МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра автоматизированных систем управления



**ОТЧЁТ**

**по курсовой работе**

**по теме:** Разработка фрагмента информационной системы ВУЗа.

**по дисциплине:** Программирование.

Вариант **№28**

Выполнил работу:

Студент гр. АП–426, АВТФ

Ширинов Дмитрий Алексеевич

Александровна

«17» декабря 2025г

Проверил работу:

Эстрайх Игорь Викторович

«\_\_\_» \_\_\_\_\_\_ 20\_\_г

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

# **Оглавление**

|  |  |
| --- | --- |
| Постановка задачи | 3 |
| Описание классов | 5 |
| Взаимосвязь классов | 36 |
| Описание алгоритмов | 40 |
| Описание разработанного приложения (руководство пользователя) | 45 |
| Заключение | 55 |
| Список используемой литературы | 56 |
| Приложение. Листинг разработанной программы | 57 |

# **Постановка задачи**

Информационная система управления данными высшего учебного заведения представляет собой программу для автоматизации процессов учета и управления данными образовательного процесса в университете: ведение учета факультетов и кафедр, управление учебными группами и студентами, учет преподавательского состава, управление учебными дисциплинами, распределение учебной нагрузки преподавателей, учет дипломных работ и ведение успеваемости студентов [3].

Необходимо реализовать: возможность добавления, редактирования и удаления всех основных сущностей системы (факультеты, кафедры, группы, студенты, преподаватели, дисциплины, учебная нагрузка, дипломные работы), возможность группировки данных для удобного отображения иерархических структур, возможность управления профилями студентов и преподавателей с отображением связанных данных, возможность автоматического вычисления курса дисциплины на основе семестра, возможность автоматического расчета итоговых оценок, возможность валидации всех вводимых данных с проверкой корректности и соблюдения бизнес–правил, возможность сохранения и загрузки всех данных в текстовые файлы для обеспечения персистентности между сеансами работы [2].

Взаимодействие с данными осуществляется через графический интерфейс, построенный на основе паттерна MVVM (Model–View–ViewModel). Все сущности представлены классами моделей данных, которые инкапсулируют информацию и предоставляют методы для сериализации и десериализации. Каждая сущность имеет уникальный идентификатор для установления связей между объектами. Сохранение и загрузка состояния системы происходит с помощью методов сериализации и десериализации соответственно, которые преобразуют объекты в строки с разделителями и обратно. Все вводимые данные проходят валидацию на соответствие допустимым форматам и диапазонам значений, обязательные поля проверяются на заполненность.

При разработке программы необходимо использовать ООП, инкапсуляцию, наследование и полиморфизм, обеспечить сохранение данных в текстовых файлах [1].

**Описание классов**

В программе присутствуют следующие классы:

1. Program – основной класс, содержащий метод Main, с него начинается запуск приложения.

Методы:

public static void Main(string[] args) – метод запуска всей программы, точка входа в приложение;

public static AppBuilder BuildAvaloniaApp() – метод настройки Avalonia приложения, конфигурирует платформу, шрифты, логирование и ReactiveUI;

2. App – класс главного приложения. Инициализирует XAML и создает главное окно с ViewModel.

Методы:

public override void Initialize() – метод загрузки XAML ресурсов приложения;

public override void OnFrameworkInitializationCompleted() – метод, вызываемый после инициализации фреймворка, создает главное окно и настраивает приложение;

private void DisableAvaloniaDataAnnotationValidation() – метод отключения встроенной валидации Avalonia для избежания конфликтов;

3. MainWindow – класс формы главного окна. В нём хранятся основные методы и обработчики событий.

Методы:

public MainWindow() – конструктор главного окна, инициализирует компоненты и подписывается на событие закрытия окна для автосохранения данных;

4. DataService – класс предназначен для хранения всех наблюдаемых массивов, а также загрузки и сохранения данных в текстовые файлы.

Методы:

public DataService() – конструктор, инициализирует все коллекции и создает папку Data для хранения файлов;

private void EnsureDataDirectoryExists() – метод создания папки Data для хранения файлов, если она не существует;

public void LoadAllData() – метод загрузки всех данных из текстовых файлов в память, вызывается при запуске приложения;

public void SaveAllData() – метод сохранения всех данных из памяти в текстовые файлы, вызывается при закрытии приложения или по запросу пользователя;

public string GetDataPath() – метод получения пути к папке с данными;

private void LoadFaculties() – метод загрузки списка факультетов из текстового файла;

private void SaveFaculties() – метод сохранения списка факультетов в текстовый файл;

private void LoadDepartments() – метод загрузки списка кафедр из текстового файла;

private void SaveDepartments() – метод сохранения списка кафедр в текстовый файл;

private void LoadGroups() – метод загрузки списка групп из текстового файла;

private void SaveGroups() – метод сохранения списка групп в текстовый файл;

private void LoadStudents() – метод загрузки списка студентов из текстового файла;

private void SaveStudents() – метод сохранения списка студентов в текстовый файл;

private void LoadTeachers() – метод загрузки списка преподавателей из текстового файла;

private void SaveTeachers() – метод сохранения списка преподавателей в текстовый файл;

private void LoadDisciplines() – метод загрузки списка дисциплин из текстового файла;

private void SaveDisciplines() – метод сохранения списка дисциплин в текстовый файл;

private void LoadWorkLoads() – метод загрузки списка учебной нагрузки из текстового файла;

private void SaveWorkLoads() – метод сохранения списка учебной нагрузки в текстовый файл;

private void LoadThesisWorks() – метод загрузки списка дипломных работ из текстового файла;

private void SaveThesisWorks() – метод сохранения списка дипломных работ в текстовый файл;

private void LoadGrades() – метод загрузки списка оценок из текстового файла;

private void SaveGrades() – метод сохранения списка оценок в текстовый файл;

private void LoadTeacherDisciplines() – метод загрузки списка связей преподавателей с дисциплинами из текстового файла;

private void SaveTeacherDisciplines() – метод сохранения списка связей преподавателей с дисциплинами в текстовый файл;

public Faculty? GetFaculty(Guid id) – метод поиска факультета по его уникальному идентификатору;

public Department? GetDepartment(Guid id) – метод поиска кафедры по её уникальному идентификатору;

public Group? GetGroup(Guid id) – метод поиска учебной группы по её уникальному идентификатору;

public Student? GetStudent(Guid id) – метод поиска студента по его уникальному идентификатору;

public Teacher? GetTeacher(Guid id) – метод поиска преподавателя по его уникальному идентификатору;

public Discipline? GetDiscipline(Guid id) – метод поиска дисциплины по её уникальному идентификатору;

public List<Department> GetDepartmentsByFaculty(Guid facultyId) – метод возврата списка всех кафедр, принадлежащих указанному факультету;

public List<Group> GetGroupsByFaculty(Guid facultyId) – метод возврата списка всех групп, принадлежащих указанному факультету;

public List<Student> GetStudentsByGroup(Guid groupId) – метод возврата списка всех студентов указанной группы;

public List<Teacher> GetTeachersByDepartment(Guid departmentId) – метод возврата списка всех преподавателей указанной кафедры;

public List<WorkLoad> GetWorkLoadsByTeacher(Guid teacherId) – метод возврата списка всей учебной нагрузки указанного преподавателя;

public List<ThesisWork> GetThesisWorksBySupervisor(Guid supervisorId) – метод возврата списка всех дипломных работ, которыми руководит указанный преподаватель;

Поля:

public ObservableCollection<Faculty> Faculties – наблюдаемый массив, состоящий из факультетов;

public ObservableCollection<Department> Departments – наблюдаемый массив, состоящий из кафедр;

public ObservableCollection<Group> Groups – наблюдаемый массив, состоящий из учебных групп;

public ObservableCollection<Student> Students – наблюдаемый массив, состоящий из студентов;

public ObservableCollection<Teacher> Teachers – наблюдаемый массив, состоящий из преподавателей;

public ObservableCollection<Discipline> Disciplines – наблюдаемый массив, состоящий из дисциплин;

public ObservableCollection<WorkLoad> WorkLoads – наблюдаемый массив, состоящий из учебной нагрузки;

public ObservableCollection<ThesisWork> ThesisWorks – наблюдаемый массив, состоящий из дипломных работ;

public ObservableCollection<StudentGrade> Grades – наблюдаемый массив, состоящий из оценок студентов;

public ObservableCollection<TeacherDiscipline> TeacherDisciplines – наблюдаемый массив, состоящий из связей преподавателей с дисциплинами;

5. ViewModelBase – родительский класс для всех ViewModel в приложении. Наследуется от ReactiveObject для поддержки реактивного программирования.

6. MainWindowViewModel – класс главной ViewModel приложения. Управляет навигацией между разными разделами и координирует работу всех ViewModel.

Методы:

public MainWindowViewModel() – конструктор, инициализирует главное окно: загружает данные и создает все ViewModel;

public void OnClosing() – метод, вызываемый при закрытии приложения для сохранения всех данных;

public void ShowStudentProfile(Student student) – метод открытия профиля выбранного студента с информацией о его предметах и оценках;

public void ShowTeacherProfile(Teacher teacher) – метод открытия профиля выбранного преподавателя с информацией о его дисциплинах;

Поля:

public ViewModelBase CurrentPage – поле текущей открытой страницы в главном окне, при изменении автоматически обновляется содержимое главного окна;

public FacultiesViewModel FacultiesViewModel – ViewModel для раздела факультетов;

public DepartmentsViewModel DepartmentsViewModel – ViewModel для раздела кафедр;

public GroupsViewModel GroupsViewModel – ViewModel для раздела групп;

public StudentsViewModel StudentsViewModel – ViewModel для раздела студентов;

public TeachersViewModel TeachersViewModel – ViewModel для раздела преподавателей;

public DisciplinesViewModel DisciplinesViewModel – ViewModel для раздела дисциплин;

public WorkLoadsViewModel WorkLoadsViewModel – ViewModel для раздела учебной нагрузки;

public ThesisWorksViewModel ThesisWorksViewModel – ViewModel для раздела дипломных работ;

public ReactiveCommand<Unit, Unit> ShowFacultiesCommand – команда для навигации к разделу факультетов;

public ReactiveCommand<Unit, Unit> ShowDepartmentsCommand – команда для навигации к разделу кафедр;

public ReactiveCommand<Unit, Unit> ShowGroupsCommand – команда для навигации к разделу групп;

public ReactiveCommand<Unit, Unit> ShowStudentsCommand – команда для навигации к разделу студентов;

public ReactiveCommand<Unit, Unit> ShowTeachersCommand – команда для навигации к разделу преподавателей;

public ReactiveCommand<Unit, Unit> ShowDisciplinesCommand – команда для навигации к разделу дисциплин;

public ReactiveCommand<Unit, Unit> ShowWorkLoadsCommand – команда для навигации к разделу учебной нагрузки;

public ReactiveCommand<Unit, Unit> ShowThesisWorksCommand – команда для навигации к разделу дипломных работ;

public ReactiveCommand<Unit, Unit> SaveDataCommand – команда для сохранения всех данных;

7. FacultiesViewModel – класс ViewModel для управления факультетами. Позволяет добавлять, редактировать и удалять факультеты.

Методы:

public FacultiesViewModel(DataService dataService) – конструктор, инициализирует ViewModel и создает команды;

private void AddFaculty() – метод добавления нового факультета в базу данных, выполняет валидацию всех полей перед добавлением;

private void UpdateFaculty() – метод обновления выбранного факультета;

private void DeleteFaculty() – метод удаления выбранного факультета;

private void ClearFields() – метод очистки полей ввода;

Поля:

public ObservableCollection<Faculty> Faculties – коллекция факультетов;

public Faculty? SelectedFaculty – выбранный факультет;

public string Name – название факультета;

public string Dean – ФИО декана факультета;

public string ErrorMessage – сообщение об ошибке валидации;

public ReactiveCommand<Unit, Unit> AddCommand – команда добавления факультета;

public ReactiveCommand<Unit, Unit> UpdateCommand – команда обновления факультета;

public ReactiveCommand<Unit, Unit> DeleteCommand – команда удаления факультета;

8. DepartmentsViewModel – класс ViewModel для управления кафедрами. Позволяет добавлять, редактировать и удалять кафедры. Группирует кафедры по факультетам для удобного отображения.

Методы:

public DepartmentsViewModel(DataService dataService) – конструктор, инициализирует ViewModel и подписывается на изменения в коллекциях;

private void AddDepartment() – метод добавления новой кафедры в базу данных;

private void UpdateDepartment() – метод обновления выбранной кафедры;

private void DeleteDepartment() – метод удаления выбранной кафедры;

private void ClearFields() – метод очистки полей ввода;

private void UpdateGroupedDepartments() – метод обновления группировки кафедр по факультетам;

Поля:

public ObservableCollection<Department> Departments – коллекция кафедр;

public ObservableCollection<Faculty> Faculties – коллекция факультетов;

public ObservableCollection<FacultyGroup> GroupedDepartments – коллекция кафедр, сгруппированных по факультетам;

public Department? SelectedDepartment – выбранная кафедра;

public string Name – название кафедры;

public string Head – ФИО заведующего кафедрой;

public Faculty? SelectedFaculty – выбранный факультет;

public string ErrorMessage – сообщение об ошибке валидации;

public ReactiveCommand<Unit, Unit> AddCommand – команда добавления кафедры;

public ReactiveCommand<Unit, Unit> UpdateCommand – команда обновления кафедры;

public ReactiveCommand<Unit, Unit> DeleteCommand – команда удаления кафедры;

9. FacultyGroup – класс группы кафедр, объединенных по факультету. Используется для отображения кафедр с группировкой по факультетам.

Поля:

public string FacultyName – название факультета;

public ObservableCollection<Department> Departments – коллекция кафедр факультета;

10. GroupsViewModel – класс ViewModel для управления учебными группами. Позволяет добавлять, редактировать и удалять группы. Группирует группы по факультетам и курсам для удобного отображения.

Методы:

public GroupsViewModel(DataService dataService) – конструктор, инициализирует ViewModel и подписывается на изменения;

private void UpdateGroupedGroups() – метод обновления группировки групп по факультетам и курсам;

private void AddGroup() – метод добавления новой группы в базу данных;

private void UpdateGroup() – метод обновления выбранной группы;

private void DeleteGroup() – метод удаления выбранной группы;

private void ClearFields() – метод очистки полей ввода;

Поля:

public ObservableCollection<Group> Groups – коллекция групп;

public ObservableCollection<Faculty> Faculties – коллекция факультетов;

public ObservableCollection<FacultyGrouping>? GroupedGroups – коллекция групп, сгруппированных по факультетам и курсам;

public Group? SelectedGroup – выбранная группа;

public string Number – номер группы;

public int YearOfAdmission – год набора;

public int Course – курс;

public Faculty? SelectedFaculty – выбранный факультет;

public string ErrorMessage – сообщение об ошибке валидации;

public ReactiveCommand<Unit, Unit> AddCommand – команда добавления группы;

public ReactiveCommand<Unit, Unit> UpdateCommand – команда обновления группы;

public ReactiveCommand<Unit, Unit> DeleteCommand – команда удаления группы;

11. CourseGrouping – класс группы учебных групп, объединенных по курсу. Используется для отображения групп с группировкой по курсам.

Поля:

public int Course – номер курса;

public string CourseName – название курса в виде строки;

public ObservableCollection<Group> Groups – коллекция групп курса;

12. FacultyGrouping – класс группы учебных групп, объединенных по факультету. Содержит вложенные группы по курсам.

Поля:

public string FacultyName – название факультета;

public ObservableCollection<CourseGrouping> Courses – коллекция курсов факультета;

13. StudentsViewModel – класс ViewModel для управления студентами. Позволяет добавлять, редактировать и удалять студентов. Группирует студентов по факультетам и группам для удобного отображения.

Методы:

public StudentsViewModel(DataService dataService, Action<Student>? showProfileAction) – конструктор, инициализирует ViewModel и подписывается на изменения;

private void UpdateGroupedStudents() – метод обновления группировки студентов по факультетам и группам;

private void AddStudent() – метод добавления нового студента в базу данных;

private void UpdateStudent() – метод обновления выбранного студента;

private void DeleteStudent() – метод удаления выбранного студента;

private void ClearFields() – метод очистки полей ввода;

private void ShowProfile() – метод открытия профиля выбранного студента;

Поля:

public ObservableCollection<Student> Students – коллекция студентов;

public ObservableCollection<Group> Groups – коллекция групп;

public ObservableCollection<StudentFacultyGrouping>? GroupedStudents – коллекция студентов, сгруппированных по факультетам и группам;

public Student? SelectedStudent – выбранный студент;

public string LastName – фамилия студента;

public string FirstName – имя студента;

public string MiddleName – отчество студента;

public string RecordBookNumber – номер зачётной книжки;

public double GPA – средний балл;

public Group? SelectedGroup – выбранная группа;

public string ErrorMessage – сообщение об ошибке валидации;

public ReactiveCommand<Unit, Unit> AddCommand – команда добавления студента;

public ReactiveCommand<Unit, Unit> UpdateCommand – команда обновления студента;

public ReactiveCommand<Unit, Unit> DeleteCommand – команда удаления студента;

public ReactiveCommand<Unit, Unit> ShowProfileCommand – команда открытия профиля студента;

14. StudentGroupGrouping – класс группы студентов, объединенных по учебной группе. Используется для отображения студентов с группировкой по группам.

Поля:

public Group Group – учебная группа;

public string GroupName – название группы;

public ObservableCollection<Student> Students – коллекция студентов группы;

15. StudentFacultyGrouping – класс группы студентов, объединенных по факультету. Содержит вложенные группы по учебным группам.

Поля:

public string FacultyName – название факультета;

public ObservableCollection<StudentGroupGrouping> Groups – коллекция групп факультета;

16. TeachersViewModel – класс ViewModel для управления преподавателями. Позволяет добавлять, редактировать и удалять преподавателей. Управляет информацией о должности, степени, звании и научной деятельности.

Методы:

public TeachersViewModel(DataService dataService, Action<Teacher>? showProfileAction) – конструктор, инициализирует ViewModel и создает команды;

private void AddTeacher() – метод добавления нового преподавателя в базу данных;

private void UpdateTeacher() – метод обновления выбранного преподавателя;

private void DeleteTeacher() – метод удаления выбранного преподавателя;

private void ClearFields() – метод очистки полей ввода;

private void ShowProfile() – метод открытия профиля выбранного преподавателя;

Поля:

public ObservableCollection<Teacher> Teachers – коллекция преподавателей;

public ObservableCollection<Department> Departments – коллекция кафедр;

public ObservableCollection<EnumItem<TeacherPosition>> Positions – коллекция должностей преподавателей;

public ObservableCollection<EnumItem<AcademicDegree>> Degrees – коллекция учёных степеней;

public ObservableCollection<EnumItem<AcademicTitle>> Titles – коллекция учёных званий;

public Teacher? SelectedTeacher – выбранный преподаватель;

public string LastName – фамилия преподавателя;

public string FirstName – имя преподавателя;

public string MiddleName – отчество преподавателя;

public TeacherPosition Position – должность преподавателя;

public AcademicDegree Degree – учёная степень;

public AcademicTitle Title – учёное звание;

public EnumItem<TeacherPosition>? SelectedPositionItem – выбранный элемент должности;

public EnumItem<AcademicDegree>? SelectedDegreeItem – выбранный элемент степени;

public EnumItem<AcademicTitle>? SelectedTitleItem – выбранный элемент звания;

public bool IsPostgraduate – обучается в аспирантуре;

public bool LeadsResearchTopics – руководит научными темами;

public bool LeadsResearchDirections – руководит научными направлениями;

public Department? SelectedDepartment – выбранная кафедра;

public string ErrorMessage – сообщение об ошибке валидации;

public ReactiveCommand<Unit, Unit> AddCommand – команда добавления преподавателя;

public ReactiveCommand<Unit, Unit> UpdateCommand – команда обновления преподавателя;

public ReactiveCommand<Unit, Unit> DeleteCommand – команда удаления преподавателя;

public ReactiveCommand<Unit, Unit> ShowProfileCommand – команда открытия профиля преподавателя;

17. DisciplinesViewModel – класс ViewModel для управления дисциплинами. Позволяет добавлять, редактировать и удалять дисциплины. Автоматически вычисляет курс на основе семестра. Группирует дисциплины по группам для удобного отображения.

Методы:

public DisciplinesViewModel(DataService dataService) – конструктор, инициализирует ViewModel и подписывается на изменения;

private void UpdateGroupedDisciplines() – метод обновления группировки дисциплин по группам;

private void AddDiscipline() – метод добавления новой дисциплины в базу данных;

private void UpdateDiscipline() – метод обновления выбранной дисциплины;

private void DeleteDiscipline() – метод удаления выбранной дисциплины;

private void ClearFields() – метод очистки полей ввода;

Поля:

public ObservableCollection<Discipline> Disciplines – коллекция дисциплин;

public ObservableCollection<Group> Groups – коллекция групп;

public ObservableCollection<EnumItem<ControlForm>> ControlForms – коллекция форм контроля;

public ObservableCollection<GroupDisciplines> GroupedDisciplines – коллекция дисциплин, сгруппированных по группам;

public Discipline? SelectedDiscipline – выбранная дисциплина;

public string Name – название дисциплины;

public int Course – курс;

public int Semester – семестр;

public int LectureHours – количество часов лекций;

public int SeminarHours – количество часов семинаров;

public int LaboratoryHours – количество часов лабораторных работ;

public ControlForm ControlForm – форма контроля;

public EnumItem<ControlForm>? SelectedControlFormItem – выбранный элемент формы контроля;

public Group? SelectedGroup – выбранная группа;

public string ErrorMessage – сообщение об ошибке валидации;

public ReactiveCommand<Unit, Unit> AddCommand – команда добавления дисциплины;

public ReactiveCommand<Unit, Unit> UpdateCommand – команда обновления дисциплины;

public ReactiveCommand<Unit, Unit> DeleteCommand – команда удаления дисциплины;

18. GroupDisciplines – класс группы дисциплин, объединенных по группе. Используется для отображения дисциплин с группировкой по группам.

Поля:

public string GroupName – название группы;

public ObservableCollection<Discipline> Disciplines – коллекция дисциплин группы;

19. WorkLoadsViewModel – класс ViewModel для управления учебной нагрузкой преподавателей. Позволяет назначать преподавателям дисциплины для конкретных групп. Валидирует правила: один преподаватель на лекции/семинары, несколько на лабораторные.

Методы:

public WorkLoadsViewModel(DataService dataService) – конструктор, инициализирует ViewModel и подписывается на изменения;

private void UpdateAvailableDisciplines() – метод обновления списка доступных дисциплин в зависимости от выбранного преподавателя;

private void AddWorkLoad() – метод добавления новой учебной нагрузки в базу данных;

private void UpdateWorkLoad() – метод обновления выбранной учебной нагрузки;

private void DeleteWorkLoad() – метод удаления выбранной учебной нагрузки;

private void ClearFields() – метод очистки полей ввода;

Поля:

public ObservableCollection<WorkLoad> WorkLoads – коллекция учебной нагрузки;

public ObservableCollection<Teacher> Teachers – коллекция преподавателей;

public ObservableCollection<Discipline> AvailableDisciplines – коллекция доступных дисциплин для выбранного преподавателя;

public ObservableCollection<Group> Groups – коллекция групп;

public ObservableCollection<EnumItem<LessonType>> LessonTypes – коллекция видов занятий;

public WorkLoad? SelectedWorkLoad – выбранная учебная нагрузка;

public Teacher? SelectedTeacher – выбранный преподаватель;

public Discipline? SelectedDiscipline – выбранная дисциплина;

public Group? SelectedGroup – выбранная группа;

public LessonType LessonType – вид занятий;

public EnumItem<LessonType>? SelectedLessonTypeItem – выбранный элемент вида занятий;

public int Hours – количество часов;

public string AcademicYear – учебный год;

public int Semester – семестр;

public string ErrorMessage – сообщение об ошибке валидации;

public ReactiveCommand<Unit, Unit> AddCommand – команда добавления учебной нагрузки;

public ReactiveCommand<Unit, Unit> UpdateCommand – команда обновления учебной нагрузки;

public ReactiveCommand<Unit, Unit> DeleteCommand – команда удаления учебной нагрузки;

20. ThesisWorksViewModel – класс ViewModel для управления дипломными работами. Позволяет добавлять, редактировать и удалять дипломные работы. Валидирует, что научный руководитель имеет право руководить дипломными работами.

Методы:

public ThesisWorksViewModel(DataService dataService) – конструктор, инициализирует ViewModel и создает команды;

private void AddThesisWork() – метод добавления новой дипломной работы в базу данных;

private void UpdateThesisWork() – метод обновления выбранной дипломной работы;

private void DeleteThesisWork() – метод удаления выбранной дипломной работы;

private void ClearFields() – метод очистки полей ввода;

Поля:

public ObservableCollection<ThesisWork> ThesisWorks – коллекция дипломных работ;

public ObservableCollection<Student> Students – коллекция студентов;

public ObservableCollection<Teacher> Supervisors – коллекция преподавателей, которые могут быть научными руководителями;

public ThesisWork? SelectedThesisWork – выбранная дипломная работа;

public string Title – тема работы;

public Student? SelectedStudent – выбранный студент;

public Teacher? SelectedSupervisor – выбранный научный руководитель;

public int Year – год защиты;

public int? Grade – оценка;

public string ErrorMessage – сообщение об ошибке валидации;

public ReactiveCommand<Unit, Unit> AddCommand – команда добавления дипломной работы;

public ReactiveCommand<Unit, Unit> UpdateCommand – команда обновления дипломной работы;

public ReactiveCommand<Unit, Unit> DeleteCommand – команда удаления дипломной работы;

21. StudentProfileViewModel – класс ViewModel для профиля студента. Отображает информацию о студенте, его группе и дисциплинах. Разделяет дисциплины на пройденные, текущие и будущие.

Методы:

public StudentProfileViewModel(DataService dataService, Student student) – конструктор, инициализирует ViewModel и загружает данные студента;

public string GetGradeForDiscipline(Guid disciplineId) – метод возврата текстового представления оценки студента по указанной дисциплине;

private async void OpenGradeInput(DisciplineWithGrade disciplineWithGrade) – метод открытия окна для ввода/редактирования оценки по выбранной дисциплине;

private void LoadStudentData() – метод загрузки и обработки данных студента для отображения в профиле;

Поля:

public Student Student – студент, профиль которого отображается;

public string FullName – полное имя студента;

public string GroupName – название группы студента;

public int CurrentSemester – текущий семестр;

public ObservableCollection<SemesterGroup> CompletedDisciplines – коллекция пройденных дисциплин, сгруппированных по семестрам;

public ObservableCollection<DisciplineWithGrade> CurrentDisciplines – коллекция текущих дисциплин;

public ObservableCollection<DisciplineWithGrade> FutureDisciplines – коллекция будущих дисциплин;

public ReactiveCommand<Unit, Unit> CloseCommand – команда закрытия профиля;

public ReactiveCommand<DisciplineWithGrade, Unit> OpenGradeInputCommand – команда открытия окна ввода оценки;

22. DisciplineWithGrade – класс обертки для дисциплины с текстом оценки. Используется для отображения дисциплины вместе с оценкой студента в профиле.

Поля:

public Discipline Discipline – дисциплина;

public string GradeText – текст оценки;

23. SemesterGroup – класс группы дисциплин, объединенных по семестру. Используется для отображения пройденных предметов с группировкой по семестрам.

Поля:

public int Semester – номер семестра;

public string SemesterName – название семестра в виде строки;

public ObservableCollection<DisciplineWithGrade> Disciplines – коллекция дисциплин семестра;

24. TeacherProfileViewModel – класс ViewModel для профиля преподавателя. Управляет списком дисциплин, которые ведет преподаватель. Позволяет добавлять и удалять дисциплины из профиля преподавателя.

Методы:

public TeacherProfileViewModel(DataService dataService, Teacher teacher) – конструктор, инициализирует ViewModel и подписывается на изменения;

private void LoadTeacherDisciplines() – метод загрузки списка дисциплин преподавателя;

private void LoadAvailableDisciplines() – метод загрузки списка доступных для добавления дисциплин;

private void AddDiscipline() – метод добавления дисциплины в профиль преподавателя;

private void RemoveDiscipline() – метод удаления дисциплины из профиля преподавателя;

Поля:

public string TeacherName – полное имя преподавателя;

public string Position – должность преподавателя;

public string Department – название кафедры преподавателя;

public ObservableCollection<Discipline> AvailableDisciplines – коллекция доступных для добавления дисциплин;

public ObservableCollection<Discipline> TeacherDisciplines – коллекция дисциплин преподавателя;

public Discipline? SelectedDisciplineToAdd – выбранная дисциплина для добавления;

public TeacherDiscipline? SelectedTeacherDiscipline – выбранная связь преподавателя с дисциплиной для удаления;

public string ErrorMessage – сообщение об ошибке валидации;

public ReactiveCommand<Unit, Unit> AddDisciplineCommand – команда добавления дисциплины;

public ReactiveCommand<Unit, Unit> RemoveDisciplineCommand – команда удаления дисциплины;

25. GradeInputViewModel – класс ViewModel для окна ввода оценки студента. Управляет вводом баллов за семестр и экзамен/зачет. Автоматически рассчитывает итоговую оценку по правилам вуза.

Методы:

public GradeInputViewModel(DataService dataService, Student student, Discipline discipline) – конструктор, инициализирует ViewModel и загружает существующую оценку;

private void LoadExistingGrade() – метод загрузки существующей оценки студента по этой дисциплине;

private void Calculate() – метод расчета итоговой оценки на основе введенных баллов;

private void SaveGrade() – метод сохранения оценки в базу данных;

Поля:

public string DisciplineName – название дисциплины;

public string StudentName – полное имя студента;

public string ControlFormName – название формы контроля;

public int MaxSemesterPoints – максимальное количество баллов за семестр;

public int MaxExamPoints – максимальное количество баллов за экзамен/зачет;

public int SemesterPoints – баллы за работу в семестре;

public int ExamPoints – баллы на экзамене или зачете;

public int TotalPoints – общая сумма баллов;

public string Grade – итоговая оценка в текстовом виде;

public string ErrorMessage – сообщение об ошибке валидации;

public ReactiveCommand<Unit, Unit> SaveCommand – команда сохранения оценки;

public ReactiveCommand<Unit, Unit> CalculateCommand – команда расчета оценки;

26. Faculty – класс модели факультета университета. Содержит информацию о факультете: название, декан, список групп и кафедр.

Методы:

public Faculty() – конструктор, инициализирует факультет с уникальным идентификатором;

public override string ToString() – метод возврата названия факультета;

public string ToFileString() – метод преобразования объекта факультета в строку для сохранения в текстовый файл;

public static Faculty FromFileString(string line) – метод создания объекта факультета из строки, загруженной из текстового файла;

Поля:

public Guid Id – уникальный идентификатор факультета;

public string Name – название факультета;

public string Dean – ФИО декана факультета;

public List<Guid> GroupIds – список идентификаторов групп, принадлежащих факультету;

public List<Guid> DepartmentIds – список идентификаторов кафедр, принадлежащих факультету;

27. Department – класс модели кафедры. Содержит информацию о кафедре: название, заведующий, факультет.

Методы:

public Department() – конструктор, инициализирует кафедру с уникальным идентификатором;

public override string ToString() – метод возврата названия кафедры;

public string ToFileString() – метод сохранения в строку для текстового файла;

public static Department FromFileString(string line) – метод загрузки из строки текстового файла;

Поля:

public Guid Id – уникальный идентификатор;

public string Name – название кафедры;

public string Head – заведующий кафедрой;

public Guid FacultyId – идентификатор факультета, к которому относится кафедра;

public List<Guid> TeacherIds – список преподавателей кафедры;

28. Group – класс модели учебной группы. Содержит информацию о группе: номер, год набора, курс, факультет.

Методы:

public Group() – конструктор, инициализирует группу с уникальным идентификатором;

public override string ToString() – метод возврата строкового представления группы;

public string ToFileString() – метод сохранения в строку для текстового файла;

public static Group FromFileString(string line) – метод загрузки из строки текстового файла;

Поля:

public Guid Id – уникальный идентификатор;

public string Number – номер группы;

public int YearOfAdmission – год набора;

public int Course – курс;

public Guid FacultyId – идентификатор факультета;

public List<Guid> StudentIds – список студентов группы;

29. Student – класс модели студента. Содержит информацию о студенте: ФИО, группа, номер зачётной книжки, средний балл.

Методы:

public Student() – конструктор, инициализирует студента с уникальным идентификатором;

public string FullName – свойство возврата полного имени студента;

public override string ToString() – метод возврата полного имени студента;

public string ToFileString() – метод сохранения в строку для текстового файла;

public static Student FromFileString(string line) – метод загрузки из строки текстового файла;

Поля:

public Guid Id – уникальный идентификатор;

public string LastName – фамилия;

public string FirstName – имя;

public string MiddleName – отчество;

public Guid GroupId – идентификатор группы;

public string RecordBookNumber – номер зачётной книжки;

public double GPA – средний балл;

30. Teacher – класс модели преподавателя. Содержит информацию о преподавателе: ФИО, должность, учёная степень, учёное звание, кафедра, научная деятельность.

Методы:

public Teacher() – конструктор, инициализирует преподавателя с уникальным идентификатором;

public string FullName – свойство возврата полного имени преподавателя;

public string FullInfo – свойство возврата полной информации о преподавателе с должностью и званиями;

public override string ToString() – метод возврата полной информации о преподавателе;

public bool IsPositionValid() – метод проверки возможности занимать текущую должность с текущим званием;

public bool CanTeachLectures() – метод проверки возможности читать лекции;

public bool CanTeachLaboratory() – метод проверки возможности проводить лабораторные работы;

public string ToFileString() – метод сохранения в строку для текстового файла;

public static Teacher FromFileString(string line) – метод загрузки из строки текстового файла;

private static string GetPositionString(TeacherPosition position) – метод возврата строкового представления должности;

private static string GetDegreeString(AcademicDegree degree) – метод возврата строкового представления учёной степени;

private static string GetTitleString(AcademicTitle title) – метод возврата строкового представления учёного звания;

Поля:

public Guid Id – уникальный идентификатор;

public string LastName – фамилия;

public string FirstName – имя;

public string MiddleName – отчество;

public TeacherPosition Position – должность;

public AcademicDegree Degree – учёная степень;

public AcademicTitle Title – учёное звание;

public Guid DepartmentId – идентификатор кафедры;

public bool IsPostgraduate – обучается в аспирантуре;

public bool LeadsResearchTopics – руководит научными темами;

public bool LeadsResearchDirections – руководит научными направлениями;

31. Discipline – класс модели учебной дисциплины. Содержит информацию о предмете: название, курс, семестр, часы, форма контроля. Дисциплина привязана к конкретной группе.

Методы:

public Discipline() – конструктор, инициализирует дисциплину с уникальным идентификатором;

public static int CalculateCourseFromSemester(int semester) – метод вычисления номера курса на основе номера семестра;

public void SetSemester(int semester) – метод установки семестра и автоматического пересчета курса;

public override string ToString() – метод возврата строкового представления дисциплины;

public string ToFileString() – метод преобразования объекта дисциплины в строку для сохранения в текстовый файл;

public static Discipline FromFileString(string line) – метод создания объекта дисциплины из строки, загруженной из текстового файла;

Поля:

public Guid Id – уникальный идентификатор;

public string Name – название дисциплины;

public int Course – курс, на котором читается дисциплина;

public int Semester – семестр;

public int LectureHours – количество часов лекций;

public int SeminarHours – количество часов семинаров;

public int LaboratoryHours – количество часов лабораторных работ;

public ControlForm ControlForm – форма контроля;

public Guid GroupId – идентификатор группы;

32. WorkLoad – класс модели учебной нагрузки преподавателя. Описывает, какой преподаватель ведет какой предмет у какой группы. Содержит информацию о типе занятий (лекции, семинары, лабораторные) и количестве часов.

Методы:

public WorkLoad() – конструктор, инициализирует учебную нагрузку с уникальным идентификатором;

public string ToFileString() – метод сохранения в строку для текстового файла;

public static WorkLoad FromFileString(string line) – метод загрузки из строки текстового файла;

public static string GetLessonTypeString(LessonType lessonType) – метод возврата строкового представления вида занятий;

Поля:

public Guid Id – уникальный идентификатор;

public Guid TeacherId – идентификатор преподавателя;

public Guid DisciplineId – идентификатор дисциплины;

public Guid GroupId – идентификатор группы;

public LessonType LessonType – вид занятий;

public int Hours – количество часов;

public string AcademicYear – учебный год;

public int Semester – семестр;

33. ThesisWork – класс модели дипломной работы. Содержит информацию о дипломной работе: название, студент, научный руководитель, год защиты, оценка.

Методы:

public ThesisWork() – конструктор, инициализирует дипломную работу с уникальным идентификатором;

public string ToFileString() – метод сохранения в строку для текстового файла;

public static ThesisWork FromFileString(string line) – метод загрузки из строки текстового файла;

Поля:

public Guid Id – уникальный идентификатор;

public string Title – название работы;

public Guid StudentId – идентификатор студента;

public Guid SupervisorId – идентификатор научного руководителя;

public int Year – год защиты;

public int? Grade – оценка (2–5, null если не выставлена);

34. StudentGrade – класс модели оценки студента по дисциплине. Хранит баллы за семестр и экзамен/зачет, автоматически рассчитывает итоговую оценку.

Методы:

public StudentGrade() – конструктор, инициализирует оценку с уникальным идентификатором;

public void CalculateGrade(ControlForm controlForm) – метод расчета итоговой оценки на основе набранных баллов и формы контроля;

public string ToFileString() – метод сохранения в строку для текстового файла;

public static StudentGrade FromFileString(string line) – метод загрузки из строки текстового файла;

public override string ToString() – метод возврата строкового представления оценки;

Поля:

public Guid Id – уникальный идентификатор;

public Guid StudentId – идентификатор студента;

public Guid DisciplineId – идентификатор дисциплины;

public int SemesterPoints – баллы за работу в семестре;

public int ExamPoints – баллы на экзамене или зачете;

public int TotalPoints – общая сумма баллов;

public string Grade – итоговая оценка в текстовом виде;

35. TeacherDiscipline – класс модели связи преподавателя с дисциплиной. Определяет, какие дисциплины может вести конкретный преподаватель.

Методы:

public TeacherDiscipline() – конструктор, инициализирует связь с уникальным идентификатором;

public string ToFileString() – метод сохранения в строку для текстового файла;

public static TeacherDiscipline FromFileString(string line) – метод загрузки из строки текстового файла;

Поля:

public Guid Id – уникальный идентификатор;

public Guid TeacherId – идентификатор преподавателя;

public Guid DisciplineId – идентификатор дисциплины;

36. BaseModel – абстрактный базовый класс для всех моделей приложения

Методы:

public abstract string ToFileString() – преобразование объекта в строку;

public override string ToString() – возврат строкового представления;

Поля:

public Guid Id – уникальный идентификатор;

37. ValidationHelper – класс вспомогательных методов для валидации вводимых данных. Содержит методы проверки корректности различных типов данных.

Методы:

public static bool IsValidName(string? name) – метод проверки, что строка содержит только буквы, пробелы, дефисы и точки;

public static bool IsValidNameWithNumbers(string? name) – метод проверки, что строка содержит только буквы, цифры, пробелы, дефисы и точки;

public static bool IsValidYear(string? year) – метод проверки, что строка является корректным годом;

public static bool IsValidCourse(string? course) – метод проверки, что строка является корректным номером курса;

public static bool IsValidSemester(string? semester) – метод проверки, что строка является корректным номером семестра;

public static bool IsValidHours(string? hours) – метод проверки, что строка является корректным количеством часов;

public static string GetErrorMessage(string fieldName, string errorType) – метод возврата понятного сообщения об ошибке валидации;

38. EnumHelper – класс вспомогательных методов для создания списков enum значений с русскими названиями. Используется во всех ViewModel для заполнения ComboBox.

Методы:

public static List<EnumItem<TeacherPosition>> GetTeacherPositions() – метод создания списка должностей преподавателей с русскими названиями;

public static List<EnumItem<AcademicDegree>> GetAcademicDegrees() – метод создания списка учёных степеней с русскими названиями;

public static List<EnumItem<AcademicTitle>> GetAcademicTitles() – метод создания списка учёных званий с русскими названиями;

public static List<EnumItem<LessonType>> GetLessonTypes() – метод создания списка видов занятий с русскими названиями;

public static List<EnumItem<ControlForm>> GetControlForms() – метод создания списка форм контроля с русскими названиями;

39. EnumItem<T> – класс обертки для enum значений с русским названием. Используется для отображения enum в ComboBox с локализованными названиями.

Методы:

public EnumItem(T value, string display) – конструктор, инициализирует обертку с значением enum и его русским названием;

public override string ToString() – метод возврата отображаемого названия;

Поля:

public T Value – значение enum;

public string Display – русское название для отображения;

40. EnumExtensions – класс методов расширения для enum'ов. Содержит методы преобразования enum значений в русские строки.

Методы:

public static string ToRussianString(this TeacherPosition position) – метод получения русского названия для должности преподавателя;

public static string ToRussianString(this AcademicDegree degree) – метод получения русского названия для учёной степени;

public static string ToRussianString(this AcademicTitle title) – метод получения русского названия для учёного звания;

public static string ToRussianString(this LessonType lessonType) – метод получения русского названия для вида занятий;

public static string ToRussianString(this ControlForm controlForm) – метод получения русского названия для формы контроля;

41. ViewLocator – класс для автоматического сопоставления ViewModel с соответствующими View. Используется для динамического создания представлений.

Методы:

public Control? Build(object? param) – метод построения View на основе переданного ViewModel;

public bool Match(object? data) – метод проверки, соответствует ли объект ViewModelBase;

42. GradeInputWindow – класс формы окна ввода оценки студента.

Методы:

public GradeInputWindow() – конструктор, инициализирует компоненты окна;

private void OnSaveClick(object? sender, RoutedEventArgs e) – событие нажатия кнопки сохранения;

private void OnCancelClick(object? sender, RoutedEventArgs e) – событие нажатия кнопки отмены;

43. FacultiesView – класс представления для раздела факультетов.

44. DepartmentsView – класс представления для раздела кафедр.

45. GroupsView – класс представления для раздела групп.

46. StudentsView – класс представления для раздела студентов.

47. TeachersView – класс представления для раздела преподавателей.

48. DisciplinesView – класс представления для раздела дисциплин.

49. WorkLoadsView – класс представления для раздела учебной нагрузки.

50. ThesisWorksView – класс представления для раздела дипломных работ.

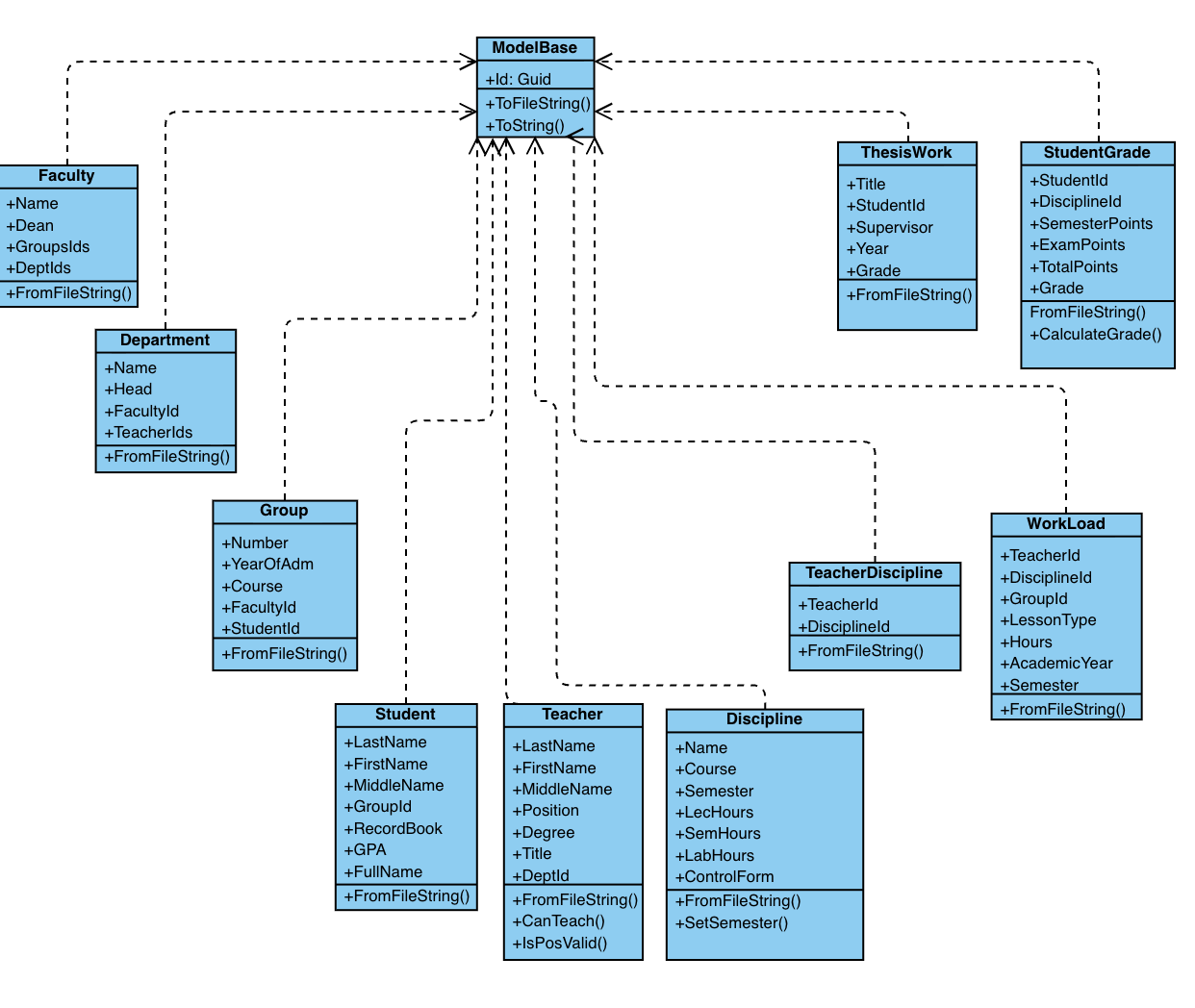
51. StudentProfileView – класс представления для профиля студента.

52. TeacherProfileView – класс представления для профиля преподавателя.

53. TeacherPosition – перечисление должностей преподавателя.

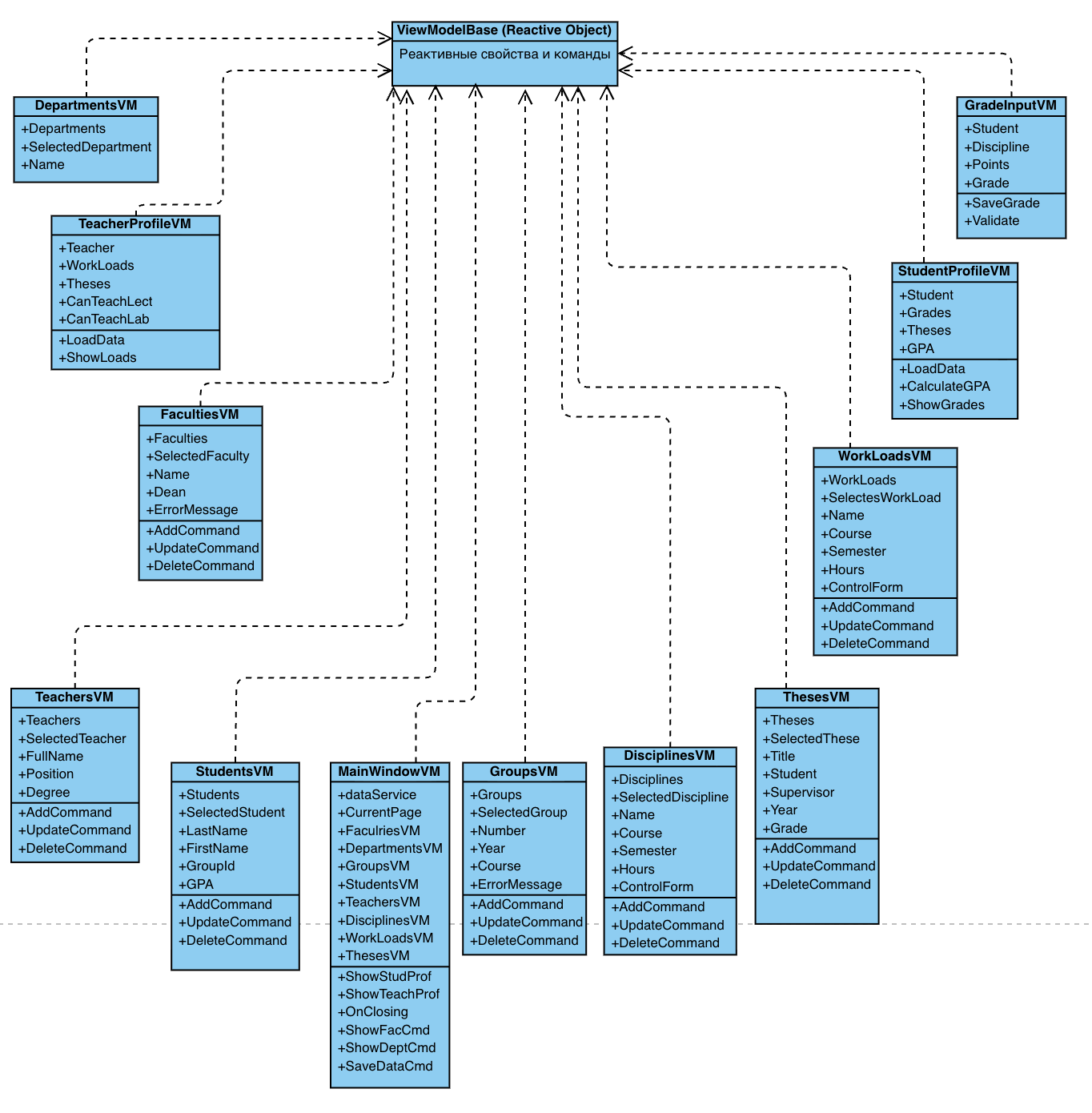
# **Взаимосвязь классов**

Класс ModelBase – абстрактный базовый класс для всех моделей данных приложения. Класс содержит общее свойство Id типа Guid и абстрактный метод ToFileString() для сериализации данных. Классы Faculty, Department, Group, Student, Teacher, Discipline, WorkLoad, ThesisWork, StudentGrade и TeacherDiscipline унаследованы от него. В классах Faculty, Department, Group, Student, Teacher и Discipline переопределён метод ToString() для возврата строкового представления объекта. Все модели данных реализуют методы ToFileString() и FromFileString() для сериализации и десериализации данных в текстовый формат. Класс Faculty содержит списки идентификаторов кафедр и групп, класс Department содержит список идентификаторов преподавателей, класс Group содержит список идентификаторов студентов. Класс Student содержит информацию о студенте и его группе, класс Teacher содержит информацию о преподавателе, его должности, учёной степени и звании. Класс Discipline содержит информацию о дисциплине и связана с группой. Класс WorkLoad связывает преподавателя, дисциплину и группу для учёта учебной нагрузки. Класс StudentGrade хранит оценки студентов по дисциплинам. Класс ThesisWork хранит информацию о дипломных работах студентов и их научных руководителях. Класс TeacherDiscipline реализует связь многие–ко–многим между преподавателями и дисциплинами. На диаграмме представлена иерархия наследования этих классов. (рисунок 1)



*Рисунок 1. – Диаграмма наследования классов от ModelBase*

Класс ViewModelBase – базовый класс для всех ViewModel в приложении. Класс содержит необходимый набор методов и свойств для поддержки реактивного программирования и автоматического обновления пользовательского интерфейса. Классы MainWindowViewModel, FacultiesViewModel, DepartmentsViewModel, GroupsViewModel, StudentsViewModel, TeachersViewModel, DisciplinesViewModel, WorkLoadsViewModel, ThesisWorksViewModel, StudentProfileViewModel, TeacherProfileViewModel и GradeInputViewModel унаследованы от него. Все ViewModel используют механизм реактивных команд (ReactiveCommand) для обработки действий пользователя и автоматического обновления интерфейса при изменении данных. В классах ViewModel переопределяются свойства и методы для работы с конкретными типами данных, при этом сохраняется единый подход к управлению состоянием через базовый класс. (рисунок 2)



*Рисунок 2. – Диаграмма наследования классов от ViewModelBase*

Класс MainWindow – класс главного окна приложения. Класс содержит логику навигации между разделами и обработку события закрытия окна для автосохранения данных. Класс GradeInputWindow – класс модального окна для ввода оценок студентов.

Классы FacultiesView, DepartmentsView, GroupsView, StudentsView, TeachersView, DisciplinesView, WorkLoadsView, ThesisWorksView, StudentProfileView и TeacherProfileView – классы представлений для отдельных разделов приложения. Все эти классы отвечают за отображение данных соответствующих ViewModel. Каждый класс View содержит разметку интерфейса в формате AXAML и минимальную логику инициализации в коде. На диаграмме представлена структура этих классов.

Класс DataService – центральный сервис для работы с данными приложения. Класс содержит наблюдаемые коллекции всех сущностей приложения и управляет их загрузкой и сохранением в текстовые файлы. Класс реализует методы LoadAllData() и SaveAllData() для загрузки и сохранения всех данных, а также методы поиска сущностей по идентификаторам и методы фильтрации. Все ViewModel используют DataService для доступа к данным.

Класс ValidationHelper – вспомогательный класс для валидации вводимых данных. Класс содержит статические методы проверки корректности различных типов данных. Все ViewModel используют ValidationHelper для проверки корректности вводимых пользователем данных.

Класс EnumHelper – вспомогательный класс для создания списков enum значений с русскими названиями. Класс содержит статические методы для всех перечислений, которые возвращают списки обёрток EnumItem<T> с локализованными названиями для отображения в ComboBox.

Класс EnumExtensions – класс методов расширения для enum'ов. Класс содержит методы расширения ToRussianString() для всех перечислений, которые преобразуют значения enum в русские строки для отображения в интерфейсе.

Класс ViewLocator – класс для автоматического сопоставления ViewModel с соответствующими View. Класс реализует интерфейс IDataTemplate и содержит методы Build() и Match() для динамического создания представлений на основе переданного ViewModel.

Класс EnumItem<T> – обобщённый класс–обёртка для enum значений с русским названием. Класс содержит свойства Value (значение enum) и Display (русское название для отображения) и используется для отображения enum в ComboBox с локализованными названиями.

**Описание алгоритмов**

В информационной системе ВУЗа реализован ряд алгоритмов для обеспечения корректной работы с данными образовательного процесса. Ниже представлено описание основных алгоритмов, используемых в приложении.

Алгоритм сериализации и десериализации данных

Для сохранения данных в текстовые файлы и их последующего восстановления реализованы алгоритмы сериализации и десериализации. Алгоритм сериализации объектов в строку (метод ToFileString) выполняет преобразование всех свойств объекта в единую строку, используя символ вертикальной черты в качестве разделителя. Для списков идентификаторов применяется метод объединения элементов через запятую, а для перечислений выполняется приведение к целому числу.

Алгоритм десериализации строки в объект (метод FromFileString) выполняет обратное преобразование. Строка разбивается по разделителю вертикальной черты, после чего каждая часть парсится в соответствующий тип данных: для идентификаторов используется Guid.Parse(), для чисел – int.Parse(), для перечислений выполняется приведение целого числа к типу перечисления. Для списков идентификаторов выполняется дополнительное разбиение по запятой с последующим парсингом каждого элемента. При возникновении ошибки парсинга некорректная строка игнорируется, что обеспечивает устойчивость работы приложения при наличии повреждённых данных.

Алгоритм загрузки и сохранения данных

Алгоритм загрузки данных (метод LoadAllData) обеспечивает восстановление всех данных приложения из текстовых файлов при запуске. Процесс начинается с проверки существования папки Data и её создания при отсутствии. Далее выполняется последовательная загрузка данных для каждой сущности: проверка существования файла, чтение всех строк из файла, для каждой непустой строки вызывается статический метод FromFileString() соответствующего класса, после чего объект добавляется в соответствующую коллекцию. Ошибки парсинга обрабатываются путём игнорирования некорректных строк, что позволяет приложению продолжить работу даже при наличии повреждённых записей. Порядок загрузки данных следующий: факультеты, кафедры, группы, студенты, преподаватели, дисциплины, учебная нагрузка, дипломные работы, оценки, связи преподавателей и дисциплин.

Алгоритм сохранения данных (метод SaveAllData) выполняет запись всех данных из памяти в текстовые файлы при закрытии приложения или по запросу пользователя. Для каждой коллекции сущностей выполняется преобразование всех объектов в строки через метод ToFileString(), после чего все строки записываются в соответствующий файл методом File.WriteAllLines(). Порядок сохранения данных: связи преподавателей и дисциплин, оценки, факультеты, кафедры, группы, студенты, преподаватели, дисциплины, учебная нагрузка, дипломные работы

Алгоритм расчёта итоговой оценки студента

Алгоритм расчёта итоговой оценки (метод CalculateGrade класса StudentGrade) выполняет автоматический расчёт итоговой оценки на основе набранных баллов и формы контроля. Входными данными являются баллы за работу в семестре (от 0 до 60 для экзамена, от 0 до 80 для зачёта), баллы на экзамене или зачёте (от 0 до 40 для экзамена, от 0 до 20 для зачёта) и форма контроля (экзамен, дифференцированный зачёт или зачёт). Алгоритм начинается с вычисления общей суммы баллов путём сложения баллов за семестр и баллов на экзамене или зачёте. Далее выполняется определение формы контроля. Для экзамена и дифференцированного зачёта применяется следующая шкала оценивания: от 0 до 49 баллов соответствует оценке "Неудовлетворительно (2)", от 50 до 72 баллов – "Удовлетворительно (3)", от 73 до 86 баллов – "Хорошо (4)", от 87 до 100 баллов – "Отлично (5)". Для зачёта используется упрощённая шкала: от 0 до 49 баллов соответствует "Незачет", от 50 до 100 баллов – "Зачет".

Алгоритм расчёта курса из семестра

Алгоритм определения номера курса на основе номера семестра (метод CalculateCourseFromSemester класса Discipline) использует формулу целочисленного деления: курс равен результату деления суммы номера семестра и единицы на два. Таким образом, семестры 1–2 соответствуют первому курсу, семестры 3–4 – второму курсу, семестры 5–6 – третьему курсу и так далее. Метод SetSemester обеспечивает установку семестра с автоматическим пересчётом курса, что гарантирует синхронизацию этих значений.

Алгоритмы валидации данных

В системе реализован набор алгоритмов валидации для проверки корректности вводимых пользователем данных. Алгоритм проверки имени (метод IsValidName) предназначен для валидации имён, фамилий и названий без цифр. Алгоритм выполняет проверку на пустоту или null, после чего применяет регулярное выражение, разрешающее использование букв (русских и латинских), пробелов, точек и дефисов.

Алгоритм проверки названия с цифрами (метод IsValidNameWithNumbers) используется для валидации названий групп, дисциплин и других сущностей, которые могут содержать цифры. Алгоритм аналогичен предыдущему, но регулярное выражение дополнительно разрешает использование цифр.

Алгоритмы проверки числовых значений (методы IsValidYear, IsValidCourse, IsValidSemester, IsValidHours) выполняют проверку на пустоту или null, попытку преобразования в целое число и проверку допустимого диапазона. Для года допустимый диапазон составляет от 1900 до 2100, для курса – от 1 до 6, для семестра – от 1 до 10, для количества часов – от 1 до 1000.

Алгоритмы поиска и фильтрации

Алгоритм поиска по идентификатору используется для нахождения сущности в коллекции по её уникальному идентификатору. Алгоритм применяет метод LINQ FirstOrDefault() для поиска элемента, у которого свойство Id равно заданному идентификатору, и возвращает найденный объект или null при отсутствии совпадения. В системе реализованы методы GetFaculty, GetDepartment, GetGroup, GetStudent, GetTeacher и GetDiscipline для поиска соответствующих сущностей.

Алгоритм фильтрации по связанному идентификатору предназначен для получения списка сущностей, связанных с другой сущностью через внешний ключ. Алгоритм использует метод LINQ Where() для фильтрации элементов, у которых внешний ключ равен заданному идентификатору, после чего преобразует результат в список методом ToList(). В системе реализованы методы GetDepartmentsByFaculty для получения кафедр факультета, GetGroupsByFaculty для получения групп факультета, GetStudentsByGroup для получения студентов группы, GetTeachersByDepartment для получения преподавателей кафедры, GetWorkLoadsByTeacher для получения нагрузки преподавателя и GetThesisWorksBySupervisor для получения дипломных работ научного руководителя.

Алгоритм проверки валидности должности преподавателя

Алгоритм проверки соответствия должности преподавателя его учёному званию (метод IsPositionValid класса Teacher) реализует бизнес–правила системы. Согласно правилам, должность доцента может занимать только преподаватель со званием доцента или профессора, а должность профессора может занимать только преподаватель со званием профессора. Алгоритм проверяет эти условия и возвращает false при несоответствии, в остальных случаях возвращает true.

Алгоритмы проверки возможности ведения занятий

Алгоритм проверки возможности чтения лекций (метод CanTeachLectures класса Teacher) определяет, может ли преподаватель читать лекции на основе его должности. Согласно правилам системы, ассистент не может читать лекции, поэтому алгоритм проверяет, что должность преподавателя не равна должности ассистента.

Алгоритм проверки возможности проведения лабораторных работ (метод CanTeachLaboratory класса Teacher) определяет, может ли преподаватель проводить лабораторные работы. Согласно правилам системы, профессор не может проводить лабораторные работы, поэтому алгоритм проверяет, что должность преподавателя не равна должности профессора.

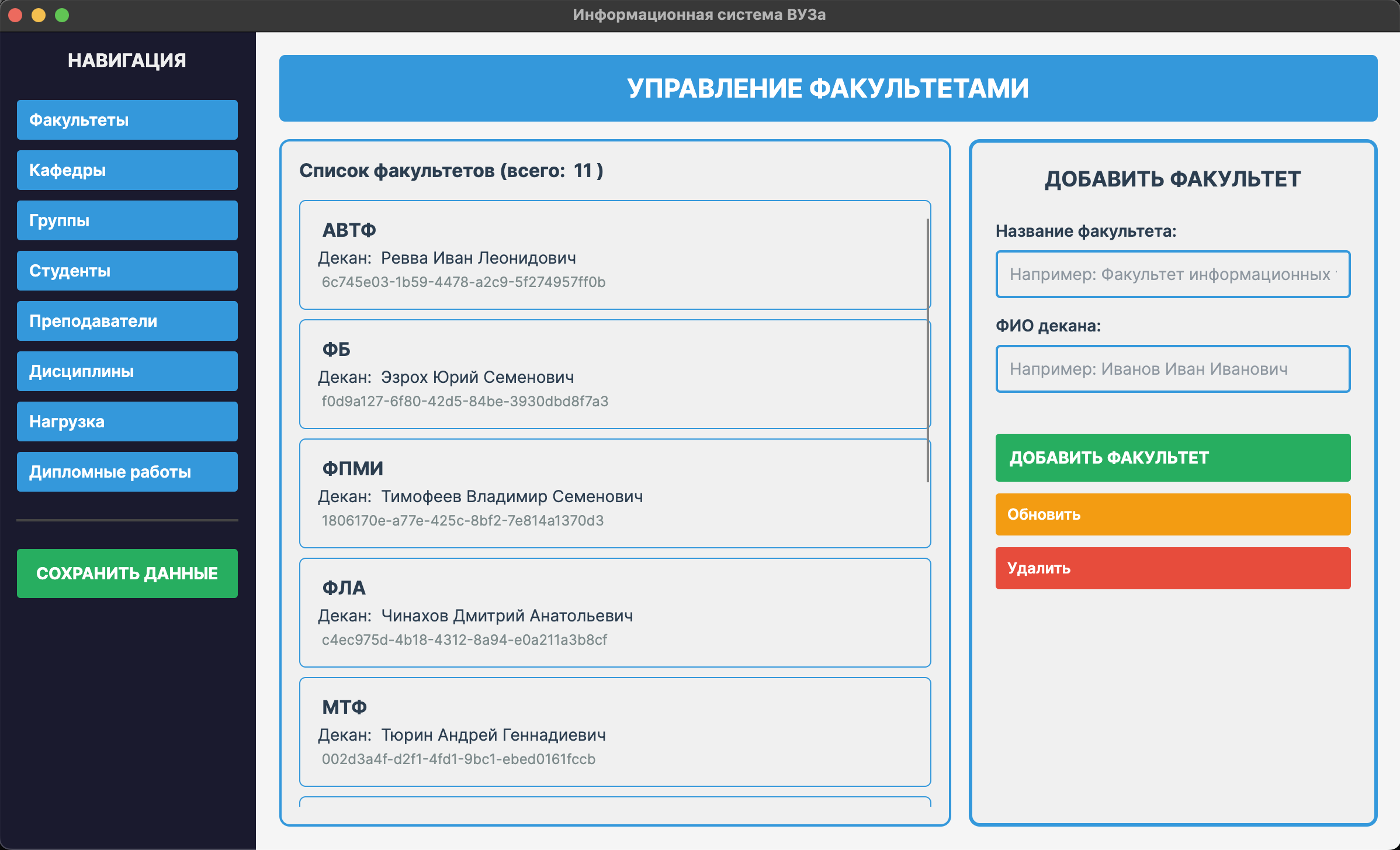
Общая характеристика алгоритмов

Особенности реализации алгоритмов включают обработку ошибок путём игнорирования некорректных строк при загрузке данных, что обеспечивает устойчивость работы приложения. Использование ObservableCollection обеспечивает автоматическое обновление пользовательского интерфейса при изменении данных благодаря механизму реактивного программирования. Разделитель данных в виде символа вертикальной черты обеспечивает простоту парсинга и надёжность сериализации. Алгоритмы десериализации поддерживают обратную совместимость со старыми форматами данных, что позволяет корректно загружать данные, сохранённые в предыдущих версиях приложения.

# **Описание разработанного приложения (руководство пользователя)**

Данное приложение предназначено для удобного ведения учёта и управления данными образовательного процесса в высшем учебном заведении. Работнику предоставляется возможность управления факультетами, кафедрами, учебными группами, студентами, преподавателями и дисциплинами. Система позволяет добавлять, редактировать и удалять записи во всех разделах, а также просматривать профили студентов и преподавателей с детальной информацией. При добавлении учебной нагрузки автоматически создаётся связь между преподавателем, дисциплиной и группой, что позволяет отслеживать распределение учебных часов. Система поддерживает учёт дипломных работ студентов с указанием научных руководителей, а также ведение оценок студентов по дисциплинам с автоматическим расчётом итоговых баллов. Все данные автоматически сохраняются в текстовые файлы при закрытии приложения, также есть возможность ручного сохранения всех данных [3]. Разработка велась на языке C# с использованием фреймворка Avalonia UI.

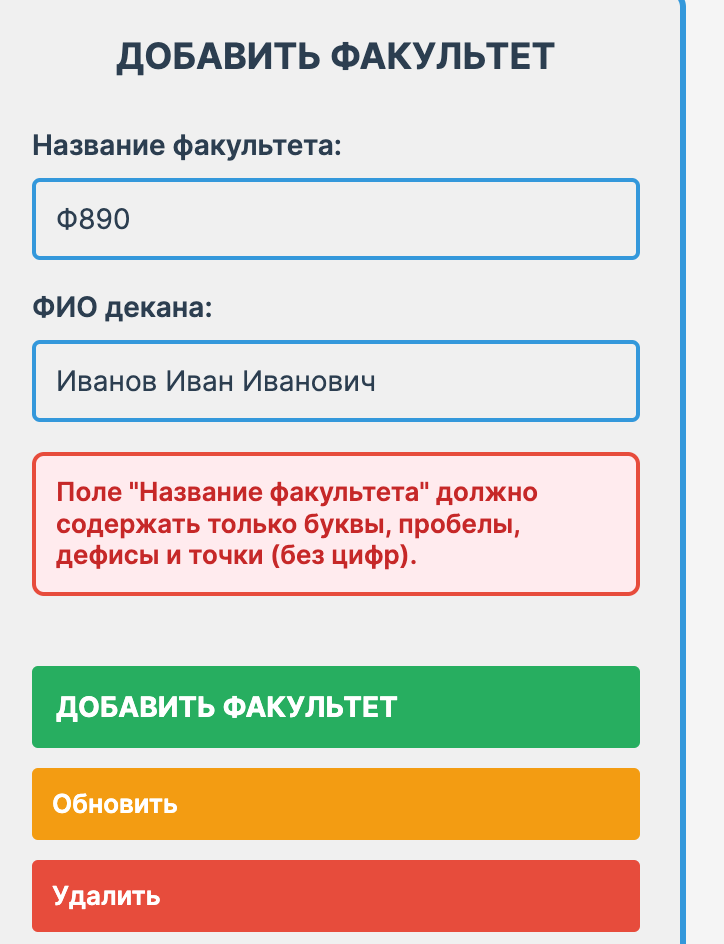
При запуске приложения пользователь попадает в главное меню, где открывается список факультетов ВУЗа. (рисунок 3)



*Рисунок 3 – Стартовое окно.*

Все факультеты уже добавлены вручную при тестировании работы приложения.

При вводе названия факультета и ФИО декана вводимые данные проходят валидацию. (рисунок 4, рисунок 5)

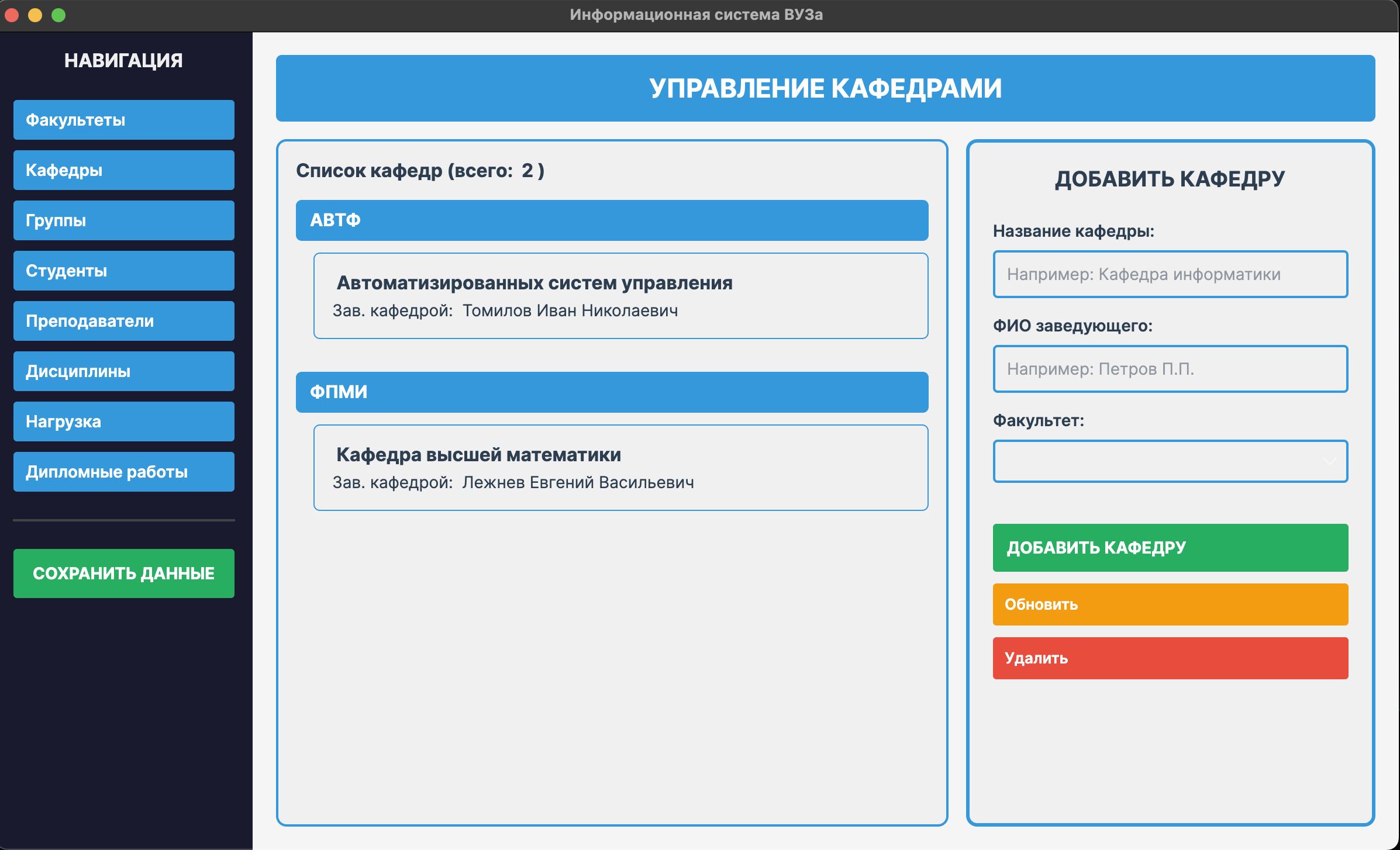


*Рисунок 4 – Информация а некорректном вводе данных.*

**

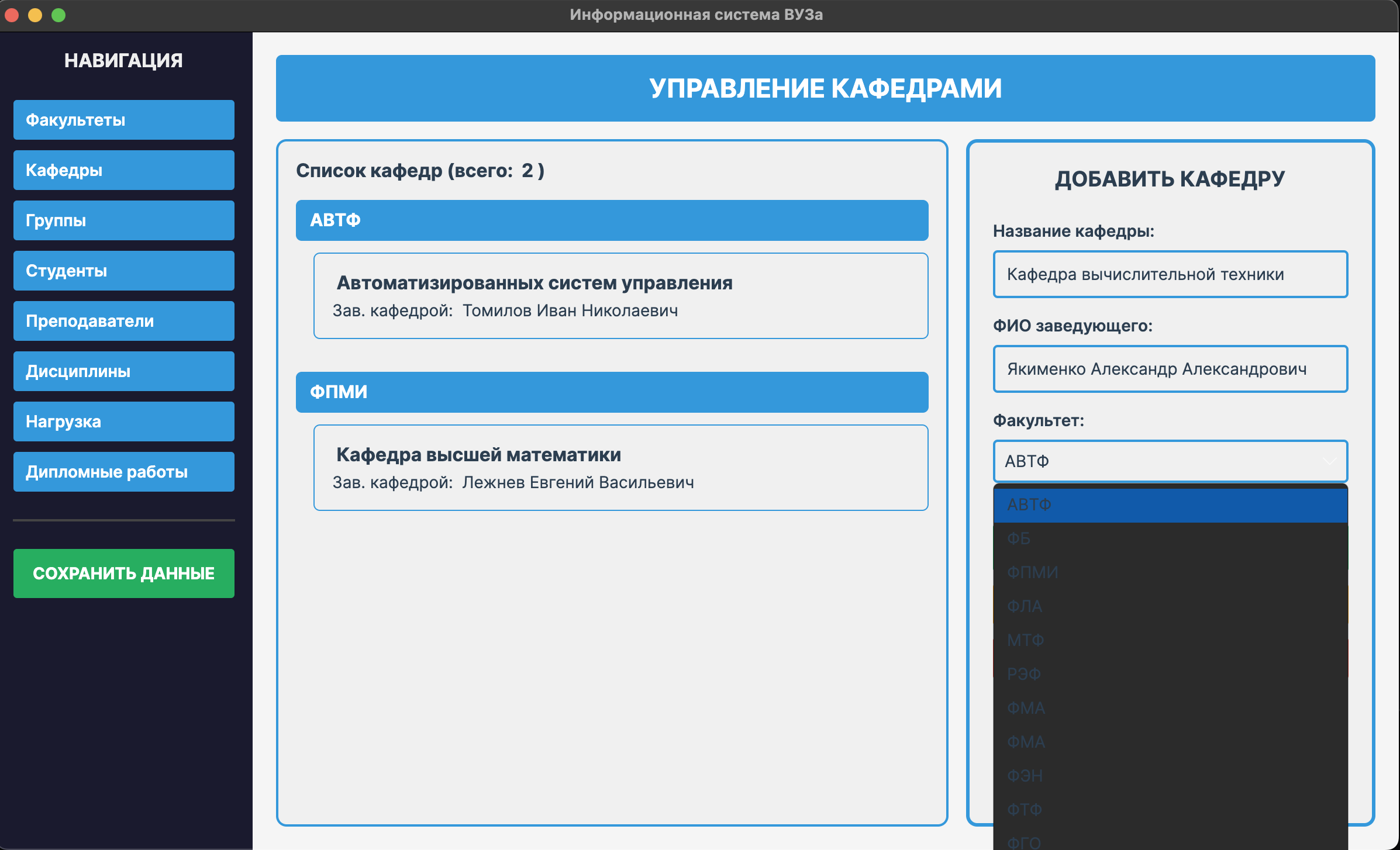
*Рисунок 5 – Информация о некорректном вводе данных.*

Через окно навигации попадаем в следующий раздел – кафедры. (рисунок 6)



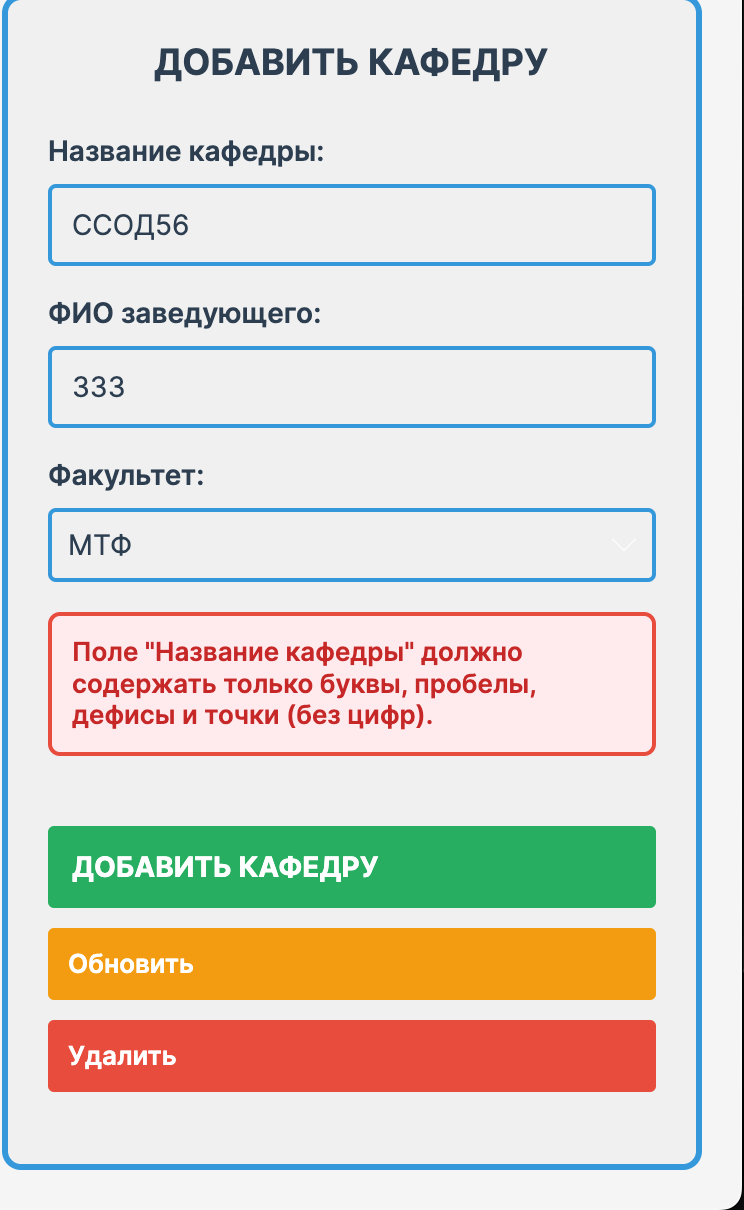
*Рисунок 6 – окно с кафедрами.*

Аналогично, как и с факультетами, при добавлении кафедры надо ввести ее название и ФИО заведующего. Поскольку кафедры относятся к факультетам, при добавлении кафедры требуется еще выбрать факультет из уже существующих, к которому она будет относиться. (рисунок 7)



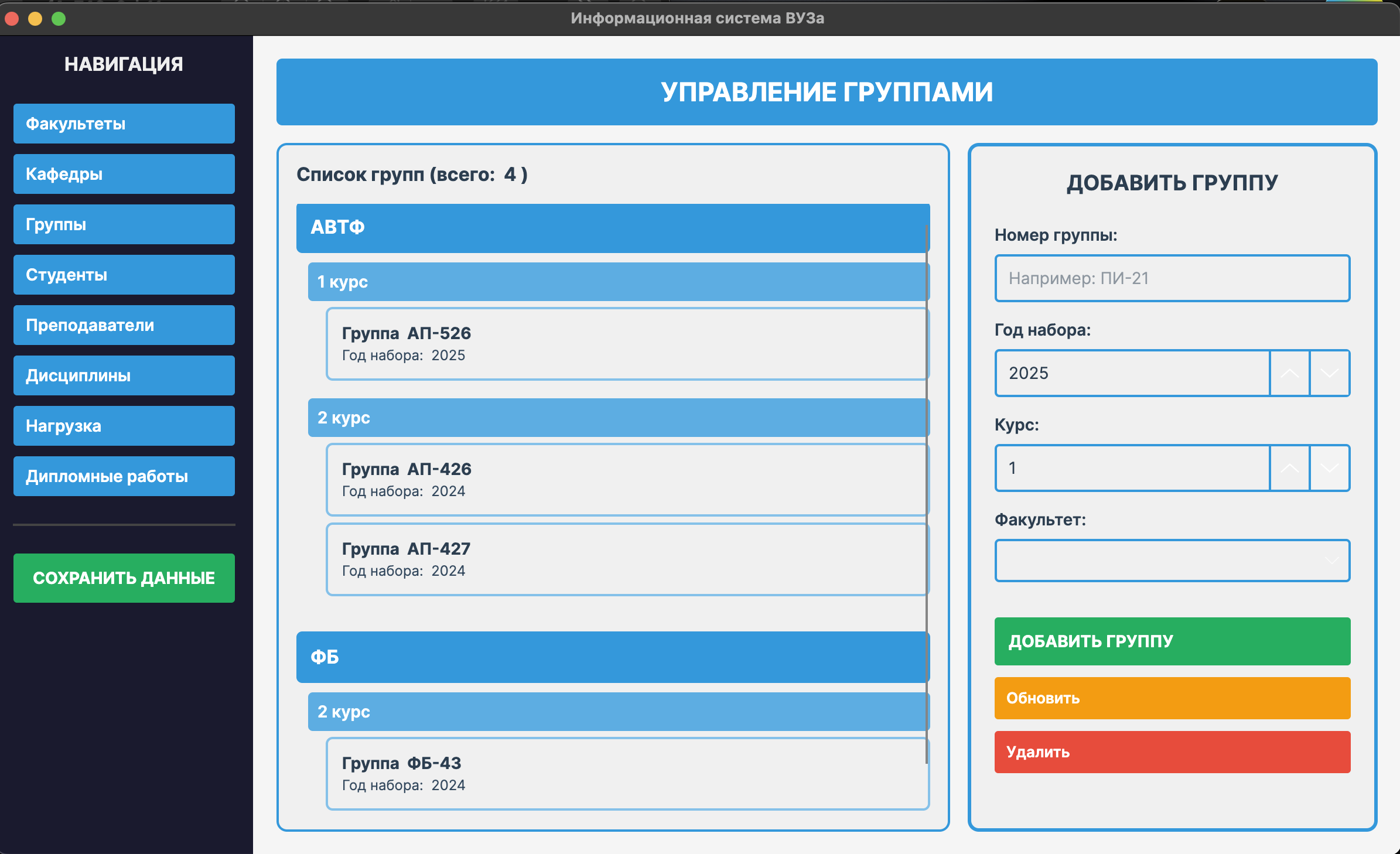
*Рисунок 7 – добавление новой кафедры.*

При вводе названия кафедры и ФИО заведующего, данные проходят валидацию. (рисунок 8)



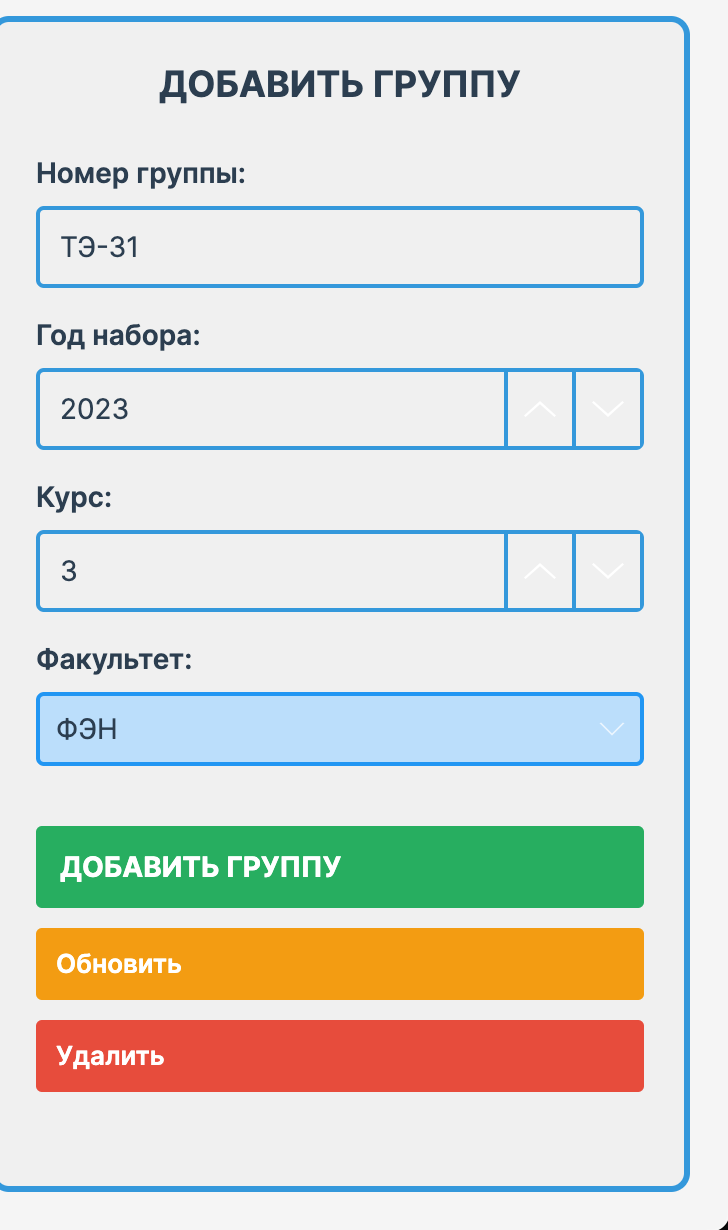
*Рисунок 8 – сообщение о некорректном вводе данных.*

Следующее окно – окно с группами. Группы распределены в первую очередь по факультетам, а уже внутри факультетов по курсам. (рисунок 9)



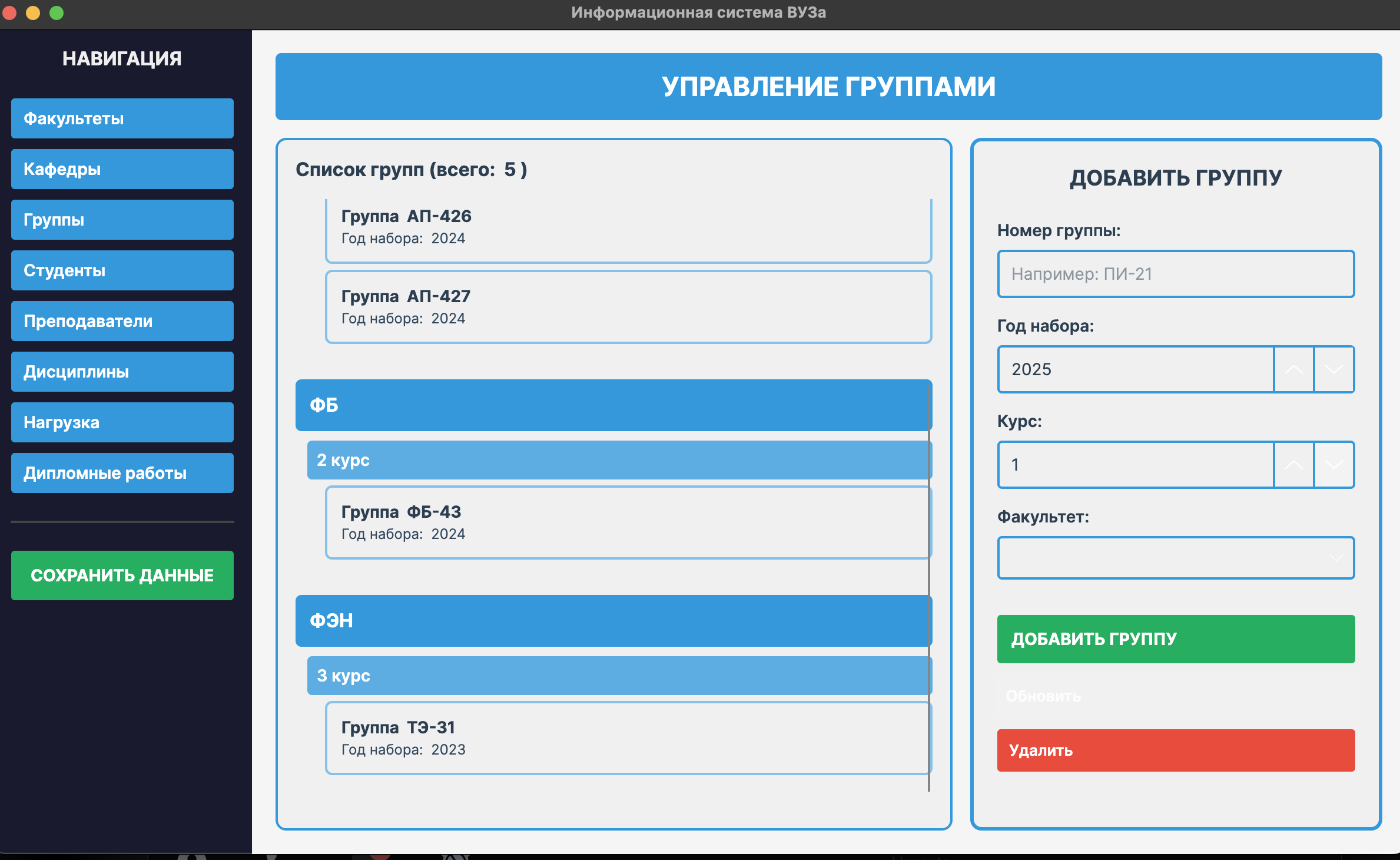
*Рисунок 9 – окно с группами*

При добавлении новой группы необходимо ввести ее название, год набора, курс и выбрать факультет, к которому она относится. (рисунок 10)



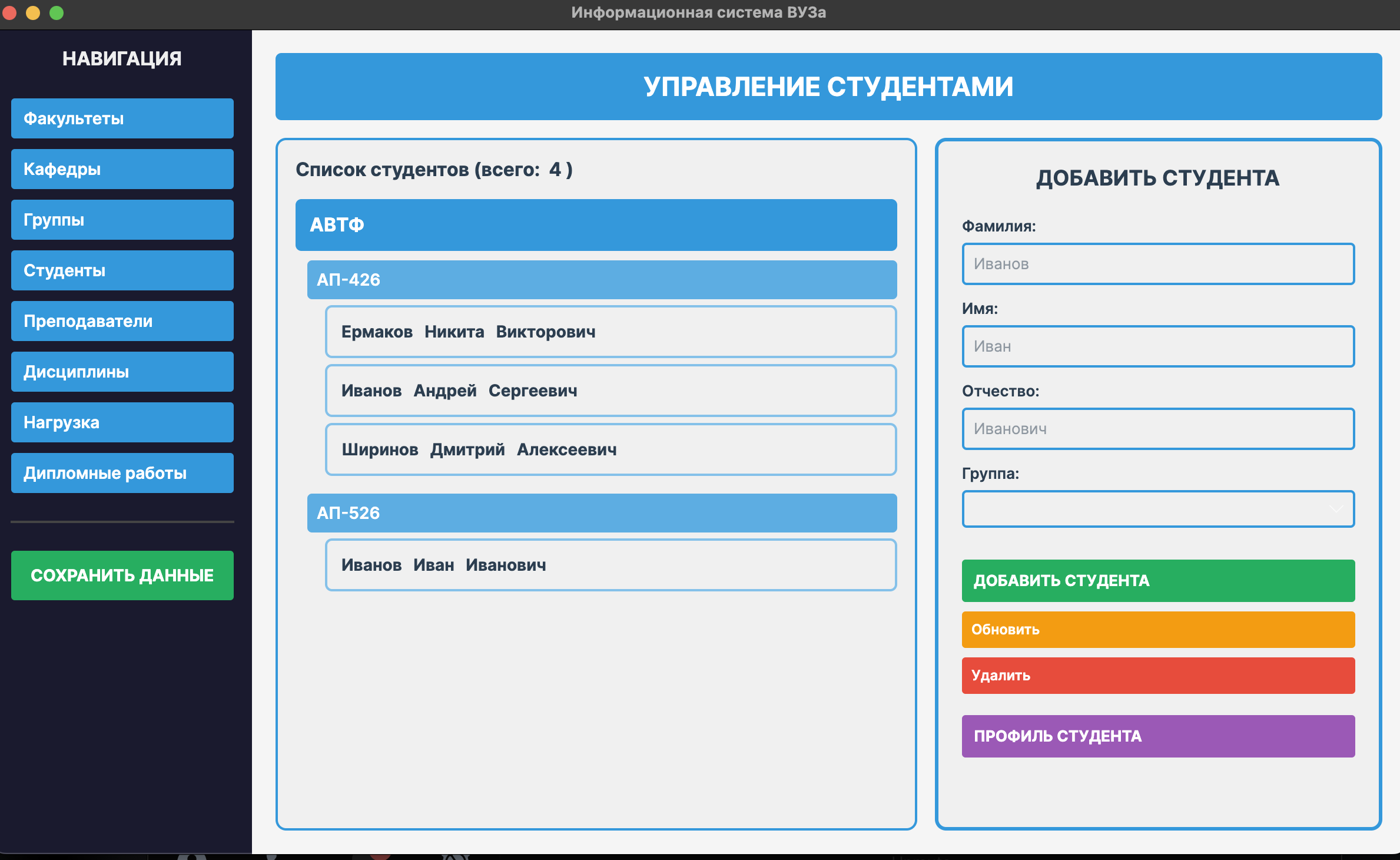
*Рисунок 10 – добавление новой группы.*

После успешного добавления (после прохождения всех фалидационных тестов) группа отобразится в общем списке. (рисунок 11)



*Рисунок 11 – Отображение добавленной группы.*

Следующее окно – окно со списками всех студентов. Списки студентов сгруппированы по своим группам и факультетам. Внутри своих групп они выстроены в алфавитном порядке. (рисунок 12)

**

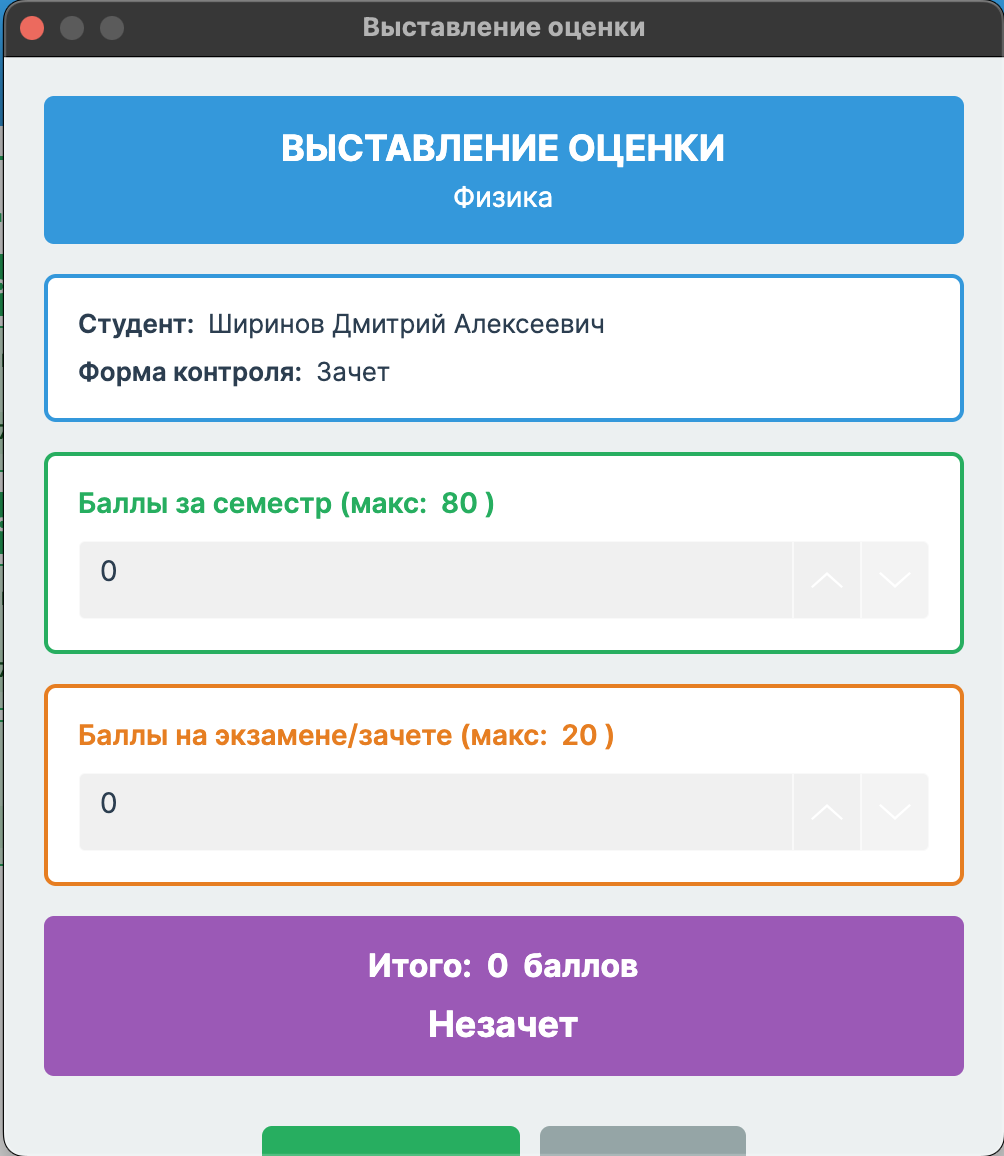
*Рисунок 12 – Окно со студентами.*

Как и в прошлых окнах, есть возможность добавления и удаления студента из списка и обновления информации о уже существующем студенте. При нажатии на студента из списка можно перейти на его профиль через кнопку в боковом меню. (рисунок 13)



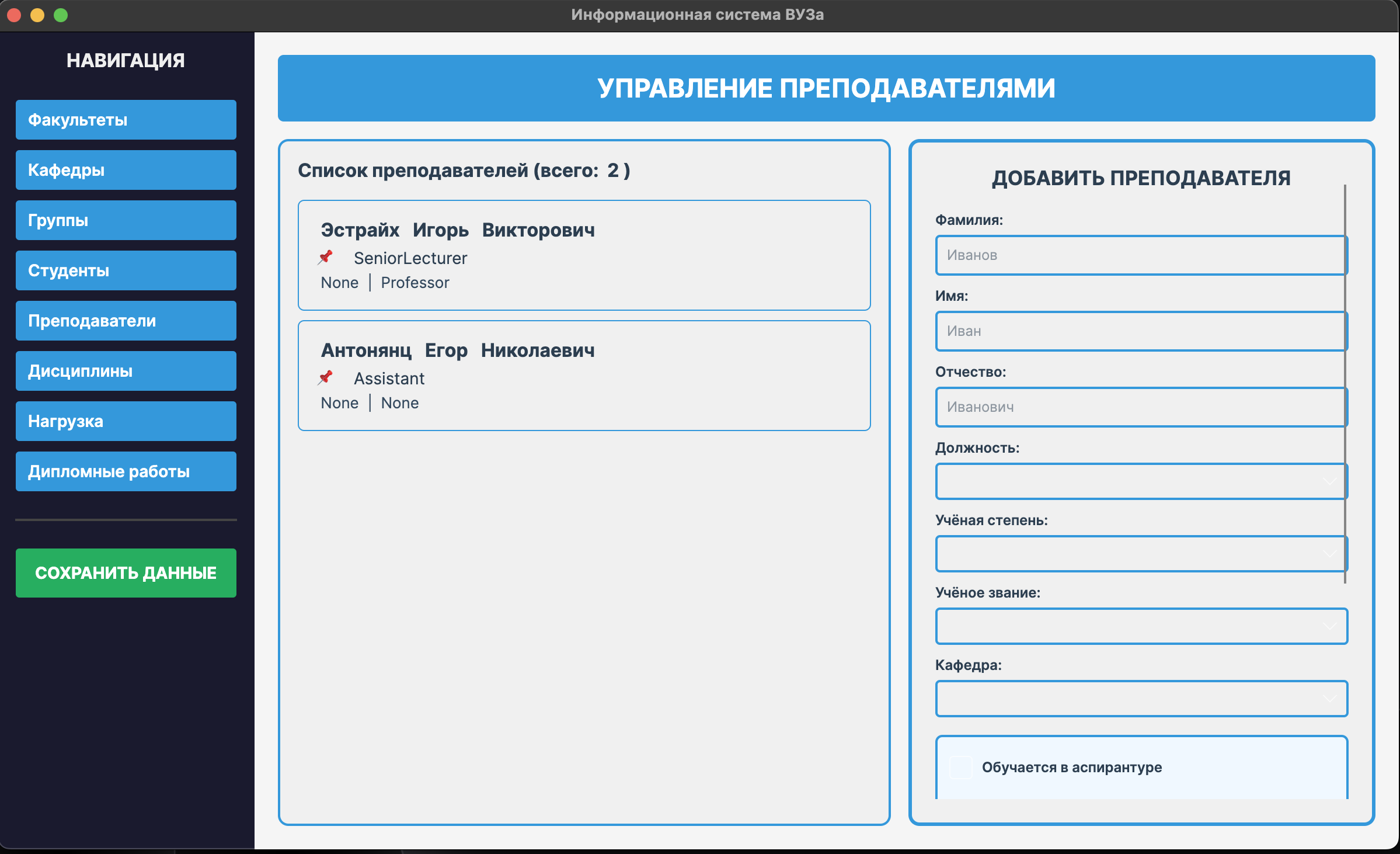
*Рисунок 13 – Профиль студента.*

В профиле отображаются уже пройденные дисциплины и полученные по ним оценки, текущие дисциплины и будущие. Есть возможность обновления оценки за пройденные предметы и выставление оценки за предмет в текущем семестре. При выборе предмета откроется окно, в котором надо будет ввести полученные студентом баллы. Далее система сама считает, какую оценку ему ставить, исходя из этих баллов. (рисунок 14)



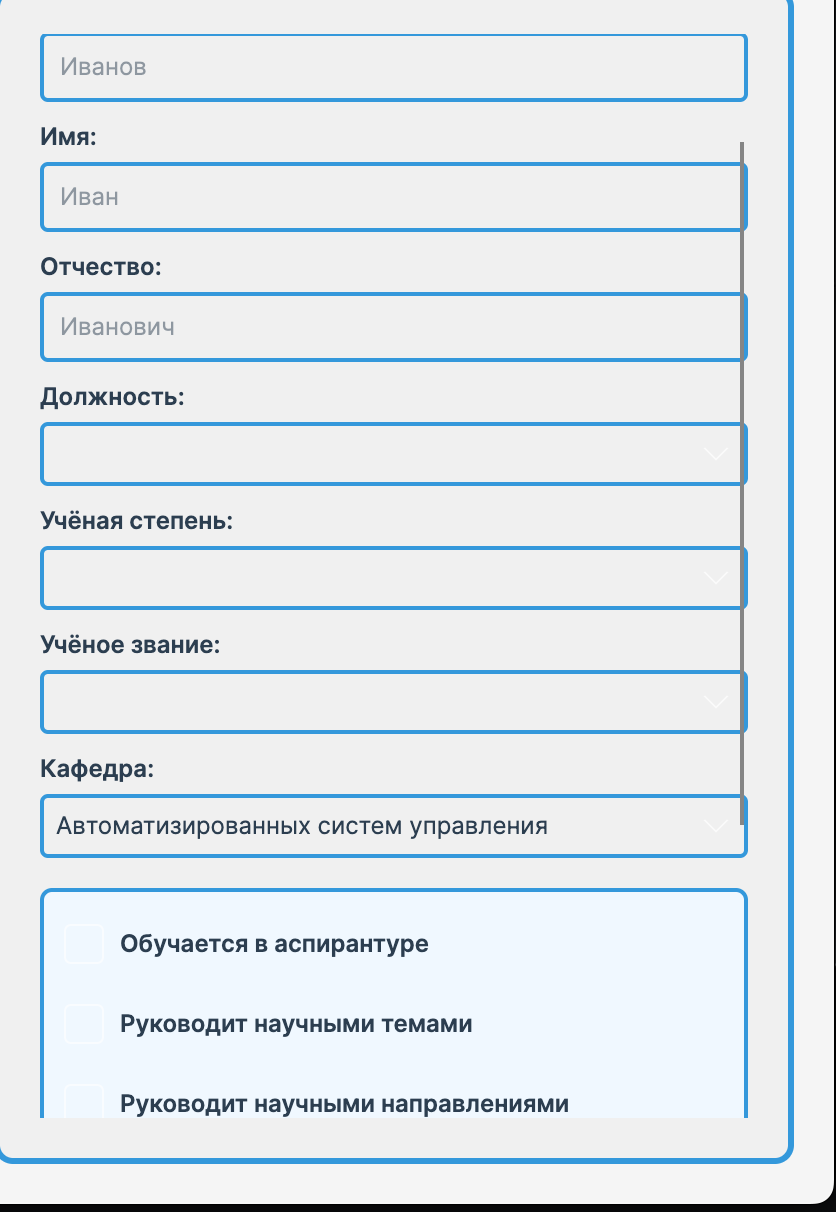
*Рисунок 14 – Окно Выставления оценки.*

Следующее окно – окно с преподавателями. Отображается список преподавателей, а также есть возможность добавления, редактирования и удаления преподавателя из списка. (рисунок 15)



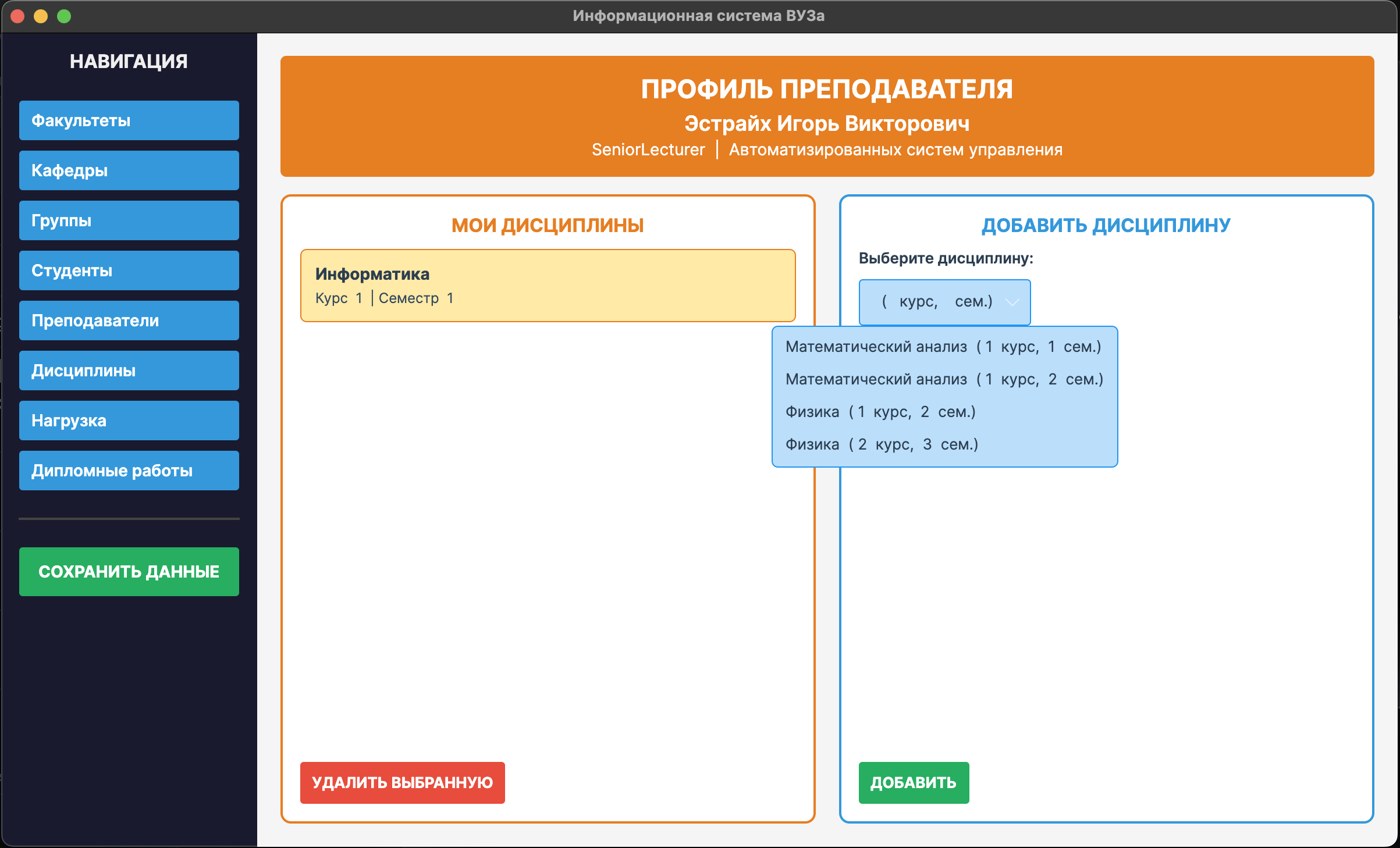
*Рисунок 15 – Окно с преподавателями.*

При добавлении преподавателя необходимо не только указать его ФИО, но и выбрать должность из предложенного списка (ассистент, преподаватель, старший преподаватель, доцент, профессор), ученую степень (без степени, кандидат наук, доктор наук), учебное звание (без звания, доцент, профессор), выбрать кафедру, на которой он будет числиться, и отметить его особенные характеристики, если они есть (обучается в аспирантуре, руководит научными темами, руководит научными направлениями). (рисунок 16)



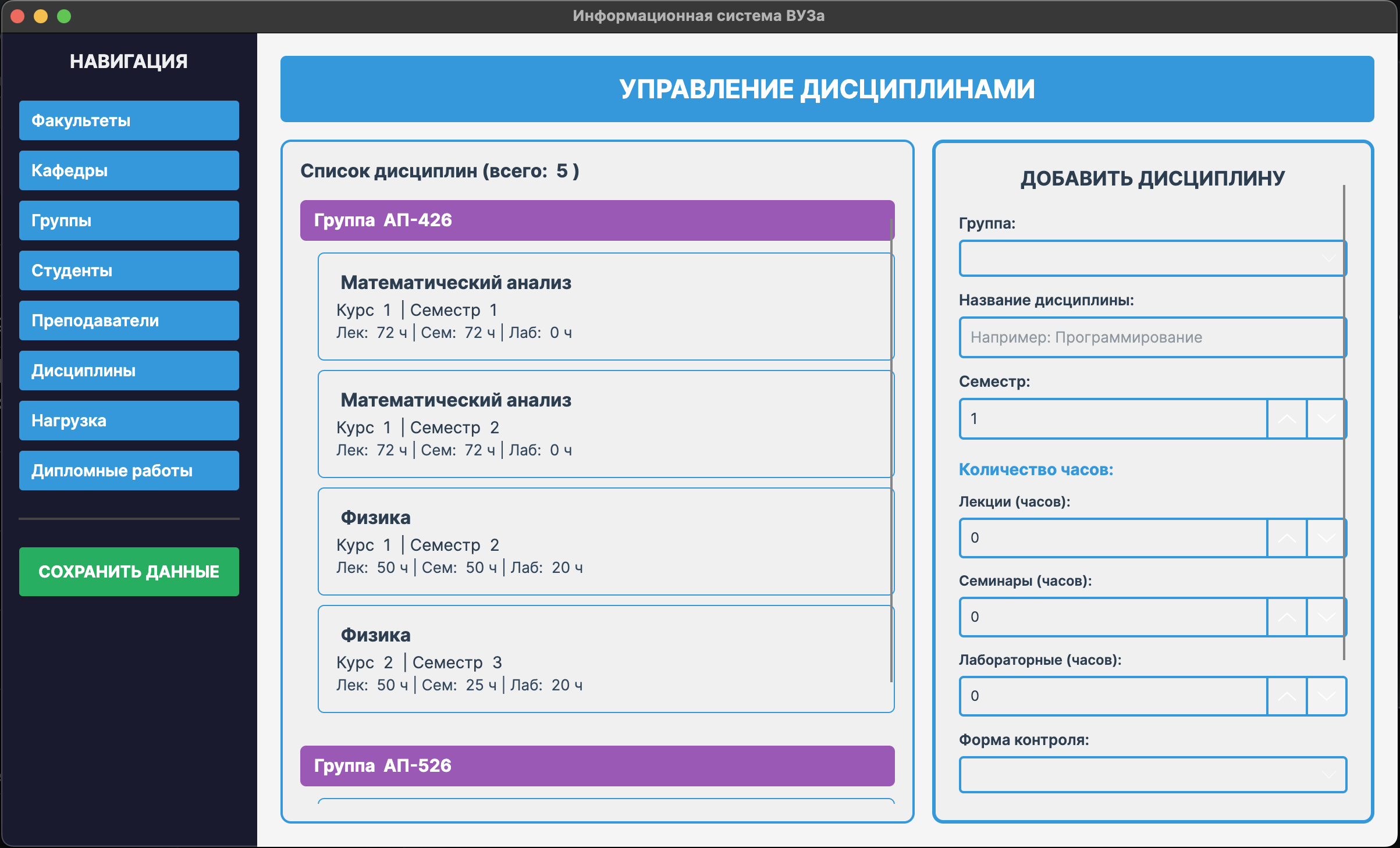
*Рисунок 16 – Добавление информации о преподавателе.*

Есть возможность перейти в профиль преподавателя, где можно увидеть, какие дисциплины он преподает, а также добавить новую. (рисунок 17)



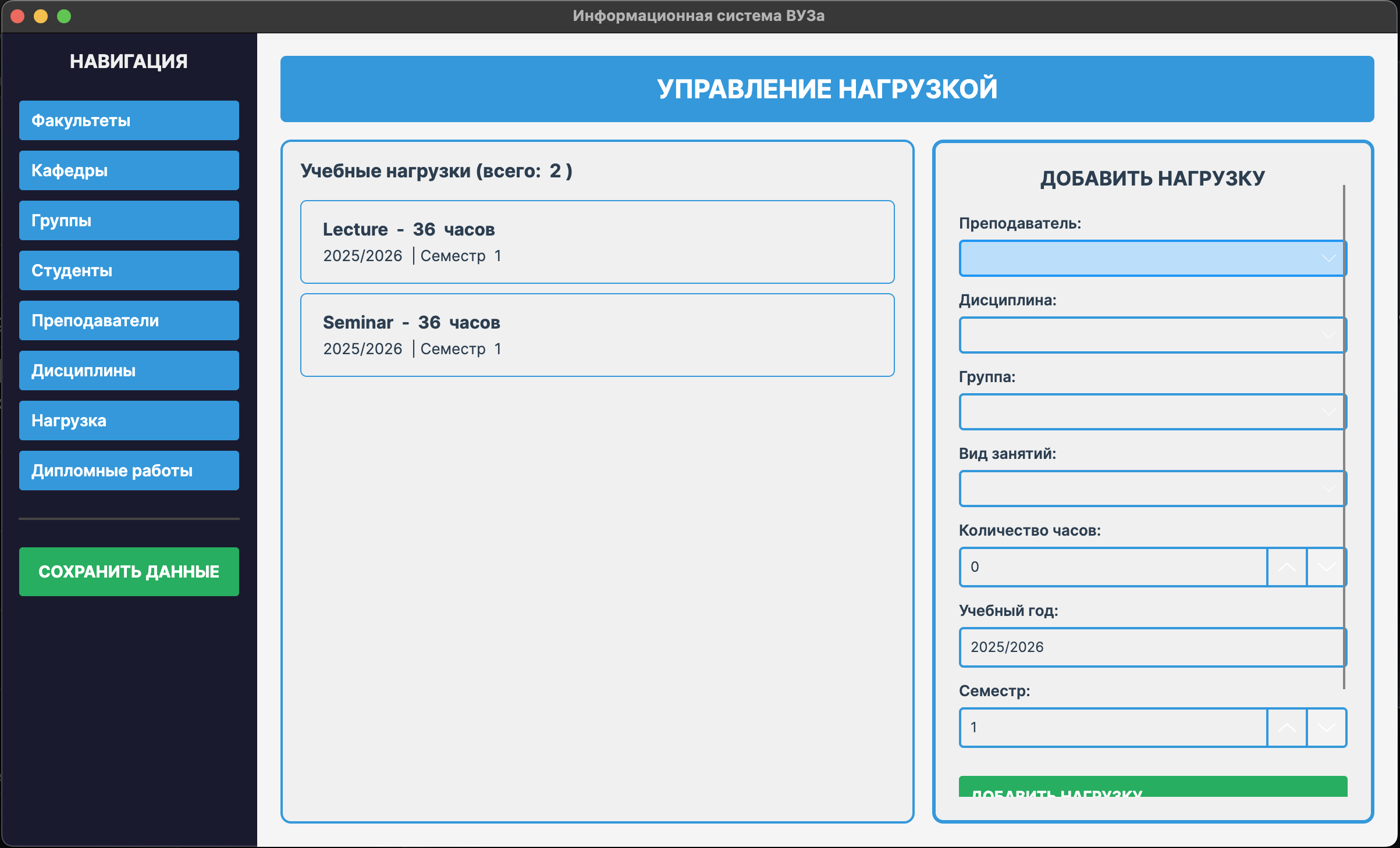
*Рисунок 17 – Профиль преподавателя.*

Следующее окно – окно с дисциплинами. Там отображаются дисциплины, которые были, есть и будут у каждой группы в хронологическом порядке. При добавлении дисциплины необходимо выбрать группу, для которой она будет, название, указать семестр, указать количество часов лекций, семинаров, лабораторных работ и выбрать форму контроля. (рисунок 18)



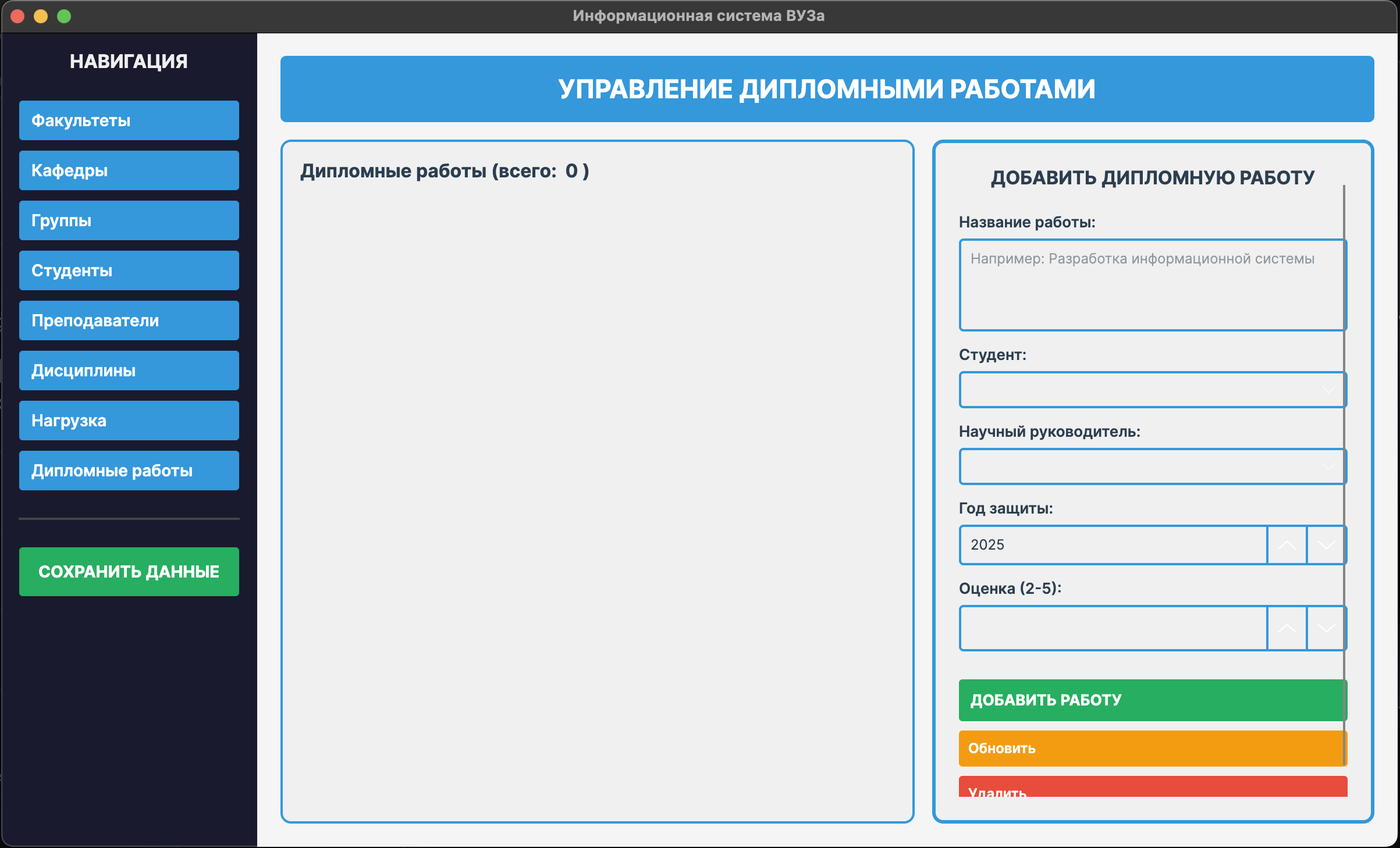
*Рисунок 18 – Окно с дисциплинами.*

Следующее окно с отображением и добавлением нагрузки преподавателям. Предлагается выбрать преподавателя, дисциплину, группу, у которой он будет ее вести, выбрать вид занятий (семинары, лекции, и т.д.), задать количество часов, учебный год и семестр. (рисунок 19)



*Рисунок 19 – Управление нагрузкой.*

Последнее окно – окно с дипломными работами. В этом окне отображается список сданных дипломных работ. Есть возможность добавления, обновления или удаления работы. Предлагается ввести название, выбрать студента, который ее выполнил, научного руководителя, год защиты и полученную оценку. (рисунок 20)



*Рисунок 20 – Окно с дипломными работами.*

# **Заключение**

В ходе выполнения курсовой работы были применены приемы, раскрывающие реальный потенциал объектно–ориентированного программирования. Парадигмы ООП позволяют упростить многие моменты разработки. Наследование и динамический полиморфизм позволили упростить и ускорить разработку [1].

Программа, написанная в ходе выполнения работы, обладает функционалом, необходимым для складского учёта. Программный продукт является устойчивым к некорректному вводу данных, интерфейс программы интуитивно понятный и приятный. Таким образом, программа обладает функционалом необходимым для использования на складах в качестве системы учёта [1].

Исходя из вышесказанного, получаем, что поставленная задача успешно выполнена.

# **Список использованной литературы**

1. Вагнер, Билл С# Эффективное программирование / Билл Вагнер. – М.: ЛОРИ, 2013. – 320 c.
2. Албахари, Джозеф C# 3.0. Справочник / Джозеф Албахари , Бен Албахари. – М.: БХВ–Петербург, 2012. – 944 c.
3. Ишкова, Э. А. Самоучитель С#. Начало программирования / Э.А. Ишкова. – М.: Наука и техника, 2013. – 496 c

# **Приложение. Листинг разработанной программы**

Файл **Program.cs** – файл инициализации программы:

using Avalonia;

using Avalonia.ReactiveUI;

using System;

namespace UniversityIS;

*// Точка входа в приложение*

*// Инициализирует Avalonia UI и запускает приложение*

sealed class Program

{

*// Главный метод приложения – точка входа*

*// Настраивает и запускает Avalonia приложение*

[STAThread]

public static void Main(string[] args) => BuildAvaloniaApp()

.StartWithClassicDesktopLifetime(args);

*// Настройка Avalonia приложения*

*// Конфигурирует платформу, шрифты, логирование и ReactiveUI*

public static AppBuilder BuildAvaloniaApp()

=> AppBuilder.Configure<App>()

.UsePlatformDetect()

.WithInterFont()

.LogToTrace()

.UseReactiveUI();

}

**App.axaml** – файл разметки главного приложения в формате AXAML, определяет ресурсы и стили приложения.

<Application xmlns="https://github.com/avaloniaui"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

x:Class="UniversityIS.App"

xmlns:local="using:UniversityIS"

RequestedThemeVariant="Default">

*<!-- "Default" ThemeVariant follows system theme variant. "Dark" or "Light" are other available options. -->*

<Application.DataTemplates>

<local:ViewLocator/>

</Application.DataTemplates>

<Application.Styles>

<FluentTheme />

*<!-- Глобальные стили для полей ввода -->*

<Style Selector="TextBox">

<Setter Property="Foreground" Value="#2c3e50"/>

</Style>

<Style Selector="TextBox:pointerover">

<Setter Property="Background" Value="#E3F2FD"/>

</Style>

<Style Selector="TextBox:focus">

<Setter Property="Background" Value="#BBDEFB"/>

<Setter Property="BorderBrush" Value="#2196F3"/>

</Style>

<Style Selector="NumericUpDown">

<Setter Property="Foreground" Value="#2c3e50"/>

</Style>

<Style Selector="NumericUpDown:pointerover /template/ Border">

<Setter Property="Background" Value="#E3F2FD"/>

</Style>

<Style Selector="NumericUpDown:focus /template/ Border">

<Setter Property="Background" Value="#BBDEFB"/>

<Setter Property="BorderBrush" Value="#2196F3"/>

</Style>

<Style Selector="ComboBox">

<Setter Property="Foreground" Value="#2c3e50"/>

</Style>

<Style Selector="ComboBox:pointerover /template/ Border">

<Setter Property="Background" Value="#E3F2FD"/>

</Style>

<Style Selector="ComboBox:focus /template/ Border">

<Setter Property="Background" Value="#BBDEFB"/>

<Setter Property="BorderBrush" Value="#2196F3"/>

</Style>

*<!-- Стили для кнопок - прозрачность при наведении -->*

<Style Selector="Button:pointerover /template/ ContentPresenter">

<Setter Property="Opacity" Value="0.85"/>

</Style>

<Style Selector="Button:pressed /template/ ContentPresenter">

<Setter Property="Opacity" Value="0.7"/>

</Style>

*<!-- Альтернативный подход - изменение фона напрямую -->*

<Style Selector="Button:pointerover">

<Setter Property="RenderTransform" Value="scale(1.02)"/>

</Style>

*<!-- Стили для улучшения читаемости текста в списках и карточках -->*

<Style Selector="TextBlock">

<Setter Property="Foreground" Value="#2c3e50"/>

</Style>

<Style Selector="ListBoxItem TextBlock">

<Setter Property="Foreground" Value="#2c3e50"/>

</Style>

<Style Selector="Border TextBlock">

<Setter Property="Foreground" Value="#2c3e50"/>

</Style>

*<!-- Стили для CheckBox - темный текст и видимые рамки -->*

<Style Selector="CheckBox">

<Setter Property="Foreground" Value="#2c3e50"/>

</Style>

<Style Selector="CheckBox /template/ Border#NormalRectangle">

<Setter Property="BorderBrush" Value="#2c3e50"/>

<Setter Property="BorderThickness" Value="2"/>

<Setter Property="Background" Value="White"/>

</Style>

<Style Selector="CheckBox:pointerover /template/ Border#NormalRectangle">

<Setter Property="BorderBrush" Value="#3498db"/>

<Setter Property="Background" Value="#E3F2FD"/>

</Style>

<Style Selector="CheckBox:checked /template/ Border#NormalRectangle">

<Setter Property="BorderBrush" Value="#27ae60"/>

<Setter Property="Background" Value="#d4edda"/>

</Style>

</Application.Styles>

</Application>

**App.axaml.cs** – файл кода главного приложения, инициализирует XAML ресурсы и создает главное окно с ViewModel.

using Avalonia;

using Avalonia.Controls.ApplicationLifetimes;

using Avalonia.Data.Core;

using Avalonia.Data.Core.Plugins;

using System.Linq;

using Avalonia.Markup.Xaml;

using UniversityIS.ViewModels;

using UniversityIS.Views;

namespace UniversityIS;

*// Главный класс приложения*

*// Инициализирует XAML и создает главное окно с ViewModel*

public partial class App : Application

{

*// Загружает XAML ресурсы приложения*

public override void Initialize()

{

AvaloniaXamlLoader.Load(this);

}

*// Вызывается после инициализации фреймворка*

*// Создает главное окно и настраивает приложение*

public override void OnFrameworkInitializationCompleted()

{

if (ApplicationLifetime is IClassicDesktopStyleApplicationLifetime desktop)

{

*// Отключаем дублирующую валидацию от Avalonia*

*// Используем только нашу собственную валидацию*

DisableAvaloniaDataAnnotationValidation();

*// Создаем главное окно и привязываем к нему ViewModel*

desktop.MainWindow = new MainWindow

{

DataContext = new MainWindowViewModel(),

};

}

base.OnFrameworkInitializationCompleted();

}

*// Отключает встроенную валидацию Avalonia для избежания конфликтов*

private void DisableAvaloniaDataAnnotationValidation()

{

var dataValidationPluginsToRemove =

BindingPlugins.DataValidators.OfType<DataAnnotationsValidationPlugin>().ToArray();

foreach (var plugin in dataValidationPluginsToRemove)

{

BindingPlugins.DataValidators.Remove(plugin);

}

}

}

Файлы моделей данных (Models)

**ModelBase.cs** – абстрактный базовый класс для всех моделей данных, содержит общее свойство Id и абстрактный метод ToFileString().

using System;

using ReactiveUI;

namespace UniversityIS.Models

{

*// Базовый класс для всех моделей данных приложения*

*// Наследуется от ReactiveObject для поддержки реактивного программирования*

*// Содержит общие свойства и методы для всех моделей*

public abstract class ModelBase : ReactiveObject

{

private Guid \_id;

*// Уникальный идентификатор модели*

public Guid Id

{

get => \_id;

set => this.RaiseAndSetIfChanged(ref \_id, value);

}

protected ModelBase()

{

Id = Guid.NewGuid();

}

*// Абстрактный метод преобразования объекта в строку для сохранения в текстовый файл*

*// Каждый дочерний класс должен реализовать свой формат сериализации*

public abstract string ToFileString();

*// Виртуальный метод для возврата строкового представления объекта*

*// Может быть переопределен в дочерних классах*

public override string ToString()

{

return Id.ToString();

}

}

}

**Faculty.cs** – модель факультета университета, содержит информацию о факультете, декане, списках кафедр и групп.

using System;

using System.Collections.Generic;

using System.Linq;

using ReactiveUI;

namespace UniversityIS.Models

{

*// Модель факультета университета*

*// Содержит информацию о факультете: название, декан, список групп и кафедр*

public class Faculty : ModelBase

{

private string \_name = string.Empty;

private string \_dean = string.Empty;

*// Название факультета*

public string Name

{

get => \_name;

set => this.RaiseAndSetIfChanged(ref \_name, value);

}

*// ФИО декана факультета*

public string Dean

{

get => \_dean;

set => this.RaiseAndSetIfChanged(ref \_dean, value);

}

*// Список идентификаторов групп, принадлежащих факультету*

public List<Guid> GroupIds { get; set; }

*// Список идентификаторов кафедр, принадлежащих факультету*

public List<Guid> DepartmentIds { get; set; }

public Faculty()

{

Name = string.Empty;

Dean = string.Empty;

GroupIds = new List<Guid>();

DepartmentIds = new List<Guid>();

}

public override string ToString()

{

return Name;

}

*// Преобразует объект факультета в строку для сохранения в текстовый файл*

*// Формат: Id|Name|Dean|GroupIds|DepartmentIds*

public override string ToFileString()

{

return $"{Id}|{Name}|{Dean}|{string.Join(",", GroupIds)}|{string.Join(",", DepartmentIds)}";

}

*// Создает объект факультета из строки, загруженной из текстового файла*

*// Парсит строку формата: Id|Name|Dean|GroupIds|DepartmentIds*

public static Faculty FromFileString(string line)

{

var parts = line.Split('|');

var faculty = new Faculty

{

Id = Guid.Parse(parts[0]),

Name = parts[1],

Dean = parts[2]

};

if (!string.IsNullOrEmpty(parts[3]))

faculty.GroupIds = parts[3].Split(',').Select(Guid.Parse).ToList();

if (parts.Length > 4 && !string.IsNullOrEmpty(parts[4]))

faculty.DepartmentIds = parts[4].Split(',').Select(Guid.Parse).ToList();

return faculty;

}

}

}

**Department.cs** – модель кафедры, содержит информацию о кафедре, заведующем и связи с факультетом.

using System;

using System.Collections.Generic;

using System.Linq;

using ReactiveUI;

namespace UniversityIS.Models

{

*// Кафедра*

public class Department : ModelBase

{

private string \_name = string.Empty;

private string \_head = string.Empty;

private Guid \_facultyId;

*// Название кафедры*

public string Name

{

get => \_name;

set => this.RaiseAndSetIfChanged(ref \_name, value);

}

*// Заведующий кафедрой*

public string Head

{

get => \_head;

set => this.RaiseAndSetIfChanged(ref \_head, value);

}

*// Идентификатор факультета, к которому относится кафедра*

public Guid FacultyId

{

get => \_facultyId;

set => this.RaiseAndSetIfChanged(ref \_facultyId, value);

}

*// Список преподавателей кафедры*

public List<Guid> TeacherIds { get; set; }

public Department()

{

Name = string.Empty;

Head = string.Empty;

FacultyId = Guid.Empty;

TeacherIds = new List<Guid>();

}

public override string ToString()

{

return Name;

}

*// Сохранение в строку для текстового файла*

public override string ToFileString()

{

return $"{Id}|{Name}|{Head}|{FacultyId}|{string.Join(",", TeacherIds)}";

}

*// Загрузка из строки текстового файла*

public static Department FromFileString(string line)

{

var parts = line.Split('|');

var department = new Department

{

Id = Guid.Parse(parts[0]),

Name = parts[1],

Head = parts[2],

FacultyId = Guid.Parse(parts[3])

};

if (parts.Length > 4 && !string.IsNullOrEmpty(parts[4]))

department.TeacherIds = parts[4].Split(',').Select(Guid.Parse).ToList();

return department;

}

}

}

**Group.cs** – модель учебной группы, содержит информацию о группе, курсе, годе набора и связи с факультетом.

using System;

using System.Collections.Generic;

using System.Linq;

using ReactiveUI;

namespace UniversityIS.Models

{

*// Учебная группа*

public class Group : ModelBase

{

private string \_number = string.Empty;

private int \_yearOfAdmission;

private int \_course;

private Guid \_facultyId;

*// Номер группы*

public string Number

{

get => \_number;

set => this.RaiseAndSetIfChanged(ref \_number, value);

}

*// Год набора*

public int YearOfAdmission

{

get => \_yearOfAdmission;

set => this.RaiseAndSetIfChanged(ref \_yearOfAdmission, value);

}

*// Курс*

public int Course

{

get => \_course;

set => this.RaiseAndSetIfChanged(ref \_course, value);

}

*// Идентификатор факультета*

public Guid FacultyId

{

get => \_facultyId;

set => this.RaiseAndSetIfChanged(ref \_facultyId, value);

}

*// Список студентов группы*

public List<Guid> StudentIds { get; set; }

public Group()

{

Number = string.Empty;

YearOfAdmission = DateTime.Now.Year;

Course = 1;

FacultyId = Guid.Empty;

StudentIds = new List<Guid>();

}

public override string ToString()

{

return $"{Number} ({Course} курс, {YearOfAdmission})";

}

*// Сохранение в строку для текстового файла*

public override string ToFileString()

{

return $"{Id}|{Number}|{YearOfAdmission}|{Course}|{FacultyId}|{string.Join(",", StudentIds)}";

}

*// Загрузка из строки текстового файла*

public static Group FromFileString(string line)

{

var parts = line.Split('|');

var group = new Group

{

Id = Guid.Parse(parts[0]),

Number = parts[1],

YearOfAdmission = int.Parse(parts[2]),

Course = int.Parse(parts[3]),

FacultyId = Guid.Parse(parts[4])

};

if (parts.Length > 5 && !string.IsNullOrEmpty(parts[5]))

group.StudentIds = parts[5].Split(',').Select(Guid.Parse).ToList();

return group;

}

}

}

**Student.cs** – модель студента, содержит информацию о студенте, его группе, номере зачётной книжки и среднем балле.

using System;

using ReactiveUI;

namespace UniversityIS.Models

{

*// Студент*

public class Student : ModelBase

{

private string \_lastName = string.Empty;

private string \_firstName = string.Empty;

private string \_middleName = string.Empty;

private Guid \_groupId;

private string \_recordBookNumber = string.Empty;

private double \_gpa;

*// Фамилия*

public string LastName

{

get => \_lastName;

set => this.RaiseAndSetIfChanged(ref \_lastName, value);

}

*// Имя*

public string FirstName

{

get => \_firstName;

set => this.RaiseAndSetIfChanged(ref \_firstName, value);

}

*// Отчество*

public string MiddleName

{

get => \_middleName;

set => this.RaiseAndSetIfChanged(ref \_middleName, value);

}

*// Идентификатор группы*

public Guid GroupId

{

get => \_groupId;

set => this.RaiseAndSetIfChanged(ref \_groupId, value);

}

*// Номер зачётной книжки*

public string RecordBookNumber

{

get => \_recordBookNumber;

set => this.RaiseAndSetIfChanged(ref \_recordBookNumber, value);

}

*// Средний балл*

public double GPA

{

get => \_gpa;

set => this.RaiseAndSetIfChanged(ref \_gpa, value);

}

public Student()

{

LastName = string.Empty;

FirstName = string.Empty;

MiddleName = string.Empty;

GroupId = Guid.Empty;

RecordBookNumber = string.Empty;

GPA = 0.0;

}

*// Полное имя студента*

public string FullName => $"{LastName} {FirstName} {MiddleName}";

public override string ToString()

{

return FullName;

}

*// Сохранение в строку для текстового файла*

public override string ToFileString()

{

return $"{Id}|{LastName}|{FirstName}|{MiddleName}|{GroupId}|{RecordBookNumber}|{GPA}";

}

*// Загрузка из строки текстового файла*

public static Student FromFileString(string line)

{

var parts = line.Split('|');

return new Student

{

Id = Guid.Parse(parts[0]),

LastName = parts[1],

FirstName = parts[2],

MiddleName = parts[3],

GroupId = Guid.Parse(parts[4]),

RecordBookNumber = parts[5],

GPA = double.Parse(parts[6])

};

}

}

}

**Teacher.cs** – модель преподавателя, содержит информацию о преподавателе, его должности, учёной степени, звании и научной деятельности.

using System;

using ReactiveUI;

namespace UniversityIS.Models

{

*// Преподаватель*

public class Teacher : ModelBase

{

private string \_lastName = string.Empty;

private string \_firstName = string.Empty;

private string \_middleName = string.Empty;

private TeacherPosition \_position;

private AcademicDegree \_degree;

private AcademicTitle \_title;

private Guid \_departmentId;

private bool \_isPostgraduate;

private bool \_leadsResearchTopics;

private bool \_leadsResearchDirections;

*// Фамилия*

public string LastName

{

get => \_lastName;

set => this.RaiseAndSetIfChanged(ref \_lastName, value);

}

*// Имя*

public string FirstName

{

get => \_firstName;

set => this.RaiseAndSetIfChanged(ref \_firstName, value);

}

*// Отчество*

public string MiddleName

{

get => \_middleName;

set => this.RaiseAndSetIfChanged(ref \_middleName, value);

}

*// Должность*

public TeacherPosition Position

{

get => \_position;

set => this.RaiseAndSetIfChanged(ref \_position, value);

}

*// Учёная степень*

public AcademicDegree Degree

{

get => \_degree;

set => this.RaiseAndSetIfChanged(ref \_degree, value);

}

*// Учёное звание*

public AcademicTitle Title

{

get => \_title;

set => this.RaiseAndSetIfChanged(ref \_title, value);

}

*// Идентификатор кафедры*

public Guid DepartmentId

{

get => \_departmentId;

set => this.RaiseAndSetIfChanged(ref \_departmentId, value);

}

*// Обучается в аспирантуре*

public bool IsPostgraduate

{

get => \_isPostgraduate;

set => this.RaiseAndSetIfChanged(ref \_isPostgraduate, value);

}

*// Руководит научными темами*

public bool LeadsResearchTopics

{

get => \_leadsResearchTopics;

set => this.RaiseAndSetIfChanged(ref \_leadsResearchTopics, value);

}

*// Руководит научными направлениями*

public bool LeadsResearchDirections

{

get => \_leadsResearchDirections;

set => this.RaiseAndSetIfChanged(ref \_leadsResearchDirections, value);

}

public Teacher()

{

LastName = string.Empty;

FirstName = string.Empty;

MiddleName = string.Empty;

Position = TeacherPosition.Assistant;

Degree = AcademicDegree.None;

Title = AcademicTitle.None;

DepartmentId = Guid.Empty;

IsPostgraduate = false;

LeadsResearchTopics = false;

LeadsResearchDirections = false;

}

*// Полное имя преподавателя*

public string FullName => $"{LastName} {FirstName} {MiddleName}";

*// Полная информация о преподавателе с должностью и званиями*

public string FullInfo

{

get

{

var info = FullName;

if (Degree != AcademicDegree.None)

info += $", {GetDegreeString(Degree)}";

if (Title != AcademicTitle.None)

info += $", {GetTitleString(Title)}";

info += $", {GetPositionString(Position)}";

return info;

}

}

public override string ToString()

{

return FullInfo;

}

*// Проверка возможности занимать текущую должность с текущим званием*

public bool IsPositionValid()

{

*// Доцент может быть только со званием доцента или выше*

if (Position == TeacherPosition.AssociateProfessor &&

Title != AcademicTitle.AssociateProfessor && Title != AcademicTitle.Professor)

return false;

*// Профессор может быть только со званием профессора*

if (Position == TeacherPosition.Professor && Title != AcademicTitle.Professor)

return false;

return true;

}

*// Может ли преподаватель читать лекции*

public bool CanTeachLectures()

{

return Position != TeacherPosition.Assistant;

}

*// Может ли преподаватель проводить лабораторные работы*

public bool CanTeachLaboratory()

{

return Position != TeacherPosition.Professor;

}

*// Сохранение в строку для текстового файла*

public override string ToFileString()

{

return $"{Id}|{LastName}|{FirstName}|{MiddleName}|{(int)Position}|{(int)Degree}|" +

$"{(int)Title}|{DepartmentId}|{IsPostgraduate}|{LeadsResearchTopics}|{LeadsResearchDirections}";

}

*// Загрузка из строки текстового файла*

public static Teacher FromFileString(string line)

{

var parts = line.Split('|');

return new Teacher

{

Id = Guid.Parse(parts[0]),

LastName = parts[1],

FirstName = parts[2],

MiddleName = parts[3],

Position = (TeacherPosition)int.Parse(parts[4]),

Degree = (AcademicDegree)int.Parse(parts[5]),

Title = (AcademicTitle)int.Parse(parts[6]),

DepartmentId = Guid.Parse(parts[7]),

IsPostgraduate = bool.Parse(parts[8]),

LeadsResearchTopics = bool.Parse(parts[9]),

LeadsResearchDirections = bool.Parse(parts[10])

};

}

private static string GetPositionString(TeacherPosition position)

{

return position switch

{

TeacherPosition.Assistant => "ассистент",

TeacherPosition.Lecturer => "преподаватель",

TeacherPosition.SeniorLecturer => "ст. преподаватель",

TeacherPosition.AssociateProfessor => "доцент",

TeacherPosition.Professor => "профессор",

\_ => ""

};

}

private static string GetDegreeString(AcademicDegree degree)

{

return degree switch

{

AcademicDegree.CandidateOfSciences => "к.н.",

AcademicDegree.DoctorOfSciences => "д.н.",

\_ => ""

};

}

private static string GetTitleString(AcademicTitle title)

{

return title switch

{

AcademicTitle.AssociateProfessor => "доцент",

AcademicTitle.Professor => "профессор",

\_ => ""

};

}

}

}

**Discipline.cs** – модель учебной дисциплины, содержит информацию о дисциплине, курсе, семестре, часах и форме контроля.

using System;

using ReactiveUI;

namespace UniversityIS.Models

{

*// Модель учебной дисциплины*

*// Содержит информацию о предмете: название, курс, семестр, часы, форма контроля*

*// Дисциплина привязана к конкретной группе*

public class Discipline : ModelBase

{

private string \_name = string.Empty;

private int \_course;

private int \_semester;

private int \_lectureHours;

private int \_seminarHours;

private int \_laboratoryHours;

private ControlForm \_controlForm;

private Guid \_groupId;

*// Название дисциплины*

public string Name

{

get => \_name;

set => this.RaiseAndSetIfChanged(ref \_name, value);

}

*// Курс, на котором читается дисциплина*

public int Course

{

get => \_course;

set => this.RaiseAndSetIfChanged(ref \_course, value);

}

*// Семестр*

public int Semester

{

get => \_semester;

set => this.RaiseAndSetIfChanged(ref \_semester, value);

}

*// Количество часов лекций*

public int LectureHours

{

get => \_lectureHours;

set => this.RaiseAndSetIfChanged(ref \_lectureHours, value);

}

*// Количество часов семинаров*

public int SeminarHours

{

get => \_seminarHours;

set => this.RaiseAndSetIfChanged(ref \_seminarHours, value);

}

*// Количество часов лабораторных работ*

public int LaboratoryHours

{

get => \_laboratoryHours;

set => this.RaiseAndSetIfChanged(ref \_laboratoryHours, value);

}

*// Форма контроля*

public ControlForm ControlForm

{

get => \_controlForm;

set => this.RaiseAndSetIfChanged(ref \_controlForm, value);

}

*// Идентификатор группы*

public Guid GroupId

{

get => \_groupId;

set => this.RaiseAndSetIfChanged(ref \_groupId, value);

}

public Discipline()

{

Name = string.Empty;

Course = 1;

Semester = 1;

LectureHours = 0;

SeminarHours = 0;

LaboratoryHours = 0;

ControlForm = ControlForm.Exam;

}

*// Вычисляет номер курса на основе номера семестра*

*// Формула: 1-2 семестр = 1 курс, 3-4 = 2 курс, 5-6 = 3 курс и т.д.*

public static int CalculateCourseFromSemester(int semester)

{

return (semester + 1) / 2;

}

*// Устанавливает семестр и автоматически пересчитывает курс*

*// Это гарантирует, что курс всегда соответствует семестру*

public void SetSemester(int semester)

{

Semester = semester;

Course = CalculateCourseFromSemester(semester);

}

public override string ToString()

{

return $"{Name} ({Course} курс, {Semester} сем.)";

}

*// Преобразует объект дисциплины в строку для сохранения в текстовый файл*

*// Формат: Id|Name|Course|Semester|LectureHours|SeminarHours|LaboratoryHours|ControlForm|GroupId*

public override string ToFileString()

{

return $"{Id}|{Name}|{Course}|{Semester}|{LectureHours}|{SeminarHours}|{LaboratoryHours}|{(int)ControlForm}|{GroupId}";

}

*// Создает объект дисциплины из строки, загруженной из текстового файла*

*// Поддерживает обратную совместимость со старым форматом (без GroupId)*

public static Discipline FromFileString(string line)

{

var parts = line.Split('|');

return new Discipline

{

Id = Guid.Parse(parts[0]),

Name = parts[1],

Course = int.Parse(parts[2]),

Semester = int.Parse(parts[3]),

LectureHours = int.Parse(parts[4]),

SeminarHours = int.Parse(parts[5]),

LaboratoryHours = int.Parse(parts[6]),

ControlForm = (ControlForm)int.Parse(parts[7]),

GroupId = parts.Length > 8 ? Guid.Parse(parts[8]) : Guid.Empty *// Обратная совместимость*

};

}

}

}

**WorkLoad.cs** – модель учебной нагрузки преподавателя, связывает преподавателя, дисциплину и группу с указанием типа занятий и часов.

using System;

using ReactiveUI;

namespace UniversityIS.Models

{

*// Модель учебной нагрузки преподавателя*

*// Описывает, какой преподаватель ведет какой предмет у какой группы*

*// Содержит информацию о типе занятий (лекции, семинары, лабораторные) и количестве часов*

public class WorkLoad : ModelBase

{

private Guid \_teacherId;

private Guid \_disciplineId;

private Guid \_groupId;

private LessonType \_lessonType;

private int \_hours;

private string \_academicYear = string.Empty;

private int \_semester;

*// Идентификатор преподавателя*

public Guid TeacherId

{

get => \_teacherId;

set => this.RaiseAndSetIfChanged(ref \_teacherId, value);

}

*// Идентификатор дисциплины*

public Guid DisciplineId

{

get => \_disciplineId;

set => this.RaiseAndSetIfChanged(ref \_disciplineId, value);

}

*// Идентификатор группы*

public Guid GroupId

{

get => \_groupId;

set => this.RaiseAndSetIfChanged(ref \_groupId, value);

}

*// Вид занятий*

public LessonType LessonType

{

get => \_lessonType;

set => this.RaiseAndSetIfChanged(ref \_lessonType, value);

}

*// Количество часов*

public int Hours

{

get => \_hours;

set => this.RaiseAndSetIfChanged(ref \_hours, value);

}

*// Учебный год*

public string AcademicYear

{

get => \_academicYear;

set => this.RaiseAndSetIfChanged(ref \_academicYear, value);

}

*// Семестр*

public int Semester

{

get => \_semester;

set => this.RaiseAndSetIfChanged(ref \_semester, value);

}

public WorkLoad()

{

TeacherId = Guid.Empty;

DisciplineId = Guid.Empty;

GroupId = Guid.Empty;

LessonType = LessonType.Lecture;

Hours = 0;

AcademicYear = $"{DateTime.Now.Year}/{DateTime.Now.Year + 1}";

Semester = 1;

}

*// Сохранение в строку для текстового файла*

public override string ToFileString()

{

return $"{Id}|{TeacherId}|{DisciplineId}|{GroupId}|{(int)LessonType}|{Hours}|{AcademicYear}|{Semester}";

}

*// Загрузка из строки текстового файла*

public static WorkLoad FromFileString(string line)

{

var parts = line.Split('|');

return new WorkLoad

{

Id = Guid.Parse(parts[0]),

TeacherId = Guid.Parse(parts[1]),

DisciplineId = Guid.Parse(parts[2]),

GroupId = Guid.Parse(parts[3]),

LessonType = (LessonType)int.Parse(parts[4]),

Hours = int.Parse(parts[5]),

AcademicYear = parts[6],

Semester = int.Parse(parts[7])

};

}

public static string GetLessonTypeString(LessonType lessonType)

{

return lessonType switch

{

LessonType.Lecture => "Лекции",

LessonType.Seminar => "Семинары",

LessonType.Laboratory => "Лабораторные",

LessonType.Consultation => "Консультации",

LessonType.CourseWork => "Курсовая работа",

LessonType.CourseProject => "Курсовой проект",

\_ => ""

};

}

}

}

**ThesisWork.cs** – модель дипломной работы, содержит информацию о дипломной работе, студенте, научном руководителе и оценке.

using System;

using ReactiveUI;

namespace UniversityIS.Models

{

*// Дипломная работа*

public class ThesisWork : ModelBase

{

private string \_title = string.Empty;

private Guid \_studentId;

private Guid \_supervisorId;

private int \_year;

private string \_grade = string.Empty;

*// Название работы*

public string Title

{

get => \_title;

set => this.RaiseAndSetIfChanged(ref \_title, value);

}

*// Идентификатор студента*

public Guid StudentId

{

get => \_studentId;

set => this.RaiseAndSetIfChanged(ref \_studentId, value);

}

*// Идентификатор научного руководителя*

public Guid SupervisorId

{

get => \_supervisorId;

set => this.RaiseAndSetIfChanged(ref \_supervisorId, value);

}

*// Год защиты*

public int Year

{

get => \_year;

set => this.RaiseAndSetIfChanged(ref \_year, value);

}

*// Оценка (2-5, null если не выставлена)*

public int? Grade

{

get => string.IsNullOrEmpty(\_grade) ? null : int.Parse(\_grade);

set => this.RaiseAndSetIfChanged(ref \_grade, value?.ToString() ?? string.Empty);

}

public ThesisWork()

{

Title = string.Empty;

StudentId = Guid.Empty;

SupervisorId = Guid.Empty;

Year = DateTime.Now.Year;

Grade = null;

}

*// Сохранение в строку для текстового файла*

public override string ToFileString()

{

return $"{Id}|{Title}|{StudentId}|{SupervisorId}|{Year}|{\_grade}";

}

*// Загрузка из строки текстового файла*

public static ThesisWork FromFileString(string line)

{

var parts = line.Split('|');

var work = new ThesisWork

{

Id = Guid.Parse(parts[0]),

Title = parts[1],

StudentId = Guid.Parse(parts[2]),

SupervisorId = Guid.Parse(parts[3]),

Year = int.Parse(parts[4])

};

*// Парсим оценку: если пустая строка или не число - null, иначе int*

if (!string.IsNullOrWhiteSpace(parts[5]) && int.TryParse(parts[5], out int gradeValue))

{

work.Grade = gradeValue;

}

else

{

work.Grade = null;

}

return work;

}

}

}

**StudentGrade.cs** – модель оценки студента по дисциплине, хранит баллы за семестр и экзамен, автоматически рассчитывает итоговую оценку.

using System;

using ReactiveUI;

namespace UniversityIS.Models

{

*// Модель оценки студента по дисциплине*

*// Хранит баллы за семестр и экзамен/зачет, автоматически рассчитывает итоговую оценку*

*// Система оценивания: экзамен (60+40), зачет (80+20), дифзачет (80+20)*

public class StudentGrade : ModelBase

{

private Guid \_studentId;

private Guid \_disciplineId;

private int \_semesterPoints;

private int \_examPoints;

private int \_totalPoints;

private string \_grade = string.Empty;

public StudentGrade()

{

}

public Guid StudentId

{

get => \_studentId;

set => this.RaiseAndSetIfChanged(ref \_studentId, value);

}

public Guid DisciplineId

{

get => \_disciplineId;

set => this.RaiseAndSetIfChanged(ref \_disciplineId, value);

}

*// Баллы за работу в семестре*

*// Максимум: 60 для экзамена, 80 для зачета/дифзачета*

public int SemesterPoints

{

get => \_semesterPoints;

set => this.RaiseAndSetIfChanged(ref \_semesterPoints, value);

}

*// Баллы на экзамене или зачете*

*// Максимум: 40 для экзамена, 20 для зачета/дифзачета*

public int ExamPoints

{

get => \_examPoints;

set => this.RaiseAndSetIfChanged(ref \_examPoints, value);

}

*// Общая сумма баллов (семестр + экзамен/зачет)*

*// Используется для определения итоговой оценки*

public int TotalPoints

{

get => \_totalPoints;

set => this.RaiseAndSetIfChanged(ref \_totalPoints, value);

}

*// Итоговая оценка в текстовом виде*

*// Для экзамена/дифзачета: "Отлично (5)", "Хорошо (4)", "Удовлетворительно (3)", "Неудовлетворительно (2)"*

*// Для зачета: "Зачет" или "Незачет"*

public string Grade

{

get => \_grade;

set => this.RaiseAndSetIfChanged(ref \_grade, value);

}

*// Рассчитывает итоговую оценку на основе набранных баллов и формы контроля*

*// Шкала оценивания:*

*// - Экзамен/Дифзачет: 0-49 (2), 50-72 (3), 73-86 (4), 87-100 (5)*

*// - Зачет: 0-49 (незачет), 50-100 (зачет)*

public void CalculateGrade(ControlForm controlForm)

{

TotalPoints = SemesterPoints + ExamPoints;

if (controlForm == ControlForm.Exam || controlForm == ControlForm.DifferentiatedPass)

{

*// Для экзамена и дифференцированного зачета*

if (TotalPoints < 50)

Grade = "Неудовлетворительно (2)";

else if (TotalPoints <= 72)

Grade = "Удовлетворительно (3)";

else if (TotalPoints <= 86)

Grade = "Хорошо (4)";

else

Grade = "Отлично (5)";

}

else *// Credit (зачет)*

{

Grade = TotalPoints >= 50 ? "Зачет" : "Незачет";

}

}

public override string ToFileString()

{

return $"{Id}|{StudentId}|{DisciplineId}|{SemesterPoints}|{ExamPoints}|{TotalPoints}|{Grade}";

}

public static StudentGrade FromFileString(string line)

{

var parts = line.Split('|');

return new StudentGrade

{

Id = Guid.Parse(parts[0]),

StudentId = Guid.Parse(parts[1]),

DisciplineId = Guid.Parse(parts[2]),

SemesterPoints = int.Parse(parts[3]),

ExamPoints = int.Parse(parts[4]),

TotalPoints = int.Parse(parts[5]),

Grade = parts[6]

};

}

public override string ToString()

{

return $"{TotalPoints} баллов - {Grade}";

}

}

}

**TeacherDiscipline.cs** – модель связи преподавателя с дисциплиной, реализует связь многие–ко–многим между преподавателями и дисциплинами.

using System;

using ReactiveUI;

namespace UniversityIS.Models

{

*// Модель связи преподавателя с дисциплиной*

*// Определяет, какие дисциплины может вести конкретный преподаватель*

*// Используется для фильтрации дисциплин при добавлении учебной нагрузки*

public class TeacherDiscipline : ModelBase

{

private Guid \_teacherId;

private Guid \_disciplineId;

public TeacherDiscipline()

{

}

public Guid TeacherId

{

get => \_teacherId;

set => this.RaiseAndSetIfChanged(ref \_teacherId, value);

}

public Guid DisciplineId

{

get => \_disciplineId;

set => this.RaiseAndSetIfChanged(ref \_disciplineId, value);

}

public override string ToFileString()

{

return $"{Id}|{TeacherId}|{DisciplineId}";

}

public static TeacherDiscipline FromFileString(string line)

{

var parts = line.Split('|');

return new TeacherDiscipline

{

Id = Guid.Parse(parts[0]),

TeacherId = Guid.Parse(parts[1]),

DisciplineId = Guid.Parse(parts[2])

};

}

}

}

Файлы перечислений (Enums)

**TeacherPosition.cs** – перечисление должностей преподавателя (ассистент, преподаватель, старший преподаватель, доцент, профессор).

namespace UniversityIS.Models

{

*// Должность преподавателя*

public enum TeacherPosition

{

*// Ассистент*

Assistant,

*// Преподаватель*

Lecturer,

*// Старший преподаватель*

SeniorLecturer,

*// Доцент*

AssociateProfessor,

*// Профессор*

Professor

}

}

**AcademicDegree.cs** – перечисление учёных степеней (нет, кандидат наук, доктор наук).

namespace UniversityIS.Models

{

*// Учёная степень*

public enum AcademicDegree

{

*// Без степени*

None,

*// Кандидат наук*

CandidateOfSciences,

*// Доктор наук*

DoctorOfSciences

}

}

**AcademicTitle.cs** – перечисление учёных званий (нет, доцент, профессор).

namespace UniversityIS.Models

{

*// Учёное звание*

public enum AcademicTitle

{

*// Без звания*

None,

*// Доцент*

AssociateProfessor,

*// Профессор*

Professor

}

}

**LessonType.cs** – перечисление видов занятий (лекции, семинары, лабораторные, консультации, курсовая работа, курсовой проект).

namespace UniversityIS.Models

{

*// Вид занятий*

public enum LessonType

{

*// Лекции*

Lecture,

*// Семинары*

Seminar,

*// Лабораторные работы*

Laboratory,

*// Консультации*

Consultation,

*// Курсовые работы*

CourseWork,

*// Курсовые проекты*

CourseProject

}

}

**ControlForm.cs** – перечисление форм контроля (зачёт, дифференцированный зачёт, экзамен).

namespace UniversityIS.Models

{

*// Форма контроля*

public enum ControlForm

{

*// Зачёт*

Pass,

*// Дифференцированный зачёт*

DifferentiatedPass,

*// Экзамен*

Exam

}

}

Файлы вспомогательных классов моделей

**EnumHelper.cs** – класс вспомогательных методов для создания списков enum значений с русскими названиями, содержит класс EnumItem<T> для обёртки enum значений.

using System;

using System.Collections.Generic;

using System.Linq;

namespace UniversityIS.Models

{

*// Обертка для enum значений с русским названием*

*// Используется для отображения enum в ComboBox с локализованными названиями*

public class EnumItem<T> where T : Enum

{

public T Value { get; set; }

public string Display { get; set; }

public EnumItem(T value, string display)

{

Value = value;

Display = display;

}

public override string ToString() => Display;

}

*// Вспомогательный класс для создания списков enum значений с русскими названиями*

*// Используется во всех ViewModel для заполнения ComboBox*

public static class EnumHelper

{

public static List<EnumItem<TeacherPosition>> GetTeacherPositions()

{

return Enum.GetValues<TeacherPosition>()

.Select(p => new EnumItem<TeacherPosition>(p, p.ToRussianString()))

.ToList();

}

public static List<EnumItem<AcademicDegree>> GetAcademicDegrees()

{

return Enum.GetValues<AcademicDegree>()

.Select(d => new EnumItem<AcademicDegree>(d, d.ToRussianString()))

.ToList();

}

public static List<EnumItem<AcademicTitle>> GetAcademicTitles()

{

return Enum.GetValues<AcademicTitle>()

.Select(t => new EnumItem<AcademicTitle>(t, t.ToRussianString()))

.ToList();

}

public static List<EnumItem<LessonType>> GetLessonTypes()

{

return Enum.GetValues<LessonType>()

.Select(l => new EnumItem<LessonType>(l, l.ToRussianString()))

.ToList();

}

public static List<EnumItem<ControlForm>> GetControlForms()

{

return Enum.GetValues<ControlForm>()

.Select(c => new EnumItem<ControlForm>(c, c.ToRussianString()))

.ToList();

}

}

}

**EnumExtensions.cs** – класс методов расширения для enum'ов, преобразует значения enum в русские строки для отображения в интерфейсе.

namespace UniversityIS.Models

{

*// Методы расширения для enum'ов*

public static class EnumExtensions

{

*// Получить русское название для TeacherPosition*

public static string ToRussianString(this TeacherPosition position)

{

return position switch

{

TeacherPosition.Assistant => "Ассистент",

TeacherPosition.Lecturer => "Преподаватель",

TeacherPosition.SeniorLecturer => "Старший преподаватель",

TeacherPosition.AssociateProfessor => "Доцент",

TeacherPosition.Professor => "Профессор",

\_ => position.ToString()

};

}

*// Получить русское название для AcademicDegree*

public static string ToRussianString(this AcademicDegree degree)

{

return degree switch

{

AcademicDegree.None => "Без степени",

AcademicDegree.CandidateOfSciences => "Кандидат наук",

AcademicDegree.DoctorOfSciences => "Доктор наук",

\_ => degree.ToString()

};

}

*// Получить русское название для AcademicTitle*

public static string ToRussianString(this AcademicTitle title)

{

return title switch

{

AcademicTitle.None => "Без звания",

AcademicTitle.AssociateProfessor => "Доцент",

AcademicTitle.Professor => "Профессор",

\_ => title.ToString()

};

}

*// Получить русское название для LessonType*

public static string ToRussianString(this LessonType lessonType)

{

return lessonType switch

{

LessonType.Lecture => "Лекции",

LessonType.Seminar => "Семинары",

LessonType.Laboratory => "Лабораторные работы",

LessonType.Consultation => "Консультации",

LessonType.CourseWork => "Курсовые работы",

LessonType.CourseProject => "Курсовые проекты",

\_ => lessonType.ToString()

};

}

*// Получить русское название для ControlForm*

public static string ToRussianString(this ControlForm controlForm)

{

return controlForm switch

{

ControlForm.Pass => "Зачёт",

ControlForm.DifferentiatedPass => "Дифференцированный зачёт",

ControlForm.Exam => "Экзамен",

\_ => controlForm.ToString()

};

}

}

Файлы сервисов (Services)

**DataService.cs** – центральный сервис для работы с данными приложения, содержит наблюдаемые коллекции всех сущностей и управляет их загрузкой и сохранением в текстовые файлы.

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.IO;

using System.Linq;

using UniversityIS.Models;

namespace UniversityIS.Services

{

*// Центральный сервис для работы с данными приложения*

*// Отвечает за загрузку и сохранение всех данных в текстовые файлы*

*// Все данные хранятся в папке Data в формате текстовых файлов*

public class DataService

{

private static readonly string DataDirectory = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "Data");

private const string FacultiesFile = "faculties.txt";

private const string DepartmentsFile = "departments.txt";

private const string GroupsFile = "groups.txt";

private const string StudentsFile = "students.txt";

private const string TeachersFile = "teachers.txt";

private const string DisciplinesFile = "disciplines.txt";

private const string WorkLoadsFile = "workloads.txt";

private const string ThesisWorksFile = "thesisworks.txt";

private const string GradesFile = "grades.txt";

private const string TeacherDisciplinesFile = "teacherdisciplines.txt";

*// Коллекции всех сущностей приложения*

*// Используются ObservableCollection для автоматического обновления UI при изменениях*

public ObservableCollection<Faculty> Faculties { get; private set; }

public ObservableCollection<Department> Departments { get; private set; }

public ObservableCollection<Group> Groups { get; private set; }

public ObservableCollection<Student> Students { get; private set; }

public ObservableCollection<Teacher> Teachers { get; private set; }

public ObservableCollection<Discipline> Disciplines { get; private set; }

public ObservableCollection<WorkLoad> WorkLoads { get; private set; }

public ObservableCollection<ThesisWork> ThesisWorks { get; private set; }

public ObservableCollection<StudentGrade> Grades { get; private set; }

public ObservableCollection<TeacherDiscipline> TeacherDisciplines { get; private set; }

public DataService()

{

Faculties = new ObservableCollection<Faculty>();

Departments = new ObservableCollection<Department>();

Groups = new ObservableCollection<Group>();

Students = new ObservableCollection<Student>();

Teachers = new ObservableCollection<Teacher>();

Disciplines = new ObservableCollection<Discipline>();

WorkLoads = new ObservableCollection<WorkLoad>();

ThesisWorks = new ObservableCollection<ThesisWork>();

Grades = new ObservableCollection<StudentGrade>();

TeacherDisciplines = new ObservableCollection<TeacherDiscipline>();

EnsureDataDirectoryExists();

}

*// Создает папку Data для хранения файлов, если она не существует*

private void EnsureDataDirectoryExists()

{

if (!Directory.Exists(DataDirectory))

{

Directory.CreateDirectory(DataDirectory);

}

}

*// Загружает все данные из текстовых файлов в память*

*// Вызывается при запуске приложения*

public void LoadAllData()

{

LoadFaculties();

LoadDepartments();

LoadGroups();

LoadStudents();

LoadTeachers();

LoadDisciplines();

LoadWorkLoads();

LoadThesisWorks();

LoadGrades();

LoadTeacherDisciplines();

}

*// Сохраняет все данные из памяти в текстовые файлы*

*// Вызывается при закрытии приложения или по запросу пользователя*

public void SaveAllData()

{

SaveTeacherDisciplines();

SaveGrades();

SaveFaculties();

SaveDepartments();

SaveGroups();

SaveStudents();

SaveTeachers();

SaveDisciplines();

SaveWorkLoads();

SaveThesisWorks();

}

*// Получить путь к папке с данными*

public string GetDataPath() => DataDirectory;

#region Faculties

*// Загружает список факультетов из текстового файла*

*// Пропускает пустые и некорректные строки*

private void LoadFaculties()

{

var path = Path.Combine(DataDirectory, FacultiesFile);

if (!File.Exists(path))

{

return;

}

Faculties.Clear();

var lines = File.ReadAllLines(path);

foreach (var line in lines)

{

if (!string.IsNullOrWhiteSpace(line))

{

try

{

Faculties.Add(Faculty.FromFileString(line));

}

catch

{

*// Игнорируем ошибочные строки для стабильности работы*

}

}

}

}

*// Сохраняет список факультетов в текстовый файл*

private void SaveFaculties()

{

var path = Path.Combine(DataDirectory, FacultiesFile);

var lines = Faculties.Select(f => f.ToFileString());

File.WriteAllLines(path, lines);

}

#endregion

#region Departments

*// Загружает список кафедр из текстового файла*

private void LoadDepartments()

{

var path = Path.Combine(DataDirectory, DepartmentsFile);

if (!File.Exists(path))

{

return;

}

Departments.Clear();

var lines = File.ReadAllLines(path);

foreach (var line in lines)

{

if (!string.IsNullOrWhiteSpace(line))

{

try

{

Departments.Add(Department.FromFileString(line));

}

catch

{

*// Игнорируем ошибочные строки*

}

}

}

}

private void SaveDepartments()

{

var path = Path.Combine(DataDirectory, DepartmentsFile);

var lines = Departments.Select(d => d.ToFileString());

File.WriteAllLines(path, lines);

}

#endregion

#region Groups

private void LoadGroups()

{

var path = Path.Combine(DataDirectory, GroupsFile);

if (!File.Exists(path))

{

return;

}

Groups.Clear();

var lines = File.ReadAllLines(path);

foreach (var line in lines)

{

if (!string.IsNullOrWhiteSpace(line))

{

try

{

Groups.Add(Group.FromFileString(line));

}

catch

{

*// Игнорируем ошибочные строки*

}

}

}

}

private void SaveGroups()

{

var path = Path.Combine(DataDirectory, GroupsFile);

var lines = Groups.Select(g => g.ToFileString());

File.WriteAllLines(path, lines);

}

#endregion

#region Students

private void LoadStudents()

{

var path = Path.Combine(DataDirectory, StudentsFile);

if (!File.Exists(path))

{

return;

}

Students.Clear();

var lines = File.ReadAllLines(path);

foreach (var line in lines)

{

if (!string.IsNullOrWhiteSpace(line))

{

try

{

Students.Add(Student.FromFileString(line));

}

catch

{

*// Игнорируем ошибочные строки*

}

}

}

}

private void SaveStudents()

{

var path = Path.Combine(DataDirectory, StudentsFile);

var lines = Students.Select(s => s.ToFileString());

File.WriteAllLines(path, lines);

}

#endregion

#region Teachers

private void LoadTeachers()

{

var path = Path.Combine(DataDirectory, TeachersFile);

if (!File.Exists(path))

{

return;

}

Teachers.Clear();

var lines = File.ReadAllLines(path);

foreach (var line in lines)

{

if (!string.IsNullOrWhiteSpace(line))

{

try

{

Teachers.Add(Teacher.FromFileString(line));

}

catch

{

*// Игнорируем ошибочные строки*

}

}

}

}

private void SaveTeachers()

{

var path = Path.Combine(DataDirectory, TeachersFile);

var lines = Teachers.Select(t => t.ToFileString());

File.WriteAllLines(path, lines);

}

#endregion

#region Disciplines

private void LoadDisciplines()

{

var path = Path.Combine(DataDirectory, DisciplinesFile);

if (!File.Exists(path))

{

return;

}

Disciplines.Clear();

var lines = File.ReadAllLines(path);

foreach (var line in lines)

{

if (!string.IsNullOrWhiteSpace(line))

{

try

{

Disciplines.Add(Discipline.FromFileString(line));

}

catch

{

*// Игнорируем ошибочные строки*

}

}

}

}

private void SaveDisciplines()

{

var path = Path.Combine(DataDirectory, DisciplinesFile);

var lines = Disciplines.Select(d => d.ToFileString());

File.WriteAllLines(path, lines);

}

#endregion

#region WorkLoads

private void LoadWorkLoads()

{

var path = Path.Combine(DataDirectory, WorkLoadsFile);

if (!File.Exists(path))

{

return;

}

WorkLoads.Clear();

var lines = File.ReadAllLines(path);

foreach (var line in lines)

{

if (!string.IsNullOrWhiteSpace(line))

{

try

{

WorkLoads.Add(WorkLoad.FromFileString(line));

}

catch

{

*// Игнорируем ошибочные строки*

}

}

}

}

private void SaveWorkLoads()

{

var path = Path.Combine(DataDirectory, WorkLoadsFile);

var lines = WorkLoads.Select(w => w.ToFileString());

File.WriteAllLines(path, lines);

}

#endregion

#region ThesisWorks

private void LoadThesisWorks()

{

var path = Path.Combine(DataDirectory, ThesisWorksFile);

if (!File.Exists(path))

{

return;

}

ThesisWorks.Clear();

var lines = File.ReadAllLines(path);

foreach (var line in lines)

{

if (!string.IsNullOrWhiteSpace(line))

{

try

{

ThesisWorks.Add(ThesisWork.FromFileString(line));

}

catch

{

*// Игнорируем ошибочные строки*

}

}

}

}

private void SaveThesisWorks()

{

var path = Path.Combine(DataDirectory, ThesisWorksFile);

var lines = ThesisWorks.Select(t => t.ToFileString());

File.WriteAllLines(path, lines);

}

private void LoadGrades()

{

var path = Path.Combine(DataDirectory, GradesFile);

if (!File.Exists(path)) return;

var lines = File.ReadAllLines(path);

foreach (var line in lines)

{

if (!string.IsNullOrWhiteSpace(line))

{

Grades.Add(StudentGrade.FromFileString(line));

}

}

}

private void SaveGrades()

{

var path = Path.Combine(DataDirectory, GradesFile);

var lines = Grades.Select(g => g.ToFileString());

File.WriteAllLines(path, lines);

}

private void LoadTeacherDisciplines()

{

var path = Path.Combine(DataDirectory, TeacherDisciplinesFile);

if (!File.Exists(path)) return;

var lines = File.ReadAllLines(path);

foreach (var line in lines)

{

if (!string.IsNullOrWhiteSpace(line))

{

TeacherDisciplines.Add(TeacherDiscipline.FromFileString(line));

}

}

}

private void SaveTeacherDisciplines()

{

var path = Path.Combine(DataDirectory, TeacherDisciplinesFile);

var lines = TeacherDisciplines.Select(td => td.ToFileString());

File.WriteAllLines(path, lines);

}

#endregion

#region Helper Methods

*// Вспомогательные методы для поиска сущностей по ID*

*// Используются во всех ViewModel для получения связанных объектов*

*// Находит факультет по его уникальному идентификатору*

public Faculty? GetFaculty(Guid id)

{

return Faculties.FirstOrDefault(f => f.Id == id);

}

*// Находит кафедру по ее уникальному идентификатору*

public Department? GetDepartment(Guid id)

{

return Departments.FirstOrDefault(d => d.Id == id);

}

*// Находит учебную группу по ее уникальному идентификатору*

public Group? GetGroup(Guid id)

{

return Groups.FirstOrDefault(g => g.Id == id);

}

*// Находит студента по его уникальному идентификатору*

public Student? GetStudent(Guid id)

{

return Students.FirstOrDefault(s => s.Id == id);

}

*// Находит преподавателя по его уникальному идентификатору*

public Teacher? GetTeacher(Guid id)

{

return Teachers.FirstOrDefault(t => t.Id == id);

}

*// Находит дисциплину по ее уникальному идентификатору*

public Discipline? GetDiscipline(Guid id)

{

return Disciplines.FirstOrDefault(d => d.Id == id);

}

*// Возвращает список всех кафедр, принадлежащих указанному факультету*

public List<Department> GetDepartmentsByFaculty(Guid facultyId)

{

return Departments.Where(d => d.FacultyId == facultyId).ToList();

}

*// Возвращает список всех групп, принадлежащих указанному факультету*

public List<Group> GetGroupsByFaculty(Guid facultyId)

{

return Groups.Where(g => g.FacultyId == facultyId).ToList();

}

*// Возвращает список всех студентов указанной группы*

public List<Student> GetStudentsByGroup(Guid groupId)

{

return Students.Where(s => s.GroupId == groupId).ToList();

}

*// Возвращает список всех преподавателей указанной кафедры*

public List<Teacher> GetTeachersByDepartment(Guid departmentId)

{

return Teachers.Where(t => t.DepartmentId == departmentId).ToList();

}

*// Возвращает список всей учебной нагрузки указанного преподавателя*

public List<WorkLoad> GetWorkLoadsByTeacher(Guid teacherId)

{

return WorkLoads.Where(w => w.TeacherId == teacherId).ToList();

}

*// Возвращает список всех дипломных работ, которыми руководит указанный преподаватель*

public List<ThesisWork> GetThesisWorksBySupervisor(Guid supervisorId)

{

return ThesisWorks.Where(t => t.SupervisorId == supervisorId).ToList();

}

#endregion

}

}

Файлы вспомогательных классов (Helpers)

**ValidationHelper.cs** – класс вспомогательных методов для валидации вводимых данных, содержит статические методы проверки корректности различных типов данных.

using System;

using System.Linq;

using System.Text.RegularExpressions;

namespace UniversityIS.Helpers

{

*// Вспомогательный класс для валидации вводимых данных*

*// Содержит методы проверки корректности различных типов данных*

public static class ValidationHelper

{

*// Проверяет, что строка содержит только буквы, пробелы, дефисы и точки*

*// Используется для валидации имен, фамилий и названий без цифр*

public static bool IsValidName(string? name)

{

if (string.IsNullOrWhiteSpace(name))

return false;

*// Разрешаем буквы (русские и латинские), пробелы, дефисы, точки*

return Regex.IsMatch(name, @"^[а-яА-ЯёЁa-zA-Z\s\.\-]+$");

}

*// Проверяет, что строка содержит только буквы, цифры, пробелы, дефисы и точки*

*// Используется для валидации названий групп, дисциплин и других сущностей с цифрами*

public static bool IsValidNameWithNumbers(string? name)

{

if (string.IsNullOrWhiteSpace(name))

return false;

*// Разрешаем буквы, цифры, пробелы, дефисы, точки*

return Regex.IsMatch(name, @"^[а-яА-ЯёЁa-zA-Z0-9\s\.\-]+$");

}

*// Проверяет, что строка является корректным годом*

*// Допустимый диапазон: 1900-2100*

public static bool IsValidYear(string? year)

{

if (string.IsNullOrWhiteSpace(year))

return false;

if (!int.TryParse(year, out int yearValue))

return false;

return yearValue >= 1900 && yearValue <= 2100;

}

*// Проверяет, что строка является корректным номером курса*

*// Допустимый диапазон: 1-6*

public static bool IsValidCourse(string? course)

{

if (string.IsNullOrWhiteSpace(course))

return false;

if (!int.TryParse(course, out int courseValue))

return false;

return courseValue >= 1 && courseValue <= 6;

}

*// Проверяет, что строка является корректным номером семестра*

*// Допустимый диапазон: 1-10*

public static bool IsValidSemester(string? semester)

{

if (string.IsNullOrWhiteSpace(semester))

return false;

if (!int.TryParse(semester, out int semesterValue))

return false;

return semesterValue >= 1 && semesterValue <= 10;

}

*// Проверяет, что строка является корректным количеством часов*

*// Допустимый диапазон: 1-1000*

public static bool IsValidHours(string? hours)

{

if (string.IsNullOrWhiteSpace(hours))

return false;

if (!int.TryParse(hours, out int hoursValue))

return false;

return hoursValue > 0 && hoursValue <= 1000;

}

*// Возвращает понятное сообщение об ошибке валидации*

*// fieldName - название поля, errorType - тип ошибки*

public static string GetErrorMessage(string fieldName, string errorType)

{

return errorType switch

{

"empty" => $"Поле \"{fieldName}\" не может быть пустым.",

"invalid\_name" => $"Поле \"{fieldName}\" должно содержать только буквы, пробелы, дефисы и точки (без цифр).",

"invalid\_name\_with\_numbers" => $"Поле \"{fieldName}\" содержит недопустимые символы.",

"invalid\_year" => $"Поле \"{fieldName}\" должно содержать корректный год (1900-2100).",

"invalid\_course" => $"Поле \"{fieldName}\" должно быть числом от 1 до 6.",

"invalid\_semester" => $"Поле \"{fieldName}\" должно быть числом от 1 до 10.",

"invalid\_hours" => $"Поле \"{fieldName}\" должно быть положительным числом (1-1000).",

"not\_selected" => $"Необходимо выбрать \"{fieldName}\".",

\_ => "Неизвестная ошибка валидации."

};

}

}

}

Файлы ViewLocator

**ViewLocator.cs** – класс для автоматического сопоставления ViewModel с соответствующими View, реализует интерфейс IDataTemplate для динамического создания представлений.

using System;

using System.Diagnostics.CodeAnalysis;

using Avalonia.Controls;

using Avalonia.Controls.Templates;

using UniversityIS.ViewModels;

namespace UniversityIS;

*// Given a view model, returns the corresponding view if possible.*

[RequiresUnreferencedCode(

"Default implementation of ViewLocator involves reflection which may be trimmed away.",

Url = "https://docs.avaloniaui.net/docs/concepts/view-locator")]

public class ViewLocator : IDataTemplate

{

public Control? Build(object? param)

{

if (param is null)

return null;

var name = param.GetType().FullName!.Replace("ViewModel", "View", StringComparison.Ordinal);

var type = Type.GetType(name);

if (type != null)

{

return (Control)Activator.CreateInstance(type)!;

}

return new TextBlock { Text = "Not Found: " + name };

}

public bool Match(object? data)

{

return data is ViewModelBase;

}

}

Файлы ViewModels (ViewModels)

**ViewModelBase.cs** – базовый класс для всех ViewModel в приложении, наследуется от ReactiveObject для поддержки реактивного программирования.

using ReactiveUI;

namespace UniversityIS.ViewModels;

*// Базовый класс для всех ViewModel в приложении*

*// Наследуется от ReactiveObject для поддержки реактивного программирования*

*// Все ViewModel должны наследоваться от этого класса*

public class ViewModelBase : ReactiveObject

{

}

**MainWindowViewModel.cs** – главная ViewModel приложения, управляет навигацией между разными разделами и координирует работу всех ViewModel.

using System.Reactive;

using ReactiveUI;

using UniversityIS.Models;

using UniversityIS.Services;

namespace UniversityIS.ViewModels;

*// Главная ViewModel приложения*

*// Управляет навигацией между разными разделами и координирует работу всех ViewModel*

public partial class MainWindowViewModel : ViewModelBase

{

private readonly DataService \_dataService;

*// Текущая открытая страница (раздел приложения)*

private ViewModelBase \_currentPage;

*// Инициализация главного окна: загрузка данных и создание всех ViewModel*

public MainWindowViewModel()

{

\_dataService = new DataService();

\_dataService.LoadAllData();

*// Инициализация ViewModels для разделов*

FacultiesViewModel = new FacultiesViewModel(\_dataService);

DepartmentsViewModel = new DepartmentsViewModel(\_dataService);

GroupsViewModel = new GroupsViewModel(\_dataService);

StudentsViewModel = new StudentsViewModel(\_dataService, ShowStudentProfile);

TeachersViewModel = new TeachersViewModel(\_dataService, ShowTeacherProfile);

DisciplinesViewModel = new DisciplinesViewModel(\_dataService);

WorkLoadsViewModel = new WorkLoadsViewModel(\_dataService);

ThesisWorksViewModel = new ThesisWorksViewModel(\_dataService);

*// По умолчанию открываем факультеты*

\_currentPage = FacultiesViewModel;

*// Команды навигации с указанием главного потока UI*

ShowFacultiesCommand = ReactiveCommand.Create(() => { CurrentPage = FacultiesViewModel; }, outputScheduler: RxApp.MainThreadScheduler);

ShowDepartmentsCommand = ReactiveCommand.Create(() => { CurrentPage = DepartmentsViewModel; }, outputScheduler: RxApp.MainThreadScheduler);

ShowGroupsCommand = ReactiveCommand.Create(() => { CurrentPage = GroupsViewModel; }, outputScheduler: RxApp.MainThreadScheduler);

ShowStudentsCommand = ReactiveCommand.Create(() => { CurrentPage = StudentsViewModel; }, outputScheduler: RxApp.MainThreadScheduler);

ShowTeachersCommand = ReactiveCommand.Create(() => { CurrentPage = TeachersViewModel; }, outputScheduler: RxApp.MainThreadScheduler);

ShowDisciplinesCommand = ReactiveCommand.Create(() => { CurrentPage = DisciplinesViewModel; }, outputScheduler: RxApp.MainThreadScheduler);

ShowWorkLoadsCommand = ReactiveCommand.Create(() => { CurrentPage = WorkLoadsViewModel; }, outputScheduler: RxApp.MainThreadScheduler);

ShowThesisWorksCommand = ReactiveCommand.Create(() => { CurrentPage = ThesisWorksViewModel; }, outputScheduler: RxApp.MainThreadScheduler);

SaveDataCommand = ReactiveCommand.Create(() =>

{

\_dataService.SaveAllData();

}, outputScheduler: RxApp.MainThreadScheduler);

}

*// Вызывается при закрытии приложения для сохранения всех данных*

public void OnClosing()

{

\_dataService.SaveAllData();

}

*// Открывает профиль выбранного студента с информацией о его предметах и оценках*

public void ShowStudentProfile(Student student)

{

var profileViewModel = new StudentProfileViewModel(\_dataService, student);

CurrentPage = profileViewModel;

}

*// Открывает профиль выбранного преподавателя с информацией о его дисциплинах*

public void ShowTeacherProfile(Teacher teacher)

{

var profileViewModel = new TeacherProfileViewModel(\_dataService, teacher);

CurrentPage = profileViewModel;

}

*// Текущая открытая страница в главном окне*

*// При изменении автоматически обновляется содержимое главного окна*

public ViewModelBase CurrentPage

{

get => \_currentPage;

set => this.RaiseAndSetIfChanged(ref \_currentPage, value);

}

*// ViewModel для всех разделов приложения*

public FacultiesViewModel FacultiesViewModel { get; }

public DepartmentsViewModel DepartmentsViewModel { get; }

public GroupsViewModel GroupsViewModel { get; }

public StudentsViewModel StudentsViewModel { get; }

public TeachersViewModel TeachersViewModel { get; }

public DisciplinesViewModel DisciplinesViewModel { get; }

public WorkLoadsViewModel WorkLoadsViewModel { get; }

public ThesisWorksViewModel ThesisWorksViewModel { get; }

*// Команды для навигации между разделами приложения*

*// Все команды выполняются в главном потоке UI для безопасности*

public ReactiveCommand<Unit, Unit> ShowFacultiesCommand { get; }

public ReactiveCommand<Unit, Unit> ShowDepartmentsCommand { get; }

public ReactiveCommand<Unit, Unit> ShowGroupsCommand { get; }

public ReactiveCommand<Unit, Unit> ShowStudentsCommand { get; }

public ReactiveCommand<Unit, Unit> ShowTeachersCommand { get; }

public ReactiveCommand<Unit, Unit> ShowDisciplinesCommand { get; }

public ReactiveCommand<Unit, Unit> ShowWorkLoadsCommand { get; }

public ReactiveCommand<Unit, Unit> ShowThesisWorksCommand { get; }

public ReactiveCommand<Unit, Unit> SaveDataCommand { get; }

}

**FacultiesViewModel.cs** – ViewModel для управления факультетами, позволяет добавлять, редактировать и удалять факультеты.

using System;

using System.Collections.ObjectModel;

using System.Linq;

using System.Reactive;

using ReactiveUI;

using UniversityIS.Models;

using UniversityIS.Services;

using UniversityIS.Helpers;

namespace UniversityIS.ViewModels

{

*// ViewModel для управления факультетами*

*// Позволяет добавлять, редактировать и удалять факультеты*

public class FacultiesViewModel : ViewModelBase

{

private readonly DataService \_dataService;

private Faculty? \_selectedFaculty;

private string \_name = string.Empty;

private string \_dean = string.Empty;

private string \_errorMessage = string.Empty;

public FacultiesViewModel(DataService dataService)

{

\_dataService = dataService;

AddCommand = ReactiveCommand.Create(AddFaculty, outputScheduler: RxApp.MainThreadScheduler);

UpdateCommand = ReactiveCommand.Create(UpdateFaculty, outputScheduler: RxApp.MainThreadScheduler);

DeleteCommand = ReactiveCommand.Create(DeleteFaculty, outputScheduler: RxApp.MainThreadScheduler);

}

public ObservableCollection<Faculty> Faculties => \_dataService.Faculties;

public Faculty? SelectedFaculty

{

get => \_selectedFaculty;

set

{

this.RaiseAndSetIfChanged(ref \_selectedFaculty, value);

if (value != null)

{

Name = value.Name;

Dean = value.Dean;

}

}

}

public string Name

{

get => \_name;

set => this.RaiseAndSetIfChanged(ref \_name, value);

}

public string Dean

{

get => \_dean;

set => this.RaiseAndSetIfChanged(ref \_dean, value);

}

public string ErrorMessage

{

get => \_errorMessage;

set => this.RaiseAndSetIfChanged(ref \_errorMessage, value);

}

public ReactiveCommand<Unit, Unit> AddCommand { get; }

public ReactiveCommand<Unit, Unit> UpdateCommand { get; }

public ReactiveCommand<Unit, Unit> DeleteCommand { get; }

*// Добавляет новый факультет в базу данных*

*// Выполняет валидацию всех полей перед добавлением*

private void AddFaculty()

{

ErrorMessage = string.Empty;

*// Валидация названия факультета*

if (string.IsNullOrWhiteSpace(Name))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Название факультета", "empty");

return;

}

if (!ValidationHelper.IsValidName(Name))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Название факультета", "invalid\_name");

return;

}

*// Валидация ФИО декана*

if (string.IsNullOrWhiteSpace(Dean))

{

ErrorMessage = ValidationHelper.GetErrorMessage("ФИО декана", "empty");

return;

}

if (!ValidationHelper.IsValidName(Dean))

{

ErrorMessage = ValidationHelper.GetErrorMessage("ФИО декана", "invalid\_name");

return;

}

*// Создаем новый факультет и добавляем в коллекцию*

var faculty = new Faculty

{

Name = Name,

Dean = Dean

};

\_dataService.Faculties.Add(faculty);

ClearFields();

}

private void UpdateFaculty()

{

ErrorMessage = string.Empty;

if (SelectedFaculty == null)

{

ErrorMessage = "Выберите факультет для обновления.";

return;

}

*// Валидация названия факультета*

if (string.IsNullOrWhiteSpace(Name))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Название факультета", "empty");

return;

}

if (!ValidationHelper.IsValidName(Name))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Название факультета", "invalid\_name");

return;

}

*// Валидация ФИО декана*

if (string.IsNullOrWhiteSpace(Dean))

{

ErrorMessage = ValidationHelper.GetErrorMessage("ФИО декана", "empty");

return;

}

if (!ValidationHelper.IsValidName(Dean))

{

ErrorMessage = ValidationHelper.GetErrorMessage("ФИО декана", "invalid\_name");

return;

}

SelectedFaculty.Name = Name;

SelectedFaculty.Dean = Dean;

*// Обновляем отображение*

var index = Faculties.IndexOf(SelectedFaculty);

if (index >= 0)

{

Faculties[index] = SelectedFaculty;

}

}

private void DeleteFaculty()

{

if (SelectedFaculty == null) return;

\_dataService.Faculties.Remove(SelectedFaculty);

Faculties.Remove(SelectedFaculty);

ClearFields();

}

private void ClearFields()

{

Name = string.Empty;

Dean = string.Empty;

SelectedFaculty = null;

ErrorMessage = string.Empty;

}

}

}

**DepartmentsViewModel.cs** – ViewModel для управления кафедрами, позволяет добавлять, редактировать и удалять кафедры, группирует кафедры по факультетам. Содержит внутренний класс FacultyGroup для группировки.

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

using System.Reactive;

using ReactiveUI;

using UniversityIS.Models;

using UniversityIS.Services;

using UniversityIS.Helpers;

namespace UniversityIS.ViewModels

{

*// Группа кафедр, объединенных по факультету*

*// Используется для отображения кафедр с группировкой по факультетам*

public class FacultyGroup

{

public string FacultyName { get; set; } = string.Empty;

public ObservableCollection<Department> Departments { get; set; } = new();

}

*// ViewModel для управления кафедрами*

*// Позволяет добавлять, редактировать и удалять кафедры*

*// Группирует кафедры по факультетам для удобного отображения*

public class DepartmentsViewModel : ViewModelBase

{

private readonly DataService \_dataService;

private Department? \_selectedDepartment;

private string \_name = string.Empty;

private string \_head = string.Empty;

private Faculty? \_selectedFaculty;

private string \_errorMessage = string.Empty;

private ObservableCollection<FacultyGroup> \_groupedDepartments = new();

public DepartmentsViewModel(DataService dataService)

{

\_dataService = dataService;

AddCommand = ReactiveCommand.Create(AddDepartment, outputScheduler: RxApp.MainThreadScheduler);

UpdateCommand = ReactiveCommand.Create(UpdateDepartment, outputScheduler: RxApp.MainThreadScheduler);

DeleteCommand = ReactiveCommand.Create(DeleteDepartment, outputScheduler: RxApp.MainThreadScheduler);

*// Подписываемся на изменения в коллекциях для обновления группировки*

\_dataService.Departments.CollectionChanged += (s, e) => UpdateGroupedDepartments();

\_dataService.Faculties.CollectionChanged += (s, e) => UpdateGroupedDepartments();

UpdateGroupedDepartments();

}

public ObservableCollection<Department> Departments => \_dataService.Departments;

public ObservableCollection<Faculty> Faculties => \_dataService.Faculties;

public ObservableCollection<FacultyGroup> GroupedDepartments

{

get => \_groupedDepartments;

set => this.RaiseAndSetIfChanged(ref \_groupedDepartments, value);

}

public Department? SelectedDepartment

{

get => \_selectedDepartment;

set

{

this.RaiseAndSetIfChanged(ref \_selectedDepartment, value);

if (value != null)

{

Name = value.Name;

Head = value.Head;

SelectedFaculty = \_dataService.GetFaculty(value.FacultyId);

}

}

}

public string Name

{

get => \_name;

set => this.RaiseAndSetIfChanged(ref \_name, value);

}

public string Head

{

get => \_head;

set => this.RaiseAndSetIfChanged(ref \_head, value);

}

public Faculty? SelectedFaculty

{

get => \_selectedFaculty;

set => this.RaiseAndSetIfChanged(ref \_selectedFaculty, value);

}

public string ErrorMessage

{

get => \_errorMessage;

set => this.RaiseAndSetIfChanged(ref \_errorMessage, value);

}

public ReactiveCommand<Unit, Unit> AddCommand { get; }

public ReactiveCommand<Unit, Unit> UpdateCommand { get; }

public ReactiveCommand<Unit, Unit> DeleteCommand { get; }

private void AddDepartment()

{

ErrorMessage = string.Empty;

*// Валидация факультета*

if (SelectedFaculty == null)

{

ErrorMessage = ValidationHelper.GetErrorMessage("Факультет", "not\_selected");

return;

}

*// Валидация названия кафедры*

if (string.IsNullOrWhiteSpace(Name))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Название кафедры", "empty");

return;

}

if (!ValidationHelper.IsValidName(Name))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Название кафедры", "invalid\_name");

return;

}

*// Валидация ФИО заведующего*

if (string.IsNullOrWhiteSpace(Head))

{

ErrorMessage = ValidationHelper.GetErrorMessage("ФИО заведующего", "empty");

return;

}

if (!ValidationHelper.IsValidName(Head))

{

ErrorMessage = ValidationHelper.GetErrorMessage("ФИО заведующего", "invalid\_name");

return;

}

var department = new Department

{

Name = Name,

Head = Head,

FacultyId = SelectedFaculty.Id

};

\_dataService.Departments.Add(department);

ClearFields();

}

private void UpdateDepartment()

{

ErrorMessage = string.Empty;

if (SelectedDepartment == null)

{

ErrorMessage = "Выберите кафедру для обновления.";

return;

}

*// Валидация факультета*

if (SelectedFaculty == null)

{

ErrorMessage = ValidationHelper.GetErrorMessage("Факультет", "not\_selected");

return;

}

*// Валидация названия кафедры*

if (string.IsNullOrWhiteSpace(Name))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Название кафедры", "empty");

return;

}

if (!ValidationHelper.IsValidName(Name))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Название кафедры", "invalid\_name");

return;

}

*// Валидация ФИО заведующего*

if (string.IsNullOrWhiteSpace(Head))

{

ErrorMessage = ValidationHelper.GetErrorMessage("ФИО заведующего", "empty");

return;

}

if (!ValidationHelper.IsValidName(Head))

{

ErrorMessage = ValidationHelper.GetErrorMessage("ФИО заведующего", "invalid\_name");

return;

}

SelectedDepartment.Name = Name;

SelectedDepartment.Head = Head;

SelectedDepartment.FacultyId = SelectedFaculty.Id;

var index = Departments.IndexOf(SelectedDepartment);

if (index >= 0)

{

Departments[index] = SelectedDepartment;

}

}

private void DeleteDepartment()

{

if (SelectedDepartment == null) return;

\_dataService.Departments.Remove(SelectedDepartment);

Departments.Remove(SelectedDepartment);

ClearFields();

}

private void ClearFields()

{

Name = string.Empty;

Head = string.Empty;

SelectedFaculty = null;

SelectedDepartment = null;

ErrorMessage = string.Empty;

}

private void UpdateGroupedDepartments()

{

var groups = new ObservableCollection<FacultyGroup>();

*// Группируем кафедры по факультетам*

var departmentsByFaculty = \_dataService.Departments

.GroupBy(d => d.FacultyId)

.OrderBy(g => \_dataService.GetFaculty(g.Key)?.Name ?? "Неизвестный факультет");

foreach (var group in departmentsByFaculty)

{

var faculty = \_dataService.GetFaculty(group.Key);

var facultyGroup = new FacultyGroup

{

FacultyName = faculty?.Name ?? "Неизвестный факультет",

Departments = new ObservableCollection<Department>(group.OrderBy(d => d.Name))

};

groups.Add(facultyGroup);

}

GroupedDepartments = groups;

}

}

}

**GroupsViewModel.cs** – ViewModel для управления учебными группами, позволяет добавлять, редактировать и удалять группы, группирует группы по факультетам и курсам. Содержит внутренние классы CourseGrouping и FacultyGrouping для группировки.

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

using System.Reactive;

using ReactiveUI;

using UniversityIS.Models;

using UniversityIS.Services;

using UniversityIS.Helpers;

namespace UniversityIS.ViewModels

{

*// Группа учебных групп, объединенных по курсу*

*// Используется для отображения групп с группировкой по курсам*

public class CourseGrouping

{

public CourseGrouping(int course, IEnumerable<Group> groups)

{

Course = course;

Groups = new ObservableCollection<Group>(groups);

}

public int Course { get; }

public string CourseName => $"{Course} курс";

public ObservableCollection<Group> Groups { get; }

}

*// Группа учебных групп, объединенных по факультету*

*// Содержит вложенные группы по курсам*

public class FacultyGrouping

{

public FacultyGrouping(string facultyName, IEnumerable<CourseGrouping> courses)

{

FacultyName = facultyName;

Courses = new ObservableCollection<CourseGrouping>(courses);

}

public string FacultyName { get; }

public ObservableCollection<CourseGrouping> Courses { get; }

}

*// ViewModel для управления учебными группами*

*// Позволяет добавлять, редактировать и удалять группы*

*// Группирует группы по факультетам и курсам для удобного отображения*

public class GroupsViewModel : ViewModelBase

{

private readonly DataService \_dataService;

private Group? \_selectedGroup;

private string \_number = string.Empty;

private int \_yearOfAdmission = DateTime.Now.Year;

private int \_course = 1;

private Faculty? \_selectedFaculty;

private string \_errorMessage = string.Empty;

private ObservableCollection<FacultyGrouping>? \_groupedGroups;

public GroupsViewModel(DataService dataService)

{

\_dataService = dataService;

*// Подписываемся на изменения для обновления группировки*

\_dataService.Groups.CollectionChanged += (s, e) => UpdateGroupedGroups();

\_dataService.Faculties.CollectionChanged += (s, e) => UpdateGroupedGroups();

UpdateGroupedGroups(); *// Первоначальная группировка*

AddCommand = ReactiveCommand.Create(AddGroup, outputScheduler: RxApp.MainThreadScheduler);

UpdateCommand = ReactiveCommand.Create(UpdateGroup, outputScheduler: RxApp.MainThreadScheduler);

DeleteCommand = ReactiveCommand.Create(DeleteGroup, outputScheduler: RxApp.MainThreadScheduler);

}

public ObservableCollection<Group> Groups => \_dataService.Groups;

public ObservableCollection<Faculty> Faculties => \_dataService.Faculties;

public ObservableCollection<FacultyGrouping>? GroupedGroups

{

get => \_groupedGroups;

set => this.RaiseAndSetIfChanged(ref \_groupedGroups, value);

}

private void UpdateGroupedGroups()

{

var grouped = \_dataService.Groups

.GroupBy(g => g.FacultyId)

.Select(facultyGroup =>

{

var faculty = \_dataService.GetFaculty(facultyGroup.Key);

var facultyName = faculty?.Name ?? "Неизвестный факультет";

var courseGroups = facultyGroup

.GroupBy(g => g.Course)

.OrderBy(cg => cg.Key)

.Select(courseGroup => new CourseGrouping(courseGroup.Key, courseGroup.OrderBy(g => g.Number)))

.ToList();

return new FacultyGrouping(facultyName, courseGroups);

})

.OrderBy(fg => fg.FacultyName)

.ToList();

GroupedGroups = new ObservableCollection<FacultyGrouping>(grouped);

}

public Group? SelectedGroup

{

get => \_selectedGroup;

set

{

this.RaiseAndSetIfChanged(ref \_selectedGroup, value);

if (value != null)

{

Number = value.Number;

YearOfAdmission = value.YearOfAdmission;

Course = value.Course;

SelectedFaculty = \_dataService.GetFaculty(value.FacultyId);

}

}

}

public string Number

{

get => \_number;

set => this.RaiseAndSetIfChanged(ref \_number, value);

}

public int YearOfAdmission

{

get => \_yearOfAdmission;

set => this.RaiseAndSetIfChanged(ref \_yearOfAdmission, value);

}

public int Course

{

get => \_course;

set => this.RaiseAndSetIfChanged(ref \_course, value);

}

public Faculty? SelectedFaculty

{

get => \_selectedFaculty;

set => this.RaiseAndSetIfChanged(ref \_selectedFaculty, value);

}

public string ErrorMessage

{

get => \_errorMessage;

set => this.RaiseAndSetIfChanged(ref \_errorMessage, value);

}

public ReactiveCommand<Unit, Unit> AddCommand { get; }

public ReactiveCommand<Unit, Unit> UpdateCommand { get; }

public ReactiveCommand<Unit, Unit> DeleteCommand { get; }

private void AddGroup()

{

ErrorMessage = string.Empty;

*// Валидация факультета*

if (SelectedFaculty == null)

{

ErrorMessage = ValidationHelper.GetErrorMessage("Факультет", "not\_selected");

return;

}

*// Валидация номера группы*

if (string.IsNullOrWhiteSpace(Number))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Номер группы", "empty");

return;

}

if (!ValidationHelper.IsValidNameWithNumbers(Number))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Номер группы", "invalid\_name\_with\_numbers");

return;

}

*// Валидация года поступления*

if (!ValidationHelper.IsValidYear(YearOfAdmission.ToString()))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Год поступления", "invalid\_year");

return;

}

*// Валидация курса*

if (!ValidationHelper.IsValidCourse(Course.ToString()))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Курс", "invalid\_course");

return;

}

var group = new Group

{

Number = Number,

YearOfAdmission = YearOfAdmission,

Course = Course,

FacultyId = SelectedFaculty.Id

};

\_dataService.Groups.Add(group);

ClearFields();

}

private void UpdateGroup()

{

ErrorMessage = string.Empty;

if (SelectedGroup == null)

{

ErrorMessage = "Выберите группу для обновления.";

return;

}

*// Валидация факультета*

if (SelectedFaculty == null)

{

ErrorMessage = ValidationHelper.GetErrorMessage("Факультет", "not\_selected");

return;

}

*// Валидация номера группы*

if (string.IsNullOrWhiteSpace(Number))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Номер группы", "empty");

return;

}

if (!ValidationHelper.IsValidNameWithNumbers(Number))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Номер группы", "invalid\_name\_with\_numbers");

return;

}

*// Валидация года поступления*

if (!ValidationHelper.IsValidYear(YearOfAdmission.ToString()))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Год поступления", "invalid\_year");

return;

}

*// Валидация курса*

if (!ValidationHelper.IsValidCourse(Course.ToString()))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Курс", "invalid\_course");

return;

}

SelectedGroup.Number = Number;

SelectedGroup.YearOfAdmission = YearOfAdmission;

SelectedGroup.Course = Course;

SelectedGroup.FacultyId = SelectedFaculty.Id;

var index = Groups.IndexOf(SelectedGroup);

if (index >= 0)

{

Groups[index] = SelectedGroup;

}

}

private void DeleteGroup()

{

if (SelectedGroup == null) return;

\_dataService.Groups.Remove(SelectedGroup);

Groups.Remove(SelectedGroup);

ClearFields();

}

private void ClearFields()

{

Number = string.Empty;

YearOfAdmission = DateTime.Now.Year;

Course = 1;

SelectedFaculty = null;

SelectedGroup = null;

ErrorMessage = string.Empty;

}

}

}

**StudentsViewModel.cs** – ViewModel для управления студентами, позволяет добавлять, редактировать и удалять студентов, группирует студентов по факультетам и группам. Содержит внутренние классы StudentGroupGrouping и StudentFacultyGrouping для группировки.

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

using System.Reactive;

using ReactiveUI;

using UniversityIS.Models;

using UniversityIS.Services;

using UniversityIS.Helpers;

namespace UniversityIS.ViewModels

{

*// Группа студентов, объединенных по учебной группе*

*// Используется для отображения студентов с группировкой по группам*

public class StudentGroupGrouping

{

public StudentGroupGrouping(Models.Group group, IEnumerable<Student> students)

{

Group = group;

Students = new ObservableCollection<Student>(students);

}

public Models.Group Group { get; }

public string GroupName => Group.Number;

public ObservableCollection<Student> Students { get; }

}

*// Группа студентов, объединенных по факультету*

*// Содержит вложенные группы по учебным группам*

public class StudentFacultyGrouping

{

public StudentFacultyGrouping(string facultyName, IEnumerable<StudentGroupGrouping> groups)

{

FacultyName = facultyName;

Groups = new ObservableCollection<StudentGroupGrouping>(groups);

}

public string FacultyName { get; }

public ObservableCollection<StudentGroupGrouping> Groups { get; }

}

*// ViewModel для управления студентами*

*// Позволяет добавлять, редактировать и удалять студентов*

*// Группирует студентов по факультетам и группам для удобного отображения*

public class StudentsViewModel : ViewModelBase

{

private readonly DataService \_dataService;

private readonly Action<Student>? \_showProfileAction;

private Student? \_selectedStudent;

private string \_lastName = string.Empty;

private string \_firstName = string.Empty;

private string \_middleName = string.Empty;

private string \_recordBookNumber = string.Empty;

private double \_gpa = 0.0;

private Group? \_selectedGroup;

private string \_errorMessage = string.Empty;

private ObservableCollection<StudentFacultyGrouping>? \_groupedStudents;

public StudentsViewModel(DataService dataService, Action<Student>? showProfileAction = null)

{

\_dataService = dataService;

\_showProfileAction = showProfileAction;

*// Подписываемся на изменения для обновления группировки*

\_dataService.Students.CollectionChanged += (s, e) => UpdateGroupedStudents();

\_dataService.Groups.CollectionChanged += (s, e) => UpdateGroupedStudents();

\_dataService.Faculties.CollectionChanged += (s, e) => UpdateGroupedStudents();

UpdateGroupedStudents(); *// Первоначальная группировка*

AddCommand = ReactiveCommand.Create(AddStudent, outputScheduler: RxApp.MainThreadScheduler);

UpdateCommand = ReactiveCommand.Create(UpdateStudent, outputScheduler: RxApp.MainThreadScheduler);

DeleteCommand = ReactiveCommand.Create(DeleteStudent, outputScheduler: RxApp.MainThreadScheduler);

ShowProfileCommand = ReactiveCommand.Create(ShowProfile, outputScheduler: RxApp.MainThreadScheduler);

}

public ObservableCollection<Student> Students => \_dataService.Students;

public ObservableCollection<Group> Groups => \_dataService.Groups;

public ObservableCollection<StudentFacultyGrouping>? GroupedStudents

{

get => \_groupedStudents;

set => this.RaiseAndSetIfChanged(ref \_groupedStudents, value);

}

*// Обновляет группировку студентов по факультетам и группам*

*// Вызывается автоматически при изменении данных студентов, групп или факультетов*

private void UpdateGroupedStudents()

{

var grouped = \_dataService.Students

.GroupBy(s => s.GroupId)

.Select(groupStudents =>

{

var group = \_dataService.GetGroup(groupStudents.Key);

if (group == null) return null;

var faculty = \_dataService.GetFaculty(group.FacultyId);

var facultyName = faculty?.Name ?? "Неизвестный факультет";

return new

{

FacultyName = facultyName,

Group = group,

Students = groupStudents.OrderBy(s => s.LastName).ThenBy(s => s.FirstName)

};

})

.Where(x => x != null)

.GroupBy(x => x!.FacultyName)

.Select(facultyGroup =>

{

var groupGroups = facultyGroup

.OrderBy(x => x!.Group.Number)

.Select(x => new StudentGroupGrouping(x!.Group, x.Students))

.ToList();

return new StudentFacultyGrouping(facultyGroup.Key, groupGroups);

})

.OrderBy(fg => fg.FacultyName)

.ToList();

GroupedStudents = new ObservableCollection<StudentFacultyGrouping>(grouped);

}

public Student? SelectedStudent

{

get => \_selectedStudent;

set

{

this.RaiseAndSetIfChanged(ref \_selectedStudent, value);

if (value != null)

{

LastName = value.LastName;

FirstName = value.FirstName;

MiddleName = value.MiddleName;

RecordBookNumber = value.RecordBookNumber;

GPA = value.GPA;

SelectedGroup = \_dataService.GetGroup(value.GroupId);

}

}

}

public string LastName

{

get => \_lastName;

set => this.RaiseAndSetIfChanged(ref \_lastName, value);

}

public string FirstName

{

get => \_firstName;

set => this.RaiseAndSetIfChanged(ref \_firstName, value);

}

public string MiddleName

{

get => \_middleName;

set => this.RaiseAndSetIfChanged(ref \_middleName, value);

}

public string RecordBookNumber

{

get => \_recordBookNumber;

set => this.RaiseAndSetIfChanged(ref \_recordBookNumber, value);

}

public double GPA

{

get => \_gpa;

set => this.RaiseAndSetIfChanged(ref \_gpa, value);

}

public Group? SelectedGroup

{

get => \_selectedGroup;

set => this.RaiseAndSetIfChanged(ref \_selectedGroup, value);

}

public string ErrorMessage

{

get => \_errorMessage;

set => this.RaiseAndSetIfChanged(ref \_errorMessage, value);

}

public ReactiveCommand<Unit, Unit> AddCommand { get; }

public ReactiveCommand<Unit, Unit> UpdateCommand { get; }

public ReactiveCommand<Unit, Unit> DeleteCommand { get; }

public ReactiveCommand<Unit, Unit> ShowProfileCommand { get; }

private void AddStudent()

{

ErrorMessage = string.Empty;

*// Валидация группы*

if (SelectedGroup == null)

{

ErrorMessage = ValidationHelper.GetErrorMessage("Группа", "not\_selected");

return;

}

*// Валидация фамилии*

if (string.IsNullOrWhiteSpace(LastName))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Фамилия", "empty");

return;

}

if (!ValidationHelper.IsValidName(LastName))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Фамилия", "invalid\_name");

return;

}

*// Валидация имени*

if (string.IsNullOrWhiteSpace(FirstName))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Имя", "empty");

return;

}

if (!ValidationHelper.IsValidName(FirstName))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Имя", "invalid\_name");

return;

}

*// Валидация отчества (может быть пустым)*

if (!string.IsNullOrWhiteSpace(MiddleName) && !ValidationHelper.IsValidName(MiddleName))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Отчество", "invalid\_name");

return;

}

var student = new Student

{

LastName = LastName,

FirstName = FirstName,

MiddleName = MiddleName,

RecordBookNumber = RecordBookNumber,

GPA = GPA,

GroupId = SelectedGroup.Id

};

\_dataService.Students.Add(student);

ClearFields();

}

private void UpdateStudent()

{

ErrorMessage = string.Empty;

if (SelectedStudent == null)

{

ErrorMessage = "Выберите студента для обновления.";

return;

}

*// Валидация группы*

if (SelectedGroup == null)

{

ErrorMessage = ValidationHelper.GetErrorMessage("Группа", "not\_selected");

return;

}

*// Валидация фамилии*

if (string.IsNullOrWhiteSpace(LastName))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Фамилия", "empty");

return;

}

if (!ValidationHelper.IsValidName(LastName))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Фамилия", "invalid\_name");

return;

}

*// Валидация имени*

if (string.IsNullOrWhiteSpace(FirstName))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Имя", "empty");

return;

}

if (!ValidationHelper.IsValidName(FirstName))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Имя", "invalid\_name");

return;

}

*// Валидация отчества (может быть пустым)*

if (!string.IsNullOrWhiteSpace(MiddleName) && !ValidationHelper.IsValidName(MiddleName))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Отчество", "invalid\_name");

return;

}

SelectedStudent.LastName = LastName;

SelectedStudent.FirstName = FirstName;

SelectedStudent.MiddleName = MiddleName;

SelectedStudent.RecordBookNumber = RecordBookNumber;

SelectedStudent.GPA = GPA;

SelectedStudent.GroupId = SelectedGroup.Id;

var index = Students.IndexOf(SelectedStudent);

if (index >= 0)

{

Students[index] = SelectedStudent;

}

}

private void DeleteStudent()

{

if (SelectedStudent == null) return;

\_dataService.Students.Remove(SelectedStudent);

Students.Remove(SelectedStudent);

ClearFields();

}

private void ClearFields()

{

LastName = string.Empty;

FirstName = string.Empty;

MiddleName = string.Empty;

RecordBookNumber = string.Empty;

GPA = 0.0;

SelectedGroup = null;

SelectedStudent = null;

ErrorMessage = string.Empty;

}

private void ShowProfile()

{

if (SelectedStudent == null)

{

ErrorMessage = "Выберите студента для просмотра профиля.";

return;

}

\_showProfileAction?.Invoke(SelectedStudent);

}

}

}

**TeachersViewModel.cs** – ViewModel для управления преподавателями, позволяет добавлять, редактировать и удалять преподавателей, управляет информацией о должности, степени, звании и научной деятельности.

using System;

using System.Collections.ObjectModel;

using System.Linq;

using System.Reactive;

using ReactiveUI;

using UniversityIS.Models;

using UniversityIS.Services;

using UniversityIS.Helpers;

namespace UniversityIS.ViewModels

{

*// ViewModel для управления преподавателями*

*// Позволяет добавлять, редактировать и удалять преподавателей*

*// Управляет информацией о должности, степени, звании и научной деятельности*

public class TeachersViewModel : ViewModelBase

{

private readonly DataService \_dataService;

private readonly Action<Teacher>? \_showProfileAction;

private Teacher? \_selectedTeacher;

private string \_lastName = string.Empty;

private string \_firstName = string.Empty;

private string \_middleName = string.Empty;

private TeacherPosition \_position = TeacherPosition.Assistant;

private AcademicDegree \_degree = AcademicDegree.None;

private AcademicTitle \_title = AcademicTitle.None;

private bool \_isPostgraduate = false;

private bool \_leadsResearchTopics = false;

private bool \_leadsResearchDirections = false;

private Department? \_selectedDepartment;

private EnumItem<TeacherPosition>? \_selectedPositionItem;

private EnumItem<AcademicDegree>? \_selectedDegreeItem;

private EnumItem<AcademicTitle>? \_selectedTitleItem;

private string \_errorMessage = string.Empty;

public TeachersViewModel(DataService dataService, Action<Teacher>? showProfileAction = null)

{

\_dataService = dataService;

\_showProfileAction = showProfileAction;

Positions = new ObservableCollection<EnumItem<TeacherPosition>>(EnumHelper.GetTeacherPositions());

Degrees = new ObservableCollection<EnumItem<AcademicDegree>>(EnumHelper.GetAcademicDegrees());

Titles = new ObservableCollection<EnumItem<AcademicTitle>>(EnumHelper.GetAcademicTitles());

AddCommand = ReactiveCommand.Create(AddTeacher, outputScheduler: RxApp.MainThreadScheduler);

UpdateCommand = ReactiveCommand.Create(UpdateTeacher, outputScheduler: RxApp.MainThreadScheduler);

DeleteCommand = ReactiveCommand.Create(DeleteTeacher, outputScheduler: RxApp.MainThreadScheduler);

ShowProfileCommand = ReactiveCommand.Create(ShowProfile, outputScheduler: RxApp.MainThreadScheduler);

}

public ObservableCollection<Teacher> Teachers => \_dataService.Teachers;

public ObservableCollection<Department> Departments => \_dataService.Departments;

public ObservableCollection<EnumItem<TeacherPosition>> Positions { get; }

public ObservableCollection<EnumItem<AcademicDegree>> Degrees { get; }

public ObservableCollection<EnumItem<AcademicTitle>> Titles { get; }

public Teacher? SelectedTeacher

{

get => \_selectedTeacher;

set

{

this.RaiseAndSetIfChanged(ref \_selectedTeacher, value);

if (value != null)

{

LastName = value.LastName;

FirstName = value.FirstName;

MiddleName = value.MiddleName;

Position = value.Position;

Degree = value.Degree;

Title = value.Title;

SelectedPositionItem = Positions.FirstOrDefault(x => x.Value == value.Position);

SelectedDegreeItem = Degrees.FirstOrDefault(x => x.Value == value.Degree);

SelectedTitleItem = Titles.FirstOrDefault(x => x.Value == value.Title);

IsPostgraduate = value.IsPostgraduate;

LeadsResearchTopics = value.LeadsResearchTopics;

LeadsResearchDirections = value.LeadsResearchDirections;

SelectedDepartment = \_dataService.GetDepartment(value.DepartmentId);

}

}

}

public string LastName

{

get => \_lastName;

set => this.RaiseAndSetIfChanged(ref \_lastName, value);

}

public string FirstName

{

get => \_firstName;

set => this.RaiseAndSetIfChanged(ref \_firstName, value);

}

public string MiddleName

{

get => \_middleName;

set => this.RaiseAndSetIfChanged(ref \_middleName, value);

}

public TeacherPosition Position

{

get => \_position;

set => this.RaiseAndSetIfChanged(ref \_position, value);

}

public AcademicDegree Degree

{

get => \_degree;

set => this.RaiseAndSetIfChanged(ref \_degree, value);

}

public AcademicTitle Title

{

get => \_title;

set => this.RaiseAndSetIfChanged(ref \_title, value);

}

public EnumItem<TeacherPosition>? SelectedPositionItem

{

get => \_selectedPositionItem;

set

{

this.RaiseAndSetIfChanged(ref \_selectedPositionItem, value);

if (value != null)

Position = value.Value;

}

}

public EnumItem<AcademicDegree>? SelectedDegreeItem

{

get => \_selectedDegreeItem;

set

{

this.RaiseAndSetIfChanged(ref \_selectedDegreeItem, value);

if (value != null)

Degree = value.Value;

}

}

public EnumItem<AcademicTitle>? SelectedTitleItem

{

get => \_selectedTitleItem;

set

{

this.RaiseAndSetIfChanged(ref \_selectedTitleItem, value);

if (value != null)

Title = value.Value;

}

}

public bool IsPostgraduate

{

get => \_isPostgraduate;

set => this.RaiseAndSetIfChanged(ref \_isPostgraduate, value);

}

public bool LeadsResearchTopics

{

get => \_leadsResearchTopics;

set => this.RaiseAndSetIfChanged(ref \_leadsResearchTopics, value);

}

public bool LeadsResearchDirections

{

get => \_leadsResearchDirections;

set => this.RaiseAndSetIfChanged(ref \_leadsResearchDirections, value);

}

public Department? SelectedDepartment

{

get => \_selectedDepartment;

set => this.RaiseAndSetIfChanged(ref \_selectedDepartment, value);

}

public string ErrorMessage

{

get => \_errorMessage;

set => this.RaiseAndSetIfChanged(ref \_errorMessage, value);

}

public ReactiveCommand<Unit, Unit> AddCommand { get; }

public ReactiveCommand<Unit, Unit> UpdateCommand { get; }

public ReactiveCommand<Unit, Unit> DeleteCommand { get; }

public ReactiveCommand<Unit, Unit> ShowProfileCommand { get; }

private void AddTeacher()

{

ErrorMessage = string.Empty;

*// Валидация кафедры*

if (SelectedDepartment == null)

{

ErrorMessage = ValidationHelper.GetErrorMessage("Кафедра", "not\_selected");

return;

}

*// Валидация фамилии*

if (string.IsNullOrWhiteSpace(LastName))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Фамилия", "empty");

return;

}

if (!ValidationHelper.IsValidName(LastName))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Фамилия", "invalid\_name");

return;

}

*// Валидация имени*

if (string.IsNullOrWhiteSpace(FirstName))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Имя", "empty");

return;

}

if (!ValidationHelper.IsValidName(FirstName))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Имя", "invalid\_name");

return;

}

*// Валидация отчества (может быть пустым)*

if (!string.IsNullOrWhiteSpace(MiddleName) && !ValidationHelper.IsValidName(MiddleName))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Отчество", "invalid\_name");

return;

}

var teacher = new Teacher

{

LastName = LastName,

FirstName = FirstName,

MiddleName = MiddleName,

Position = Position,

Degree = Degree,

Title = Title,

IsPostgraduate = IsPostgraduate,

LeadsResearchTopics = LeadsResearchTopics,

LeadsResearchDirections = LeadsResearchDirections,

DepartmentId = SelectedDepartment.Id

};

\_dataService.Teachers.Add(teacher);

ClearFields();

}

private void UpdateTeacher()

{

ErrorMessage = string.Empty;

if (SelectedTeacher == null)

{

ErrorMessage = "Выберите преподавателя для обновления.";

return;

}

*// Валидация кафедры*

if (SelectedDepartment == null)

{

ErrorMessage = ValidationHelper.GetErrorMessage("Кафедра", "not\_selected");

return;

}

*// Валидация фамилии*

if (string.IsNullOrWhiteSpace(LastName))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Фамилия", "empty");

return;

}

if (!ValidationHelper.IsValidName(LastName))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Фамилия", "invalid\_name");

return;

}

*// Валидация имени*

if (string.IsNullOrWhiteSpace(FirstName))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Имя", "empty");

return;

}

if (!ValidationHelper.IsValidName(FirstName))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Имя", "invalid\_name");

return;

}

*// Валидация отчества (может быть пустым)*

if (!string.IsNullOrWhiteSpace(MiddleName) && !ValidationHelper.IsValidName(MiddleName))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Отчество", "invalid\_name");

return;

}

SelectedTeacher.LastName = LastName;

SelectedTeacher.FirstName = FirstName;

SelectedTeacher.MiddleName = MiddleName;

SelectedTeacher.Position = Position;

SelectedTeacher.Degree = Degree;

SelectedTeacher.Title = Title;

SelectedTeacher.IsPostgraduate = IsPostgraduate;

SelectedTeacher.LeadsResearchTopics = LeadsResearchTopics;

SelectedTeacher.LeadsResearchDirections = LeadsResearchDirections;

SelectedTeacher.DepartmentId = SelectedDepartment.Id;

var index = Teachers.IndexOf(SelectedTeacher);

if (index >= 0)

{

Teachers[index] = SelectedTeacher;

}

}

private void DeleteTeacher()

{

if (SelectedTeacher == null) return;

\_dataService.Teachers.Remove(SelectedTeacher);

Teachers.Remove(SelectedTeacher);

ClearFields();

}

private void ClearFields()

{

LastName = string.Empty;

FirstName = string.Empty;

MiddleName = string.Empty;

Position = TeacherPosition.Assistant;

Degree = AcademicDegree.None;

Title = AcademicTitle.None;

IsPostgraduate = false;

LeadsResearchTopics = false;

LeadsResearchDirections = false;

SelectedDepartment = null;

SelectedTeacher = null;

ErrorMessage = string.Empty;

}

private void ShowProfile()

{

if (SelectedTeacher == null)

{

ErrorMessage = "Выберите преподавателя для просмотра профиля.";

return;

}

\_showProfileAction?.Invoke(SelectedTeacher);

}

}

}

**DisciplinesViewModel.cs** – ViewModel для управления дисциплинами, позволяет добавлять, редактировать и удалять дисциплины, автоматически вычисляет курс на основе семестра. Содержит внутренний класс GroupDisciplines для группировки.

using System;

using System.Collections.ObjectModel;

using System.Linq;

using System.Reactive;

using ReactiveUI;

using UniversityIS.Models;

using UniversityIS.Services;

using UniversityIS.Helpers;

namespace UniversityIS.ViewModels

{

*// Группа дисциплин, объединенных по группе*

*// Используется для отображения дисциплин с группировкой по группам*

public class GroupDisciplines

{

public string GroupName { get; set; } = string.Empty;

public ObservableCollection<Discipline> Disciplines { get; set; } = new();

}

*// ViewModel для управления дисциплинами*

*// Позволяет добавлять, редактировать и удалять дисциплины*

*// Автоматически вычисляет курс на основе семестра*

*// Группирует дисциплины по группам для удобного отображения*

public class DisciplinesViewModel : ViewModelBase

{

private readonly DataService \_dataService;

private Discipline? \_selectedDiscipline;

private string \_name = string.Empty;

private int \_course = 1;

private int \_semester = 1;

private int \_lectureHours = 0;

private int \_seminarHours = 0;

private int \_laboratoryHours = 0;

private ControlForm \_controlForm = ControlForm.Exam;

private EnumItem<ControlForm>? \_selectedControlFormItem;

private string \_errorMessage = string.Empty;

private Models.Group? \_selectedGroup;

private ObservableCollection<GroupDisciplines> \_groupedDisciplines = new();

public DisciplinesViewModel(DataService dataService)

{

\_dataService = dataService;

ControlForms = new ObservableCollection<EnumItem<ControlForm>>(EnumHelper.GetControlForms());

AddCommand = ReactiveCommand.Create(AddDiscipline, outputScheduler: RxApp.MainThreadScheduler);

UpdateCommand = ReactiveCommand.Create(UpdateDiscipline, outputScheduler: RxApp.MainThreadScheduler);

DeleteCommand = ReactiveCommand.Create(DeleteDiscipline, outputScheduler: RxApp.MainThreadScheduler);

*// Подписываемся на изменения для обновления группировки*

\_dataService.Disciplines.CollectionChanged += (s, e) => UpdateGroupedDisciplines();

\_dataService.Groups.CollectionChanged += (s, e) => UpdateGroupedDisciplines();

UpdateGroupedDisciplines();

}

public ObservableCollection<Discipline> Disciplines => \_dataService.Disciplines;

public ObservableCollection<Models.Group> Groups => \_dataService.Groups;

public ObservableCollection<EnumItem<ControlForm>> ControlForms { get; }

public ObservableCollection<GroupDisciplines> GroupedDisciplines

{

get => \_groupedDisciplines;

set => this.RaiseAndSetIfChanged(ref \_groupedDisciplines, value);

}

public Discipline? SelectedDiscipline

{

get => \_selectedDiscipline;

set

{

this.RaiseAndSetIfChanged(ref \_selectedDiscipline, value);

if (value != null)

{

Name = value.Name;

Course = value.Course;

Semester = value.Semester;

LectureHours = value.LectureHours;

SeminarHours = value.SeminarHours;

LaboratoryHours = value.LaboratoryHours;

ControlForm = value.ControlForm;

SelectedControlFormItem = ControlForms.FirstOrDefault(x => x.Value == value.ControlForm);

SelectedGroup = \_dataService.GetGroup(value.GroupId);

}

}

}

public string Name

{

get => \_name;

set => this.RaiseAndSetIfChanged(ref \_name, value);

}

public int Course

{

get => \_course;

set => this.RaiseAndSetIfChanged(ref \_course, value);

}

public int Semester

{

get => \_semester;

set => this.RaiseAndSetIfChanged(ref \_semester, value);

}

public int LectureHours

{

get => \_lectureHours;

set => this.RaiseAndSetIfChanged(ref \_lectureHours, value);

}

public int SeminarHours

{

get => \_seminarHours;

set => this.RaiseAndSetIfChanged(ref \_seminarHours, value);

}

public int LaboratoryHours

{

get => \_laboratoryHours;

set => this.RaiseAndSetIfChanged(ref \_laboratoryHours, value);

}

public ControlForm ControlForm

{

get => \_controlForm;

set => this.RaiseAndSetIfChanged(ref \_controlForm, value);

}

public EnumItem<ControlForm>? SelectedControlFormItem

{

get => \_selectedControlFormItem;

set

{

this.RaiseAndSetIfChanged(ref \_selectedControlFormItem, value);

if (value != null)

ControlForm = value.Value;

}

}

public Models.Group? SelectedGroup

{

get => \_selectedGroup;

set => this.RaiseAndSetIfChanged(ref \_selectedGroup, value);

}

public string ErrorMessage

{

get => \_errorMessage;

set => this.RaiseAndSetIfChanged(ref \_errorMessage, value);

}

public ReactiveCommand<Unit, Unit> AddCommand { get; }

public ReactiveCommand<Unit, Unit> UpdateCommand { get; }

public ReactiveCommand<Unit, Unit> DeleteCommand { get; }

private void AddDiscipline()

{

ErrorMessage = string.Empty;

*// Валидация группы*

if (SelectedGroup == null)

{

ErrorMessage = ValidationHelper.GetErrorMessage("Группа", "not\_selected");

return;

}

*// Валидация названия дисциплины*

if (string.IsNullOrWhiteSpace(Name))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Название дисциплины", "empty");

return;

}

if (!ValidationHelper.IsValidNameWithNumbers(Name))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Название дисциплины", "invalid\_name\_with\_numbers");

return;

}

*// Валидация курса*

if (!ValidationHelper.IsValidCourse(Course.ToString()))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Курс", "invalid\_course");

return;

}

*// Валидация семестра*

if (Semester < 1 || Semester > 12)

{

ErrorMessage = "Поле \"Семестр\" должно быть числом от 1 до 12.";

return;

}

*// Валидация часов лекций*

if (LectureHours < 0 || LectureHours > 1000)

{

ErrorMessage = "Поле \"Часы лекций\" должно быть от 0 до 1000.";

return;

}

*// Валидация часов семинаров*

if (SeminarHours < 0 || SeminarHours > 1000)

{

ErrorMessage = "Поле \"Часы семинаров\" должно быть от 0 до 1000.";

return;

}

*// Валидация часов лабораторных*

if (LaboratoryHours < 0 || LaboratoryHours > 1000)

{

ErrorMessage = "Поле \"Часы лабораторных\" должно быть от 0 до 1000.";

return;

}

*// Проверка, что хотя бы один тип занятий имеет часы*

if (LectureHours == 0 && SeminarHours == 0 && LaboratoryHours == 0)

{

ErrorMessage = "Хотя бы один тип занятий должен иметь ненулевое количество часов.";

return;

}

var discipline = new Discipline

{

Name = Name,

LectureHours = LectureHours,

SeminarHours = SeminarHours,

LaboratoryHours = LaboratoryHours,

ControlForm = ControlForm,

GroupId = SelectedGroup.Id

};

*// Устанавливаем семестр и автоматически вычисляем курс*

discipline.SetSemester(Semester);

\_dataService.Disciplines.Add(discipline);

ClearFields();

}

private void UpdateDiscipline()

{

ErrorMessage = string.Empty;

if (SelectedDiscipline == null)

{

ErrorMessage = "Выберите дисциплину для обновления.";

return;

}

*// Валидация группы*

if (SelectedGroup == null)

{

ErrorMessage = ValidationHelper.GetErrorMessage("Группа", "not\_selected");

return;

}

*// Валидация названия дисциплины*

if (string.IsNullOrWhiteSpace(Name))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Название дисциплины", "empty");

return;

}

if (!ValidationHelper.IsValidNameWithNumbers(Name))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Название дисциплины", "invalid\_name\_with\_numbers");

return;

}

*// Валидация семестра*

if (!ValidationHelper.IsValidSemester(Semester.ToString()))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Семестр", "invalid\_semester");

return;

}

*// Валидация часов лекций*

if (LectureHours < 0 || LectureHours > 1000)

{

ErrorMessage = "Поле \"Часы лекций\" должно быть от 0 до 1000.";

return;

}

*// Валидация часов семинаров*

if (SeminarHours < 0 || SeminarHours > 1000)

{

ErrorMessage = "Поле \"Часы семинаров\" должно быть от 0 до 1000.";

return;

}

*// Валидация часов лабораторных*

if (LaboratoryHours < 0 || LaboratoryHours > 1000)

{

ErrorMessage = "Поле \"Часы лабораторных\" должно быть от 0 до 1000.";

return;

}

*// Проверка, что хотя бы один тип занятий имеет часы*

if (LectureHours == 0 && SeminarHours == 0 && LaboratoryHours == 0)

{

ErrorMessage = "Хотя бы один тип занятий должен иметь ненулевое количество часов.";

return;

}

SelectedDiscipline.Name = Name;

SelectedDiscipline.LectureHours = LectureHours;

SelectedDiscipline.SeminarHours = SeminarHours;

SelectedDiscipline.LaboratoryHours = LaboratoryHours;

SelectedDiscipline.ControlForm = ControlForm;

SelectedDiscipline.GroupId = SelectedGroup.Id;

*// Устанавливаем семестр и автоматически вычисляем курс*

SelectedDiscipline.SetSemester(Semester);

var index = Disciplines.IndexOf(SelectedDiscipline);

if (index >= 0)

{

Disciplines[index] = SelectedDiscipline;

}

}

private void DeleteDiscipline()

{

if (SelectedDiscipline == null) return;

\_dataService.Disciplines.Remove(SelectedDiscipline);

Disciplines.Remove(SelectedDiscipline);

ClearFields();

}

private void ClearFields()

{

Name = string.Empty;

Course = 1;

Semester = 1;

LectureHours = 0;

SeminarHours = 0;

LaboratoryHours = 0;

ControlForm = ControlForm.Exam;

SelectedGroup = null;

SelectedDiscipline = null;

ErrorMessage = string.Empty;

}

private void UpdateGroupedDisciplines()

{

var groups = new ObservableCollection<GroupDisciplines>();

*// Группируем дисциплины по группам*

var disciplinesByGroup = \_dataService.Disciplines

.GroupBy(d => d.GroupId)

.OrderBy(g => \_dataService.GetGroup(g.Key)?.Number ?? "Неизвестная группа");

foreach (var group in disciplinesByGroup)

{

var groupObj = \_dataService.GetGroup(group.Key);

if (groupObj == null) continue;

var groupDisciplines = new GroupDisciplines

{

GroupName = groupObj.Number,

Disciplines = new ObservableCollection<Discipline>(group.OrderBy(d => d.Name))

};

groups.Add(groupDisciplines);

}

GroupedDisciplines = groups;

}

}

}

**WorkLoadsViewModel.cs** – ViewModel для управления учебной нагрузкой преподавателей, позволяет назначать преподавателям дисциплины для конкретных групп, валидирует правила распределения нагрузки.

using System;

using System.Collections.ObjectModel;

using System.Linq;

using System.Reactive;

using System.Text.RegularExpressions;

using ReactiveUI;

using UniversityIS.Models;

using UniversityIS.Services;

using UniversityIS.Helpers;

namespace UniversityIS.ViewModels

{

*// ViewModel для управления учебной нагрузкой преподавателей*

*// Позволяет назначать преподавателям дисциплины для конкретных групп*

*// Валидирует правила: один преподаватель на лекции/семинары, несколько на лабораторные*

public class WorkLoadsViewModel : ViewModelBase

{

private readonly DataService \_dataService;

private WorkLoad? \_selectedWorkLoad;

private Teacher? \_selectedTeacher;

private Discipline? \_selectedDiscipline;

private Models.Group? \_selectedGroup;

private LessonType \_lessonType = LessonType.Lecture;

private int \_hours = 0;

private string \_academicYear = $"{DateTime.Now.Year}/{DateTime.Now.Year + 1}";

private int \_semester = 1;

private EnumItem<LessonType>? \_selectedLessonTypeItem;

private string \_errorMessage = string.Empty;

private ObservableCollection<Discipline> \_availableDisciplines = new();

public WorkLoadsViewModel(DataService dataService)

{

\_dataService = dataService;

LessonTypes = new ObservableCollection<EnumItem<LessonType>>(EnumHelper.GetLessonTypes());

AddCommand = ReactiveCommand.Create(AddWorkLoad, outputScheduler: RxApp.MainThreadScheduler);

UpdateCommand = ReactiveCommand.Create(UpdateWorkLoad, outputScheduler: RxApp.MainThreadScheduler);

DeleteCommand = ReactiveCommand.Create(DeleteWorkLoad, outputScheduler: RxApp.MainThreadScheduler);

*// Подписываемся на изменения для обновления списка дисциплин*

\_dataService.TeacherDisciplines.CollectionChanged += (s, e) => UpdateAvailableDisciplines();

this.WhenAnyValue(x => x.SelectedTeacher).Subscribe(\_ => UpdateAvailableDisciplines());

UpdateAvailableDisciplines();

}

public ObservableCollection<WorkLoad> WorkLoads => \_dataService.WorkLoads;

public ObservableCollection<Teacher> Teachers => \_dataService.Teachers;

public ObservableCollection<Discipline> AvailableDisciplines

{

get => \_availableDisciplines;

set => this.RaiseAndSetIfChanged(ref \_availableDisciplines, value);

}

public ObservableCollection<Models.Group> Groups => \_dataService.Groups;

public ObservableCollection<EnumItem<LessonType>> LessonTypes { get; }

public WorkLoad? SelectedWorkLoad

{

get => \_selectedWorkLoad;

set

{

this.RaiseAndSetIfChanged(ref \_selectedWorkLoad, value);

if (value != null)

{

SelectedTeacher = \_dataService.GetTeacher(value.TeacherId);

SelectedDiscipline = \_dataService.GetDiscipline(value.DisciplineId);

SelectedGroup = \_dataService.GetGroup(value.GroupId);

LessonType = value.LessonType;

SelectedLessonTypeItem = LessonTypes.FirstOrDefault(x => x.Value == value.LessonType);

Hours = value.Hours;

AcademicYear = value.AcademicYear;

Semester = value.Semester;

}

}

}

public Teacher? SelectedTeacher

{

get => \_selectedTeacher;

set => this.RaiseAndSetIfChanged(ref \_selectedTeacher, value);

}

public Discipline? SelectedDiscipline

{

get => \_selectedDiscipline;

set => this.RaiseAndSetIfChanged(ref \_selectedDiscipline, value);

}

public Models.Group? SelectedGroup

{

get => \_selectedGroup;

set => this.RaiseAndSetIfChanged(ref \_selectedGroup, value);

}

public LessonType LessonType

{

get => \_lessonType;

set => this.RaiseAndSetIfChanged(ref \_lessonType, value);

}

public EnumItem<LessonType>? SelectedLessonTypeItem

{

get => \_selectedLessonTypeItem;

set

{

this.RaiseAndSetIfChanged(ref \_selectedLessonTypeItem, value);

if (value != null)

LessonType = value.Value;

}

}

public int Hours

{

get => \_hours;

set => this.RaiseAndSetIfChanged(ref \_hours, value);

}

public string AcademicYear

{

get => \_academicYear;

set => this.RaiseAndSetIfChanged(ref \_academicYear, value);

}

public int Semester

{

get => \_semester;

set => this.RaiseAndSetIfChanged(ref \_semester, value);

}

public string ErrorMessage

{

get => \_errorMessage;

set => this.RaiseAndSetIfChanged(ref \_errorMessage, value);

}

public ReactiveCommand<Unit, Unit> AddCommand { get; }

public ReactiveCommand<Unit, Unit> UpdateCommand { get; }

public ReactiveCommand<Unit, Unit> DeleteCommand { get; }

private void AddWorkLoad()

{

ErrorMessage = string.Empty;

*// Валидация преподавателя*

if (SelectedTeacher == null)

{

ErrorMessage = ValidationHelper.GetErrorMessage("Преподаватель", "not\_selected");

return;

}

*// Валидация дисциплины*

if (SelectedDiscipline == null)

{

ErrorMessage = ValidationHelper.GetErrorMessage("Дисциплина", "not\_selected");

return;

}

*// Валидация группы*

if (SelectedGroup == null)

{

ErrorMessage = ValidationHelper.GetErrorMessage("Группа", "not\_selected");

return;

}

*// Валидация часов*

if (!ValidationHelper.IsValidHours(Hours.ToString()))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Часы", "invalid\_hours");

return;

}

*// Валидация учебного года (формат: 2023/2024)*

if (!Regex.IsMatch(AcademicYear, @"^\d{4}/\d{4}$"))

{

ErrorMessage = "Поле \"Учебный год\" должно иметь формат ГГГГ/ГГГГ (например: 2023/2024).";

return;

}

*// Валидация семестра*

if (Semester < 1 || Semester > 12)

{

ErrorMessage = "Поле \"Семестр\" должно быть числом от 1 до 12.";

return;

}

*// Проверка: дисциплина должна быть у выбранной группы*

if (SelectedDiscipline.GroupId != SelectedGroup.Id)

{

ErrorMessage = "Выбранная дисциплина не относится к данной группе.";

return;

}

*// Проверка правила: один преподаватель на лекции/семинары*

if (LessonType == LessonType.Lecture || LessonType == LessonType.Seminar)

{

var existingWorkLoad = \_dataService.WorkLoads.FirstOrDefault(wl =>

wl.DisciplineId == SelectedDiscipline.Id &&

wl.GroupId == SelectedGroup.Id &&

wl.Semester == Semester &&

(wl.LessonType == LessonType.Lecture || wl.LessonType == LessonType.Seminar) &&

wl.TeacherId != SelectedTeacher.Id);

if (existingWorkLoad != null)

{

var lessonTypeName = LessonType == LessonType.Lecture ? "лекции" : "семинары";

ErrorMessage = $"У этой группы уже есть преподаватель на {lessonTypeName} по данной дисциплине.";

return;

}

}

var workLoad = new WorkLoad

{

TeacherId = SelectedTeacher.Id,

DisciplineId = SelectedDiscipline.Id,

GroupId = SelectedGroup.Id,

LessonType = LessonType,

Hours = Hours,

AcademicYear = AcademicYear,

Semester = Semester

};

\_dataService.WorkLoads.Add(workLoad);

ClearFields();

}

private void UpdateWorkLoad()

{

ErrorMessage = string.Empty;

if (SelectedWorkLoad == null)

{

ErrorMessage = "Выберите нагрузку для обновления.";

return;

}

*// Валидация преподавателя*

if (SelectedTeacher == null)

{

ErrorMessage = ValidationHelper.GetErrorMessage("Преподаватель", "not\_selected");

return;

}

*// Валидация дисциплины*

if (SelectedDiscipline == null)

{

ErrorMessage = ValidationHelper.GetErrorMessage("Дисциплина", "not\_selected");

return;

}

*// Валидация группы*

if (SelectedGroup == null)

{

ErrorMessage = ValidationHelper.GetErrorMessage("Группа", "not\_selected");

return;

}

*// Валидация часов*

if (!ValidationHelper.IsValidHours(Hours.ToString()))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Часы", "invalid\_hours");

return;

}

*// Валидация учебного года (формат: 2023/2024)*

if (!Regex.IsMatch(AcademicYear, @"^\d{4}/\d{4}$"))

{

ErrorMessage = "Поле \"Учебный год\" должно иметь формат ГГГГ/ГГГГ (например: 2023/2024).";

return;

}

*// Валидация семестра*

if (Semester < 1 || Semester > 12)

{

ErrorMessage = "Поле \"Семестр\" должно быть числом от 1 до 12.";

return;

}

*// Проверка: дисциплина должна быть у выбранной группы*

if (SelectedDiscipline.GroupId != SelectedGroup.Id)

{

ErrorMessage = "Выбранная дисциплина не относится к данной группе.";

return;

}

*// Проверка правила: один преподаватель на лекции/семинары*

if (LessonType == LessonType.Lecture || LessonType == LessonType.Seminar)

{

var existingWorkLoad = \_dataService.WorkLoads.FirstOrDefault(wl =>

wl.Id != SelectedWorkLoad.Id && *// Исключаем текущую запись*

wl.DisciplineId == SelectedDiscipline.Id &&

wl.GroupId == SelectedGroup.Id &&

wl.Semester == Semester &&

(wl.LessonType == LessonType.Lecture || wl.LessonType == LessonType.Seminar) &&

wl.TeacherId != SelectedTeacher.Id);

if (existingWorkLoad != null)

{

var lessonTypeName = LessonType == LessonType.Lecture ? "лекции" : "семинары";

ErrorMessage = $"У этой группы уже есть преподаватель на {lessonTypeName} по данной дисциплине.";

return;

}

}

SelectedWorkLoad.TeacherId = SelectedTeacher.Id;

SelectedWorkLoad.DisciplineId = SelectedDiscipline.Id;

SelectedWorkLoad.GroupId = SelectedGroup.Id;

SelectedWorkLoad.LessonType = LessonType;

SelectedWorkLoad.Hours = Hours;

SelectedWorkLoad.AcademicYear = AcademicYear;

SelectedWorkLoad.Semester = Semester;

var index = WorkLoads.IndexOf(SelectedWorkLoad);

if (index >= 0)

{

WorkLoads[index] = SelectedWorkLoad;

}

}

*// Обновляет список доступных дисциплин в зависимости от выбранного преподавателя*

*// Показывает только те дисциплины, которые преподаватель может вести*

*// Вызывается автоматически при изменении выбранного преподавателя*

private void UpdateAvailableDisciplines()

{

if (SelectedTeacher == null)

{

AvailableDisciplines = new ObservableCollection<Discipline>();

return;

}

*// Получаем идентификаторы дисциплин, которые ведет выбранный преподаватель*

var teacherDisciplineIds = \_dataService.TeacherDisciplines

.Where(td => td.TeacherId == SelectedTeacher.Id)

.Select(td => td.DisciplineId)

.ToHashSet();

*// Фильтруем все дисциплины, оставляя только те, что есть у преподавателя*

var disciplines = \_dataService.Disciplines

.Where(d => teacherDisciplineIds.Contains(d.Id))

.OrderBy(d => d.Name)

.ToList();

AvailableDisciplines = new ObservableCollection<Discipline>(disciplines);

}

*// Удаляет выбранную учебную нагрузку из базы данных*

private void DeleteWorkLoad()

{

if (SelectedWorkLoad == null) return;

\_dataService.WorkLoads.Remove(SelectedWorkLoad);

WorkLoads.Remove(SelectedWorkLoad);

ClearFields();

}

private void ClearFields()

{

SelectedTeacher = null;

SelectedDiscipline = null;

SelectedGroup = null;

LessonType = LessonType.Lecture;

Hours = 0;

AcademicYear = $"{DateTime.Now.Year}/{DateTime.Now.Year + 1}";

Semester = 1;

SelectedWorkLoad = null;

ErrorMessage = string.Empty;

}

}

}

**ThesisWorksViewModel.cs** – ViewModel для управления дипломными работами, позволяет добавлять, редактировать и удалять дипломные работы, валидирует права научного руководителя.

using System;

using System.Collections.ObjectModel;

using System.Linq;

using System.Reactive;

using ReactiveUI;

using UniversityIS.Models;

using UniversityIS.Services;

using UniversityIS.Helpers;

namespace UniversityIS.ViewModels

{

*// ViewModel для управления дипломными работами*

*// Позволяет добавлять, редактировать и удалять дипломные работы*

*// Валидирует, что научный руководитель имеет право руководить дипломными работами*

public class ThesisWorksViewModel : ViewModelBase

{

private readonly DataService \_dataService;

private ThesisWork? \_selectedThesisWork;

private string \_title = string.Empty;

private Student? \_selectedStudent;

private Teacher? \_selectedSupervisor;

private int \_year = DateTime.Now.Year;

private int? \_grade;

private string \_errorMessage = string.Empty;

public ThesisWorksViewModel(DataService dataService)

{

\_dataService = dataService;

AddCommand = ReactiveCommand.Create(AddThesisWork, outputScheduler: RxApp.MainThreadScheduler);

UpdateCommand = ReactiveCommand.Create(UpdateThesisWork, outputScheduler: RxApp.MainThreadScheduler);

DeleteCommand = ReactiveCommand.Create(DeleteThesisWork, outputScheduler: RxApp.MainThreadScheduler);

}

public ObservableCollection<ThesisWork> ThesisWorks => \_dataService.ThesisWorks;

public ObservableCollection<Student> Students => \_dataService.Students;

*// Только преподаватели, которые руководят научными темами или направлениями*

public ObservableCollection<Teacher> Supervisors => new ObservableCollection<Teacher>(

\_dataService.Teachers.Where(t => t.LeadsResearchTopics || t.LeadsResearchDirections)

);

public ThesisWork? SelectedThesisWork

{

get => \_selectedThesisWork;

set

{

this.RaiseAndSetIfChanged(ref \_selectedThesisWork, value);

if (value != null)

{

Title = value.Title;

SelectedStudent = \_dataService.GetStudent(value.StudentId);

SelectedSupervisor = \_dataService.GetTeacher(value.SupervisorId);

Year = value.Year;

Grade = value.Grade;

}

}

}

public string Title

{

get => \_title;

set => this.RaiseAndSetIfChanged(ref \_title, value);

}

public Student? SelectedStudent

{

get => \_selectedStudent;

set => this.RaiseAndSetIfChanged(ref \_selectedStudent, value);

}

public Teacher? SelectedSupervisor

{

get => \_selectedSupervisor;

set => this.RaiseAndSetIfChanged(ref \_selectedSupervisor, value);

}

public int Year

{

get => \_year;

set => this.RaiseAndSetIfChanged(ref \_year, value);

}

public int? Grade

{

get => \_grade;

set => this.RaiseAndSetIfChanged(ref \_grade, value);

}

public string ErrorMessage

{

get => \_errorMessage;

set => this.RaiseAndSetIfChanged(ref \_errorMessage, value);

}

public ReactiveCommand<Unit, Unit> AddCommand { get; }

public ReactiveCommand<Unit, Unit> UpdateCommand { get; }

public ReactiveCommand<Unit, Unit> DeleteCommand { get; }

private void AddThesisWork()

{

ErrorMessage = string.Empty;

*// Валидация студента*

if (SelectedStudent == null)

{

ErrorMessage = ValidationHelper.GetErrorMessage("Студент", "not\_selected");

return;

}

*// Валидация научного руководителя*

if (SelectedSupervisor == null)

{

ErrorMessage = ValidationHelper.GetErrorMessage("Научный руководитель", "not\_selected");

return;

}

*// Проверка: преподаватель должен руководить научными темами или направлениями*

if (!SelectedSupervisor.LeadsResearchTopics && !SelectedSupervisor.LeadsResearchDirections)

{

ErrorMessage = "Выбранный преподаватель не может быть научным руководителем. " +

"Научный руководитель должен иметь отметку \"Руководит научными темами\" " +

"или \"Руководит научными направлениями\".";

return;

}

*// Валидация темы работы*

if (string.IsNullOrWhiteSpace(Title))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Тема работы", "empty");

return;

}

if (!ValidationHelper.IsValidNameWithNumbers(Title))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Тема работы", "invalid\_name\_with\_numbers");

return;

}

*// Валидация года защиты*

if (!ValidationHelper.IsValidYear(Year.ToString()))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Год защиты", "invalid\_year");

return;

}

*// Валидация оценки (может быть пустой, но если не пустая - должна быть от 2 до 5)*

if (Grade.HasValue && (Grade.Value < 2 || Grade.Value > 5))

{

ErrorMessage = "Оценка должна быть числом от 2 до 5.";

return;

}

var thesisWork = new ThesisWork

{

Title = Title,

StudentId = SelectedStudent.Id,

SupervisorId = SelectedSupervisor.Id,

Year = Year,

Grade = Grade

};

\_dataService.ThesisWorks.Add(thesisWork);

ClearFields();

}

private void UpdateThesisWork()

{

ErrorMessage = string.Empty;

if (SelectedThesisWork == null)

{

ErrorMessage = "Выберите дипломную работу для обновления.";

return;

}

*// Валидация студента*

if (SelectedStudent == null)

{

ErrorMessage = ValidationHelper.GetErrorMessage("Студент", "not\_selected");

return;

}

*// Валидация научного руководителя*

if (SelectedSupervisor == null)

{

ErrorMessage = ValidationHelper.GetErrorMessage("Научный руководитель", "not\_selected");

return;

}

*// Проверка: преподаватель должен руководить научными темами или направлениями*

if (!SelectedSupervisor.LeadsResearchTopics && !SelectedSupervisor.LeadsResearchDirections)

{

ErrorMessage = "Выбранный преподаватель не может быть научным руководителем. " +

"Научный руководитель должен иметь отметку \"Руководит научными темами\" " +

"или \"Руководит научными направлениями\".";

return;

}

*// Валидация темы работы*

if (string.IsNullOrWhiteSpace(Title))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Тема работы", "empty");

return;

}

if (!ValidationHelper.IsValidNameWithNumbers(Title))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Тема работы", "invalid\_name\_with\_numbers");

return;

}

*// Валидация года защиты*

if (!ValidationHelper.IsValidYear(Year.ToString()))

{

ErrorMessage = ValidationHelper.GetErrorMessage("Год защиты", "invalid\_year");

return;

}

*// Валидация оценки (может быть пустой, но если не пустая - должна быть от 2 до 5)*

if (Grade.HasValue && (Grade.Value < 2 || Grade.Value > 5))

{

ErrorMessage = "Оценка должна быть числом от 2 до 5.";

return;

}

SelectedThesisWork.Title = Title;

SelectedThesisWork.StudentId = SelectedStudent.Id;

SelectedThesisWork.SupervisorId = SelectedSupervisor.Id;

SelectedThesisWork.Year = Year;

SelectedThesisWork.Grade = Grade;

var index = ThesisWorks.IndexOf(SelectedThesisWork);

if (index >= 0)

{

ThesisWorks[index] = SelectedThesisWork;

}

}

private void DeleteThesisWork()

{

if (SelectedThesisWork == null) return;

\_dataService.ThesisWorks.Remove(SelectedThesisWork);

ThesisWorks.Remove(SelectedThesisWork);

ClearFields();

}

private void ClearFields()

{

Title = string.Empty;

SelectedStudent = null;

SelectedSupervisor = null;

Year = DateTime.Now.Year;

Grade = null;

SelectedThesisWork = null;

ErrorMessage = string.Empty;

}

}

}

**StudentProfileViewModel.cs** – ViewModel для профиля студента, отображает информацию о студенте, его группе и дисциплинах, разделяет дисциплины на пройденные, текущие и будущие. Содержит внутренние классы DisciplineWithGrade и SemesterGroup для группировки.

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

using System.Reactive;

using Avalonia;

using Avalonia.Controls;

using Avalonia.Controls.ApplicationLifetimes;

using ReactiveUI;

using UniversityIS.Models;

using UniversityIS.Services;

using UniversityIS.Views;

namespace UniversityIS.ViewModels

{

*// Обертка для дисциплины с текстом оценки*

*// Используется для отображения дисциплины вместе с оценкой студента в профиле*

public class DisciplineWithGrade

{

public DisciplineWithGrade(Discipline discipline, string gradeText)

{

Discipline = discipline;

GradeText = gradeText;

}

public Discipline Discipline { get; }

public string GradeText { get; }

}

*// Группа дисциплин, объединенных по семестру*

*// Используется для отображения пройденных предметов с группировкой по семестрам*

public class SemesterGroup

{

public SemesterGroup(int semester, IEnumerable<DisciplineWithGrade> disciplines)

{

Semester = semester;

Disciplines = new ObservableCollection<DisciplineWithGrade>(disciplines);

}

public int Semester { get; }

public string SemesterName => $"{Semester} семестр ({(Semester + 1) / 2} курс)";

public ObservableCollection<DisciplineWithGrade> Disciplines { get; }

}

*// ViewModel для профиля студента*

*// Отображает информацию о студенте, его группе и дисциплинах*

*// Разделяет дисциплины на пройденные, текущие и будущие*

public class StudentProfileViewModel : ViewModelBase

{

private readonly DataService \_dataService;

private readonly Student \_student;

private Models.Group? \_group;

private int \_currentSemester;

private ObservableCollection<SemesterGroup> \_completedDisciplines = new();

private ObservableCollection<DisciplineWithGrade> \_currentDisciplines = new();

private ObservableCollection<DisciplineWithGrade> \_futureDisciplines = new();

public StudentProfileViewModel(DataService dataService, Student student)

{

\_dataService = dataService;

\_student = student;

CloseCommand = ReactiveCommand.Create(() => { });

OpenGradeInputCommand = ReactiveCommand.Create<DisciplineWithGrade>(OpenGradeInput, outputScheduler: RxApp.MainThreadScheduler);

LoadStudentData();

*// Подписываемся на изменения оценок для обновления отображения*

\_dataService.Grades.CollectionChanged += (s, e) => LoadStudentData();

}

public Student Student => \_student;

public string FullName => $"{\_student.LastName} {\_student.FirstName} {\_student.MiddleName}".Trim();

public string GroupName => \_group?.Number ?? "Группа не найдена";

public int CurrentSemester

{

get => \_currentSemester;

set => this.RaiseAndSetIfChanged(ref \_currentSemester, value);

}

public ObservableCollection<SemesterGroup> CompletedDisciplines

{

get => \_completedDisciplines;

set => this.RaiseAndSetIfChanged(ref \_completedDisciplines, value);

}

public ObservableCollection<DisciplineWithGrade> CurrentDisciplines

{

get => \_currentDisciplines;

set => this.RaiseAndSetIfChanged(ref \_currentDisciplines, value);

}

public ObservableCollection<DisciplineWithGrade> FutureDisciplines

{

get => \_futureDisciplines;

set => this.RaiseAndSetIfChanged(ref \_futureDisciplines, value);

}

public ReactiveCommand<Unit, Unit> CloseCommand { get; }

public ReactiveCommand<DisciplineWithGrade, Unit> OpenGradeInputCommand { get; }

*// Возвращает текстовое представление оценки студента по указанной дисциплине*

*// Если оценка не выставлена, возвращает "Оценка не выставлена"*

public string GetGradeForDiscipline(Guid disciplineId)

{

var grade = \_dataService.Grades.FirstOrDefault(g =>

g.StudentId == \_student.Id && g.DisciplineId == disciplineId);

return grade != null ? $"{grade.TotalPoints} б. - {grade.Grade}" : "Оценка не выставлена";

}

*// Открывает окно для ввода/редактирования оценки по выбранной дисциплине*

*// Окно открывается модально поверх главного окна приложения*

private async void OpenGradeInput(DisciplineWithGrade disciplineWithGrade)

{

var viewModel = new GradeInputViewModel(\_dataService, \_student, disciplineWithGrade.Discipline);

var window = new GradeInputWindow

{

DataContext = viewModel

};

*// Получаем главное окно приложения для модального отображения*

if (Application.Current?.ApplicationLifetime is IClassicDesktopStyleApplicationLifetime desktop)

{

await window.ShowDialog(desktop.MainWindow);

}

}

*// Загружает и обрабатывает данные студента для отображения в профиле*

*// Определяет текущий семестр на основе времени года и курса*

*// Разделяет дисциплины на пройденные, текущие и будущие*

private void LoadStudentData()

{

*// Получаем группу студента*

\_group = \_dataService.GetGroup(\_student.GroupId);

if (\_group == null) return;

*// Вычисляем текущий семестр группы с учетом времени года*

*// Сентябрь (9) - Январь (1): нечетный семестр (1, 3, 5, 7, 9)*

*// Февраль (2) - Август (8): четный семестр (2, 4, 6, 8, 10)*

var currentMonth = DateTime.Now.Month;

bool isFirstHalfOfYear = currentMonth >= 9 || currentMonth <= 1; *// сентябрь-январь*

if (isFirstHalfOfYear)

{

*// Первая половина учебного года - нечетный семестр*

CurrentSemester = \_group.Course \* 2 - 1;

}

else

{

*// Вторая половина учебного года - четный семестр*

CurrentSemester = \_group.Course \* 2;

}

*// Получаем все дисциплины группы и сортируем по семестру и названию*

var groupDisciplines = \_dataService.Disciplines

.Where(d => d.GroupId == \_group.Id)

.OrderBy(d => d.Semester)

.ThenBy(d => d.Name)

.ToList();

*// Разделяем дисциплины на три категории: пройденные, текущие и будущие*

var completedDiscs = groupDisciplines.Where(d => d.Semester < CurrentSemester).ToList();

*// Группируем пройденные предметы по семестрам с оценками*

*// Каждый семестр отображается отдельной группой*

var completedGrouped = completedDiscs

.GroupBy(d => d.Semester)

.OrderBy(g => g.Key)

.Select(g => new SemesterGroup(g.Key,

g.OrderBy(d => d.Name)

.Select(d => new DisciplineWithGrade(d, GetGradeForDiscipline(d.Id)))))

.ToList();

CompletedDisciplines = new ObservableCollection<SemesterGroup>(completedGrouped);

*// Текущие дисциплины - только те, что в текущем семестре*

CurrentDisciplines = new ObservableCollection<DisciplineWithGrade>(

groupDisciplines.Where(d => d.Semester == CurrentSemester)

.Select(d => new DisciplineWithGrade(d, GetGradeForDiscipline(d.Id)))

);

*// Будущие дисциплины - те, что в семестрах после текущего*

FutureDisciplines = new ObservableCollection<DisciplineWithGrade>(

groupDisciplines.Where(d => d.Semester > CurrentSemester)

.Select(d => new DisciplineWithGrade(d, GetGradeForDiscipline(d.Id)))

);

}

}

}

**TeacherProfileViewModel.cs** – ViewModel для профиля преподавателя, управляет списком дисциплин, которые ведет преподаватель, позволяет добавлять и удалять дисциплины из профиля.

using System;

using System.Collections.ObjectModel;

using System.Linq;

using System.Reactive;

using ReactiveUI;

using UniversityIS.Models;

using UniversityIS.Services;

namespace UniversityIS.ViewModels

{

*// ViewModel для профиля преподавателя*

*// Управляет списком дисциплин, которые ведет преподаватель*

*// Позволяет добавлять и удалять дисциплины из профиля преподавателя*

public class TeacherProfileViewModel : ViewModelBase

{

private readonly DataService \_dataService;

private readonly Teacher \_teacher;

private Discipline? \_selectedDisciplineToAdd;

private TeacherDiscipline? \_selectedTeacherDiscipline;

private string \_errorMessage = string.Empty;

private ObservableCollection<Discipline> \_availableDisciplines = new();

private ObservableCollection<Discipline> \_teacherDisciplines = new();

public TeacherProfileViewModel(DataService dataService, Teacher teacher)

{

\_dataService = dataService;

\_teacher = teacher;

AddDisciplineCommand = ReactiveCommand.Create(AddDiscipline, outputScheduler: RxApp.MainThreadScheduler);

RemoveDisciplineCommand = ReactiveCommand.Create(RemoveDiscipline, outputScheduler: RxApp.MainThreadScheduler);

*// Подписываемся на изменения для обновления списков*

\_dataService.TeacherDisciplines.CollectionChanged += (s, e) => LoadTeacherDisciplines();

\_dataService.Disciplines.CollectionChanged += (s, e) => LoadAvailableDisciplines();

LoadTeacherDisciplines();

LoadAvailableDisciplines();

}

public string TeacherName => \_teacher.FullName;

public string Position => \_teacher.Position.ToString();

public string Department

{

get

{

var dept = \_dataService.GetDepartment(\_teacher.DepartmentId);

return dept?.Name ?? "Кафедра не найдена";

}

}

public ObservableCollection<Discipline> AvailableDisciplines

{

get => \_availableDisciplines;

set => this.RaiseAndSetIfChanged(ref \_availableDisciplines, value);

}

public ObservableCollection<Discipline> TeacherDisciplines

{

get => \_teacherDisciplines;

set => this.RaiseAndSetIfChanged(ref \_teacherDisciplines, value);

}

public Discipline? SelectedDisciplineToAdd

{

get => \_selectedDisciplineToAdd;

set => this.RaiseAndSetIfChanged(ref \_selectedDisciplineToAdd, value);

}

public TeacherDiscipline? SelectedTeacherDiscipline

{

get => \_selectedTeacherDiscipline;

set => this.RaiseAndSetIfChanged(ref \_selectedTeacherDiscipline, value);

}

public string ErrorMessage

{

get => \_errorMessage;

set => this.RaiseAndSetIfChanged(ref \_errorMessage, value);

}

public ReactiveCommand<Unit, Unit> AddDisciplineCommand { get; }

public ReactiveCommand<Unit, Unit> RemoveDisciplineCommand { get; }

private void LoadTeacherDisciplines()

{

var disciplineIds = \_dataService.TeacherDisciplines

.Where(td => td.TeacherId == \_teacher.Id)

.Select(td => td.DisciplineId)

.ToList();

var disciplines = disciplineIds

.Select(id => \_dataService.GetDiscipline(id))

.Where(d => d != null)

.OrderBy(d => d!.Name)

.ToList();

TeacherDisciplines = new ObservableCollection<Discipline>(disciplines!);

LoadAvailableDisciplines();

}

private void LoadAvailableDisciplines()

{

var teacherDisciplineIds = \_dataService.TeacherDisciplines

.Where(td => td.TeacherId == \_teacher.Id)

.Select(td => td.DisciplineId)

.ToHashSet();

var available = \_dataService.Disciplines

.Where(d => !teacherDisciplineIds.Contains(d.Id))

.OrderBy(d => d.Name)

.ToList();

AvailableDisciplines = new ObservableCollection<Discipline>(available);

}

private void AddDiscipline()

{

ErrorMessage = string.Empty;

if (SelectedDisciplineToAdd == null)

{

ErrorMessage = "Выберите дисциплину для добавления.";

return;

}

*// Проверяем, не добавлена ли уже эта дисциплина*

var exists = \_dataService.TeacherDisciplines.Any(td =>

td.TeacherId == \_teacher.Id &&

td.DisciplineId == SelectedDisciplineToAdd.Id);

if (exists)

{

ErrorMessage = "Эта дисциплина уже добавлена для данного преподавателя.";

return;

}

var teacherDiscipline = new TeacherDiscipline

{

TeacherId = \_teacher.Id,

DisciplineId = SelectedDisciplineToAdd.Id

};

\_dataService.TeacherDisciplines.Add(teacherDiscipline);

SelectedDisciplineToAdd = null;

}

private void RemoveDiscipline()

{

ErrorMessage = string.Empty;

if (SelectedTeacherDiscipline == null)

{

ErrorMessage = "Выберите дисциплину для удаления.";

return;

}

var toRemove = \_dataService.TeacherDisciplines.FirstOrDefault(td =>

td.TeacherId == \_teacher.Id &&

td.DisciplineId == SelectedTeacherDiscipline.Id);

if (toRemove != null)

{

\_dataService.TeacherDisciplines.Remove(toRemove);

SelectedTeacherDiscipline = null;

}

}

}

}

**GradeInputViewModel.cs** – ViewModel для окна ввода оценки студента, управляет вводом баллов за семестр и экзамен/зачет, автоматически рассчитывает итоговую оценку по правилам вуза.

using System;

using System.Linq;

using System.Reactive;

using ReactiveUI;

using UniversityIS.Models;

using UniversityIS.Services;

namespace UniversityIS.ViewModels

{

*// ViewModel для окна ввода оценки студента*

*// Управляет вводом баллов за семестр и экзамен/зачет*

*// Автоматически рассчитывает итоговую оценку по правилам вуза*

public class GradeInputViewModel : ViewModelBase

{

private readonly DataService \_dataService;

private readonly Student \_student;

private readonly Discipline \_discipline;

private int \_semesterPoints;

private int \_examPoints;

private int \_totalPoints;

private string \_grade = string.Empty;

private string \_errorMessage = string.Empty;

private int \_maxSemesterPoints;

private int \_maxExamPoints;

public GradeInputViewModel(DataService dataService, Student student, Discipline discipline)

{

\_dataService = dataService;

\_student = student;

\_discipline = discipline;

*// Определяем максимальные баллы в зависимости от формы контроля*

if (discipline.ControlForm == ControlForm.Exam)

{

MaxSemesterPoints = 60;

MaxExamPoints = 40;

ControlFormName = "Экзамен";

}

else *// Credit или DifferentiatedCredit*

{

MaxSemesterPoints = 80;

MaxExamPoints = 20;

ControlFormName = discipline.ControlForm == ControlForm.Pass ? "Зачет" : "Дифференцированный зачет";

}

*// Загружаем существующую оценку, если есть*

LoadExistingGrade();

SaveCommand = ReactiveCommand.Create(SaveGrade, outputScheduler: RxApp.MainThreadScheduler);

CalculateCommand = ReactiveCommand.Create(Calculate, outputScheduler: RxApp.MainThreadScheduler);

*// Подписываемся на изменение баллов для автоматического расчета*

this.WhenAnyValue(x => x.SemesterPoints, x => x.ExamPoints)

.Subscribe(\_ => Calculate());

}

public string DisciplineName => \_discipline.Name;

public string StudentName => $"{\_student.LastName} {\_student.FirstName} {\_student.MiddleName}".Trim();

public string ControlFormName { get; }

public int MaxSemesterPoints

{

get => \_maxSemesterPoints;

set => this.RaiseAndSetIfChanged(ref \_maxSemesterPoints, value);

}

public int MaxExamPoints

{

get => \_maxExamPoints;

set => this.RaiseAndSetIfChanged(ref \_maxExamPoints, value);

}

public int SemesterPoints

{

get => \_semesterPoints;

set => this.RaiseAndSetIfChanged(ref \_semesterPoints, Math.Clamp(value, 0, MaxSemesterPoints));

}

public int ExamPoints

{

get => \_examPoints;

set => this.RaiseAndSetIfChanged(ref \_examPoints, Math.Clamp(value, 0, MaxExamPoints));

}

public int TotalPoints

{

get => \_totalPoints;

set => this.RaiseAndSetIfChanged(ref \_totalPoints, value);

}

public string Grade

{

get => \_grade;

set => this.RaiseAndSetIfChanged(ref \_grade, value);

}

public string ErrorMessage

{

get => \_errorMessage;

set => this.RaiseAndSetIfChanged(ref \_errorMessage, value);

}

public ReactiveCommand<Unit, Unit> SaveCommand { get; }

public ReactiveCommand<Unit, Unit> CalculateCommand { get; }

*// Загружает существующую оценку студента по этой дисциплине, если она есть*

*// Заполняет поля ввода уже имеющимися баллами*

private void LoadExistingGrade()

{

var existingGrade = \_dataService.Grades.FirstOrDefault(g =>

g.StudentId == \_student.Id && g.DisciplineId == \_discipline.Id);

if (existingGrade != null)

{

SemesterPoints = existingGrade.SemesterPoints;

ExamPoints = existingGrade.ExamPoints;

Calculate();

}

}

*// Рассчитывает итоговую оценку на основе введенных баллов*

*// Шкала оценивания зависит от формы контроля (экзамен/зачет/дифзачет)*

private void Calculate()

{

TotalPoints = SemesterPoints + ExamPoints;

*// Для экзамена и дифференцированного зачета используется 5-балльная шкала*

if (\_discipline.ControlForm == ControlForm.Exam || \_discipline.ControlForm == ControlForm.DifferentiatedPass)

{

if (TotalPoints < 50)

Grade = "Неудовлетворительно (2) - Долг";

else if (TotalPoints <= 72)

Grade = "Удовлетворительно (3)";

else if (TotalPoints <= 86)

Grade = "Хорошо (4)";

else

Grade = "Отлично (5)";

}

else *// Для обычного зачета - зачет/незачет*

{

Grade = TotalPoints >= 50 ? "Зачет" : "Незачет";

}

}

*// Сохраняет оценку в базу данных*

*// Если оценка уже существует - обновляет ее, иначе создает новую*

private void SaveGrade()

{

ErrorMessage = string.Empty;

var existingGrade = \_dataService.Grades.FirstOrDefault(g =>

g.StudentId == \_student.Id && g.DisciplineId == \_discipline.Id);

if (existingGrade != null)

{

*// Обновляем существующую оценку*

existingGrade.SemesterPoints = SemesterPoints;

existingGrade.ExamPoints = ExamPoints;

existingGrade.CalculateGrade(\_discipline.ControlForm);

}

else

{

*// Создаем новую оценку*

var newGrade = new StudentGrade

{

StudentId = \_student.Id,

DisciplineId = \_discipline.Id,

SemesterPoints = SemesterPoints,

ExamPoints = ExamPoints

};

newGrade.CalculateGrade(\_discipline.ControlForm);

\_dataService.Grades.Add(newGrade);

}

\_dataService.SaveAllData();

}

}

}

Файлы представлений (Views)

**MainWindow.axaml** – файл разметки главного окна приложения в формате AXAML, определяет структуру главного окна с меню навигации и областью отображения разделов.

<Window xmlns="https://github.com/avaloniaui"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:vm="using:UniversityIS.ViewModels"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:views="using:UniversityIS.Views"

mc:Ignorable="d" d:DesignWidth="1200" d:DesignHeight="700"

x:Class="UniversityIS.Views.MainWindow"

x:DataType="vm:MainWindowViewModel"

Icon="/Assets/avalonia-logo.ico"

Title="Информационная система ВУЗа"

Width="1200" Height="700">

<Design.DataContext>

<vm:MainWindowViewModel/>

</Design.DataContext>

<Grid ColumnDefinitions="220,\*">

*<!-- Боковое меню навигации -->*

<Border Grid.Column="0" Background="#1a1a2e" Padding="15">

<StackPanel Spacing="8">

<TextBlock Text="НАВИГАЦИЯ"

Foreground="#eee"

FontSize="16"

FontWeight="Bold"

Margin="0,0,0,15"

HorizontalAlignment="Center"/>

<Button Content="Факультеты"

Command="{Binding ShowFacultiesCommand}"

HorizontalAlignment="Stretch"

HorizontalContentAlignment="Left"

Background="#3498db"

Foreground="White"

FontSize="14"

FontWeight="SemiBold"

Padding="10,8"/>

<Button Content="Кафедры"

Command="{Binding ShowDepartmentsCommand}"

HorizontalAlignment="Stretch"

HorizontalContentAlignment="Left"

Background="#3498db"

Foreground="White"

FontSize="14"

FontWeight="SemiBold"

Padding="10,8"/>

<Button Content="Группы"

Command="{Binding ShowGroupsCommand}"

HorizontalAlignment="Stretch"

HorizontalContentAlignment="Left"

Background="#3498db"

Foreground="White"

FontSize="14"

FontWeight="SemiBold"

Padding="10,8"/>

<Button Content="Студенты"

Command="{Binding ShowStudentsCommand}"

HorizontalAlignment="Stretch"

HorizontalContentAlignment="Left"

Background="#3498db"

Foreground="White"

FontSize="14"

FontWeight="SemiBold"

Padding="10,8"/>

<Button Content="Преподаватели"

Command="{Binding ShowTeachersCommand}"

HorizontalAlignment="Stretch"

HorizontalContentAlignment="Left"

Background="#3498db"

Foreground="White"

FontSize="14"

FontWeight="SemiBold"

Padding="10,8"/>

<Button Content="Дисциплины"

Command="{Binding ShowDisciplinesCommand}"

HorizontalAlignment="Stretch"

HorizontalContentAlignment="Left"

Background="#3498db"

Foreground="White"

FontSize="14"

FontWeight="SemiBold"

Padding="10,8"/>

<Button Content="Нагрузка"

Command="{Binding ShowWorkLoadsCommand}"

HorizontalAlignment="Stretch"

HorizontalContentAlignment="Left"

Background="#3498db"

Foreground="White"

FontSize="14"

FontWeight="SemiBold"

Padding="10,8"/>

<Button Content="Дипломные работы"

Command="{Binding ShowThesisWorksCommand}"

HorizontalAlignment="Stretch"

HorizontalContentAlignment="Left"

Background="#3498db"

Foreground="White"

FontSize="14"

FontWeight="SemiBold"

Padding="10,8"/>

<Separator Margin="0,15,0,15" Background="#444" Height="2"/>

<Button Content="СОХРАНИТЬ ДАННЫЕ"

Command="{Binding SaveDataCommand}"

HorizontalAlignment="Stretch"

HorizontalContentAlignment="Center"

Background="#27ae60"

Foreground="White"

FontSize="14"

FontWeight="Bold"

Padding="10,12"/>

</StackPanel>

</Border>

*<!-- Основное содержимое -->*

<Border Grid.Column="1" Background="#f5f5f5" Padding="20">

<ContentControl Content="{Binding CurrentPage}">

<ContentControl.DataTemplates>

<DataTemplate DataType="vm:FacultiesViewModel">

<views:FacultiesView />

</DataTemplate>

<DataTemplate DataType="vm:DepartmentsViewModel">

<views:DepartmentsView />

</DataTemplate>

<DataTemplate DataType="vm:GroupsViewModel">

<views:GroupsView />

</DataTemplate>

<DataTemplate DataType="vm:StudentsViewModel">

<views:StudentsView />

</DataTemplate>

<DataTemplate DataType="vm:TeachersViewModel">

<views:TeachersView />

</DataTemplate>

<DataTemplate DataType="vm:DisciplinesViewModel">

<views:DisciplinesView />

</DataTemplate>

<DataTemplate DataType="vm:WorkLoadsViewModel">

<views:WorkLoadsView />

</DataTemplate>

<DataTemplate DataType="vm:ThesisWorksViewModel">

<views:ThesisWorksView />

</DataTemplate>

</ContentControl.DataTemplates>

</ContentControl>

</Border>

</Grid>

</Window>

**MainWindow.axaml.cs** – файл кода главного окна, содержит логику инициализации и обработку события закрытия окна для автосохранения данных.

using Avalonia.Controls;

using UniversityIS.ViewModels;

namespace UniversityIS.Views;

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

*// Автосохранение при закрытии окна*

Closing += (sender, e) =>

{

if (DataContext is MainWindowViewModel viewModel)

{

viewModel.OnClosing();

}

};

}

}

**FacultiesView.axaml** – файл разметки представления для раздела факультетов в формате AXAML, определяет интерфейс управления факультетами.

<UserControl xmlns="https://github.com/avaloniaui"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:vm="using:UniversityIS.ViewModels"

mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="600"

x:Class="UniversityIS.Views.FacultiesView"

x:DataType="vm:FacultiesViewModel">

<Grid RowDefinitions="Auto,\*">

<Border Grid.Row="0" Background="#3498db" Padding="15" CornerRadius="5" Margin="0,0,0,15">

<TextBlock Text="УПРАВЛЕНИЕ ФАКУЛЬТЕТАМИ"

FontSize="22"

FontWeight="Bold"

Foreground="White"

HorizontalAlignment="Center"/>

</Border>

<Grid Grid.Row="1" ColumnDefinitions="\*,350">

*<!-- Список факультетов -->*

<Border Grid.Column="0"

Background="#F0F0F0"

CornerRadius="8"

Padding="15"

Margin="0,0,15,0"

BorderBrush="#3498db"

BorderThickness="2">

<Grid RowDefinitions="Auto,\*">

<StackPanel Grid.Row="0" Margin="0,0,0,15">

<TextBlock FontSize="16"

FontWeight="Bold"

Foreground="#2c3e50">

<Run Text="Список факультетов (всего: "/>

<Run Text="{Binding Faculties.Count}"/>

<Run Text=")"/>

</TextBlock>

</StackPanel>

<ListBox Grid.Row="1"

ItemsSource="{Binding Faculties}"

SelectedItem="{Binding SelectedFaculty}"

Background="Transparent"

BorderThickness="0">

<ListBox.ItemTemplate>

<DataTemplate>

<Border Background="#F0F0F0"

Margin="0,0,0,8"

Padding="15"

CornerRadius="5"

BorderBrush="#3498db"

BorderThickness="1">

<StackPanel>

<TextBlock FontSize="16" FontWeight="Bold" Foreground="#2c3e50">

<Run Text=""/>

<Run Text="{Binding Name}"/>

</TextBlock>

<TextBlock FontSize="14" Foreground="#2c3e50" Margin="0,5,0,0">

<Run Text="Декан: "/>

<Run Text="{Binding Dean}"/>

</TextBlock>

<TextBlock FontSize="12" Foreground="#7f8c8d" Margin="0,5,0,0">

<Run Text=""/>

<Run Text="{Binding Id}"/>

</TextBlock>

</StackPanel>

</Border>

</DataTemplate>

</ListBox.ItemTemplate>

<ListBox.Styles>

<Style Selector="ListBoxItem">

<Setter Property="Padding" Value="0"/>

<Setter Property="Margin" Value="0"/>

</Style>

<Style Selector="ListBoxItem:selected /template/ ContentPresenter">

<Setter Property="Background" Value="Transparent"/>

</Style>

<Style Selector="ListBoxItem:selected Border">

<Setter Property="Background" Value="#e3f2fd"/>

<Setter Property="BorderBrush" Value="#2196f3"/>

<Setter Property="BorderThickness" Value="3"/>

</Style>

</ListBox.Styles>

</ListBox>

</Grid>

</Border>

*<!-- Форма редактирования -->*

<Border Grid.Column="1"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="3"

CornerRadius="8"

Padding="20">

<StackPanel Spacing="15">

<TextBlock Text="ДОБАВИТЬ ФАКУЛЬТЕТ"

FontSize="18"

FontWeight="Bold"

Foreground="#2c3e50"

HorizontalAlignment="Center"

Margin="0,0,0,10"/>

<StackPanel Spacing="8">

<TextBlock Text="Название факультета:"

FontWeight="SemiBold"

FontSize="14"

Foreground="#2c3e50"/>

<TextBox Text="{Binding Name}"

Watermark="Например: Факультет информационных технологий"

FontSize="14"

Padding="10"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="8">

<TextBlock Text="ФИО декана:"

FontWeight="SemiBold"

FontSize="14"

Foreground="#2c3e50"/>

<TextBox Text="{Binding Dean}"

Watermark="Например: Иванов Иван Иванович"

FontSize="14"

Padding="10"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="2"/>

</StackPanel>

*<!-- Сообщение об ошибке -->*

<Border Background="#ffebee"

BorderBrush="#e74c3c"

BorderThickness="2"

CornerRadius="5"

Padding="10"

IsVisible="{Binding ErrorMessage, Converter={x:Static StringConverters.IsNotNullOrEmpty}}">

<TextBlock Text="{Binding ErrorMessage}"

Foreground="#c62828"

FontWeight="SemiBold"

FontSize="13"

TextWrapping="Wrap"/>

</Border>

<StackPanel Spacing="10" Margin="0,20,0,0">

<Button Content="ДОБАВИТЬ ФАКУЛЬТЕТ"

Command="{Binding AddCommand}"

HorizontalAlignment="Stretch"

Background="#27ae60"

Foreground="White"

FontSize="14"

FontWeight="Bold"

Padding="12"

BorderThickness="0"/>

<Button Content="Обновить"

Command="{Binding UpdateCommand}"

HorizontalAlignment="Stretch"

Background="#f39c12"

Foreground="White"

FontSize="13"

FontWeight="SemiBold"

Padding="10"

BorderThickness="0"/>

<Button Content="Удалить"

Command="{Binding DeleteCommand}"

HorizontalAlignment="Stretch"

Background="#e74c3c"

Foreground="White"

FontSize="13"

FontWeight="SemiBold"

Padding="10"

BorderThickness="0"/>

</StackPanel>

</StackPanel>

</Border>

</Grid>

</Grid>

</UserControl>

**FacultiesView.axaml.cs** – файл кода представления для раздела факультетов, содержит конструктор для инициализации компонентов.

using Avalonia.Controls;

namespace UniversityIS.Views

{

public partial class FacultiesView : UserControl

{

public FacultiesView()

{

InitializeComponent();

}

}

}

**DepartmentsView.axaml** – файл разметки представления для раздела кафедр в формате AXAML, определяет интерфейс управления кафедрами с группировкой по факультетам.

<UserControl xmlns="https://github.com/avaloniaui"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:vm="using:UniversityIS.ViewModels"

mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="600"

x:Class="UniversityIS.Views.DepartmentsView"

x:DataType="vm:DepartmentsViewModel">

<Grid RowDefinitions="Auto,\*">

<Border Grid.Row="0" Background="#3498db" Padding="15" CornerRadius="5" Margin="0,0,0,15">

<TextBlock Text="УПРАВЛЕНИЕ КАФЕДРАМИ"

FontSize="22"

FontWeight="Bold"

Foreground="White"

HorizontalAlignment="Center"/>

</Border>

<Grid Grid.Row="1" ColumnDefinitions="\*,350">

*<!-- Список кафедр -->*

<Border Grid.Column="0"

Background="#F0F0F0"

CornerRadius="8"

Padding="15"

Margin="0,0,15,0"

BorderBrush="#3498db"

BorderThickness="2">

<Grid RowDefinitions="Auto,\*">

<StackPanel Grid.Row="0" Margin="0,0,0,15">

<TextBlock FontSize="16"

FontWeight="Bold"

Foreground="#2c3e50">

<Run Text="Список кафедр (всего: "/>

<Run Text="{Binding Departments.Count}"/>

<Run Text=")"/>

</TextBlock>

</StackPanel>

<ScrollViewer Grid.Row="1" VerticalScrollBarVisibility="Auto">

<ItemsControl ItemsSource="{Binding GroupedDepartments}">

<ItemsControl.ItemTemplate>

<DataTemplate>

<StackPanel Margin="0,0,0,20">

*<!-- Заголовок факультета -->*

<Border Background="#3498db"

Padding="12,8"

CornerRadius="5"

Margin="0,0,0,10">

<TextBlock Text="{Binding FacultyName}"

FontSize="15"

FontWeight="Bold"

Foreground="White">

<TextBlock.Inlines>

<Run Text=""/>

<Run Text="{Binding FacultyName}"/>

</TextBlock.Inlines>

</TextBlock>

</Border>

*<!-- Кафедры этого факультета -->*

<ListBox ItemsSource="{Binding Departments}"

SelectedItem="{Binding $parent[UserControl].((vm:DepartmentsViewModel)DataContext).SelectedDepartment}"

Background="Transparent"

BorderThickness="0"

Margin="15,0,0,0">

<ListBox.ItemTemplate>

<DataTemplate>

<Border Background="#F0F0F0"

Margin="0,0,0,8"

Padding="15"

CornerRadius="5"

BorderBrush="#3498db"

BorderThickness="1">

<StackPanel>

<TextBlock FontSize="16" FontWeight="Bold" Foreground="#2c3e50">

<Run Text=""/>

<Run Text="{Binding Name}"/>

</TextBlock>

<TextBlock FontSize="14" Foreground="#2c3e50" Margin="0,5,0,0">

<Run Text="Зав. кафедрой: "/>

<Run Text="{Binding Head}"/>

</TextBlock>

</StackPanel>

</Border>

</DataTemplate>

</ListBox.ItemTemplate>

<ListBox.Styles>

<Style Selector="ListBoxItem">

<Setter Property="Padding" Value="0"/>

<Setter Property="Margin" Value="0"/>

</Style>

<Style Selector="ListBoxItem:selected /template/ ContentPresenter">

<Setter Property="Background" Value="Transparent"/>

</Style>

<Style Selector="ListBoxItem:selected Border">

<Setter Property="Background" Value="#e3f2fd"/>

<Setter Property="BorderBrush" Value="#2196f3"/>

<Setter Property="BorderThickness" Value="3"/>

</Style>

</ListBox.Styles>

</ListBox>

</StackPanel>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

</ScrollViewer>

</Grid>

</Border>

*<!-- Форма редактирования -->*

<Border Grid.Column="1"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="3"

CornerRadius="8"

Padding="20">

<StackPanel Spacing="15">

<TextBlock Text="ДОБАВИТЬ КАФЕДРУ"

FontSize="18"

FontWeight="Bold"

Foreground="#2c3e50"

HorizontalAlignment="Center"

Margin="0,0,0,10"/>

<StackPanel Spacing="8">

<TextBlock Text="Название кафедры:"

FontWeight="SemiBold"

FontSize="14"

Foreground="#2c3e50"/>

<TextBox Text="{Binding Name}"

Watermark="Например: Кафедра информатики"

FontSize="14"

Padding="10"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="8">

<TextBlock Text="ФИО заведующего:"

FontWeight="SemiBold"

FontSize="14"

Foreground="#2c3e50"/>

<TextBox Text="{Binding Head}"

Watermark="Например: Петров П.П."

FontSize="14"

Padding="10"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="8">

<TextBlock Text="Факультет:"

FontWeight="SemiBold"

FontSize="14"

Foreground="#2c3e50"/>

<ComboBox ItemsSource="{Binding Faculties}"

SelectedItem="{Binding SelectedFaculty}"

HorizontalAlignment="Stretch"

FontSize="14"

Padding="10"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="2"/>

</StackPanel>

*<!-- Сообщение об ошибке -->*

<Border Background="#ffebee"

BorderBrush="#e74c3c"

BorderThickness="2"

CornerRadius="5"

Padding="10"

IsVisible="{Binding ErrorMessage, Converter={x:Static StringConverters.IsNotNullOrEmpty}}">

<TextBlock Text="{Binding ErrorMessage}"

Foreground="#c62828"

FontWeight="SemiBold"

FontSize="13"

TextWrapping="Wrap"/>

</Border>

<StackPanel Spacing="10" Margin="0,20,0,0">

<Button Content="ДОБАВИТЬ КАФЕДРУ"

Command="{Binding AddCommand}"

HorizontalAlignment="Stretch"

Background="#27ae60"

Foreground="White"

FontSize="14"

FontWeight="Bold"

Padding="12"

BorderThickness="0"/>

<Button Content="Обновить"

Command="{Binding UpdateCommand}"

HorizontalAlignment="Stretch"

Background="#f39c12"

Foreground="White"

FontSize="13"

FontWeight="SemiBold"

Padding="10"

BorderThickness="0"/>

<Button Content="Удалить"

Command="{Binding DeleteCommand}"

HorizontalAlignment="Stretch"

Background="#e74c3c"

Foreground="White"

FontSize="13"

FontWeight="SemiBold"

Padding="10"

BorderThickness="0"/>

</StackPanel>

</StackPanel>

</Border>

</Grid>

</Grid>

</UserControl>

**DepartmentsView.axaml.cs** – файл кода представления для раздела кафедр, содержит конструктор для инициализации компонентов.

using Avalonia.Controls;

namespace UniversityIS.Views

{

public partial class DepartmentsView : UserControl

{

public DepartmentsView()

{

InitializeComponent();

}

}

}

**GroupsView.axaml** – файл разметки представления для раздела групп в формате AXAML, определяет интерфейс управления учебными группами с группировкой по факультетам и курсам.

<UserControl xmlns="https://github.com/avaloniaui"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:vm="using:UniversityIS.ViewModels"

mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="600"

x:Class="UniversityIS.Views.GroupsView"

x:DataType="vm:GroupsViewModel">

<Grid RowDefinitions="Auto,\*">

<Border Grid.Row="0" Background="#3498db" Padding="15" CornerRadius="5" Margin="0,0,0,15">

<TextBlock Text="УПРАВЛЕНИЕ ГРУППАМИ"

FontSize="22"

FontWeight="Bold"

Foreground="White"

HorizontalAlignment="Center"/>

</Border>

<Grid Grid.Row="1" ColumnDefinitions="\*,350">

*<!-- Список групп -->*

<Border Grid.Column="0"

Background="#F0F0F0"

CornerRadius="8"

Padding="15"

Margin="0,0,15,0"

BorderBrush="#3498db"

BorderThickness="2">

<Grid RowDefinitions="Auto,\*">

<StackPanel Grid.Row="0" Margin="0,0,0,15">

<TextBlock FontSize="16"

FontWeight="Bold"

Foreground="#2c3e50">

<Run Text="Список групп (всего: "/>

<Run Text="{Binding Groups.Count}"/>

<Run Text=")"/>

</TextBlock>

</StackPanel>

<ScrollViewer Grid.Row="1" VerticalScrollBarVisibility="Auto">

<ItemsControl ItemsSource="{Binding GroupedGroups}">

<ItemsControl.ItemTemplate>

<DataTemplate>

<StackPanel Margin="0,0,0,15">

*<!-- Заголовок факультета -->*

<Border Background="#3498db"

CornerRadius="5"

Padding="12"

Margin="0,0,0,8">

<TextBlock Text="{Binding FacultyName}"

FontSize="16"

FontWeight="Bold"

Foreground="White"/>

</Border>

*<!-- Курсы внутри факультета -->*

<ItemsControl ItemsSource="{Binding Courses}" Margin="10,0,0,0">

<ItemsControl.ItemTemplate>

<DataTemplate>

<StackPanel Margin="0,0,0,10">

*<!-- Заголовок курса -->*

<Border Background="#5dade2"

CornerRadius="4"

Padding="8"

Margin="0,0,0,5">

<TextBlock Text="{Binding CourseName}"

FontSize="14"

FontWeight="SemiBold"

Foreground="White"/>

</Border>

*<!-- Группы на курсе -->*

<ListBox ItemsSource="{Binding Groups}"

SelectedItem="{Binding $parent[UserControl].((vm:GroupsViewModel)DataContext).SelectedGroup}"

Margin="15,0,0,0"

Background="Transparent"

BorderThickness="0">

<ListBox.ItemTemplate>

<DataTemplate>

<Border Background="#F0F0F0"

Margin="0,0,0,5"

Padding="12"

CornerRadius="5"

BorderBrush="#85c1e9"

BorderThickness="2">

<StackPanel>

<TextBlock FontSize="14" FontWeight="Bold" Foreground="#2c3e50">

<Run Text="Группа "/>

<Run Text="{Binding Number}"/>

</TextBlock>

<TextBlock FontSize="12" Foreground="#34495e" Margin="0,3,0,0">

<Run Text="Год набора: "/>

<Run Text="{Binding YearOfAdmission}"/>

</TextBlock>

</StackPanel>

</Border>

</DataTemplate>

</ListBox.ItemTemplate>

<ListBox.Styles>

<Style Selector="ListBoxItem">

<Setter Property="Padding" Value="0"/>

<Setter Property="Margin" Value="0"/>

</Style>

<Style Selector="ListBoxItem:selected /template/ ContentPresenter">

<Setter Property="Background" Value="Transparent"/>

</Style>

<Style Selector="ListBoxItem:selected Border">

<Setter Property="Background" Value="#e3f2fd"/>

<Setter Property="BorderBrush" Value="#2196f3"/>

<Setter Property="BorderThickness" Value="3"/>

</Style>

</ListBox.Styles>

</ListBox>

</StackPanel>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

</StackPanel>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

</ScrollViewer>

</Grid>

</Border>

*<!-- Форма редактирования -->*

<Border Grid.Column="1"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="3"

CornerRadius="8"

Padding="20">

<StackPanel Spacing="15">

<TextBlock Text="ДОБАВИТЬ ГРУППУ"

FontSize="18"

FontWeight="Bold"

Foreground="#2c3e50"

HorizontalAlignment="Center"

Margin="0,0,0,10"/>

<StackPanel Spacing="8">

<TextBlock Text="Номер группы:"

FontWeight="SemiBold"

FontSize="14"

Foreground="#2c3e50"/>

<TextBox Text="{Binding Number}"

Watermark="Например: ПИ-21"

FontSize="14"

Padding="10"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="8">

<TextBlock Text="Год набора:"

FontWeight="SemiBold"

FontSize="14"

Foreground="#2c3e50"/>

<NumericUpDown Value="{Binding YearOfAdmission}"

Minimum="2000"

Maximum="2100"

FontSize="14"

Padding="10"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="8">

<TextBlock Text="Курс:"

FontWeight="SemiBold"

FontSize="14"

Foreground="#2c3e50"/>

<NumericUpDown Value="{Binding Course}"

Minimum="1"

Maximum="6"

FontSize="14"

Padding="10"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="8">

<TextBlock Text="Факультет:"

FontWeight="SemiBold"

FontSize="14"

Foreground="#2c3e50"/>

<ComboBox ItemsSource="{Binding Faculties}"

SelectedItem="{Binding SelectedFaculty}"

HorizontalAlignment="Stretch"

FontSize="14"

Padding="10"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="2"/>

</StackPanel>

*<!-- Сообщение об ошибке -->*

<Border Background="#ffebee"

BorderBrush="#e74c3c"

BorderThickness="2"

CornerRadius="5"

Padding="10"

IsVisible="{Binding ErrorMessage, Converter={x:Static StringConverters.IsNotNullOrEmpty}}">

<TextBlock Text="{Binding ErrorMessage}"

Foreground="#c62828"

FontWeight="SemiBold"

FontSize="13"

TextWrapping="Wrap"/>

</Border>

<StackPanel Spacing="10" Margin="0,15,0,0">

<Button Content="ДОБАВИТЬ ГРУППУ"

Command="{Binding AddCommand}"

HorizontalAlignment="Stretch"

Background="#27ae60"

Foreground="White"

FontSize="14"

FontWeight="Bold"

Padding="12"

BorderThickness="0"/>

<Button Content="Обновить"

Command="{Binding UpdateCommand}"

HorizontalAlignment="Stretch"

Background="#f39c12"

Foreground="White"

FontSize="13"

FontWeight="SemiBold"

Padding="10"

BorderThickness="0"/>

<Button Content="Удалить"

Command="{Binding DeleteCommand}"

HorizontalAlignment="Stretch"

Background="#e74c3c"

Foreground="White"

FontSize="13"

FontWeight="SemiBold"

Padding="10"

BorderThickness="0"/>

</StackPanel>

</StackPanel>

</Border>

</Grid>

</Grid>

</UserControl>

**GroupsView.axaml.cs** – файл кода представления для раздела групп, содержит конструктор для инициализации компонентов.

using Avalonia.Controls;

namespace UniversityIS.Views

{

public partial class GroupsView : UserControl

{

public GroupsView()

{

InitializeComponent();

}

}

}

**StudentsView.axaml** – файл разметки представления для раздела студентов в формате AXAML, определяет интерфейс управления студентами с группировкой по факультетам и группам.

<UserControl xmlns="https://github.com/avaloniaui"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:vm="using:UniversityIS.ViewModels"

mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="600"

x:Class="UniversityIS.Views.StudentsView"

x:DataType="vm:StudentsViewModel">

<Grid RowDefinitions="Auto,\*">

<Border Grid.Row="0" Background="#3498db" Padding="15" CornerRadius="5" Margin="0,0,0,15">

<TextBlock Text="УПРАВЛЕНИЕ СТУДЕНТАМИ"

FontSize="22"

FontWeight="Bold"

Foreground="White"

HorizontalAlignment="Center"/>

</Border>

<Grid Grid.Row="1" ColumnDefinitions="\*,380">

*<!-- Список студентов -->*

<Border Grid.Column="0"

Background="#F0F0F0"

CornerRadius="8"

Padding="15"

Margin="0,0,15,0"

BorderBrush="#3498db"

BorderThickness="2">

<Grid RowDefinitions="Auto,\*">

<StackPanel Grid.Row="0" Margin="0,0,0,15">

<TextBlock FontSize="16"

FontWeight="Bold"

Foreground="#2c3e50">

<Run Text="Список студентов (всего: "/>

<Run Text="{Binding Students.Count}"/>

<Run Text=")"/>

</TextBlock>

</StackPanel>

<ScrollViewer Grid.Row="1" VerticalScrollBarVisibility="Auto">

<ItemsControl ItemsSource="{Binding GroupedStudents}">

<ItemsControl.ItemTemplate>

<DataTemplate>

<StackPanel Margin="0,0,0,15">

*<!-- Заголовок факультета -->*

<Border Background="#3498db"

CornerRadius="5"

Padding="12"

Margin="0,0,0,8">

<TextBlock Text="{Binding FacultyName}"

FontSize="16"

FontWeight="Bold"

Foreground="White"/>

</Border>

*<!-- Группы внутри факультета -->*

<ItemsControl ItemsSource="{Binding Groups}" Margin="10,0,0,0">

<ItemsControl.ItemTemplate>

<DataTemplate>

<StackPanel Margin="0,0,0,10">

*<!-- Заголовок группы -->*

<Border Background="#5dade2"

CornerRadius="4"

Padding="8"

Margin="0,0,0,5">

<TextBlock Text="{Binding GroupName}"

FontSize="14"

FontWeight="SemiBold"

Foreground="White"/>

</Border>

*<!-- Студенты в группе -->*

<ListBox ItemsSource="{Binding Students}"

SelectedItem="{Binding $parent[UserControl].((vm:StudentsViewModel)DataContext).SelectedStudent}"

Margin="15,0,0,0"

Background="Transparent"

BorderThickness="0">

<ListBox.ItemTemplate>

<DataTemplate>

<Border Background="#F0F0F0"

Margin="0,0,0,5"

Padding="12"

CornerRadius="5"

BorderBrush="#85c1e9"

BorderThickness="2">

<StackPanel>

<TextBlock FontSize="14" FontWeight="Bold" Foreground="#2c3e50">

<Run Text="{Binding LastName}"/>

<Run Text=" "/>

<Run Text="{Binding FirstName}"/>

<Run Text=" "/>

<Run Text="{Binding MiddleName}"/>

</TextBlock>

</StackPanel>

</Border>

</DataTemplate>

</ListBox.ItemTemplate>

<ListBox.Styles>

<Style Selector="ListBoxItem">

<Setter Property="Padding" Value="0"/>

<Setter Property="Margin" Value="0"/>

</Style>

<Style Selector="ListBoxItem:selected /template/ ContentPresenter">

<Setter Property="Background" Value="Transparent"/>

</Style>

<Style Selector="ListBoxItem:selected Border">

<Setter Property="Background" Value="#e3f2fd"/>

<Setter Property="BorderBrush" Value="#2196f3"/>

<Setter Property="BorderThickness" Value="3"/>

</Style>

</ListBox.Styles>

</ListBox>

</StackPanel>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

</StackPanel>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

</ScrollViewer>

</Grid>

</Border>

*<!-- Форма редактирования -->*

<Border Grid.Column="1"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="3"

CornerRadius="8"

Padding="20">

<ScrollViewer>

<StackPanel Spacing="12">

<TextBlock Text="ДОБАВИТЬ СТУДЕНТА"

FontSize="18"

FontWeight="Bold"

Foreground="#2c3e50"

HorizontalAlignment="Center"

Margin="0,0,0,10"/>

<StackPanel Spacing="6">

<TextBlock Text="Фамилия:"

FontWeight="SemiBold"

FontSize="13"

Foreground="#2c3e50"/>

<TextBox Text="{Binding LastName}"

Watermark="Иванов"

FontSize="13"

Padding="8"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="6">

<TextBlock Text="Имя:"

FontWeight="SemiBold"

FontSize="13"

Foreground="#2c3e50"/>

<TextBox Text="{Binding FirstName}"

Watermark="Иван"

FontSize="13"

Padding="8"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="6">

<TextBlock Text="Отчество:"

FontWeight="SemiBold"

FontSize="13"

Foreground="#2c3e50"/>

<TextBox Text="{Binding MiddleName}"

Watermark="Иванович"

FontSize="13"

Padding="8"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="6">

<TextBlock Text="Группа:"

FontWeight="SemiBold"

FontSize="13"

Foreground="#2c3e50"/>

<ComboBox ItemsSource="{Binding Groups}"

SelectedItem="{Binding SelectedGroup}"

HorizontalAlignment="Stretch"

FontSize="13"

Padding="8"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="2"/>

</StackPanel>

*<!-- Сообщение об ошибке -->*

<Border Background="#ffebee"

BorderBrush="#e74c3c"

BorderThickness="2"

CornerRadius="5"

Padding="10"

IsVisible="{Binding ErrorMessage, Converter={x:Static StringConverters.IsNotNullOrEmpty}}">

<TextBlock Text="{Binding ErrorMessage}"

Foreground="#c62828"

FontWeight="SemiBold"

FontSize="12"

TextWrapping="Wrap"/>

</Border>

<StackPanel Spacing="8" Margin="0,15,0,0">

<Button Content="ДОБАВИТЬ СТУДЕНТА"

Command="{Binding AddCommand}"

HorizontalAlignment="Stretch"

Background="#27ae60"

Foreground="White"

FontSize="13"

FontWeight="Bold"

Padding="10"

BorderThickness="0"/>

<Button Content="Обновить"

Command="{Binding UpdateCommand}"

HorizontalAlignment="Stretch"

Background="#f39c12"

Foreground="White"

FontSize="12"

FontWeight="SemiBold"

Padding="8"

BorderThickness="0"/>

<Button Content="Удалить"

Command="{Binding DeleteCommand}"

HorizontalAlignment="Stretch"

Background="#e74c3c"

Foreground="White"

FontSize="12"

FontWeight="SemiBold"

Padding="8"

BorderThickness="0"/>

<Button Content="ПРОФИЛЬ СТУДЕНТА"

Command="{Binding ShowProfileCommand}"

HorizontalAlignment="Stretch"

Background="#9b59b6"

Foreground="White"

FontSize="13"

FontWeight="Bold"

Padding="10"

BorderThickness="0"

Margin="0,10,0,0"/>

</StackPanel>

</StackPanel>

</ScrollViewer>

</Border>

</Grid>

</Grid>

</UserControl>

**StudentsView.axaml.cs** – файл кода представления для раздела студентов, содержит конструктор для инициализации компонентов.

using Avalonia.Controls;

namespace UniversityIS.Views

{

public partial class StudentsView : UserControl

{

public StudentsView()

{

InitializeComponent();

}

}

}

**TeachersView.axaml** – файл разметки представления для раздела преподавателей в формате AXAML, определяет интерфейс управления преподавателями.

<UserControl xmlns="https://github.com/avaloniaui"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:vm="using:UniversityIS.ViewModels"

mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="600"

x:Class="UniversityIS.Views.TeachersView"

x:DataType="vm:TeachersViewModel">

<Grid RowDefinitions="Auto,\*">

<Border Grid.Row="0" Background="#3498db" Padding="15" CornerRadius="5" Margin="0,0,0,15">

<TextBlock Text="УПРАВЛЕНИЕ ПРЕПОДАВАТЕЛЯМИ"

FontSize="22"

FontWeight="Bold"

Foreground="White"

HorizontalAlignment="Center"/>

</Border>

<Grid Grid.Row="1" ColumnDefinitions="\*,400">

*<!-- Список преподавателей -->*

<Border Grid.Column="0"

Background="#F0F0F0"

CornerRadius="8"

Padding="15"

Margin="0,0,15,0"

BorderBrush="#3498db"

BorderThickness="2">

<Grid RowDefinitions="Auto,\*">

<StackPanel Grid.Row="0" Margin="0,0,0,15">

<TextBlock FontSize="16"

FontWeight="Bold"

Foreground="#2c3e50">

<Run Text="Список преподавателей (всего: "/>

<Run Text="{Binding Teachers.Count}"/>

<Run Text=")"/>

</TextBlock>

</StackPanel>

<ListBox Grid.Row="1"

ItemsSource="{Binding Teachers}"

SelectedItem="{Binding SelectedTeacher}"

Background="Transparent"

BorderThickness="0">

<ListBox.ItemTemplate>

<DataTemplate>

<Border Background="#F0F0F0"

Margin="0,0,0,8"

Padding="15"

CornerRadius="5"

BorderBrush="#3498db"

BorderThickness="1">

<StackPanel>

<TextBlock FontSize="16" FontWeight="Bold" Foreground="#2c3e50">

<Run Text=""/>

<Run Text="{Binding LastName}"/>

<Run Text=" "/>

<Run Text="{Binding FirstName}"/>

<Run Text=" "/>

<Run Text="{Binding MiddleName}"/>

</TextBlock>

<TextBlock FontSize="14" Foreground="#2c3e50" Margin="0,5,0,0">

<Run Text="📌 "/>

<Run Text="{Binding Position}"/>

</TextBlock>

<TextBlock FontSize="13" Foreground="#34495e" Margin="0,3,0,0">

<Run Text=""/>

<Run Text="{Binding Degree}"/>

<Run Text=" | "/>

<Run Text="{Binding Title}"/>

</TextBlock>

</StackPanel>

</Border>

</DataTemplate>

</ListBox.ItemTemplate>

<ListBox.Styles>

<Style Selector="ListBoxItem">

<Setter Property="Padding" Value="0"/>

<Setter Property="Margin" Value="0"/>

</Style>

<Style Selector="ListBoxItem:selected /template/ ContentPresenter">

<Setter Property="Background" Value="Transparent"/>

</Style>

<Style Selector="ListBoxItem:selected Border">

<Setter Property="Background" Value="#e3f2fd"/>

<Setter Property="BorderBrush" Value="#2196f3"/>

<Setter Property="BorderThickness" Value="3"/>

</Style>

</ListBox.Styles>

</ListBox>

</Grid>

</Border>

*<!-- Форма редактирования -->*

<Border Grid.Column="1"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="3"

CornerRadius="8"

Padding="20">

<ScrollViewer>

<StackPanel Spacing="10">

<TextBlock Text="ДОБАВИТЬ ПРЕПОДАВАТЕЛЯ"

FontSize="17"

FontWeight="Bold"

Foreground="#2c3e50"

HorizontalAlignment="Center"

Margin="0,0,0,8"/>

<StackPanel Spacing="5">

<TextBlock Text="Фамилия:" FontWeight="SemiBold" FontSize="12" Foreground="#2c3e50"/>

<TextBox Text="{Binding LastName}" Watermark="Иванов" FontSize="12" Padding="8" Background="#F0F0F0" BorderBrush="#3498db" BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="5">

<TextBlock Text="Имя:" FontWeight="SemiBold" FontSize="12" Foreground="#2c3e50"/>

<TextBox Text="{Binding FirstName}" Watermark="Иван" FontSize="12" Padding="8" Background="#F0F0F0" BorderBrush="#3498db" BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="5">

<TextBlock Text="Отчество:" FontWeight="SemiBold" FontSize="12" Foreground="#2c3e50"/>

<TextBox Text="{Binding MiddleName}" Watermark="Иванович" FontSize="12" Padding="8" Background="#F0F0F0" BorderBrush="#3498db" BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="5">

<TextBlock Text="Должность:" FontWeight="SemiBold" FontSize="12" Foreground="#2c3e50"/>

<ComboBox ItemsSource="{Binding Positions}" SelectedItem="{Binding SelectedPositionItem}" HorizontalAlignment="Stretch" FontSize="12" Padding="8" Background="#F0F0F0" BorderBrush="#3498db" BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="5">

<TextBlock Text="Учёная степень:" FontWeight="SemiBold" FontSize="12" Foreground="#2c3e50"/>

<ComboBox ItemsSource="{Binding Degrees}" SelectedItem="{Binding SelectedDegreeItem}" HorizontalAlignment="Stretch" FontSize="12" Padding="8" Background="#F0F0F0" BorderBrush="#3498db" BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="5">

<TextBlock Text="Учёное звание:" FontWeight="SemiBold" FontSize="12" Foreground="#2c3e50"/>

<ComboBox ItemsSource="{Binding Titles}" SelectedItem="{Binding SelectedTitleItem}" HorizontalAlignment="Stretch" FontSize="12" Padding="8" Background="#F0F0F0" BorderBrush="#3498db" BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="5">

<TextBlock Text="Кафедра:" FontWeight="SemiBold" FontSize="12" Foreground="#2c3e50"/>

<ComboBox ItemsSource="{Binding Departments}" SelectedItem="{Binding SelectedDepartment}" HorizontalAlignment="Stretch" FontSize="12" Padding="8" Background="#F0F0F0" BorderBrush="#3498db" BorderThickness="2"/>

</StackPanel>

*<!-- Сообщение об ошибке -->*

<Border Background="#ffebee"

BorderBrush="#e74c3c"

BorderThickness="2"

CornerRadius="5"

Padding="8"

IsVisible="{Binding ErrorMessage, Converter={x:Static StringConverters.IsNotNullOrEmpty}}">

<TextBlock Text="{Binding ErrorMessage}"

Foreground="#c62828"

FontWeight="SemiBold"

FontSize="11"

TextWrapping="Wrap"/>

</Border>

<Border Background="#f0f8ff"

Padding="10"

CornerRadius="5"

Margin="0,5,0,0"

BorderBrush="#3498db"

BorderThickness="2">

<StackPanel Spacing="8">

<CheckBox IsChecked="{Binding IsPostgraduate}" Content="Обучается в аспирантуре" FontSize="12" FontWeight="SemiBold"/>

<CheckBox IsChecked="{Binding LeadsResearchTopics}" Content="Руководит научными темами" FontSize="12" FontWeight="SemiBold"/>

<CheckBox IsChecked="{Binding LeadsResearchDirections}" Content="Руководит научными направлениями" FontSize="12" FontWeight="SemiBold"/>

</StackPanel>

</Border>

<StackPanel Spacing="8" Margin="0,12,0,0">

<Button Content="ДОБАВИТЬ ПРЕПОДАВАТЕЛЯ" Command="{Binding AddCommand}" HorizontalAlignment="Stretch" Background="#27ae60" Foreground="White" FontSize="12" FontWeight="Bold" Padding="10" BorderThickness="0"/>

<Button Content="Обновить" Command="{Binding UpdateCommand}" HorizontalAlignment="Stretch" Background="#f39c12" Foreground="White" FontSize="11" FontWeight="SemiBold" Padding="8" BorderThickness="0"/>

<Button Content="Удалить" Command="{Binding DeleteCommand}" HorizontalAlignment="Stretch" Background="#e74c3c" Foreground="White" FontSize="11" FontWeight="SemiBold" Padding="8" BorderThickness="0"/>

<Button Content="ПРОФИЛЬ ПРЕПОДАВАТЕЛЯ"

Command="{Binding ShowProfileCommand}"

HorizontalAlignment="Stretch"

Background="#e67e22"

Foreground="White"

FontSize="13"

FontWeight="Bold"

Padding="10"

BorderThickness="0"

Margin="0,10,0,0"/>

</StackPanel>

</StackPanel>

</ScrollViewer>

</Border>

</Grid>

</Grid>

</UserControl>

**TeachersView.axaml.cs** – файл кода представления для раздела преподавателей, содержит конструктор для инициализации компонентов.

using Avalonia.Controls;

namespace UniversityIS.Views

{

public partial class TeachersView : UserControl

{

public TeachersView()

{

InitializeComponent();

}

}

}

**DisciplinesView.axaml** – файл разметки представления для раздела дисциплин в формате AXAML, определяет интерфейс управления дисциплинами с группировкой по группам.

<UserControl xmlns="https://github.com/avaloniaui"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:vm="using:UniversityIS.ViewModels"

mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="600"

x:Class="UniversityIS.Views.DisciplinesView"

x:DataType="vm:DisciplinesViewModel">

<Grid RowDefinitions="Auto,\*">

<Border Grid.Row="0" Background="#3498db" Padding="15" CornerRadius="5" Margin="0,0,0,15">

<TextBlock Text="УПРАВЛЕНИЕ ДИСЦИПЛИНАМИ"

FontSize="22"

FontWeight="Bold"

Foreground="White"

HorizontalAlignment="Center"/>

</Border>

<Grid Grid.Row="1" ColumnDefinitions="\*,380">

*<!-- Список дисциплин -->*

<Border Grid.Column="0"

Background="#F0F0F0"

CornerRadius="8"

Padding="15"

Margin="0,0,15,0"

BorderBrush="#3498db"

BorderThickness="2">

<Grid RowDefinitions="Auto,\*">

<StackPanel Grid.Row="0" Margin="0,0,0,15">

<TextBlock FontSize="16"

FontWeight="Bold"

Foreground="#2c3e50">

<Run Text="Список дисциплин (всего: "/>

<Run Text="{Binding Disciplines.Count}"/>

<Run Text=")"/>

</TextBlock>

</StackPanel>

<ScrollViewer Grid.Row="1" VerticalScrollBarVisibility="Auto">

<ItemsControl ItemsSource="{Binding GroupedDisciplines}">

<ItemsControl.ItemTemplate>

<DataTemplate>

<StackPanel Margin="0,0,0,20">

*<!-- Заголовок группы -->*

<Border Background="#9b59b6"

Padding="12,8"

CornerRadius="5"

Margin="0,0,0,10">

<TextBlock FontSize="15"

FontWeight="Bold"

Foreground="White">

<TextBlock.Inlines>

<Run Text="Группа "/>

<Run Text="{Binding GroupName}"/>

</TextBlock.Inlines>

</TextBlock>

</Border>

*<!-- Дисциплины этой группы -->*

<ListBox ItemsSource="{Binding Disciplines}"

SelectedItem="{Binding $parent[UserControl].((vm:DisciplinesViewModel)DataContext).SelectedDiscipline}"

Background="Transparent"

BorderThickness="0"

Margin="15,0,0,0">

<ListBox.ItemTemplate>

<DataTemplate>

<Border Background="#F0F0F0"

Margin="0,0,0,8"

Padding="15"

CornerRadius="5"

BorderBrush="#3498db"

BorderThickness="1">

<StackPanel>

<TextBlock FontSize="16" FontWeight="Bold" Foreground="#2c3e50">

<Run Text=""/>

<Run Text="{Binding Name}"/>

</TextBlock>

<TextBlock FontSize="14" Foreground="#2c3e50" Margin="0,5,0,0">

<Run Text="Курс "/>

<Run Text="{Binding Course}"/>

<Run Text=" | Семестр "/>

<Run Text="{Binding Semester}"/>

</TextBlock>

<TextBlock FontSize="13" Foreground="#34495e" Margin="0,3,0,0">

<Run Text="Лек: "/>

<Run Text="{Binding LectureHours}"/>

<Run Text="ч | Сем: "/>

<Run Text="{Binding SeminarHours}"/>

<Run Text="ч | Лаб: "/>

<Run Text="{Binding LaboratoryHours}"/>

<Run Text="ч"/>

</TextBlock>

</StackPanel>

</Border>

</DataTemplate>

</ListBox.ItemTemplate>

<ListBox.Styles>

<Style Selector="ListBoxItem">

<Setter Property="Padding" Value="0"/>

<Setter Property="Margin" Value="0"/>

</Style>

<Style Selector="ListBoxItem:selected /template/ ContentPresenter">

<Setter Property="Background" Value="Transparent"/>

</Style>

<Style Selector="ListBoxItem:selected Border">

<Setter Property="Background" Value="#e3f2fd"/>

<Setter Property="BorderBrush" Value="#2196f3"/>

<Setter Property="BorderThickness" Value="3"/>

</Style>

</ListBox.Styles>

</ListBox>

</StackPanel>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

</ScrollViewer>

</Grid>

</Border>

*<!-- Форма редактирования -->*

<Border Grid.Column="1"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="3"

CornerRadius="8"

Padding="20">

<ScrollViewer>

<StackPanel Spacing="12">

<TextBlock Text="ДОБАВИТЬ ДИСЦИПЛИНУ"

FontSize="17"

FontWeight="Bold"

Foreground="#2c3e50"

HorizontalAlignment="Center"

Margin="0,0,0,8"/>

<StackPanel Spacing="6">

<TextBlock Text="Группа:" FontWeight="SemiBold" FontSize="13" Foreground="#2c3e50"/>

<ComboBox ItemsSource="{Binding Groups}" SelectedItem="{Binding SelectedGroup}" HorizontalAlignment="Stretch" FontSize="13" Padding="8" Background="#F0F0F0" BorderBrush="#3498db" BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="6">

<TextBlock Text="Название дисциплины:" FontWeight="SemiBold" FontSize="13" Foreground="#2c3e50"/>

<TextBox Text="{Binding Name}" Watermark="Например: Программирование" FontSize="13" Padding="8" Background="#F0F0F0" BorderBrush="#3498db" BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="6">

<TextBlock Text="Семестр:" FontWeight="SemiBold" FontSize="13" Foreground="#2c3e50"/>

<NumericUpDown Value="{Binding Semester}" Minimum="1" Maximum="10" FontSize="13" Padding="8" Background="#F0F0F0" BorderBrush="#3498db" BorderThickness="2"/>

</StackPanel>

<TextBlock Text="Количество часов:" FontWeight="Bold" FontSize="14" Foreground="#3498db" Margin="0,5,0,0"/>

<StackPanel Spacing="6">

<TextBlock Text="Лекции (часов):" FontWeight="SemiBold" FontSize="12" Foreground="#2c3e50"/>

<NumericUpDown Value="{Binding LectureHours}" Minimum="0" Maximum="500" FontSize="12" Padding="8" Background="#F0F0F0" BorderBrush="#3498db" BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="6">

<TextBlock Text="Семинары (часов):" FontWeight="SemiBold" FontSize="12" Foreground="#2c3e50"/>

<NumericUpDown Value="{Binding SeminarHours}" Minimum="0" Maximum="500" FontSize="12" Padding="8" Background="#F0F0F0" BorderBrush="#3498db" BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="6">

<TextBlock Text="Лабораторные (часов):" FontWeight="SemiBold" FontSize="12" Foreground="#2c3e50"/>

<NumericUpDown Value="{Binding LaboratoryHours}" Minimum="0" Maximum="500" FontSize="12" Padding="8" Background="#F0F0F0" BorderBrush="#3498db" BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="6">

<TextBlock Text="Форма контроля:" FontWeight="SemiBold" FontSize="13" Foreground="#2c3e50"/>

<ComboBox ItemsSource="{Binding ControlForms}" SelectedItem="{Binding SelectedControlFormItem}" HorizontalAlignment="Stretch" FontSize="13" Padding="8" Background="#F0F0F0" BorderBrush="#3498db" BorderThickness="2"/>

</StackPanel>

*<!-- Сообщение об ошибке -->*

<Border Background="#ffebee"

BorderBrush="#e74c3c"

BorderThickness="2"

CornerRadius="5"

Padding="10"

IsVisible="{Binding ErrorMessage, Converter={x:Static StringConverters.IsNotNullOrEmpty}}">

<TextBlock Text="{Binding ErrorMessage}"

Foreground="#c62828"

FontWeight="SemiBold"

FontSize="12"

TextWrapping="Wrap"/>

</Border>

<StackPanel Spacing="8" Margin="0,12,0,0">

<Button Content="ДОБАВИТЬ ДИСЦИПЛИНУ" Command="{Binding AddCommand}" HorizontalAlignment="Stretch" Background="#27ae60" Foreground="White" FontSize="13" FontWeight="Bold" Padding="10" BorderThickness="0"/>

<Button Content="Обновить" Command="{Binding UpdateCommand}" HorizontalAlignment="Stretch" Background="#f39c12" Foreground="White" FontSize="12" FontWeight="SemiBold" Padding="8" BorderThickness="0"/>

<Button Content="Удалить" Command="{Binding DeleteCommand}" HorizontalAlignment="Stretch" Background="#e74c3c" Foreground="White" FontSize="12" FontWeight="SemiBold" Padding="8" BorderThickness="0"/>

</StackPanel>

</StackPanel>

</ScrollViewer>

</Border>

</Grid>

</Grid>

</UserControl>

**DisciplinesView.axaml.cs** – файл кода представления для раздела дисциплин, содержит конструктор для инициализации компонентов.

using Avalonia.Controls;

namespace UniversityIS.Views

{

public partial class DisciplinesView : UserControl

{

public DisciplinesView()

{

InitializeComponent();

}

}

}

**WorkLoadsView.axaml** – файл разметки представления для раздела учебной нагрузки в формате AXAML, определяет интерфейс управления учебной нагрузкой преподавателей.

<UserControl xmlns="https://github.com/avaloniaui"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:vm="using:UniversityIS.ViewModels"

mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="600"

x:Class="UniversityIS.Views.WorkLoadsView"

x:DataType="vm:WorkLoadsViewModel">

<Grid RowDefinitions="Auto,\*">

<Border Grid.Row="0" Background="#3498db" Padding="15" CornerRadius="5" Margin="0,0,0,15">

<TextBlock Text="УПРАВЛЕНИЕ НАГРУЗКОЙ"

FontSize="22"

FontWeight="Bold"

Foreground="White"

HorizontalAlignment="Center"/>

</Border>

<Grid Grid.Row="1" ColumnDefinitions="\*,380">

*<!-- Список нагрузок -->*

<Border Grid.Column="0"

Background="#F0F0F0"

CornerRadius="8"

Padding="15"

Margin="0,0,15,0"

BorderBrush="#3498db"

BorderThickness="2">

<Grid RowDefinitions="Auto,\*">

<StackPanel Grid.Row="0" Margin="0,0,0,15">

<TextBlock FontSize="16"

FontWeight="Bold"

Foreground="#2c3e50">

<Run Text="Учебные нагрузки (всего: "/>

<Run Text="{Binding WorkLoads.Count}"/>

<Run Text=")"/>

</TextBlock>

</StackPanel>

<ListBox Grid.Row="1"

ItemsSource="{Binding WorkLoads}"

SelectedItem="{Binding SelectedWorkLoad}"

Background="Transparent"

BorderThickness="0">

<ListBox.ItemTemplate>

<DataTemplate>

<Border Background="#F0F0F0"

Margin="0,0,0,8"

Padding="15"

CornerRadius="5"

BorderBrush="#3498db"

BorderThickness="1">

<StackPanel>

<TextBlock FontSize="15" FontWeight="Bold" Foreground="#2c3e50">

<Run Text=""/>

<Run Text="{Binding LessonType}"/>

<Run Text=" - "/>

<Run Text="{Binding Hours}"/>

<Run Text=" часов"/>

</TextBlock>

<TextBlock FontSize="13" Foreground="#2c3e50" Margin="0,5,0,0">

<Run Text=""/>

<Run Text="{Binding AcademicYear}"/>

<Run Text=" | Семестр "/>

<Run Text="{Binding Semester}"/>

</TextBlock>

</StackPanel>

</Border>

</DataTemplate>

</ListBox.ItemTemplate>

<ListBox.Styles>

<Style Selector="ListBoxItem">

<Setter Property="Padding" Value="0"/>

<Setter Property="Margin" Value="0"/>

</Style>

<Style Selector="ListBoxItem:selected /template/ ContentPresenter">

<Setter Property="Background" Value="Transparent"/>

</Style>

<Style Selector="ListBoxItem:selected Border">

<Setter Property="Background" Value="#e3f2fd"/>

<Setter Property="BorderBrush" Value="#2196f3"/>

<Setter Property="BorderThickness" Value="3"/>

</Style>

</ListBox.Styles>

</ListBox>

</Grid>

</Border>

*<!-- Форма редактирования -->*

<Border Grid.Column="1"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="3"

CornerRadius="8"

Padding="20">

<ScrollViewer>

<StackPanel Spacing="12">

<TextBlock Text="ДОБАВИТЬ НАГРУЗКУ"

FontSize="17"

FontWeight="Bold"

Foreground="#2c3e50"

HorizontalAlignment="Center"

Margin="0,0,0,8"/>

<StackPanel Spacing="6">

<TextBlock Text="Преподаватель:" FontWeight="SemiBold" FontSize="13" Foreground="#2c3e50"/>

<ComboBox ItemsSource="{Binding Teachers}" SelectedItem="{Binding SelectedTeacher}" HorizontalAlignment="Stretch" FontSize="12" Padding="8" Background="#F0F0F0" BorderBrush="#3498db" BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="6">

<TextBlock Text="Дисциплина:" FontWeight="SemiBold" FontSize="13" Foreground="#2c3e50"/>

<ComboBox ItemsSource="{Binding AvailableDisciplines}" SelectedItem="{Binding SelectedDiscipline}" HorizontalAlignment="Stretch" FontSize="12" Padding="8" Background="#F0F0F0" BorderBrush="#3498db" BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="6">

<TextBlock Text="Группа:" FontWeight="SemiBold" FontSize="13" Foreground="#2c3e50"/>

<ComboBox ItemsSource="{Binding Groups}" SelectedItem="{Binding SelectedGroup}" HorizontalAlignment="Stretch" FontSize="12" Padding="8" Background="#F0F0F0" BorderBrush="#3498db" BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="6">

<TextBlock Text="Вид занятий:" FontWeight="SemiBold" FontSize="13" Foreground="#2c3e50"/>

<ComboBox ItemsSource="{Binding LessonTypes}" SelectedItem="{Binding SelectedLessonTypeItem}" HorizontalAlignment="Stretch" FontSize="12" Padding="8" Background="#F0F0F0" BorderBrush="#3498db" BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="6">

<TextBlock Text="Количество часов:" FontWeight="SemiBold" FontSize="13" Foreground="#2c3e50"/>

<NumericUpDown Value="{Binding Hours}" Minimum="0" Maximum="500" FontSize="12" Padding="8" Background="#F0F0F0" BorderBrush="#3498db" BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="6">

<TextBlock Text="Учебный год:" FontWeight="SemiBold" FontSize="13" Foreground="#2c3e50"/>

<TextBox Text="{Binding AcademicYear}" Watermark="2024-2025" FontSize="12" Padding="8" Background="#F0F0F0" BorderBrush="#3498db" BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="6">

<TextBlock Text="Семестр:" FontWeight="SemiBold" FontSize="13" Foreground="#2c3e50"/>

<NumericUpDown Value="{Binding Semester}" Minimum="1" Maximum="12" FontSize="12" Padding="8" Background="#F0F0F0" BorderBrush="#3498db" BorderThickness="2"/>

</StackPanel>

*<!-- Сообщение об ошибке -->*

<Border Background="#ffebee"

BorderBrush="#e74c3c"

BorderThickness="2"

CornerRadius="5"

Padding="10"

IsVisible="{Binding ErrorMessage, Converter={x:Static StringConverters.IsNotNullOrEmpty}}">

<TextBlock Text="{Binding ErrorMessage}"

Foreground="#c62828"

FontWeight="SemiBold"

FontSize="12"

TextWrapping="Wrap"/>

</Border>

<StackPanel Spacing="8" Margin="0,12,0,0">

<Button Content="ДОБАВИТЬ НАГРУЗКУ" Command="{Binding AddCommand}" HorizontalAlignment="Stretch" Background="#27ae60" Foreground="White" FontSize="13" FontWeight="Bold" Padding="10" BorderThickness="0"/>

<Button Content="Обновить" Command="{Binding UpdateCommand}" HorizontalAlignment="Stretch" Background="#f39c12" Foreground="White" FontSize="12" FontWeight="SemiBold" Padding="8" BorderThickness="0"/>

<Button Content="Удалить" Command="{Binding DeleteCommand}" HorizontalAlignment="Stretch" Background="#e74c3c" Foreground="White" FontSize="12" FontWeight="SemiBold" Padding="8" BorderThickness="0"/>

</StackPanel>

</StackPanel>

</ScrollViewer>

</Border>

</Grid>

</Grid>

</UserControl>

**WorkLoadsView.axaml.cs** – файл кода представления для раздела учебной нагрузки, содержит конструктор для инициализации компонентов.

using Avalonia.Controls;

namespace UniversityIS.Views

{

public partial class WorkLoadsView : UserControl

{

public WorkLoadsView()

{

InitializeComponent();

}

}

}

**ThesisWorksView.axaml** – файл разметки представления для раздела дипломных работ в формате AXAML, определяет интерфейс управления дипломными работами.

<UserControl xmlns="https://github.com/avaloniaui"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:vm="using:UniversityIS.ViewModels"

mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="600"

x:Class="UniversityIS.Views.ThesisWorksView"

x:DataType="vm:ThesisWorksViewModel">

<Grid RowDefinitions="Auto,\*">

<Border Grid.Row="0" Background="#3498db" Padding="15" CornerRadius="5" Margin="0,0,0,15">

<TextBlock Text="УПРАВЛЕНИЕ ДИПЛОМНЫМИ РАБОТАМИ"

FontSize="22"

FontWeight="Bold"

Foreground="White"

HorizontalAlignment="Center"/>

</Border>

<Grid Grid.Row="1" ColumnDefinitions="\*,380">

*<!-- Список дипломных работ -->*

<Border Grid.Column="0"

Background="#F0F0F0"

CornerRadius="8"

Padding="15"

Margin="0,0,15,0"

BorderBrush="#3498db"

BorderThickness="2">

<Grid RowDefinitions="Auto,\*">

<StackPanel Grid.Row="0" Margin="0,0,0,15">

<TextBlock FontSize="16"

FontWeight="Bold"

Foreground="#2c3e50">

<Run Text="Дипломные работы (всего: "/>

<Run Text="{Binding ThesisWorks.Count}"/>

<Run Text=")"/>

</TextBlock>

</StackPanel>

<ListBox Grid.Row="1"

ItemsSource="{Binding ThesisWorks}"

SelectedItem="{Binding SelectedThesisWork}"

Background="Transparent"

BorderThickness="0">

<ListBox.ItemTemplate>

<DataTemplate>

<Border Background="#F0F0F0"

Margin="0,0,0,8"

Padding="15"

CornerRadius="5"

BorderBrush="#3498db"

BorderThickness="1">

<StackPanel>

<TextBlock FontSize="15" FontWeight="Bold" Foreground="#2c3e50" TextWrapping="Wrap">

<Run Text=""/>

<Run Text="{Binding Title}"/>

</TextBlock>

<TextBlock FontSize="13" Foreground="#2c3e50" Margin="0,5,0,0">

<Run Text="Год: "/>

<Run Text="{Binding Year}"/>

<Run Text=" | Оценка: "/>

<Run Text="{Binding Grade}" FontWeight="Bold"/>

</TextBlock>

</StackPanel>

</Border>

</DataTemplate>

</ListBox.ItemTemplate>

<ListBox.Styles>

<Style Selector="ListBoxItem">

<Setter Property="Padding" Value="0"/>

<Setter Property="Margin" Value="0"/>

</Style>

<Style Selector="ListBoxItem:selected /template/ ContentPresenter">

<Setter Property="Background" Value="Transparent"/>

</Style>

<Style Selector="ListBoxItem:selected Border">

<Setter Property="Background" Value="#e3f2fd"/>

<Setter Property="BorderBrush" Value="#2196f3"/>

<Setter Property="BorderThickness" Value="3"/>

</Style>

</ListBox.Styles>

</ListBox>

</Grid>

</Border>

*<!-- Форма редактирования -->*

<Border Grid.Column="1"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="3"

CornerRadius="8"

Padding="20">

<ScrollViewer>

<StackPanel Spacing="12">

<TextBlock Text="ДОБАВИТЬ ДИПЛОМНУЮ РАБОТУ"

FontSize="16"

FontWeight="Bold"

Foreground="#2c3e50"

HorizontalAlignment="Center"

Margin="0,0,0,8"

TextWrapping="Wrap"/>

<StackPanel Spacing="6">

<TextBlock Text="Название работы:" FontWeight="SemiBold" FontSize="13" Foreground="#2c3e50"/>

<TextBox Text="{Binding Title}"

Watermark="Например: Разработка информационной системы"

FontSize="12"

Padding="8"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="2"

TextWrapping="Wrap"

AcceptsReturn="True"

Height="80"/>

</StackPanel>

<StackPanel Spacing="6">

<TextBlock Text="Студент:" FontWeight="SemiBold" FontSize="13" Foreground="#2c3e50"/>

<ComboBox ItemsSource="{Binding Students}"

SelectedItem="{Binding SelectedStudent}"

HorizontalAlignment="Stretch"

FontSize="12"

Padding="8"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="6">

<TextBlock Text="Научный руководитель:" FontWeight="SemiBold" FontSize="13" Foreground="#2c3e50"/>

<ComboBox ItemsSource="{Binding Supervisors}"

SelectedItem="{Binding SelectedSupervisor}"

HorizontalAlignment="Stretch"

FontSize="12"

Padding="8"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="6">

<TextBlock Text="Год защиты:" FontWeight="SemiBold" FontSize="13" Foreground="#2c3e50"/>

<NumericUpDown Value="{Binding Year}"

Minimum="2000"

Maximum="2100"

FontSize="12"

Padding="8"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="2"/>

</StackPanel>

<StackPanel Spacing="6">

<TextBlock Text="Оценка (2-5):" FontWeight="SemiBold" FontSize="13" Foreground="#2c3e50"/>

<NumericUpDown Value="{Binding Grade}"

Minimum="2"

Maximum="5"

Increment="1"

AllowSpin="True"

FontSize="12"

Padding="8"

Background="#F0F0F0"

BorderBrush="#3498db"

BorderThickness="2"

Height="40"/>

</StackPanel>

*<!-- Сообщение об ошибке -->*

<Border Background="#ffebee"

BorderBrush="#e74c3c"

BorderThickness="2"

CornerRadius="5"

Padding="10"

IsVisible="{Binding ErrorMessage, Converter={x:Static StringConverters.IsNotNullOrEmpty}}">

<TextBlock Text="{Binding ErrorMessage}"

Foreground="#c62828"

FontWeight="SemiBold"

FontSize="12"

TextWrapping="Wrap"/>

</Border>

<StackPanel Spacing="8" Margin="0,12,0,0">

<Button Content="ДОБАВИТЬ РАБОТУ"

Command="{Binding AddCommand}"

HorizontalAlignment="Stretch"

Background="#27ae60"

Foreground="White"

FontSize="13"

FontWeight="Bold"

Padding="10"

BorderThickness="0"/>

<Button Content="Обновить"

Command="{Binding UpdateCommand}"

HorizontalAlignment="Stretch"

Background="#f39c12"

Foreground="White"

FontSize="12"

FontWeight="SemiBold"

Padding="8"

BorderThickness="0"/>

<Button Content="Удалить"

Command="{Binding DeleteCommand}"

HorizontalAlignment="Stretch"

Background="#e74c3c"

Foreground="White"

FontSize="12"

FontWeight="SemiBold"

Padding="8"

BorderThickness="0"/>

</StackPanel>

</StackPanel>

</ScrollViewer>

</Border>

</Grid>

</Grid>

</UserControl>

**ThesisWorksView.axaml.cs** – файл кода представления для раздела дипломных работ, содержит конструктор для инициализации компонентов.

using Avalonia.Controls;

namespace UniversityIS.Views

{

public partial class ThesisWorksView : UserControl

{

public ThesisWorksView()

{

InitializeComponent();

}

}

}

**StudentProfileView.axaml** – файл разметки представления для профиля студента в формате AXAML, определяет интерфейс отображения информации о студенте, его дисциплинах и оценках.

<UserControl xmlns="https://github.com/avaloniaui"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:vm="using:UniversityIS.ViewModels"

mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="600"

x:Class="UniversityIS.Views.StudentProfileView"

x:DataType="vm:StudentProfileViewModel">

<Grid RowDefinitions="Auto,\*">

*<!-- Заголовок профиля -->*

<Border Grid.Row="0" Background="#3498db" Padding="15" CornerRadius="5" Margin="0,0,0,15">

<StackPanel>

<TextBlock Text="ПРОФИЛЬ СТУДЕНТА"

FontSize="22"

FontWeight="Bold"

Foreground="White"

HorizontalAlignment="Center"/>

<TextBlock Text="{Binding FullName}"

FontSize="18"

FontWeight="SemiBold"

Foreground="White"

HorizontalAlignment="Center"

Margin="0,5,0,0"/>

<TextBlock FontSize="14"

Foreground="White"

HorizontalAlignment="Center"

Margin="0,3,0,0">

<Run Text="Группа: "/>

<Run Text="{Binding GroupName}" FontWeight="Bold"/>

<Run Text=" | Текущий семестр: "/>

<Run Text="{Binding CurrentSemester}" FontWeight="Bold"/>

</TextBlock>

</StackPanel>

</Border>

*<!-- Списки дисциплин -->*

<Grid Grid.Row="1" ColumnDefinitions="\*,\*,\*" ColumnSpacing="15">

*<!-- Пройденные предметы -->*

<Border Grid.Column="0"

Background="White"

CornerRadius="8"

Padding="15"

BorderBrush="#27ae60"

BorderThickness="2">

<Grid RowDefinitions="Auto,\*">

<TextBlock Grid.Row="0"

Text="ПРОЙДЕННЫЕ ПРЕДМЕТЫ"

FontSize="14"

FontWeight="Bold"

Foreground="#27ae60"

HorizontalAlignment="Center"

Margin="0,0,0,10"/>

<ScrollViewer Grid.Row="1" VerticalScrollBarVisibility="Auto">

<ItemsControl ItemsSource="{Binding CompletedDisciplines}">

<ItemsControl.ItemTemplate>

<DataTemplate>

<StackPanel>

*<!-- Заголовок семестра -->*

<Border Background="#27ae60"

CornerRadius="5"

Padding="8"

Margin="0,5,0,5">

<TextBlock Text="{Binding SemesterName}"

FontSize="12"

FontWeight="Bold"

Foreground="White"

HorizontalAlignment="Center"/>

</Border>

*<!-- Дисциплины семестра -->*

<ItemsControl ItemsSource="{Binding Disciplines}">

<ItemsControl.ItemTemplate>

<DataTemplate>

<Border Background="#d4edda"

Margin="0,0,0,5"

Padding="8"

CornerRadius="5"

BorderBrush="#27ae60"

BorderThickness="1">

<Button Command="{Binding $parent[UserControl].((vm:StudentProfileViewModel)DataContext).OpenGradeInputCommand}"

CommandParameter="{Binding}"

Background="Transparent"

BorderThickness="0"

Padding="0"

HorizontalAlignment="Stretch"

HorizontalContentAlignment="Stretch"

Cursor="Hand">

<StackPanel>

<TextBlock Text="{Binding Discipline.Name}"

FontSize="12"

FontWeight="Bold"

Foreground="#155724"

TextWrapping="Wrap"/>

<TextBlock FontSize="10" Foreground="#155724" Margin="0,2,0,0">

<Run Text="Лек: "/>

<Run Text="{Binding Discipline.LectureHours}"/>

<Run Text="ч | Сем: "/>

<Run Text="{Binding Discipline.SeminarHours}"/>

<Run Text="ч | Лаб: "/>

<Run Text="{Binding Discipline.LaboratoryHours}"/>

<Run Text="ч"/>

</TextBlock>

<Border Background="#c3e6cb"

Padding="4"

CornerRadius="3"

Margin="0,4,0,0">

<TextBlock Text="{Binding GradeText}"

FontSize="10"

FontWeight="SemiBold"

Foreground="#155724"

TextAlignment="Center"

TextWrapping="Wrap"/>

</Border>

</StackPanel>

</Button>

</Border>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

</StackPanel>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

</ScrollViewer>

</Grid>

</Border>

*<!-- Текущие предметы -->*

<Border Grid.Column="1"

Background="White"

CornerRadius="8"

Padding="15"

BorderBrush="#3498db"

BorderThickness="2">

<Grid RowDefinitions="Auto,\*">

<TextBlock Grid.Row="0"

Text="ТЕКУЩИЙ СЕМЕСТР"

FontSize="14"

FontWeight="Bold"

Foreground="#3498db"

HorizontalAlignment="Center"

Margin="0,0,0,10"/>

<ScrollViewer Grid.Row="1" VerticalScrollBarVisibility="Auto">

<ItemsControl ItemsSource="{Binding CurrentDisciplines}">

<ItemsControl.ItemTemplate>

<DataTemplate>

<Border Background="#d1ecf1"

Margin="0,0,0,8"

Padding="10"

CornerRadius="5"

BorderBrush="#3498db"

BorderThickness="1">

<Button Command="{Binding $parent[UserControl].((vm:StudentProfileViewModel)DataContext).OpenGradeInputCommand}"

CommandParameter="{Binding}"

Background="Transparent"

BorderThickness="0"

Padding="0"

HorizontalAlignment="Stretch"

HorizontalContentAlignment="Stretch"

Cursor="Hand">

<StackPanel>

<TextBlock Text="{Binding Discipline.Name}"

FontSize="13"

FontWeight="Bold"

Foreground="#0c5460"

TextWrapping="Wrap"/>

<TextBlock FontSize="11" Foreground="#0c5460" Margin="0,3,0,0">

<Run Text="Семестр "/>

<Run Text="{Binding Discipline.Semester}"/>

<Run Text=" ("/>

<Run Text="{Binding Discipline.Course}"/>

<Run Text=" курс)"/>

</TextBlock>

<TextBlock FontSize="10" Foreground="#0c5460" Margin="0,2,0,0">

<Run Text="Лек: "/>

<Run Text="{Binding Discipline.LectureHours}"/>

<Run Text="ч | Сем: "/>

<Run Text="{Binding Discipline.SeminarHours}"/>

<Run Text="ч | Лаб: "/>

<Run Text="{Binding Discipline.LaboratoryHours}"/>

<Run Text="ч"/>

</TextBlock>

<Border Background="#bee5eb"

Padding="4"

CornerRadius="3"

Margin="0,4,0,0">

<TextBlock Text="{Binding GradeText}"

FontSize="10"

FontWeight="SemiBold"

Foreground="#0c5460"

TextAlignment="Center"

TextWrapping="Wrap"/>

</Border>

</StackPanel>

</Button>

</Border>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

</ScrollViewer>

</Grid>

</Border>

*<!-- Будущие предметы -->*

<Border Grid.Column="2"

Background="White"

CornerRadius="8"

Padding="15"

BorderBrush="#95a5a6"

BorderThickness="2">

<Grid RowDefinitions="Auto,\*">

<TextBlock Grid.Row="0"

Text="БУДУЩИЕ ПРЕДМЕТЫ"

FontSize="14"

FontWeight="Bold"

Foreground="#95a5a6"

HorizontalAlignment="Center"

Margin="0,0,0,10"/>

<ScrollViewer Grid.Row="1" VerticalScrollBarVisibility="Auto">

<ItemsControl ItemsSource="{Binding FutureDisciplines}">

<ItemsControl.ItemTemplate>

<DataTemplate>

<Border Background="#f8f9fa"

Margin="0,0,0,8"

Padding="10"

CornerRadius="5"

BorderBrush="#95a5a6"

BorderThickness="1">

<Button Command="{Binding $parent[UserControl].((vm:StudentProfileViewModel)DataContext).OpenGradeInputCommand}"

CommandParameter="{Binding}"

Background="Transparent"

BorderThickness="0"

Padding="0"

HorizontalAlignment="Stretch"

HorizontalContentAlignment="Stretch"

Cursor="Hand">

<StackPanel>

<TextBlock Text="{Binding Discipline.Name}"

FontSize="13"

FontWeight="Bold"

Foreground="#6c757d"

TextWrapping="Wrap"/>

<TextBlock FontSize="11" Foreground="#6c757d" Margin="0,3,0,0">

<Run Text="Семестр "/>

<Run Text="{Binding Discipline.Semester}"/>

<Run Text=" ("/>

<Run Text="{Binding Discipline.Course}"/>

<Run Text=" курс)"/>

</TextBlock>

<TextBlock FontSize="10" Foreground="#6c757d" Margin="0,2,0,0">

<Run Text="Лек: "/>

<Run Text="{Binding Discipline.LectureHours}"/>

<Run Text="ч | Сем: "/>

<Run Text="{Binding Discipline.SeminarHours}"/>

<Run Text="ч | Лаб: "/>

<Run Text="{Binding Discipline.LaboratoryHours}"/>

<Run Text="ч"/>

</TextBlock>

<Border Background="#e2e3e5"

Padding="4"

CornerRadius="3"

Margin="0,4,0,0">

<TextBlock Text="{Binding GradeText}"

FontSize="10"

FontWeight="SemiBold"

Foreground="#6c757d"

TextAlignment="Center"

TextWrapping="Wrap"/>

</Border>

</StackPanel>

</Button>

</Border>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

</ScrollViewer>

</Grid>

</Border>

</Grid>

</Grid>

</UserControl>

**StudentProfileView.axaml.cs** – файл кода представления для профиля студента, содержит конструктор для инициализации компонентов.

using Avalonia.Controls;

namespace UniversityIS.Views

{

public partial class StudentProfileView : UserControl

{

public StudentProfileView()

{

InitializeComponent();

}

}

}

**TeacherProfileView.axaml** – файл разметки представления для профиля преподавателя в формате AXAML, определяет интерфейс управления списком дисциплин преподавателя.

<UserControl xmlns="https://github.com/avaloniaui"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:vm="using:UniversityIS.ViewModels"

mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="600"

x:Class="UniversityIS.Views.TeacherProfileView"

x:DataType="vm:TeacherProfileViewModel">

<Grid RowDefinitions="Auto,\*">

*<!-- Заголовок профиля -->*

<Border Grid.Row="0" Background="#e67e22" Padding="15" CornerRadius="5" Margin="0,0,0,15">

<StackPanel>

<TextBlock Text="ПРОФИЛЬ ПРЕПОДАВАТЕЛЯ"

FontSize="22"

FontWeight="Bold"

Foreground="White"

HorizontalAlignment="Center"/>

<TextBlock Text="{Binding TeacherName}"

FontSize="18"

FontWeight="SemiBold"

Foreground="White"

HorizontalAlignment="Center"

Margin="0,5,0,0"/>

<TextBlock FontSize="14"

Foreground="White"

HorizontalAlignment="Center"

Margin="0,3,0,0">

<Run Text="{Binding Position}"/>

<Run Text=" | "/>

<Run Text="{Binding Department}"/>

</TextBlock>

</StackPanel>

</Border>

*<!-- Управление дисциплинами -->*

<Grid Grid.Row="1" ColumnDefinitions="\*,\*">

*<!-- Дисциплины преподавателя -->*

<Border Grid.Column="0"

Background="White"

CornerRadius="8"

Padding="15"

Margin="0,0,10,0"

BorderBrush="#e67e22"

BorderThickness="2">

<Grid RowDefinitions="Auto,\*,Auto">

<TextBlock Grid.Row="0"

Text="МОИ ДИСЦИПЛИНЫ"

FontSize="16"

FontWeight="Bold"

Foreground="#e67e22"

HorizontalAlignment="Center"

Margin="0,0,0,10"/>

<ScrollViewer Grid.Row="1" VerticalScrollBarVisibility="Auto">

<ItemsControl ItemsSource="{Binding TeacherDisciplines}">

<ItemsControl.ItemTemplate>

<DataTemplate>

<Border Background="#ffeaa7"

Margin="0,0,0,8"

Padding="12"

CornerRadius="5"

BorderBrush="#e67e22"

BorderThickness="1">

<StackPanel>

<TextBlock Text="{Binding Name}"

FontSize="14"

FontWeight="Bold"

Foreground="#2c3e50"

TextWrapping="Wrap"/>

<TextBlock FontSize="12" Foreground="#34495e" Margin="0,5,0,0">

<Run Text="Курс "/>

<Run Text="{Binding Course}"/>

<Run Text=" | Семестр "/>

<Run Text="{Binding Semester}"/>

</TextBlock>

</StackPanel>

</Border>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

</ScrollViewer>

<Button Grid.Row="2"

Content="УДАЛИТЬ ВЫБРАННУЮ"

Command="{Binding RemoveDisciplineCommand}"

Background="#e74c3c"

Foreground="White"

FontSize="13"

FontWeight="Bold"

Padding="10"

BorderThickness="0"

Margin="0,10,0,0"/>

</Grid>

</Border>

*<!-- Добавление новых дисциплин -->*

<Border Grid.Column="1"

Background="White"

CornerRadius="8"

Padding="15"

Margin="10,0,0,0"

BorderBrush="#3498db"

BorderThickness="2">

<Grid RowDefinitions="Auto,\*,Auto">

<TextBlock Grid.Row="0"

Text="ДОБАВИТЬ ДИСЦИПЛИНУ"

FontSize="16"

FontWeight="Bold"

Foreground="#3498db"

HorizontalAlignment="Center"

Margin="0,0,0,10"/>

<ScrollViewer Grid.Row="1" VerticalScrollBarVisibility="Auto">

<StackPanel Spacing="10">

<TextBlock Text="Выберите дисциплину:"

FontWeight="SemiBold"

FontSize="13"

Foreground="#2c3e50"/>

<ComboBox ItemsSource="{Binding AvailableDisciplines}"

SelectedItem="{Binding SelectedDisciplineToAdd}"

Background="#F0F0F0"

FontSize="13"

Height="40">

<ComboBox.ItemTemplate>

<DataTemplate>

<TextBlock>

<Run Text="{Binding Name}"/>

<Run Text=" ("/>

<Run Text="{Binding Course}"/>

<Run Text=" курс, "/>

<Run Text="{Binding Semester}"/>

<Run Text=" сем.)"/>

</TextBlock>

</DataTemplate>

</ComboBox.ItemTemplate>

</ComboBox>

<TextBlock Text="{Binding ErrorMessage}"

Foreground="Red"

FontSize="12"

TextWrapping="Wrap"

Margin="0,5,0,0"

IsVisible="{Binding ErrorMessage, Converter={x:Static StringConverters.IsNotNullOrEmpty}}"/>

</StackPanel>

</ScrollViewer>

<Button Grid.Row="2"

Content="ДОБАВИТЬ"

Command="{Binding AddDisciplineCommand}"

Background="#27ae60"

Foreground="White"

FontSize="13"

FontWeight="Bold"

Padding="10"

BorderThickness="0"

Margin="0,10,0,0"/>

</Grid>

</Border>

</Grid>

</Grid>

</UserControl>

**TeacherProfileView.axaml.cs** – файл кода представления для профиля преподавателя, содержит конструктор для инициализации компонентов.

using Avalonia.Controls;

namespace UniversityIS.Views

{

public partial class TeacherProfileView : UserControl

{

public TeacherProfileView()

{

InitializeComponent();

}

}

}

**GradeInputWindow.axaml** – файл разметки модального окна ввода оценки студента в формате AXAML, определяет интерфейс для ввода баллов за семестр и экзамен/зачет.

<Window xmlns="https://github.com/avaloniaui"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:vm="using:UniversityIS.ViewModels"

mc:Ignorable="d" d:DesignWidth="500" d:DesignHeight="550"

x:Class="UniversityIS.Views.GradeInputWindow"

x:DataType="vm:GradeInputViewModel"

Title="Выставление оценки"

Width="500"

Height="550"

WindowStartupLocation="CenterOwner"

CanResize="False">

<Border Background="#ecf0f1" Padding="20">

<StackPanel Spacing="15">

*<!-- Заголовок -->*

<Border Background="#3498db" Padding="15" CornerRadius="5">

<StackPanel>

<TextBlock Text="ВЫСТАВЛЕНИЕ ОЦЕНКИ"

FontSize="18"

FontWeight="Bold"

Foreground="White"

HorizontalAlignment="Center"/>

<TextBlock Text="{Binding DisciplineName}"

FontSize="14"

Foreground="White"

HorizontalAlignment="Center"

Margin="0,5,0,0"/>

</StackPanel>

</Border>

*<!-- Информация о студенте и предмете -->*

<Border Background="White" Padding="15" CornerRadius="5" BorderBrush="#3498db" BorderThickness="2">

<StackPanel Spacing="8">

<TextBlock FontSize="13" Foreground="#2c3e50">

<Run Text="Студент: " FontWeight="SemiBold"/>

<Run Text="{Binding StudentName}"/>

</TextBlock>

<TextBlock FontSize="13" Foreground="#2c3e50">

<Run Text="Форма контроля: " FontWeight="SemiBold"/>

<Run Text="{Binding ControlFormName}"/>

</TextBlock>

</StackPanel>

</Border>

*<!-- Ввод баллов за семестр -->*

<Border Background="White" Padding="15" CornerRadius="5" BorderBrush="#27ae60" BorderThickness="2">

<StackPanel Spacing="10">

<TextBlock FontSize="14" FontWeight="SemiBold" Foreground="#27ae60">

<Run Text="Баллы за семестр (макс: "/>

<Run Text="{Binding MaxSemesterPoints}"/>

<Run Text=")"/>

</TextBlock>

<NumericUpDown Value="{Binding SemesterPoints}"

Minimum="0"

Maximum="{Binding MaxSemesterPoints}"

Increment="1"

Background="#F0F0F0"

FontSize="14"

Height="40"/>

</StackPanel>

</Border>

*<!-- Ввод баллов на экзамене/зачете -->*

<Border Background="White" Padding="15" CornerRadius="5" BorderBrush="#e67e22" BorderThickness="2">

<StackPanel Spacing="10">

<TextBlock FontSize="14" FontWeight="SemiBold" Foreground="#e67e22">

<Run Text="Баллы на экзамене/зачете (макс: "/>

<Run Text="{Binding MaxExamPoints}"/>

<Run Text=")"/>

</TextBlock>

<NumericUpDown Value="{Binding ExamPoints}"

Minimum="0"

Maximum="{Binding MaxExamPoints}"

Increment="1"

Background="#F0F0F0"

FontSize="14"

Height="40"/>

</StackPanel>

</Border>

*<!-- Результат -->*

<Border Background="#9b59b6" Padding="15" CornerRadius="5">

<StackPanel Spacing="8">

<TextBlock FontSize="16" FontWeight="Bold" Foreground="White" HorizontalAlignment="Center">

<Run Text="Итого: "/>

<Run Text="{Binding TotalPoints}"/>

<Run Text=" баллов"/>

</TextBlock>

<TextBlock Text="{Binding Grade}"

FontSize="18"

FontWeight="Bold"

Foreground="White"

HorizontalAlignment="Center"/>

</StackPanel>

</Border>

*<!-- Сообщение об ошибке -->*

<TextBlock Text="{Binding ErrorMessage}"

Foreground="Red"

FontSize="12"

TextWrapping="Wrap"

IsVisible="{Binding ErrorMessage, Converter={x:Static StringConverters.IsNotNullOrEmpty}}"/>

*<!-- Кнопки -->*

<StackPanel Orientation="Horizontal" HorizontalAlignment="Center" Spacing="10" Margin="0,10,0,0">

<Button Content="СОХРАНИТЬ"

Command="{Binding SaveCommand}"

Click="OnSaveClick"

Background="#27ae60"

Foreground="White"

FontSize="14"

FontWeight="Bold"

Padding="20,12"

BorderThickness="0"

CornerRadius="5"/>

<Button Content="ОТМЕНА"

Click="OnCancelClick"

Background="#95a5a6"

Foreground="White"

FontSize="14"

FontWeight="Bold"

Padding="20,12"

BorderThickness="0"

CornerRadius="5"/>

</StackPanel>

</StackPanel>

</Border>

</Window>

**GradeInputWindow.axaml.cs** – файл кода модального окна ввода оценки, содержит конструктор и обработчики событий для сохранения и отмены.

using Avalonia.Controls;

using Avalonia.Interactivity;

namespace UniversityIS.Views

{

public partial class GradeInputWindow : Window

{

public GradeInputWindow()

{

InitializeComponent();

}

private void OnSaveClick(object? sender, RoutedEventArgs e)

{

Close(true); *// Закрыть окно с результатом "сохранено"*

}

private void OnCancelClick(object? sender, RoutedEventArgs e)

{

Close(false); *// Закрыть окно без сохранения*

}

}

}