавателе;
- *bool AddInAuditoriums(string, int)* – метода добавления новой аудитории;
- *bool DeleteInAuditoriums(Auditorium)* – метод удаления информации об аудитории;
- *bool AddInCurriculum(string, int, string)* – метод добавления новой записи в учебный план;
- *bool DeleteInCurriculum(Curriculum)* – метод удаления записи из учебного плана;
- *List<Schedule> FindByGroup(int)* – метод поиска в расписании по группе;
- *List<Schedule> FindByEducator(string)* – метод поиска в расписании по преподавателю;
- *List<Schedule> FindByDay(string)* – метод поиска в расписании по дню недели;
- *List<Schedule> FindByGroupAndEducator(int, string)* – метод поиска в расписании по группе и преподавателю;
- *List<Schedule> FindByGroupAndDay(int, string)* – метод поиска в расписании по группе и дню недели;
- *List<Schedule> FindByEducatorAndDay(string, string)* – метод поиска в расписании по преподавателю и дню недели;
- *void GenerateSchedule()* – метод генерации расписания.

Кроме того, был реализован и класс Форма(*FormSchedule*), который осуществляет взаимодействие с элементами формы и связывает её с *User*.

Наиболее сложными алгоритмами являются алгоритм добавления записи в расписание и алгоритм генерации расписания.

Алгоритм добавления записи в расписание включает в себя следующие шаги:

1. Начало алгоритма;
2. Получение данных записи с формы;
3. Проверка на то, что у данной группы нет пар в данное время;
4. Проверка на то, что данная аудитория пустует, в случае если тип данного предмета не является лекцией;
5. Проверка на то, что данная аудитория не занята;
6. Проверка на то, что данная аудитория вмещает данное количество студентов;
7. Проверка на то, что данный преподаватель не занят в данное время;
8. Проверка на то, что нагрузка данного преподавателя не будет превышена:
9. Проверка на то, что у данной группы нет лекций в данное время;
10. Если тип данного предмета является лекцией, то проверка на то, что данная лекция проходит в одно время с потоком;
11. Если все проверки пройдены, уменьшение оставшегося времени нагрузки данного преподавателя, иначе пункт 13;
12. Добавление записи в расписание;
13. Обновление данных формы;
14. Конец алгоритма.

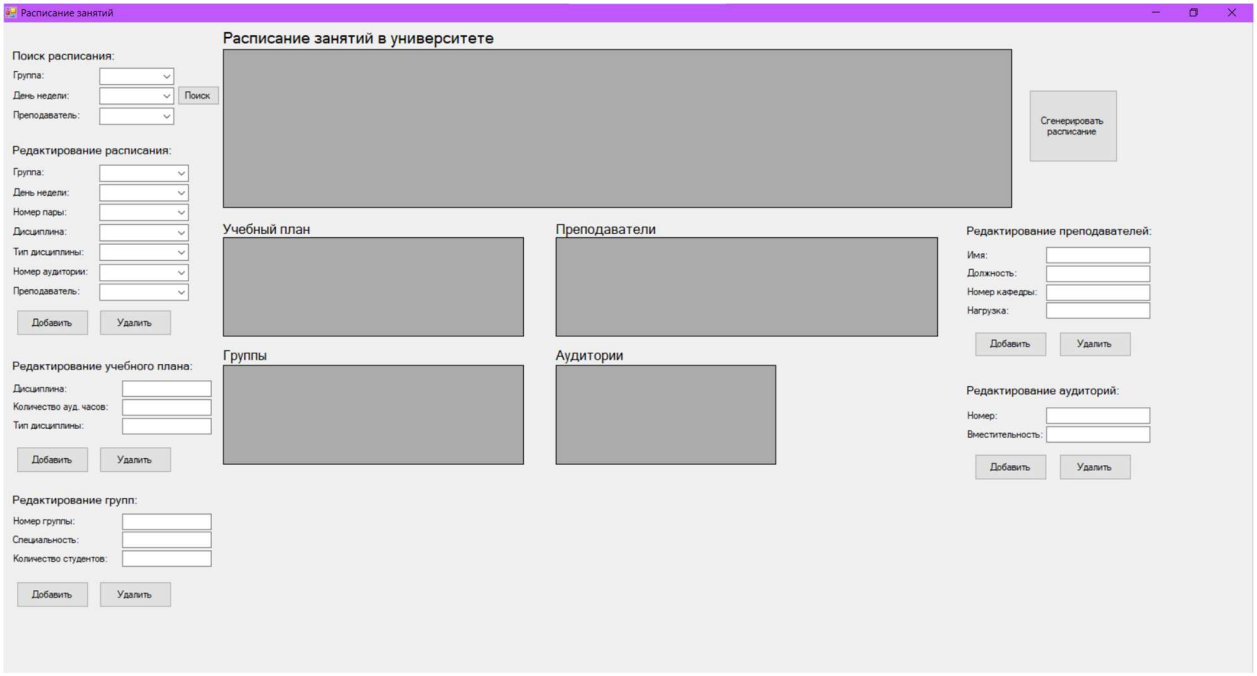
Алгоритм генерации расписания включает в себя следующие шаги:

1. Начало алгоритма;
2. Добавление групп в список групп;
3. Добавление преподавателей в список преподавателей;
4. Добавление аудиторий в список аудиторий;
5. Добавление записей учебного плана в список учебного плана;

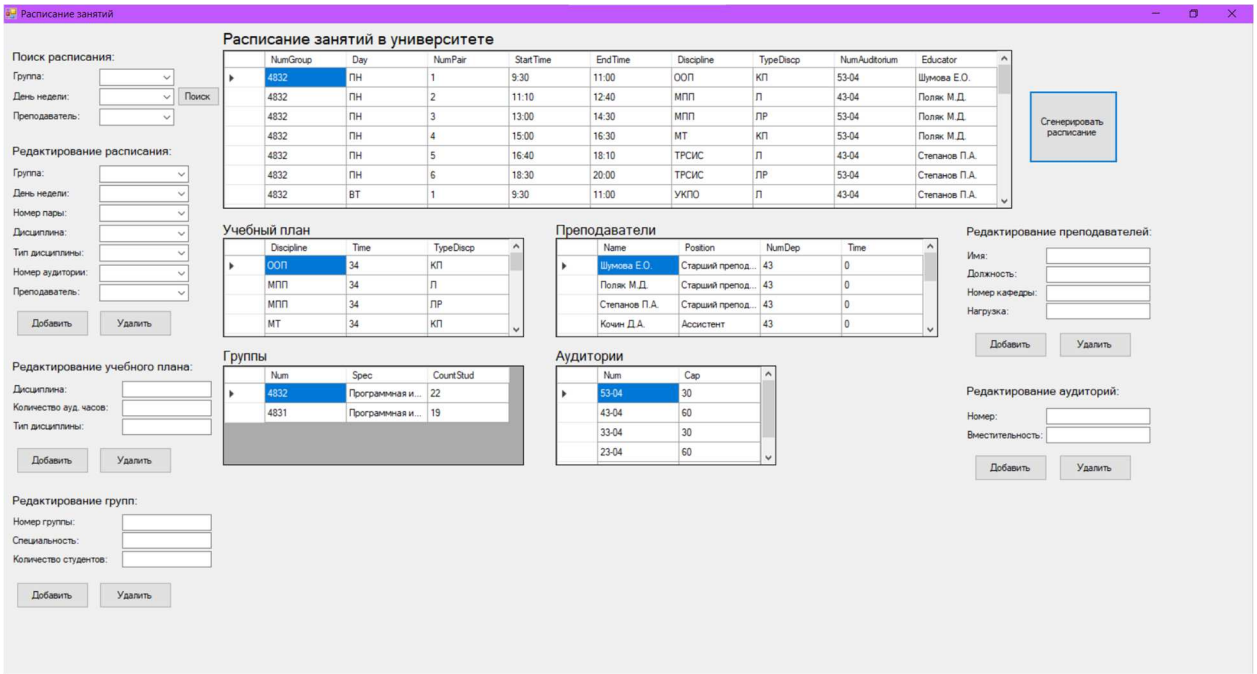
6. Если еще не все группы были опробованы, то выбор группы из списка групп, иначе пункт 12;
7. Если еще не все записи учебного плана были опробованы, то выбор записи учебного плана из списка учебного плана, иначе пункт 6;
8. Если еще не все дни недели были опробованы, то выбор дня недели, иначе пункт 7;
9. Если еще не все номера пар были опробованы, то выбор номера пары, иначе пункт 8;
10. Попытка добавления записи в расписание согласно алгоритму добавления записи в расписание, описанному выше;
11. Если попытка успешна, то пункт 7, иначе пункт 9;
12. Конец алгоритма.

4. Тестирование

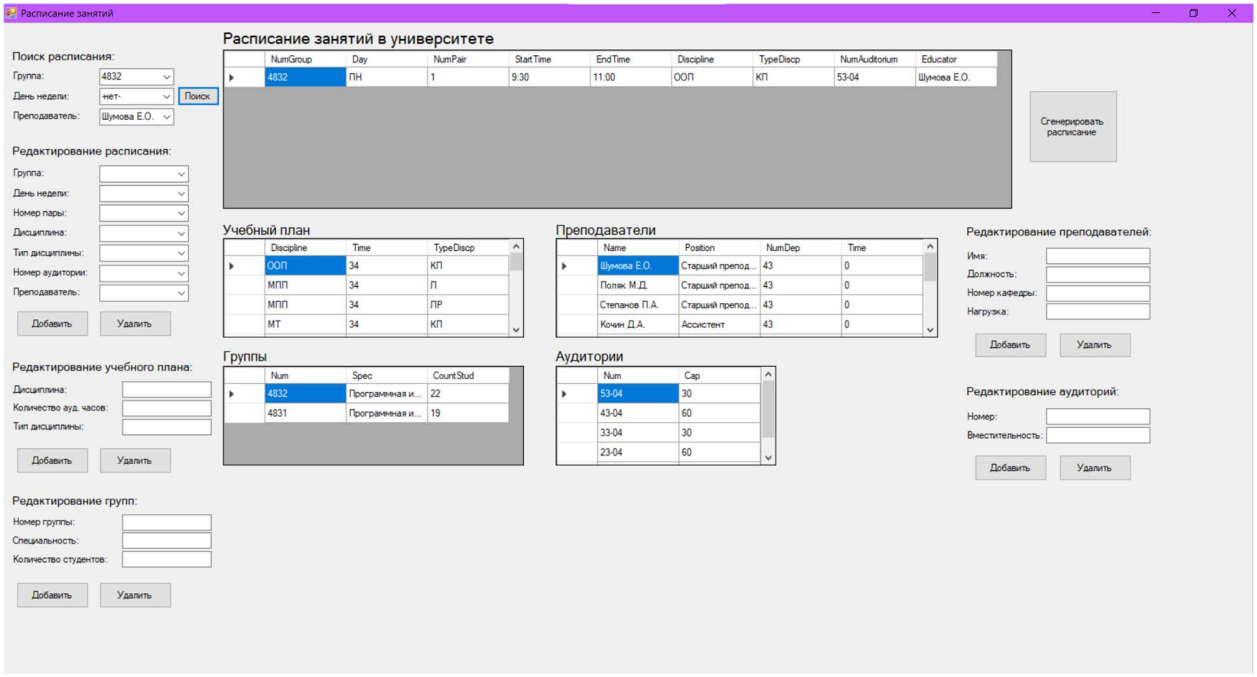
До выполнения всех тестов:



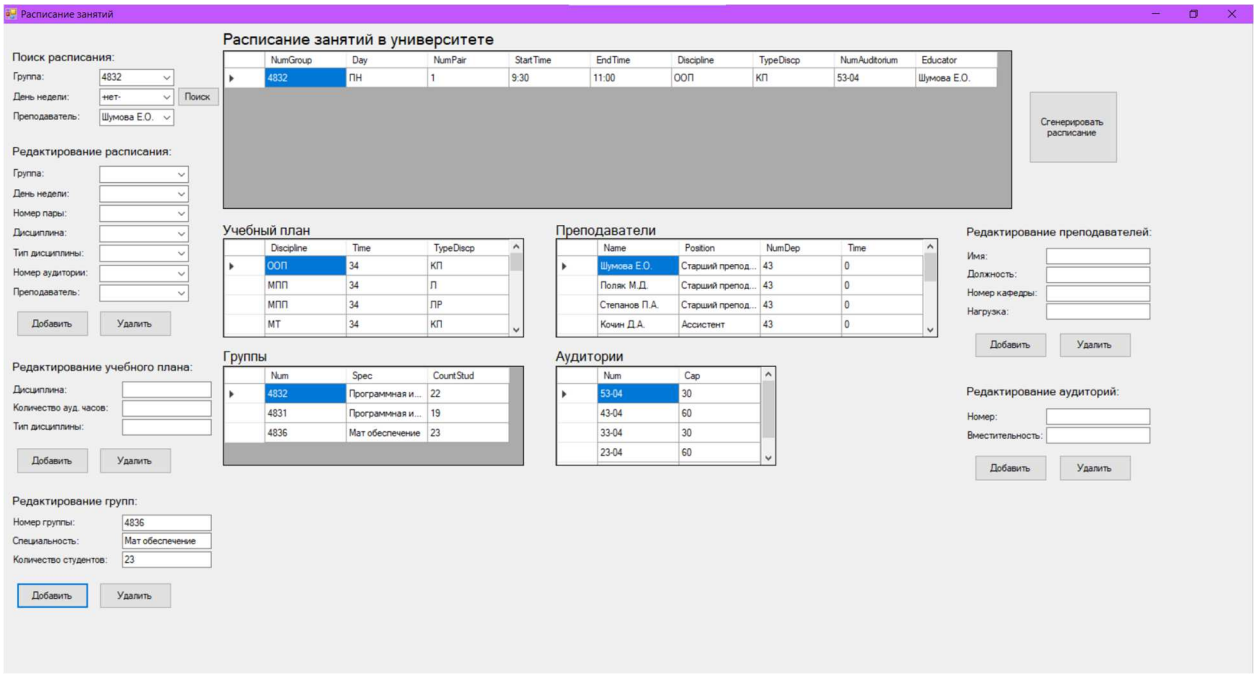
- Генерация расписания:



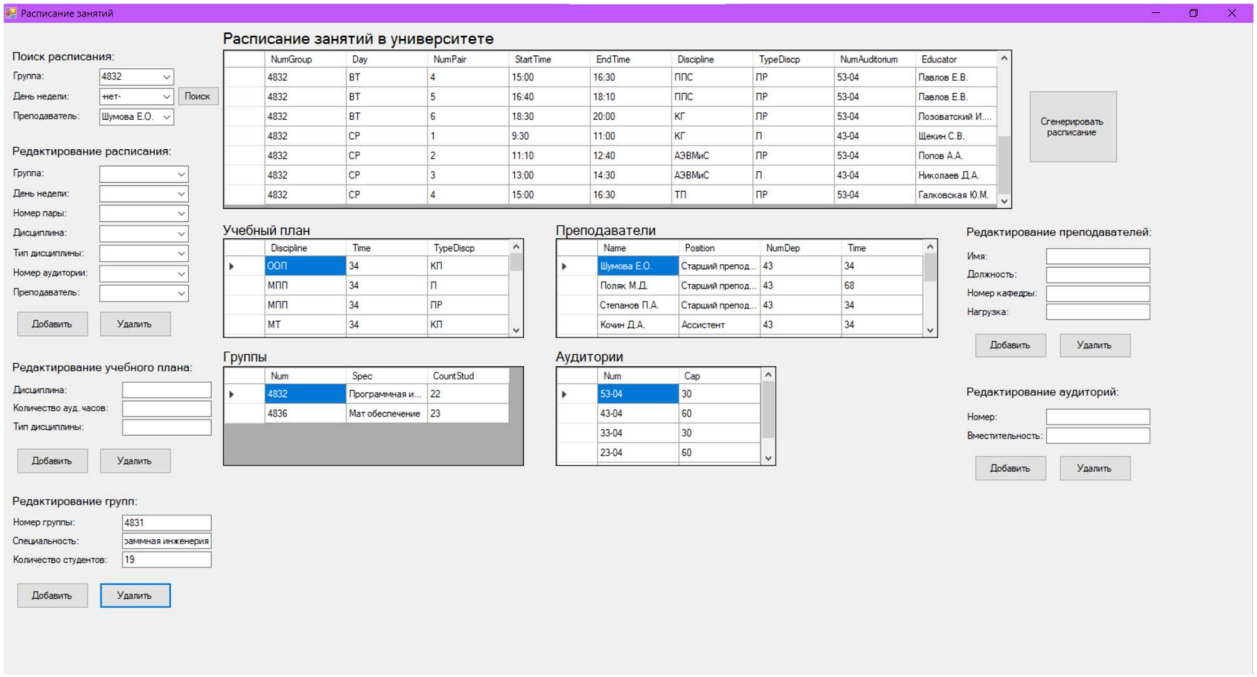
- Поиск в расписании (группа и преподаватель):



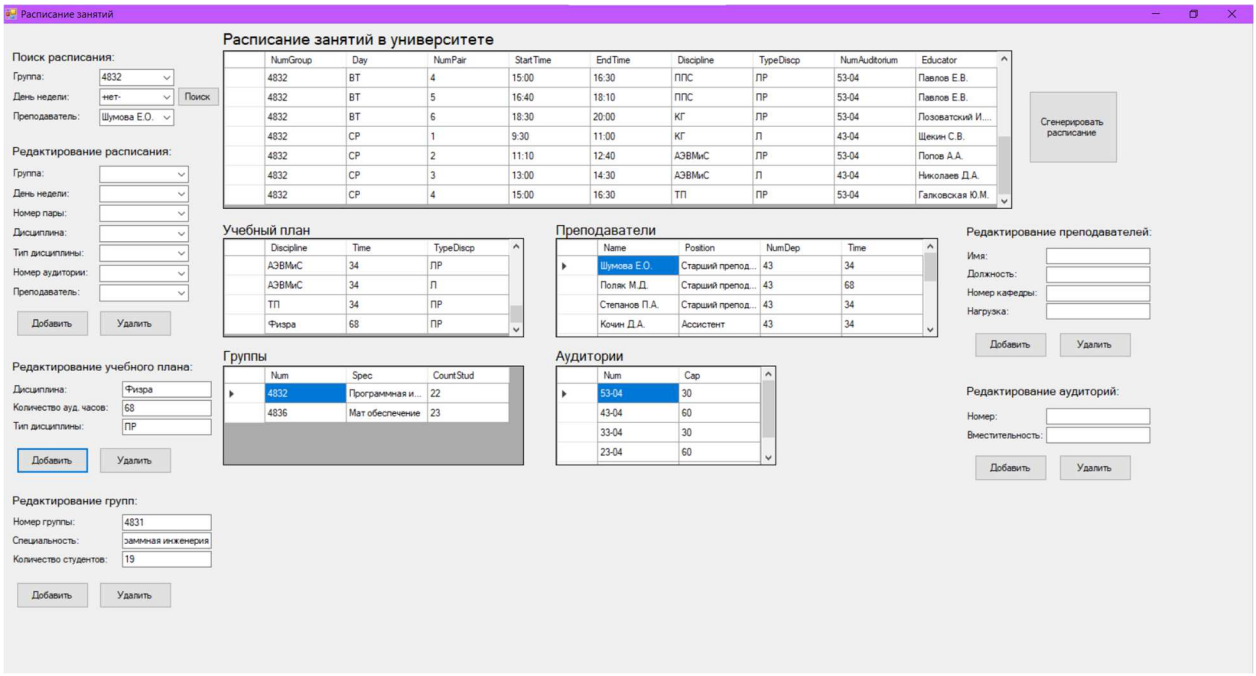
- Добавление группы (4836):



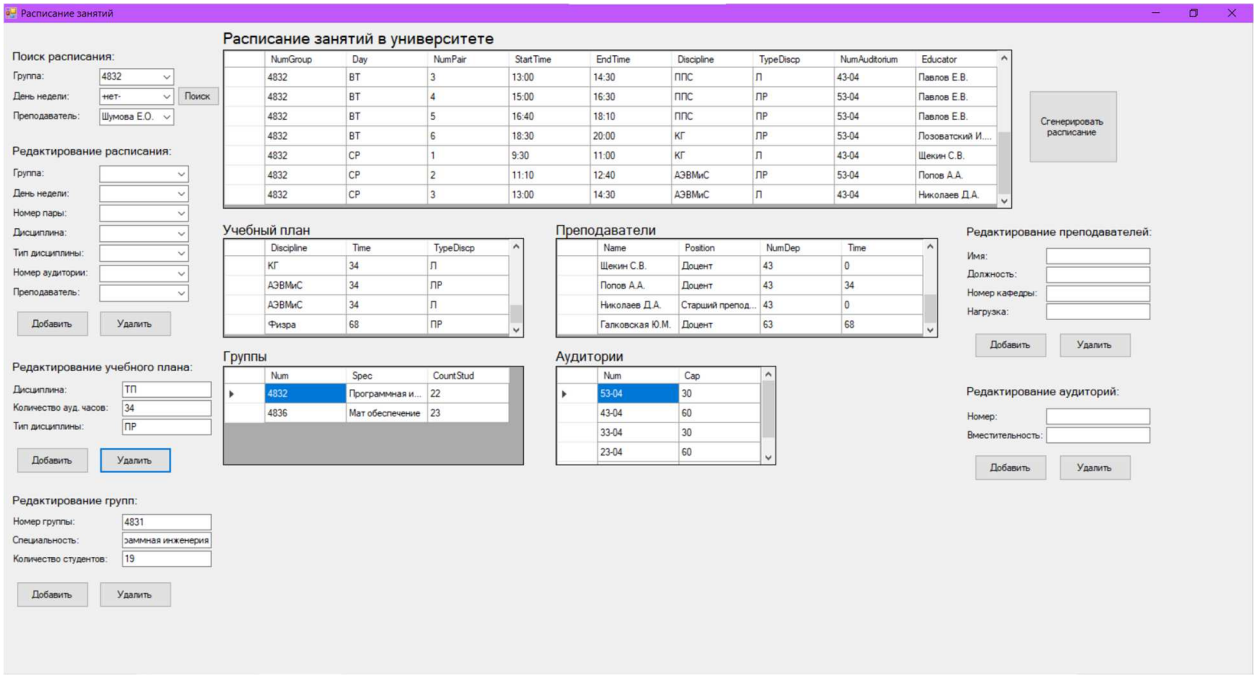
- Удаление группы (4831):



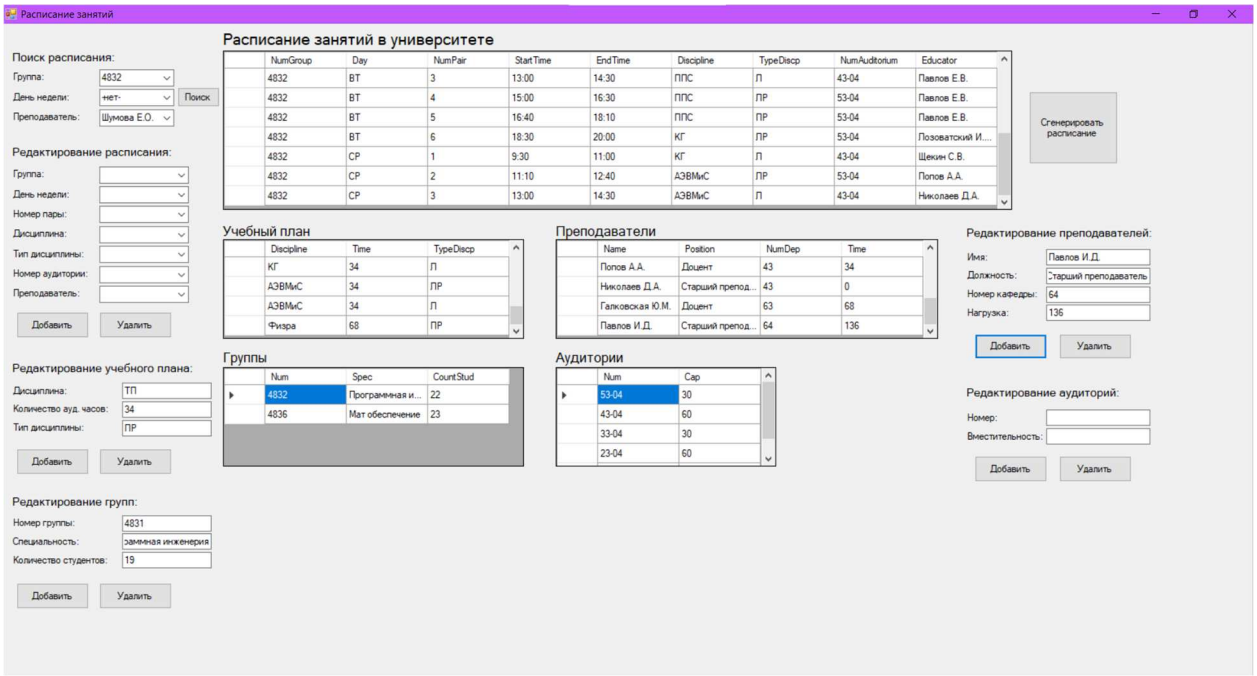
- Добавление записи в учебный план (Физра, ПР):



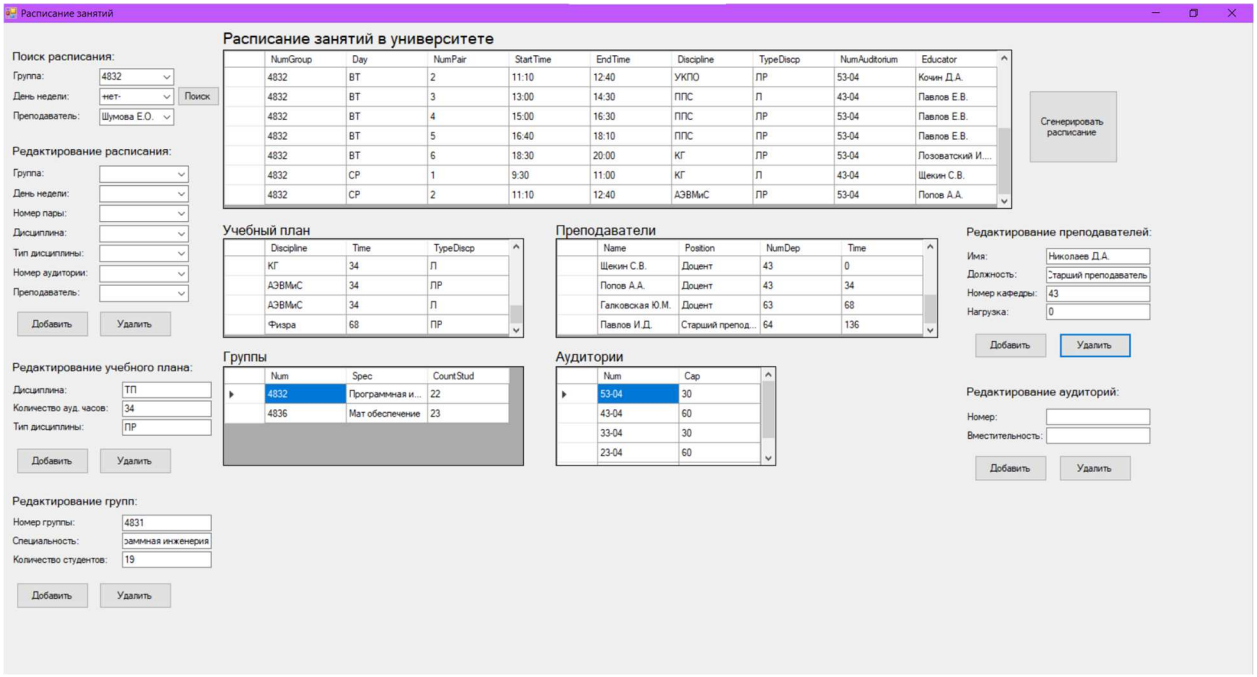
- Удаление записи из учебного плана (ТП, ПР):



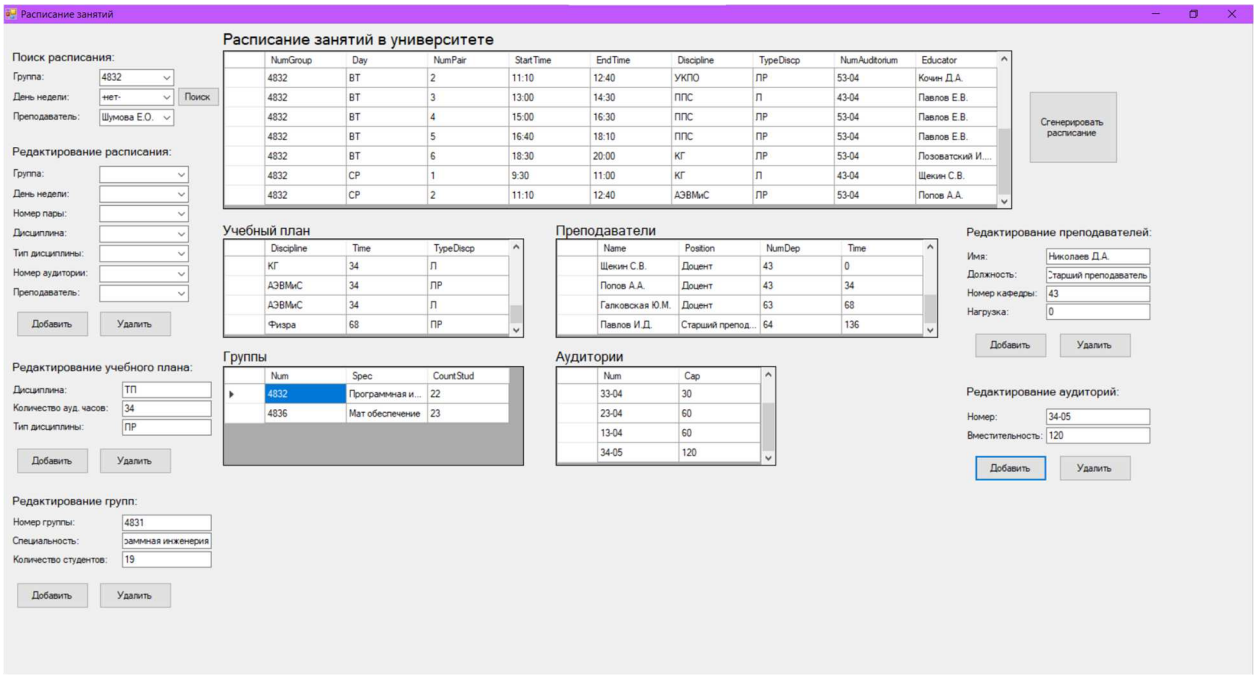
- Добавление преподавателя (Павлов И.Д.):



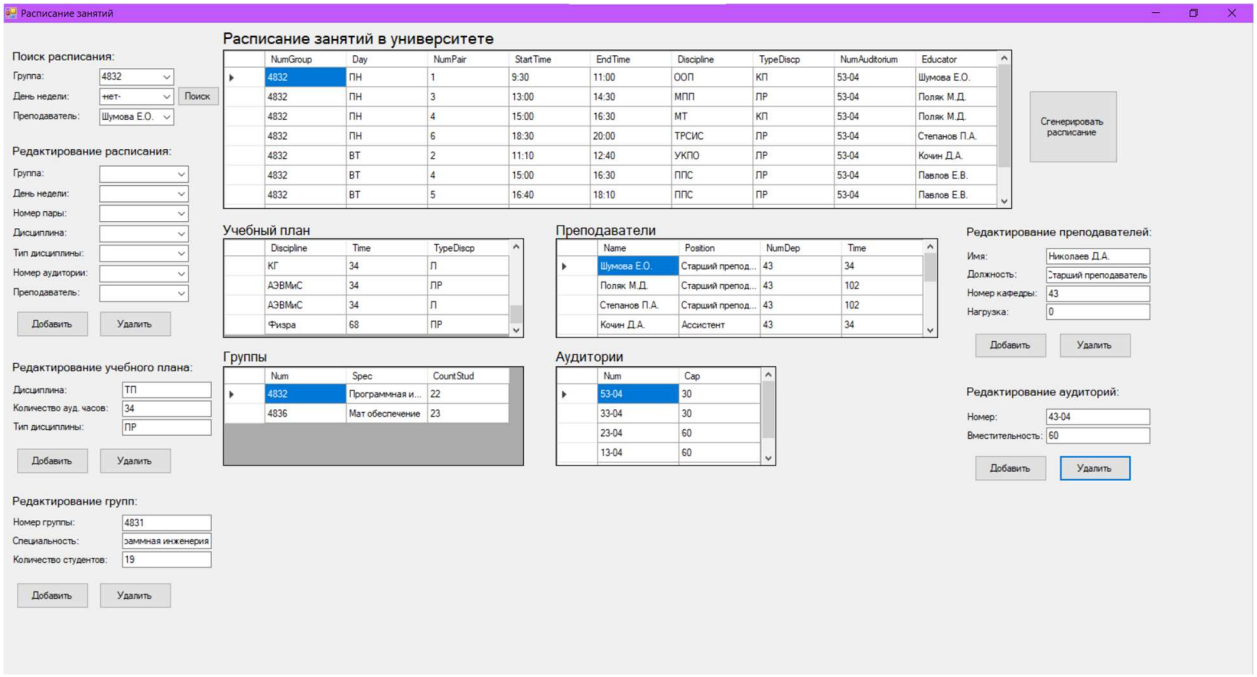
- Удаление преподавателя (Николаев Д.А.):



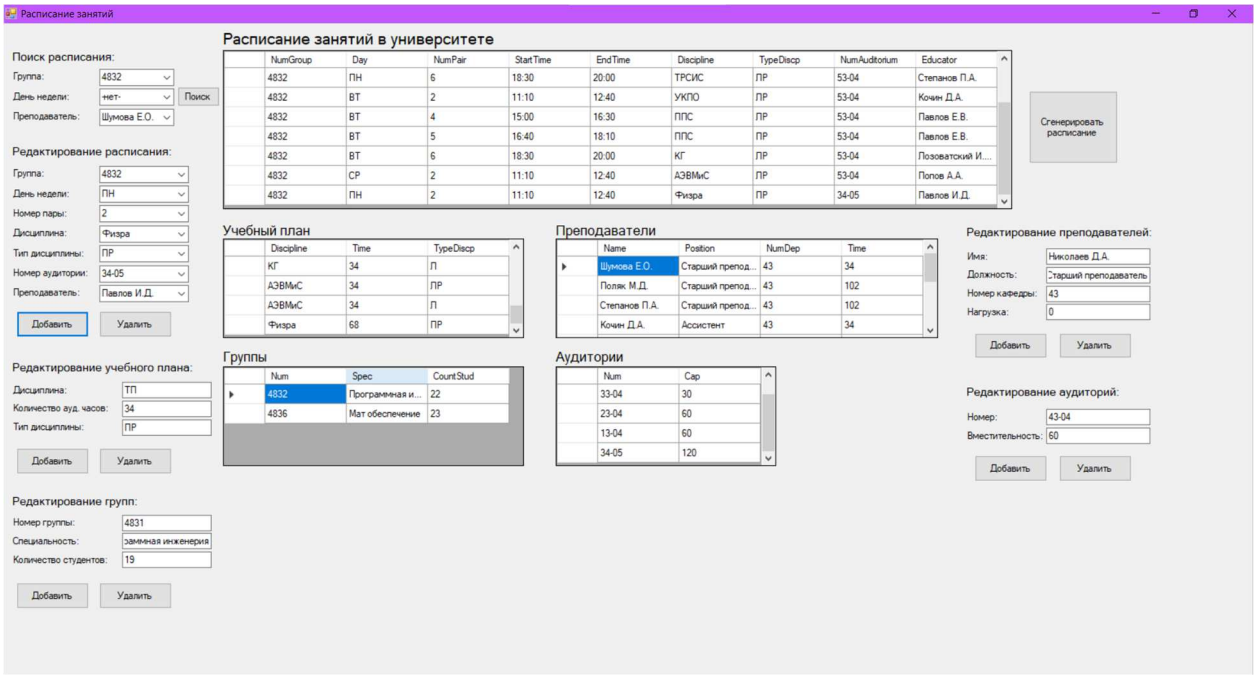
- Добавление аудитории (34-05):



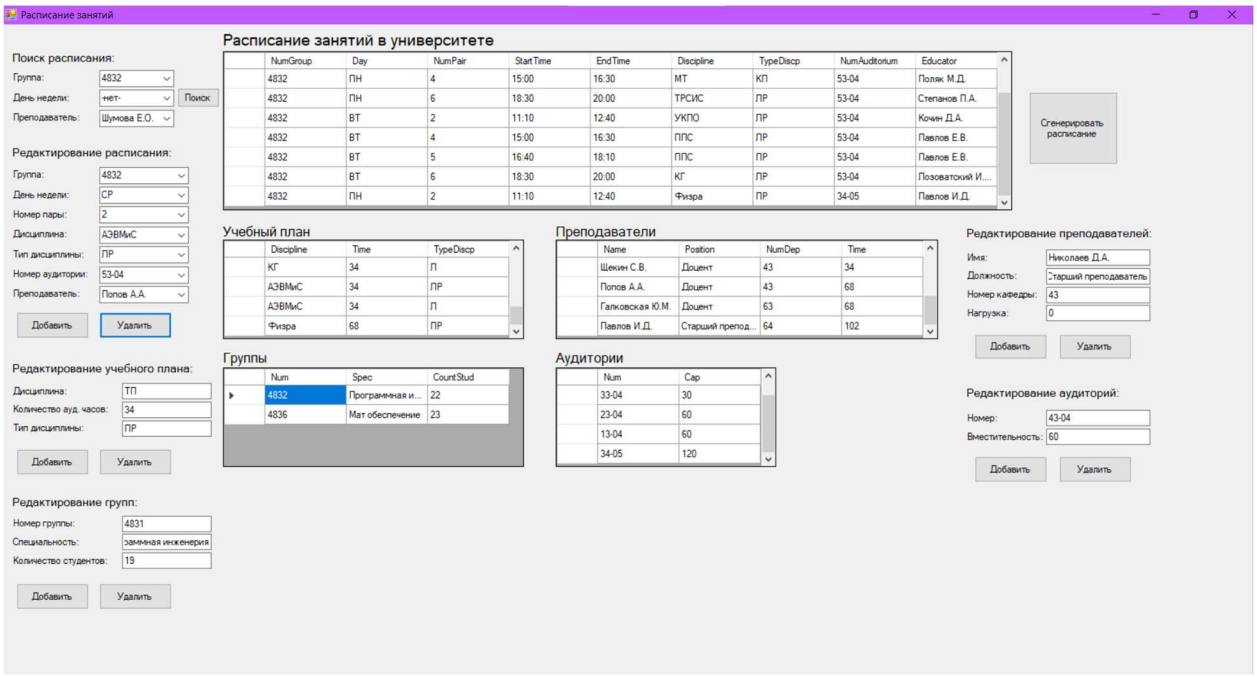
- Удаление аудитории (43-04):



- Добавление записи в расписание (4832, ПН, 2 пара, Физра, ПР, 34-05, Павлов И.Д.):



- Удаление записи в расписании (4832, СР, 2 пара, АЭВМиС, ЛР, 53-04, Попов А.А.):



ЗАКЛЮЧЕНИЕ

В ходе работы над курсовым проектом было разработано оконное приложение для предметной области «Расписание занятий в университете». Приложение было реализовано с применением технологии .NET.

Из достоинств работы можно отметить приятный и дружественный к пользователю интерфейс, а также хорошую скорость работы приложения за счет использования довольно быстрого языка программирования C# и, что самое главное, корректное выполнение всех функций программы.

Также можно дать рекомендацию по совершенствования данной программы, в частности вынесения некоторых элементов формы, служащих для операций добавления и удаления в отдельные вкладки или формы.

В дальнейшем реализованное ПО может быть использовано в качестве средства для работы с расписанием занятий в любом университете.

СПИСОК ИСПОЛЬЗОВАНЫХ ИСТОЧНИКОВ

1. Шумова Е.О. Объектно-ориентированное программирование. Методические указания к выполнению курсового проекта. – СПб: СПбГУАП, 2020, 14 с.
2. https://metanit.com/sharp/tutorial/
3. https://metanit.com/sharp/windowsforms/
4. https://www.bestprog.net/ru/2018/02/17/the-datagridview-control_ru/
5. https://metanit.com/sharp/tutorial/3.4.php

ПРИЛОЖЕНИЕ

Group.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace kursachOOP
{
    class Group : IEquatable<Group>
    {
        public int Num { get; set; }
        public string Spec { get; set; }
        public int CountStud { get; set; }

        public Group(int num, string spec, int countStud)
        {
            Num = num;
            Spec = spec;
            CountStud = countStud;
        }

        public Group() {}

        public override bool Equals(object obj)
        {
            return Equals(obj as Group);
        }

        public bool Equals(Group other)
        {
            return other != null &&
                   Num == other.Num;
        }

        public override int GetHashCode()
        {
            return 159832395 + Num.GetHashCode();
        }
    }
}
```

Educator.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace kursachOOP
{
    class Educator : IEquatable<Educator>
    {
        public string Name { get; set; }
        public string Position { get; set; }
        public int NumDep { get; set; }
        public int Time { get; set; }

        public Educator(string name, string position, int numDep, int time)
        {
            Name = name;
```

```
            Position = position;
            NumDep = numDep;
            Time = time;
        }

        public Educator() { }

        public void Workload(int time)
        {
            Time -= time;
        }

        public override bool Equals(object obj)
        {
            return Equals(obj as Educator);
        }

        public bool Equals(Educator other)
        {
            return other != null &&
                   Name == other.Name;
        }

        public override int GetHashCode()
        {
            return 539060726 + EqualityComparer<string>.Default.GetHashCode(Name);
        }
    }
}
```

Auditorium.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace kursachOOP
{
    class Auditorium : IEquatable<Auditorium>
    {
        public string Num { get; set; }
        public int Cap { get; set; }

        public Auditorium(string num, int cap)
        {
            Num = num;
            Cap = cap;
        }

        public Auditorium() { }

        public override bool Equals(object obj)
        {
            return Equals(obj as Auditorium);
        }

        public bool Equals(Auditorium other)
        {
            return other != null &&
                   Num == other.Num;
        }

        public override int GetHashCode()
```

```
        {
            return 159832395 + EqualityComparer<string>.Default.GetHashCode(Num);
        }
    }
}
```

Curriculum.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace kursachOOP
{
    class Curriculum : IEquatable<Curriculum>
    {
        public string Discipline { get; set; }
        public int Time { get; set; }
        public string TypeDiscp { get; set; }

        public Curriculum(string discipline, int time, string typeDiscp)
        {
            Discipline = discipline;
            Time = time;
            TypeDiscp = typeDiscp;
        }

        public Curriculum() { }

        public override bool Equals(object obj)
        {
            return Equals(obj as Curriculum);
        }

        public bool Equals(Curriculum other)
        {
            return other != null &&
                   Discipline == other.Discipline &&
                   TypeDiscp == other.TypeDiscp;
        }

        public override int GetHashCode()
        {
            int hashCode = 904361603;
            hashCode = hashCode * -1521134295 +
EqualityComparer<string>.Default.GetHashCode(Discipline);
            hashCode = hashCode * -1521134295 +
EqualityComparer<string>.Default.GetHashCode(TypeDiscp);
            return hashCode;
        }
    }
}
```

Schedule.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace kursachOOP
{
```

```
    class Schedule : IEquatable<Schedule>
    {
        public int NumGroup { get; set; }
        public string Day { get; set; }
        public int NumPair { get; set; }
        public string StartTime { get; set; }
        public string EndTime { get; set; }
        public string Discipline { get; set; }
        public string TypeDiscp { get; set; }
        public string NumAuditorium { get; set; }
        public string Educator { get; set; }

        public Schedule(int numGroup, string day, int numPair, string discipline, string
typeDiscp, string numAuditorium, string educator)
        {
            NumGroup = numGroup;
            Day = day;
            NumPair = numPair;
            SetTime();
            Discipline = discipline;
            TypeDiscp = typeDiscp;
            NumAuditorium = numAuditorium;
            Educator = educator;
        }

        public Schedule() { }

        public void SetTime()
        {
            if (NumPair == 1)
            {
                StartTime = "9:30";
                EndTime = "11:00";
            }
            else if (NumPair == 2)
            {
                StartTime = "11:10";
                EndTime = "12:40";
            }
            else if (NumPair == 3)
            {
                StartTime = "13:00";
                EndTime = "14:30";
            }
            else if (NumPair == 4)
            {
                StartTime = "15:00";
                EndTime = "16:30";
            }
            else if (NumPair == 5)
            {
                StartTime = "16:40";
                EndTime = "18:10";
            }
            else if (NumPair == 6)
            {
                StartTime = "18:30";
                EndTime = "20:00";
            }
        }

        public override bool Equals(object obj)
        {
            return Equals(obj as Schedule);
        }
```

```
        public bool Equals(Schedule other)
        {
            return other != null &&
                   NumGroup == other.NumGroup &&
                   Day == other.Day &&
                   NumPair == other.NumPair &&
                   Discipline == other.Discipline &&
                   TypeDiscp == other.TypeDiscp &&
                   NumAuditorium == other.NumAuditorium &&
                   Educator == other.Educator;
        }

        public override int GetHashCode()
        {
            int hashCode = -2112368879;
            hashCode = hashCode * -1521134295 + NumGroup.GetHashCode();
            hashCode = hashCode * -1521134295 +
EqualityComparer<string>.Default.GetHashCode(Day);
            hashCode = hashCode * -1521134295 + NumPair.GetHashCode();
            hashCode = hashCode * -1521134295 +
EqualityComparer<string>.Default.GetHashCode(Discipline);
            hashCode = hashCode * -1521134295 +
EqualityComparer<string>.Default.GetHashCode(TypeDiscp);
            hashCode = hashCode * -1521134295 +
EqualityComparer<string>.Default.GetHashCode(NumAuditorium);
            hashCode = hashCode * -1521134295 +
EqualityComparer<string>.Default.GetHashCode(Educator);
            return hashCode;
        }
    }
}
```

User.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace kursachOOP
{
    class User
    {
        public List<Group> GroupsVector = new List<Group>();
        public List<Educator> EducatorsVector = new List<Educator>();
        public List<Auditorium> AuditoriumsVector = new List<Auditorium>();
        public List<Curriculum> CurriculumsVector = new List<Curriculum>();
        public List<Schedule> ScheduleVector = new List<Schedule>();

        public User() {}

        public bool AddInSchedule(Schedule tempSchedule, bool flag)
        {
            if (ScheduleVector.Count() != 0)
            {
                if (!GroupHaveTime(tempSchedule, flag))
                    return false;
                else if (!AuditoriumFull(tempSchedule, flag))
                    return false;
                else if (!EducatorHavePair(tempSchedule, flag))
                    return false;
                else if (!EducatorHaveTime(tempSchedule, flag))
```

```
                    return false;
                else if (!Lecture(tempSchedule, flag))
                    return false;
                else if (!HaveLection(tempSchedule, flag))
                    return false;
                else
                {
                    EducatorWorkloadLower(tempSchedule);
                    ScheduleVector.Add(tempSchedule);
                    return true;
                }
            }
            else
            {
                EducatorWorkloadLower(tempSchedule);
                ScheduleVector.Add(tempSchedule);
                return true;
            }
        }

        private bool GroupHaveTime(Schedule schedule, bool flag)
        {
            foreach (Schedule tmpSchedule in ScheduleVector)
            {
                if (tmpSchedule.NumGroup == schedule.NumGroup && tmpSchedule.Day ==
schedule.Day && tmpSchedule.NumPair == schedule.NumPair)
                {
                    if (flag == true)
                    {
                        MessageBox.Show("У данной группы уже есть занятие в это время!");
                    }
                    return false;
                }
            }
            return true;
        }

        private bool AuditoriumFull(Schedule schedule, bool flag)
        {
            if (schedule.TypeDiscp == "Л")
            {
                foreach (Schedule tmpSchedule in ScheduleVector)
                {
                    if (tmpSchedule.Day == schedule.Day && tmpSchedule.NumPair ==
schedule.NumPair && tmpSchedule.NumAuditorium == schedule.NumAuditorium)
                    {
                        if (tmpSchedule.Discipline == schedule.Discipline)
                        {
                            foreach (Group tmpGroup in GroupsVector)
                            {
                                if (tmpGroup.Num == schedule.NumGroup)
                                {
                                    foreach (Auditorium tmpAuditorium in
AuditoriumsVector)
                                    {
                                        if (tmpAuditorium.Num == schedule.NumAuditorium)
                                        {
                                            if (tmpGroup.CountStud > tmpAuditorium.Cap)
                                            {
                                                if (flag == true)
                                                {
                                                    MessageBox.Show("К сожалению,
аудитория не может поместить столько людей!");
                                                }
                                                return false;
```

```
                                        }
                                    }
                                }
                            }
                        }
                        else
                        {
                            if (flag == true)
                            {
                                MessageBox.Show("В данной аудитории уже проводится
занятие по другой дисциплине!");
                            }
                            return false;
                        }
                    }
                }
            }
            else
            {
                foreach (Schedule tmpSchedule in ScheduleVector)
                {
                    if (tmpSchedule.Day == schedule.Day && tmpSchedule.NumPair ==
schedule.NumPair && tmpSchedule.NumAuditorium == schedule.NumAuditorium)
                    {
                        if (flag == true)
                        {
                            MessageBox.Show("В данной аудитории уже проводится
занятие!");
                        }
                        return false;
                    }
                }
            }
            foreach (Group tmpGroup in GroupsVector)
            {
                if (tmpGroup.Num == schedule.NumGroup)
                {
                    foreach (Auditorium tmpAuditorium in AuditoriumsVector)
                    {
                        if (tmpAuditorium.Num == schedule.NumAuditorium)
                        {
                            if (tmpGroup.CountStud > tmpAuditorium.Cap)
                            {
                                if (flag == true)
                                {
                                    MessageBox.Show("Данная аудитория слишком
маленькая!");
                                }
                                return false;
                            }
                        }
                    }
                }
            }
            return true;
        }

        private bool EducatorHavePair(Schedule schedule, bool flag)
        {
            foreach (Schedule tmpSchedule in ScheduleVector)
            {
                if (tmpSchedule.Day == schedule.Day && tmpSchedule.NumPair ==
schedule.NumPair && tmpSchedule.Educator == schedule.Educator)
                {
```

```
                    if (tmpSchedule.Discipline == schedule.Discipline &&
schedule.TypeDiscp == "Л")
                    {
                        return true;
                    }
                    else
                    {
                        if (flag == true)
                        {
                            MessageBox.Show("Преподаватель занят в это время!");
                        }
                        return false;
                    }
                }
            }
            return true;
        }

        private bool EducatorHaveTime(Schedule schedule, bool flag)
        {
            foreach (Schedule tmpSchedule in ScheduleVector)
            {
                if (tmpSchedule.Discipline == schedule.Discipline &&
tmpSchedule.TypeDiscp == schedule.TypeDiscp && schedule.TypeDiscp == "Л")
                    return true;
            }

            foreach (Educator tmpEducator in EducatorsVector)
            {
                if (tmpEducator.Name == schedule.Educator)
                {
                    foreach (Curriculum tmpCurriculum in CurriculumsVector)
                    {
                        if (tmpCurriculum.Discipline == schedule.Discipline &&
tmpCurriculum.TypeDiscp == schedule.TypeDiscp )
                        {
                            if ((tmpCurriculum.Time == 68) && ((tmpEducator.Time -
tmpCurriculum.Time / 2) < 0))
                            {
                                if (flag == true)
                                {
                                    MessageBox.Show("Нагрузка преподавателя будет
превышена!");
                                }
                                return false;
                            }
                            else if ((tmpCurriculum.Time == 34) && (tmpEducator.Time -
tmpCurriculum.Time) < 0)
                            {
                                if (flag == true)
                                {
                                    MessageBox.Show("Нагрузка преподавателя будет
превышена!");
                                }
                                return false;
                            }
                            else
                                return true;
                        }
                    }
                }
            }
            return true;
        }
```

```
        private bool Lecture(Schedule schedule, bool flag)
        {
            if (schedule.TypeDiscp == "Л")
            {
                foreach (Schedule tmpSchedule in ScheduleVector)
                {
                    if (tmpSchedule.Discipline == schedule.Discipline &&
tmpSchedule.TypeDiscp == schedule.TypeDiscp)
                    {
                        if (tmpSchedule.Day != schedule.Day)
                        {
                            if (flag == true)
                            {
                                MessageBox.Show("Лекции по одному предмету должны
проходить у всех групп в один день!");
                            }
                            return false;
                        }
                        else if (tmpSchedule.NumPair != schedule.NumPair)
                        {
                            if (flag == true)
                            {
                                MessageBox.Show("Лекции по одному предмету должны
проходить у всех групп в одно время!");
                            }
                            return false;
                        }
                        else if (tmpSchedule.NumAuditorium != schedule.NumAuditorium)
                        {
                            if (flag == true)
                            {
                                MessageBox.Show("Лекции по одному предмету должны
проходить у всех групп в одной аудитории!");
                            }
                            return false;
                        }
                        else if (tmpSchedule.Educator != schedule.Educator)
                        {
                            if (flag == true)
                            {
                                MessageBox.Show("Лекции по одному предмету должны
проводиться у всех групп одним преподавателем!");
                            }
                            return false;
                        }
                        else
                        {
                            return true;
                        }
                    }
                }
                return true;
            }
            else
            {
                return true;
            }
        }

        private bool HaveLection(Schedule schedule, bool flag)
        {
            if (schedule.TypeDiscp != "Л")
            {
                foreach (Schedule tmpSchedule in ScheduleVector)
                {
```

```
                    if (tmpSchedule.TypeDiscp == "Л" && tmpSchedule.Day == schedule.Day
&& tmpSchedule.NumPair == schedule.NumPair)
                    {
                        if (flag == true)
                        {
                            MessageBox.Show("Лекции по одному предмету должны проводиться
у всех групп в одно время!");
                        }
                        return false;
                    }
                }
                return true;
            }
            else
            {
                return true;
            }
        }

        private void EducatorWorkloadLower(Schedule schedule)
        {
            foreach (Educator tmpEducator in EducatorsVector)
            {
                if (tmpEducator.Name == schedule.Educator)
                {
                    foreach (Curriculum tmpCurriculum in CurriculumsVector)
                    {
                        if (tmpCurriculum.Discipline == schedule.Discipline &&
tmpCurriculum.TypeDiscp == schedule.TypeDiscp && tmpCurriculum.TypeDiscp == "Л")
                        {
                            foreach (Schedule tmpSchedule in ScheduleVector)
                            {
                                if (tmpSchedule.Discipline == tmpCurriculum.Discipline &&
tmpSchedule.TypeDiscp == tmpCurriculum.TypeDiscp)
                                    return;
                            }
                            if (tmpCurriculum.Time == 68)
                            {
                                tmpEducator.Workload(tmpCurriculum.Time / 2);
                                return;
                            }
                            else
                            {
                                tmpEducator.Workload(tmpCurriculum.Time);
                                return;
                            }
                        }
                        else if (tmpCurriculum.Discipline == schedule.Discipline &&
tmpCurriculum.TypeDiscp == schedule.TypeDiscp)
                        {
                            if (tmpCurriculum.Time == 68)
                            {
                                tmpEducator.Workload(tmpCurriculum.Time / 2);
                                return;
                            }
                            else
                            {
                                tmpEducator.Workload(tmpCurriculum.Time);
                                return;
                            }
                        }
                    }
                }
            }
        }
```

```
        public bool DeleteInSchedule(Schedule schedule)
        {
            if (!ScheduleVector.Remove(schedule))
            {
                MessageBox.Show("Такой записи нет в расписании!");
                return false;
            }
            else
            {
                EducatorWorkloadUpper(schedule);
                return true;
            }
        }

        private void EducatorWorkloadUpper(Schedule schedule)
        {
            foreach (Schedule tmpSchedule in ScheduleVector)
            {
                if (tmpSchedule.Discipline == schedule.Discipline &&
tmpSchedule.TypeDiscp == schedule.TypeDiscp && schedule.TypeDiscp == "Л")
                {
                    return;
                }
            }
            foreach (Educator educator in EducatorsVector)
            {
                if (schedule.Educator == educator.Name)
                {
                    educator.Time += 34;
                    return;
                }
            }
        }


        public bool AddInGroups(int num, string spec, int countStud)
        {
            Group group = new Group(num, spec, countStud);
            if (GroupsVector.Count() != 0)
            {
                foreach (Group tmpGroup in GroupsVector)
                {
                    if (tmpGroup.Equals(group))
                    {
                        MessageBox.Show("Такая группа уже имеется!");
                        return false;
                    }
                }
                GroupsVector.Add(group);
                return true;
            }
            else
            {
                GroupsVector.Add(group);
                return true;
            }
        }

        public bool DeleteInGroups(Group group)
        {
            if (!GroupsVector.Remove(group))
            {
                MessageBox.Show("Такой группы нет!");
                return false;
```

```
        }
        else
        {
            List<Schedule> list = new List<Schedule>();
            foreach (Schedule tmpSchedule in ScheduleVector)
            {
                list.Add(tmpSchedule);
            }
            foreach (Schedule schedule in list)
            {
                if (schedule.NumGroup == group.Num)
                {
                    ScheduleVector.Remove(schedule);
                    EducatorWorkloadUpper(schedule);
                }
            }
            return true;
        }
    }

    public bool AddInEducators(string name, string position, int numDep, int time)
    {
        Educator educator = new Educator(name, position, numDep, time);
        if (EducatorsVector.Count() != 0)
        {
            foreach (Educator tmpEducator in EducatorsVector)
            {
                if (tmpEducator.Equals(educator))
                {
                    MessageBox.Show("Такой преподаватель уже существует!");
                    return false;
                }
            }
            EducatorsVector.Add(educator);
            return true;
        }
        else
        {
            EducatorsVector.Add(educator);
            return true;
        }
    }

    public bool DeleteInEducators(Educator educator)
    {
        if (!EducatorsVector.Remove(educator))
        {
            MessageBox.Show("Такого преподавателя нет!");
            return false;
        }
        else
        {
            List<Schedule> list = new List<Schedule>();
            foreach (Schedule tmpSchedule in ScheduleVector)
            {
                list.Add(tmpSchedule);
            }
            foreach (Schedule schedule in list)
            {
                if (schedule.Educator == educator.Name)
                {
                    ScheduleVector.Remove(schedule);
                }
            }
            return true;
```

```
    }
}

public bool AddInAuditoriums(string num, int cap)
{
    Auditorium auditorium = new Auditorium(num, cap);
    if (AuditoriumsVector.Count() != 0)
    {
        foreach (Auditorium tmpAuditorium in AuditoriumsVector)
        {
            if (tmpAuditorium.Equals(auditorium))
            {
                MessageBox.Show("Такая аудитория уже существует!");
                return false;
            }
        }
        AuditoriumsVector.Add(auditorium);
        return true;
    }
    else
    {
        AuditoriumsVector.Add(auditorium);
        return true;
    }
}

public bool DeleteInAuditoriums(Auditorium auditorium)
{
    if (!AuditoriumsVector.Remove(auditorium))
    {
        MessageBox.Show("Такой аудитории нет!");
        return false;
    }
    else
    {
        List<Schedule> list = new List<Schedule>();
        foreach (Schedule tmpSchedule in ScheduleVector)
        {
            list.Add(tmpSchedule);
        }
        foreach (Schedule schedule in list)
        {
            if (schedule.NumAuditorium == auditorium.Num)
            {
                ScheduleVector.Remove(schedule);
                EducatorWorkloadUpper(schedule);
            }
        }
        return true;
    }
}

public bool AddInCurriculum(string discipline, int time, string typeDiscp)
{
    Curriculum curriculum = new Curriculum(discipline, time, typeDiscp);
    if (CurriculumsVector.Count() != 0)
    {
        foreach (Curriculum tmpCurriculum in CurriculumsVector)
        {
            if (tmpCurriculum.Equals(curriculum))
            {
                MessageBox.Show("Такая запись в учебном плане уже существует!");
                return false;
            }
        }
```

```
                CurriculumsVector.Add(curriculum);
                return true;
            }
            else
            {
                CurriculumsVector.Add(curriculum);
                return true;
            }
        }

        public bool DeleteInCurriculum(Curriculum curriculum)
        {
            if (!CurriculumsVector.Remove(curriculum))
            {
                MessageBox.Show("Такой записи в учебном плане нет!");
                return false;
            }
            else
            {
                List<Schedule> list = new List<Schedule>();
                foreach (Schedule tmpSchedule in ScheduleVector)
                {
                    list.Add(tmpSchedule);
                }
                foreach (Schedule schedule in list)
                {
                    if (schedule.Discipline == curriculum.Discipline &&
schedule.TypeDiscp == curriculum.TypeDiscp)
                    {
                        ScheduleVector.Remove(schedule);
                        EducatorWorkloadUpper(schedule);
                    }
                }
                return true;
            }
        }

        public List<Schedule> FindByGroup(int numGroup)
        {
            List<Schedule> ScheduleByGroupVector = new List<Schedule>();
            foreach (Schedule schedule in ScheduleVector)
            {
                if (schedule.NumGroup == numGroup)
                {
                    ScheduleByGroupVector.Add(schedule);
                }
            }
            return ScheduleByGroupVector;
        }

        public List<Schedule> FindByEducator(string educator)
        {
            List<Schedule> ScheduleByEducatorVector = new List<Schedule>();
            foreach (Schedule schedule in ScheduleVector)
            {
                if (schedule.Educator == educator)
                {
                    ScheduleByEducatorVector.Add(schedule);
                }
            }
            return ScheduleByEducatorVector;
        }

        public List<Schedule> FindByDay(string day)
        {
```

```
            List<Schedule> ScheduleByDayVector = new List<Schedule>();
            foreach (Schedule schedule in ScheduleVector)
            {
                if (schedule.Day == day)
                {
                    ScheduleByDayVector.Add(schedule);
                }
            }
            return ScheduleByDayVector;
        }

        public List<Schedule> FindByGroupAndEducator(int numGroup, string educator)
        {
            List<Schedule> ScheduleByGroupAndEducatorVector = new List<Schedule>();
            foreach (Schedule schedule in ScheduleVector)
            {
                if (schedule.NumGroup == numGroup && schedule.Educator == educator)
                {
                    ScheduleByGroupAndEducatorVector.Add(schedule);
                }
            }
            return ScheduleByGroupAndEducatorVector;
        }

        public List<Schedule> FindByGroupAndDay(int numGroup, string day)
        {
            List<Schedule> ScheduleByGroupAndDayVector = new List<Schedule>();
            foreach (Schedule schedule in ScheduleVector)
            {
                if (schedule.NumGroup == numGroup && schedule.Day == day)
                {
                    ScheduleByGroupAndDayVector.Add(schedule);
                }
            }
            return ScheduleByGroupAndDayVector;
        }

        public List<Schedule> FindByEducatorAndDay(string educator, string day)
        {
            List<Schedule> ScheduleByEducatorAndDayVector = new List<Schedule>();
            foreach (Schedule schedule in ScheduleVector)
            {
                if (schedule.Educator == educator && schedule.Day == day)
                {
                    ScheduleByEducatorAndDayVector.Add(schedule);
                }
            }
            return ScheduleByEducatorAndDayVector;
        }

        public void GenerateSchedule()
        {
            GroupsVector.Add(new Group(4832, "Программная инженерия", 22));
            GroupsVector.Add(new Group(4831, "Программная инженерия", 19));

            EducatorsVector.Add(new Educator("Шумова Е.О.", "Старший преподаватель", 43,
68));
            EducatorsVector.Add(new Educator("Поляк М.Д.", "Старший преподаватель", 43,
170));
            EducatorsVector.Add(new Educator("Степанов П.А.", "Старший преподаватель",
43, 136));
            EducatorsVector.Add(new Educator("Кочин Д.А.", "Ассистент", 43, 68));
            EducatorsVector.Add(new Educator("Павлов Е.В.", "Старший преподаватель", 43,
170));
            EducatorsVector.Add(new Educator("Лозоватский И.М.", "Ассистент", 43, 102));
```

```
            EducatorsVector.Add(new Educator("Щекин С.В.", "Доцент", 43, 34));
            EducatorsVector.Add(new Educator("Попов А.А.", "Доцент", 43, 68));
            EducatorsVector.Add(new Educator("Николаев Д.А.", "Старший преподаватель",
43, 34));
            EducatorsVector.Add(new Educator("Галковская Ю.М.", "Доцент", 63, 68));

            AuditoriumsVector.Add(new Auditorium("53-04", 30));
            AuditoriumsVector.Add(new Auditorium("43-04", 60));
            AuditoriumsVector.Add(new Auditorium("33-04", 30));
            AuditoriumsVector.Add(new Auditorium("23-04", 60));
            AuditoriumsVector.Add(new Auditorium("13-04", 60));

            CurriculumsVector.Add(new Curriculum("ООП", 34, "КП"));
            CurriculumsVector.Add(new Curriculum("МПП", 34, "Л"));
            CurriculumsVector.Add(new Curriculum("МПП", 34, "ЛР"));
            CurriculumsVector.Add(new Curriculum("МТ", 34, "КП"));
            CurriculumsVector.Add(new Curriculum("ТРСИС", 34, "Л"));
            CurriculumsVector.Add(new Curriculum("ТРСИС", 34, "ЛР"));
            CurriculumsVector.Add(new Curriculum("УКПО", 34, "Л"));
            CurriculumsVector.Add(new Curriculum("УКПО", 34, "ЛР"));
            CurriculumsVector.Add(new Curriculum("ППС", 34, "Л"));
            CurriculumsVector.Add(new Curriculum("ППС", 34, "ЛР"));
            CurriculumsVector.Add(new Curriculum("ППС", 34, "ПР"));
            CurriculumsVector.Add(new Curriculum("КГ", 34, "ЛР"));
            CurriculumsVector.Add(new Curriculum("КГ", 34, "Л"));
            CurriculumsVector.Add(new Curriculum("АЭВМиС", 34, "ЛР"));
            CurriculumsVector.Add(new Curriculum("АЭВМиС", 34, "Л"));
            CurriculumsVector.Add(new Curriculum("ТП", 34, "ПР"));

            List<string> Days = new List<string>
            {
                "ПН",
                "ВТ",
                "СР",
                "ЧТ",
                "ПТ",
                "СБ"
            };

            List<int> NumPairs = new List<int>
            {
                1,
                2,
                3,
                4,
                5,
                6
            };

            foreach (Group group in GroupsVector)
            {
                foreach (Curriculum curriculum in CurriculumsVector)
                {
                    bool f = false;
                    Schedule schedule = new Schedule
                    {
                        NumGroup = group.Num,
                        Discipline = curriculum.Discipline,
                        TypeDiscp = curriculum.TypeDiscp
                    };
                    if (schedule.Discipline == "ООП")
                        schedule.Educator = "Шумова Е.О.";
                    else if (schedule.Discipline == "МПП" || schedule.Discipline == "МТ")
                        schedule.Educator = "Поляк М.Д.";
```

```
                    else if (schedule.Discipline == "ТРСИС" || (schedule.Discipline ==
"УКПО" && schedule.TypeDiscp == "Л"))
                        schedule.Educator = "Степанов П.А.";
                    else if (schedule.Discipline == "УКПО" && schedule.TypeDiscp == "ЛР")
                        schedule.Educator = "Кочин Д.А.";
                    else if (schedule.Discipline == "ППС")
                        schedule.Educator = "Павлов Е.В.";
                    else if (schedule.Discipline == "АЭВМиС" && schedule.TypeDiscp ==
"Л")
                        schedule.Educator = "Николаев Д.А.";
                    else if (schedule.Discipline == "АЭВМиС" && schedule.TypeDiscp ==
"ЛР")
                        schedule.Educator = "Попов А.А.";
                    else if (schedule.Discipline == "КГ" && schedule.TypeDiscp == "Л")
                        schedule.Educator = "Щекин С.В.";
                    else if (schedule.Discipline == "КГ" && schedule.TypeDiscp == "ЛР")
                        schedule.Educator = "Лозоватский И.М.";
                    else if (schedule.Discipline == "ТП")
                        schedule.Educator = "Галковская Ю.М.";
                    foreach (string day in Days)
                    {
                        schedule.Day = day;
                        foreach (int numPair in NumPairs)
                        {
                            schedule.NumPair = numPair;
                            schedule.SetTime();
                            foreach (Auditorium auditorium in AuditoriumsVector)
                            {
                                if (schedule.TypeDiscp == "Л" && auditorium.Cap < 41)
                                    continue;
                                else
                                {
                                    schedule.NumAuditorium = auditorium.Num;
                                    if (AddInSchedule(schedule, false))
                                    {
                                        f = true;
                                    }
                                }
                                if (f == true)
                                {
                                    break;
                                }
                            }
                            if (f == true)
                            {
                                break;
                            }
                        }
                        if (f == true)
                        {
                            break;
                        }
                    }
                }
            }
        }
    }
}
```

FormSchedule.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
```

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

namespace kursachOOP
{
    public partial class FormSchedule : Form
    {
        User user = new User();
        public FormSchedule()
        {
            InitializeComponent();
        }

        private void FormSchedule_Load(object sender, EventArgs e)
        {

        }

        private void generation_Click(object sender, EventArgs e)
        {
            user.GenerateSchedule();
            dataSchedule.DataSource = null;
            dataSchedule.DataSource = user.ScheduleVector;

            dataCurriculum.DataSource = null;
            dataCurriculum.DataSource = user.CurriculumsVector;

            dataEducators.DataSource = null;
            dataEducators.DataSource = user.EducatorsVector;

            dataGroups.DataSource = null;
            dataGroups.DataSource = user.GroupsVector;

            dataAuditoriums.DataSource = null;
            dataAuditoriums.DataSource = user.AuditoriumsVector;
        }

        private void findGroupUpdate(object sender, EventArgs e)
        {
            List<string> list = user.GroupsVector.Select(g => g.Num.ToString()).ToList();
            list.Add("-нет-");
            ((ComboBox)sender).DataSource = list;
        }

        private void findEducatorUpdate(object sender, EventArgs e)
        {
            List<string> list = user.EducatorsVector.Select(g =>
g.Name.ToString()).ToList();
            list.Add("-нет-");
            ((ComboBox)sender).DataSource = list;
        }

        private void groupsComboUpdate(object sender, EventArgs e)
        {
            ((ComboBox)sender).DataSource = user.GroupsVector.Select(g =>
g.Num.ToString()).ToList();
        }

        private void disciplinesComboUpdate(object sender, EventArgs e)
        {
            List<string> listDisciplineFalse = user.CurriculumsVector.Select(g =>
g.Discipline.ToString()).ToList();
```

```
            List<string> listDisciplineTrue = new List<string>();
            foreach (string tmpDisciplineFalse in listDisciplineFalse)
            {
                bool flag = true;
                if (listDisciplineTrue.Count() != 0)
                {
                    foreach (string tmpDisiplineTrue in listDisciplineTrue)
                    {
                        if (tmpDisciplineFalse == tmpDisiplineTrue)
                            flag = false;
                    }
                    if (flag)
                        listDisciplineTrue.Add(tmpDisciplineFalse);
                }
                else
                    listDisciplineTrue.Add(tmpDisciplineFalse);
            }
            ((ComboBox)sender).DataSource = listDisciplineTrue;
        }

        private void auditoriumsComboUpdate(object sender, EventArgs e)
        {
            ((ComboBox)sender).DataSource = user.AuditoriumsVector.Select(g =>
g.Num.ToString()).ToList();
        }

        private void educatorsComboUpdate(object sender, EventArgs e)
        {
            ((ComboBox)sender).DataSource = user.EducatorsVector.Select(g =>
g.Name.ToString()).ToList();
        }

        private void addSchedule_Click(object sender, EventArgs e)
        {
            int Group = int.Parse(GroupsAD.SelectedItem.ToString());
            string Day = DayAD.SelectedItem.ToString();
            int NumPair = int.Parse(NumPairAD.SelectedItem.ToString());
            string Discipline = DisciplineAD.SelectedItem.ToString();
            string TypeDiscp = TypeDiscpAD.SelectedItem.ToString();
            string NumAuditorium = NumAudAD.SelectedItem.ToString();
            string Educator = EducatorAD.SelectedItem.ToString();
            Schedule schedule = new Schedule(Group, Day, NumPair, Discipline, TypeDiscp,
NumAuditorium, Educator);
            if (user.AddInSchedule(schedule, true))
            {
                dataSchedule.DataSource = null;
                dataSchedule.DataSource = user.ScheduleVector;

                dataEducators.DataSource = null;
                dataEducators.DataSource = user.EducatorsVector;
            }
        }

        private void deleteSchedule_Click(object sender, EventArgs e)
        {
            int Group = int.Parse(GroupsAD.SelectedItem.ToString());
            string Day = DayAD.SelectedItem.ToString();
            int NumPair = int.Parse(NumPairAD.SelectedItem.ToString());
            string Discipline = DisciplineAD.SelectedItem.ToString();
            string TypeDiscp = TypeDiscpAD.SelectedItem.ToString();
            string NumAuditorium = NumAudAD.SelectedItem.ToString();
            string Educator = EducatorAD.SelectedItem.ToString();
            Schedule schedule = new Schedule(Group, Day, NumPair, Discipline, TypeDiscp,
NumAuditorium, Educator);
            if (user.DeleteInSchedule(schedule))
```

```
            {
                dataSchedule.DataSource = null;
                dataSchedule.DataSource = user.ScheduleVector;

                dataEducators.DataSource = null;
                dataEducators.DataSource = user.EducatorsVector;
            }
        }

        private void find_Click(object sender, EventArgs e)
        {
            if (findGroup.SelectedItem.ToString() == "-нет-" &&
findDay.SelectedItem.ToString() == "-нет-" && findEducator.SelectedItem.ToString() == "-
нет-")
            {
                dataSchedule.DataSource = user.ScheduleVector;
            }
            else if (findGroup.SelectedItem.ToString() != "-нет-" &&
findDay.SelectedItem.ToString() == "-нет-" && findEducator.SelectedItem.ToString() == "-
нет-")
            {
                dataSchedule.DataSource =
user.FindByGroup(int.Parse(findGroup.SelectedItem.ToString()));
            }
            else if (findGroup.SelectedItem.ToString() == "-нет-" &&
findDay.SelectedItem.ToString() != "-нет-" && findEducator.SelectedItem.ToString() == "-
нет-")
            {
                dataSchedule.DataSource =
user.FindByDay(findDay.SelectedItem.ToString());
            }
            else if (findGroup.SelectedItem.ToString() == "-нет-" &&
findDay.SelectedItem.ToString() == "-нет-" && findEducator.SelectedItem.ToString() != "-
нет-")
            {
                dataSchedule.DataSource =
user.FindByEducator(findEducator.SelectedItem.ToString());
            }
            else if (findGroup.SelectedItem.ToString() != "-нет-" &&
findDay.SelectedItem.ToString() != "-нет-" && findEducator.SelectedItem.ToString() == "-
нет-")
            {
                dataSchedule.DataSource =
user.FindByGroupAndDay(int.Parse(findGroup.SelectedItem.ToString()),
findDay.SelectedItem.ToString());
            }
            else if (findGroup.SelectedItem.ToString() != "-нет-" &&
findDay.SelectedItem.ToString() == "-нет-" && findEducator.SelectedItem.ToString() != "-
нет-")
            {
                dataSchedule.DataSource =
user.FindByGroupAndEducator(int.Parse(findGroup.SelectedItem.ToString()),
findEducator.SelectedItem.ToString());
            }
            else if (findGroup.SelectedItem.ToString() == "-нет-" &&
findDay.SelectedItem.ToString() != "-нет-" && findEducator.SelectedItem.ToString() != "-
нет-")
            {
                dataSchedule.DataSource =
user.FindByEducatorAndDay(findEducator.SelectedItem.ToString(),
findDay.SelectedItem.ToString());
            }
            else
            {
                MessageBox.Show("Нельзя найти расписание по всем трем параметрам!");
```

```
    }
}

private void addCurriculum_Click(object sender, EventArgs e)
{
    string Discipline = discpCurric.Text;
    int Time = int.Parse(timeCurric.Text);
    string TypeDiscp = typeDiscpCurric.Text;
    if (user.AddInCurriculum(Discipline, Time, TypeDiscp))
    {
        dataCurriculum.DataSource = null;
        dataCurriculum.DataSource = user.CurriculumsVector;
    }
}

private void deleteCurriculum_Click(object sender, EventArgs e)
{
    string Discipline = discpCurric.Text;
    int Time = int.Parse(timeCurric.Text);
    string TypeDiscp = typeDiscpCurric.Text;
    Curriculum curriculum = new Curriculum(Discipline, Time, TypeDiscp);
    if (user.DeleteInCurriculum(curriculum))
    {
        dataCurriculum.DataSource = null;
        dataCurriculum.DataSource = user.CurriculumsVector;

        dataSchedule.DataSource = null;
        dataSchedule.DataSource = user.ScheduleVector;

        dataEducators.DataSource = null;
        dataEducators.DataSource = user.EducatorsVector;
    }
}

private void addGroups_Click(object sender, EventArgs e)
{
    int Num = int.Parse(numGroupGroup.Text);
    string Spec = specGroup.Text;
    int CountStud = int.Parse(countStudGroup.Text);
    if (user.AddInGroups(Num, Spec, CountStud))
    {
        dataGroups.DataSource = null;
        dataGroups.DataSource = user.GroupsVector;
    }
}

private void deleteGroups_Click(object sender, EventArgs e)
{
    int Num = int.Parse(numGroupGroup.Text);
    string Spec = specGroup.Text;
    int CountStud = int.Parse(countStudGroup.Text);
    Group group = new Group(Num, Spec, CountStud);
    if (user.DeleteInGroups(group))
    {
        dataGroups.DataSource = null;
        dataGroups.DataSource = user.GroupsVector;

        dataSchedule.DataSource = null;
        dataSchedule.DataSource = user.ScheduleVector;

        dataEducators.DataSource = null;
        dataEducators.DataSource = user.EducatorsVector;
    }
}
```

```
        private void addEducators_Click(object sender, EventArgs e)
        {
            string Name = nameEducator.Text;
            string Position = positionEducator.Text;
            int NumDep = int.Parse(numDepEducator.Text);
            int Time = int.Parse(timeEducator.Text);
            if (user.AddInEducators(Name, Position, NumDep, Time))
            {
                dataEducators.DataSource = null;
                dataEducators.DataSource = user.EducatorsVector;
            }
        }

        private void deleteEducators_Click(object sender, EventArgs e)
        {
            string Name = nameEducator.Text;
            string Position = positionEducator.Text;
            int NumDep = int.Parse(numDepEducator.Text);
            int Time = int.Parse(timeEducator.Text);
            Educator educator = new Educator(Name, Position, NumDep, Time);
            if (user.DeleteInEducators(educator))
            {
                dataEducators.DataSource = null;
                dataEducators.DataSource = user.EducatorsVector;

                dataSchedule.DataSource = null;
                dataSchedule.DataSource = user.ScheduleVector;
            }
        }

        private void addAuditoriums_Click(object sender, EventArgs e)
        {
            string Num = numAudAud.Text;
            int Cap = int.Parse(capAud.Text);
            if (user.AddInAuditoriums(Num, Cap))
            {
                dataAuditoriums.DataSource = null;
                dataAuditoriums.DataSource = user.AuditoriumsVector;
            }
        }

        private void deleteAuditoriums_Click(object sender, EventArgs e)
        {
            string Num = numAudAud.Text;
            int Cap = int.Parse(capAud.Text);
            Auditorium auditorium = new Auditorium(Num, Cap);
            if (user.DeleteInAuditoriums(auditorium))
            {
                dataAuditoriums.DataSource = null;
                dataAuditoriums.DataSource = user.AuditoriumsVector;

                dataSchedule.DataSource = null;
                dataSchedule.DataSource = user.ScheduleVector;

                dataEducators.DataSource = null;
                dataEducators.DataSource = user.EducatorsVector;
            }
        }
    }
}
```