

## **ЛАБОРАТОРНАЯ РАБОТА №9**

### **Знакомство с основами LINQ to Entities. Знакомство с многоуровневой архитектурой приложения. Знакомство с XML**

#### **1 Цель работы**

Изучить возможности построения запросов к базе данных с помощью LINQ to Entities.

Познакомиться с разработкой многоуровневой архитектуры приложения.

Познакомиться с методами сохранения состояния объектов классов в файл.

#### **2 Постановка задачи**

Для заданной в индивидуальном задании предметной области, используя подход «Code First» создать базу данных, содержащую две таблицы, связанные отношением «один-ко-многим».

Для объектов одной из таблиц предусмотреть вывод изображения. Изображения должны храниться в папке Images, а в базе данных должно храниться только имя файла (без имени папки). (Для привязки данных реализовать конвертор значений – ValueConverter)

В программе на основе LINQ to Entities реализовать для спроектированной БД возможность просмотра и редактирования данных.

В программе использовать многослойную архитектуру, выделив в отдельные сборки уровень доступа к базе данных (DAL – Data Access Level) и уровень бизнес-логики (BLL – Business Logic Level).

В главном окне реализовать отображение информации в виде Master-Slave (Главный – Подчиненный), когда при выборе в списке строки одной таблицы автоматически отображается содержимое подчиненной таблицы

Составить UML диаграмму классов для спроектированной системы.

Реализовать в лабораторной работе №5 сохранения/чтения данных в XML файл, используя технологию LINQ to XML.

### 3 Индивидуальные задания

Варианты предметной области для проектирования БД:

1. Автотранспорт
2. Жилищно-коммунальная сфера
3. Здравоохранение
4. Бытовое обслуживание населения
5. Образование
6. Муниципальное управление
7. Железнодорожный транспорт
8. Авиаперевозки
9. Компьютерная техника
10. Энергетика

### 4 Пример работы приложения

#### 4.1 Диаграмма классов

Диаграмма классов приведена на Рисунок 1

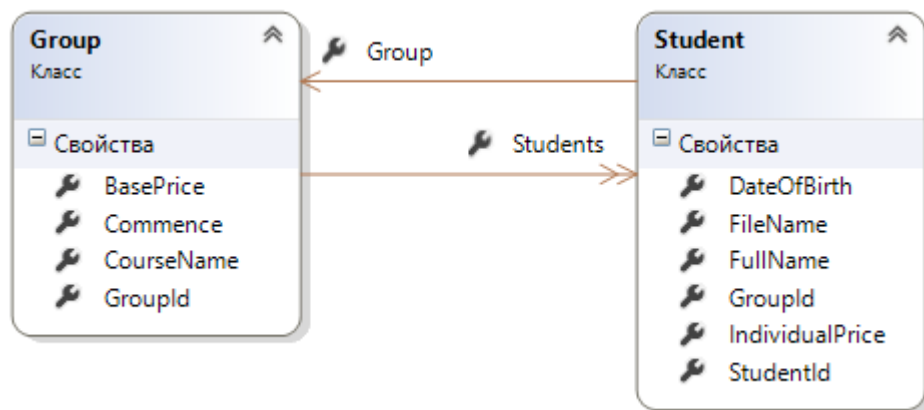


Рисунок 1 Диаграмма классов

Отношения между классами: в одной группе может быть несколько студентов.

Поле IndividualPrice – стоимость курса для конкретного студента. Стоимость может отличаться от базовой стоимости курса, если у студента есть скидка.

Полученная схема базы данных приведена на Рисунок 2

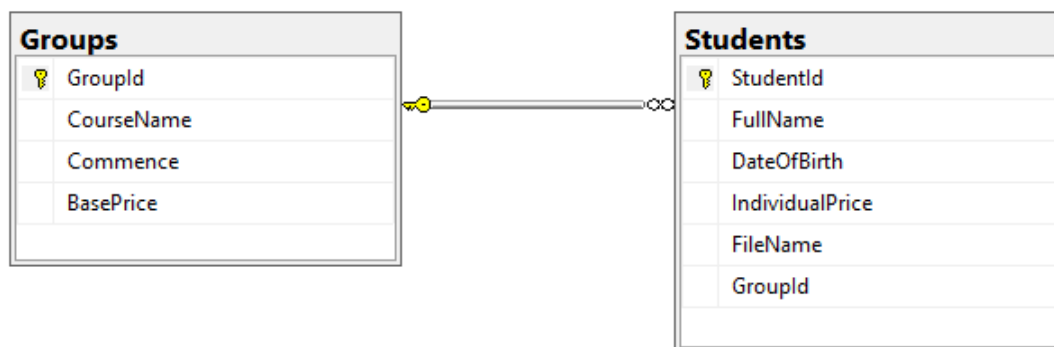


Рисунок 2 Схема базы данных

## 4.2 Пример главного окна приложения

Вариант главного окна приложения приведен на Рисунок 3

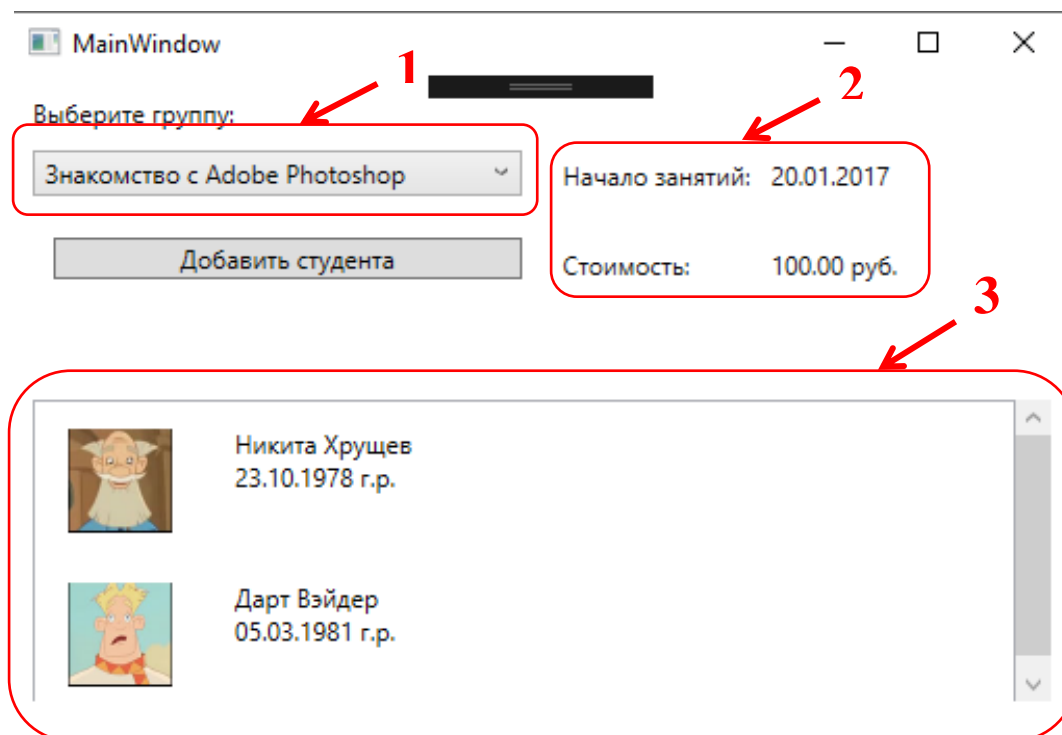


Рисунок 3 Пример главного окна приложения

Поле 1 – ComboBox со списком всех групп.

Поле 2 – подробная информация о группе.

Поле 3 – список студентов группы.

При изменении выбора в списке групп поля 2 и 3 автоматически обновляются в соответствии с выбором.

## 5 Пример решения

### 5.1 Предварительная информация

В данном примере приложение называется Lab9. Поэтому все создаваемые пространства имен начинаются с «Lab9». Например, Lab9.Infrastructure.

### 5.2 Создание уровня доступа к данным


Добавьте в решение новый проект – библиотеку классов. Присвойте проекту имя Lab9.DataLayer (см. п.5.1).

Добавьте в проект пакет Entity Framework (см. материалы лабораторной работы №8).

Добавьте в созданный проект папки: EFContext, Entities, Interfaces, Repositories.


В папке Entities создайте классы, описывающие сущности:

```
namespace Lab9.DataLayer.Entities
{
    public class Student
    {
        public int StudentId { get; set; }
        public string FullName { get; set; }
        public DateTime DateOfBirth { get; set; }
        public decimal IndividualPrice { get; set; }
        public string FileName { get; set; }
        // Навигационные свойства
        public int GroupId { get; set; }
        public Group Group { get; set; }
    }
}
```



```
namespace Lab9.DataLayer.Entities
{
    public class Group
    {
        public Group()
        {

```



```

        Students = new List<Student>();
    }
    public int GroupId { get; set; }
    public string CourseName { get; set; }
    public DateTime Commence { get; set; }
    public decimal BasePrice { get; set; }
    // НАВИГАЦИОННОЕ СВОЙСТВО
    public List<Student> Students { get; set; }
}
}


```

В папке EFContext опишите класс контекста данных:

```

using System.Data.Entity;
using Lab9.DataLayer.Entities;
namespace Lab9.DataLayer.EFContext
{
    public class CoursesContext:DbContext
    {
        public CoursesContext(string name) :base (name)
        {
            Database.SetInitializer(new CoursesInitializer());
        }
        public DbSet<Student> Students { get; set; }
        public DbSet<Group> Groups { get; set; }
    }
}

```



**Пример класса инициализации данных:**

```

using System;
using System.Collections.Generic;
using System.Data.Entity;
using Lab9.DataLayer.Entities;
namespace Lab9.DataLayer.EFContext
{
    class CoursesInitializer: CreateDatabaseIfNotExists<CoursesContext>
    {
        protected override void Seed(CoursesContext context)
        {
            context.Groups.AddRange(new Group[] {
                new Group { BasePrice=100, Commence=new DateTime(2017,
01, 20), CourseName="Знакомство с Adobe Photoshop",
                Students=new List<Student>
                {

```

```

        new Student { IndividualPrice=100,
DateOfBirth=new DateTime (1978, 10,23),
        FullName="Никита Хрущев", FileName="1.jpg" },
        new Student { IndividualPrice=100,
DateOfBirth=new DateTime (1981, 03,05),
        FullName="Дарт Вейдер", FileName="2.jpg" }
    }
},
    new Group { BasePrice=150, Commence=new DateTime(2017,
02, 11), CourseName="Вязание крючком",
        Students=new List<Student>
        {
            new Student { IndividualPrice=120,
DateOfBirth=new DateTime (1991, 06,14),
                FullName="Иван Драго", FileName="3.jpg" },
            new Student { IndividualPrice=150,
DateOfBirth=new DateTime (1989, 12,06),
                FullName="Геракл Зевсович", FileName="4.jpg"
            },
            new Student { IndividualPrice=150,
DateOfBirth=new DateTime (1985, 09,11),
                FullName="Мерлин Мэнсон", FileName="5.jpg" }
        }
    }
});
context.SaveChanges();
}
}
}

```

Для удобства работы с базой данных создадим классы репозитория, реализующих базовые операции CRUD (Create-Read-Update-Delete).

В папке Interfaces создайте обобщенный интерфейс, описывающий такой репозиторий:

```

namespace Lab9.DataLayer.Interfaces
{
    public interface IRepository<T>
    {
        IEnumerable<T> GetAll();
        T Get(int id);
        IEnumerable<T> Find(Func<T, bool> predicate);
        void Create(T t);
        void Update(T t);
        void Delete(int id);
    }
}

```

```
}
```

Метод GetAll возвращает список всех объектов типа <T>.

Метод Get ищет объект в базе данных по заданному id.

Метод Find осуществляет поиск объекта, удовлетворяющего условию predicate.

Метод Create добавляет объект типа <T> в базу данных.

Метод Update принимает измененный объект типа <T> и вносит изменения в базу данных.

В папке Repositories создайте классы репозитория для каждой таблицы базы данных.

Пример репозитория для таблицы Group:

```
using System;
using System.Collections.Generic;
using System.Linq;
using Lab9.DataLayer.Interfaces;
using Lab9.DataLayer.Entities;
using Lab9.DataLayer.EFContext;
using System.Data.Entity;

namespace Lab9.DataLayer.Repositories
{
    class GroupsRepository : IRepository<Group>
    {
        CoursesContext context;
        public GroupsRepository(CoursesContext context)
        {
            this.context = context;
        }
        public void Create(Group t)
        {
            context.Groups.Add(t);
        }

        public void Delete(int id)
        {
            var group = context.Groups.Find(id);
            context.Groups.Remove(group);
        }

        public IEnumerable<Group> Find(Func<Group, bool> predicate)
        {
            return context
                .Groups
```

```

        .Include(g =>g.Students)
        .Where(predicate)
        .ToList();
    }

    public Group Get(int id)
    {
        return context.Groups.Find(id);
    }

    public IEnumerable<Group> GetAll()
    {
        return context.Groups.Include(g => g.Students);
    }

    public void Update(Group t)
    {
        context.Entry<Group>(t).State = EntityState.Modified;
    }
}

```

Каждый репозиторий работает с контекстом Entity Framework.

Для того, чтобы этот контекст был один для всех репозиторий, используем шаблон проектирования Unit Of Work.

В папке Interfaces опишите интерфейс IUnitOfWork:

```

namespace Lab9.DataLayer.Interfaces
{
    public interface IUnitOfWork : IDisposable
    {
        IRepository<Group> Groups { get; }
        IRepository<Student> Students { get; }
        void Save();
    }
}

```

Как видим, Unit of Work должен создавать и возвращать репозитории, а также сохранять изменения в базе данных. Внутри класса будет создан один объект контекста для всех репозиторий.

Создайте класс EFUnitOfWork (EF будет означать, что данный класс работает с базой данных через классы Entity Framework) в папке Repositories, например:

```

using System;
using Lab9.DataLayer.Entities;

```



```

using Lab9.DataLayer.Interfaces;
using Lab9.DataLayer.EFContext;

namespace Lab9.DataLayer.Repositories
{
    public class EntityUnitOfWork : IUnitOfWork
    {
        CoursesContext context;
        GroupsRepository groupsRepository;
        StudentRepository studentsRepository;

        public EntityUnitOfWork(string name)
        {
            context = new CoursesContext(name);
        }
        public IRepository<Group> Groups
        {
            get
            {
                if (groupsRepository == null)
                    groupsRepository = new GroupsRepository(context);
                return groupsRepository;
            }
        }

        public IRepository<Student> Students
        {
            get
            {
                if (studentsRepository == null)
                    studentsRepository =
                        new StudentRepository(context);
                return studentsRepository;
            }
        }

        public void Save()
        {
            context.SaveChanges();
        }
        //
        // Реализация интерфейса IDisposable
        // см. https://msdn.microsoft.com/ru-ru/library/system.idisposable\(v=vs.110\).aspx
        //
        private bool disposed = false;
        public virtual void Dispose(bool disposing)
        {
            if (!this.disposed)
            {
                if (disposing)

```

```

        {
            context.Dispose();
        }
        this.disposed = true;
    }
}
public void Dispose()
{
    Dispose(true);
    GC.SuppressFinalize(this);
}
}
}

```

Окончательный вид проекта приведен на Рисунок 4

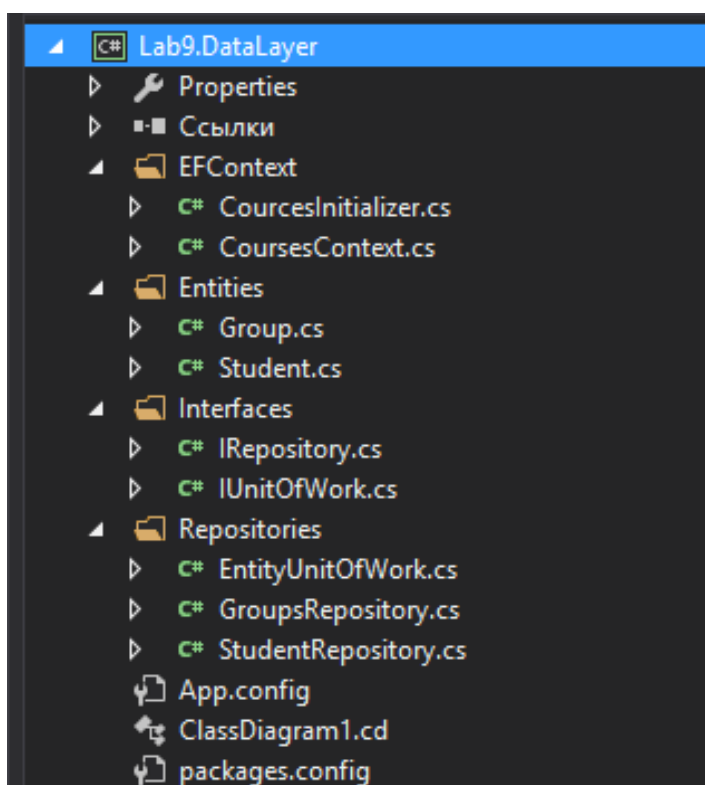


Рисунок 4 Окончательный вид проекта уровня доступа к данным

### 5.3 Создание уровня бизнес-логики

Добавьте в решение новый проект – библиотеку классов. Присвойте проекту имя Lab9.BusinessLayer (см. п.5.1).

Добавьте в созданный проект ссылку на проект Lab9.DataLayer (см. Рисунок 5, Рисунок 6)

Добавьте в созданный проект папки: Interfaces, Models, Services.

В папке Models создайте классы, которые будут использоваться в приложении для отображения/редактирования информации. Эти классы могут полностью дублировать классы Entities, а могут и отличаться. В данном примере для объектов Student не нужны свойства GroupId и Group, но понадобится поле bool HasDiscount для вычисления цены при создании нового студента.

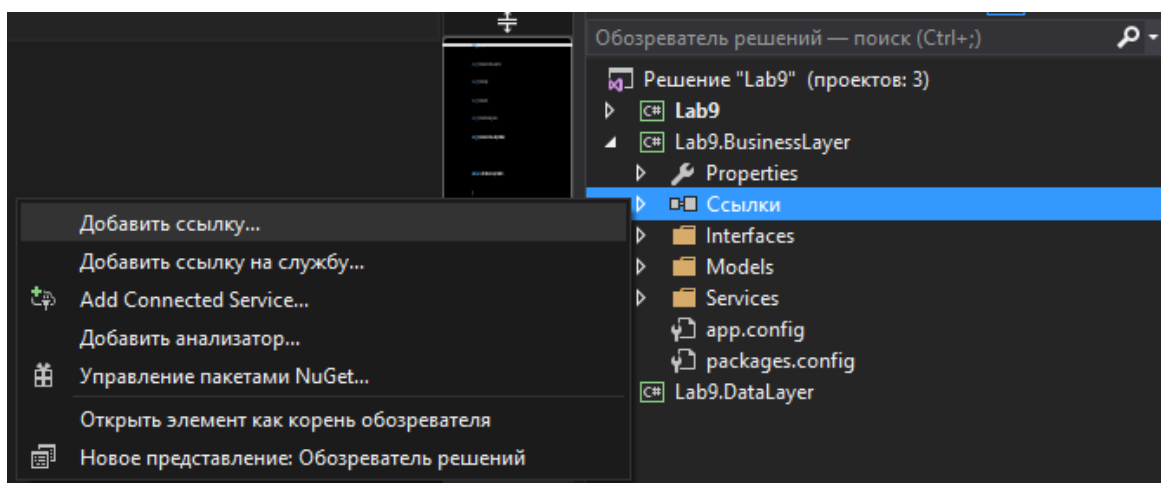


Рисунок 5 Добавление ссылки

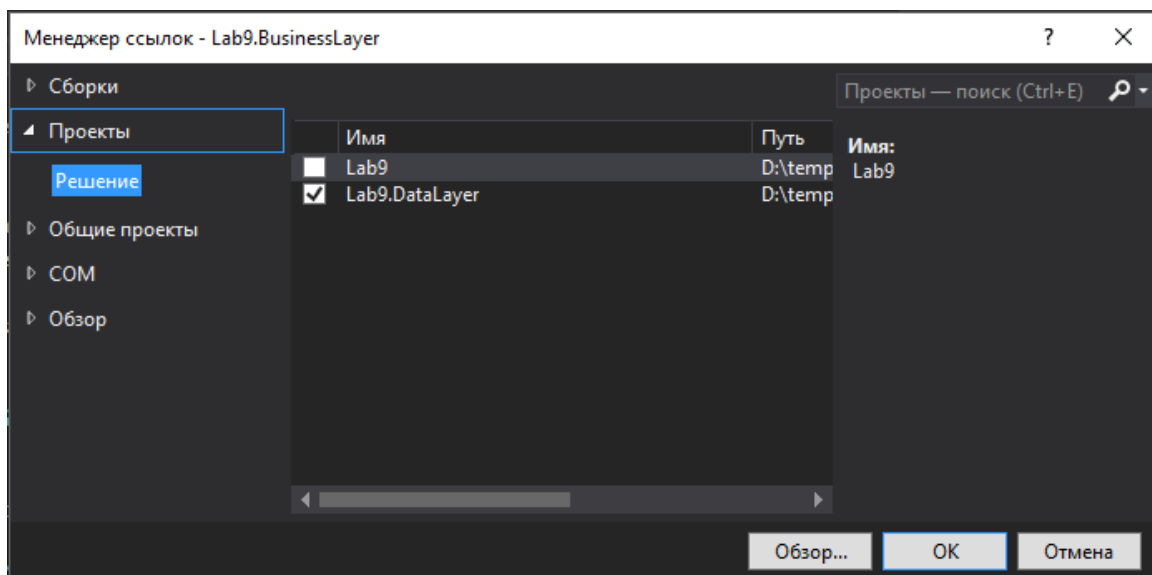


Рисунок 6 Выбор ссылки

В литературе можно найти различные варианты названий для таких классов. Чаще всего используются имена ViewModel (модель представления) или DTO (Data Transfer Object – объект для передачи данных).

Пример реализации классов GroupViewModel и StudentViewModel:

```
namespace Lab9.BusinessLayer.Models
```

```
{  
    public class StudentViewModel  
    {  
        public int StudentId { get; set; }  
        public string FullName { get; set; }  
        public DateTime DateOfBirth { get; set; }  
        public decimal IndividualPrice { get; set; }  
        public string FileName { get; set; }  
        public bool HasDiscount { get; set; }  
    }  
}
```



```
using System.Collections.ObjectModel;
```

```
namespace Lab9.BusinessLayer.Models
```

```
{  
    public class GroupViewModel  
    {  
        public int GroupId { get; set; }  
        public string CourseName { get; set; }  
        public DateTime Commence { get; set; }  
        public decimal BasePrice { get; set; }  
        public ObservableCollection<StudentViewModel> Students  
        { get; set; }  
    }  
}
```



Для реализации бизнес-логики создадим сервисный класс.

Сначала папке Interfaces опишем интерфейс сервиса для работы с группами:

```
using System.Collections.ObjectModel;
```

```
using Lab9.BusinessLayer.Models;
```

```
namespace Lab9.BusinessLayer.Interfaces
```

```
{  
    public interface IGroupService  
    {  
        ObservableCollection<GroupViewModel> GetAll();  
        GroupViewModel Get(int id);  
        void AddStudentToGroup(int droupId, StudentViewModel student);  
        void RemoveStudentFromGroup(int droupId, int studenIdt);  
        void CreateGroup(GroupViewModel group);  
        void DeleteGroup(int groupId);  
        void UpdateGroup(GroupViewModel group);  
    }  
}
```

}

В процессе работы сервису придется преобразовывать объекты Entities из уровня доступа к данным в объекты ViewModel и обратно. Для упрощения преобразований воспользуемся библиотекой AutoMapper. Для этого добавьте в проект библиотеку NuGet AutoMapper.

Познакомиться с основами работы AutoMapper можно здесь:  
<https://github.com/AutoMapper/AutoMapper/wiki/Getting-started>

Реализацию интерфейса IGroupService опишем в папке Services:

```
using System;
using System.Collections.ObjectModel;
using Lab9.BusinessLayer.Interfaces;
using Lab9.DataLayer.Entities;
using Lab9.BusinessLayer.Models;
using Lab9.DataLayer.Interfaces;
using Lab9.DataLayer.Repositories;
using AutoMapper;

namespace Lab9.BusinessLayer.Services
{
    public class GroupService : IGroupService
    {
        IUnitOfWork dataBase;
        public GroupService(string name)
        {
            dataBase = new EntityUnitOfWork(name);
        }

        public void AddStudentToGroup(int droupId, StudentViewModel student)
        {
            var group = dataBase.Groups.Get(droupId);

            // Конфигурирование AutoMapper
            Mapper.Initialize(cfg => cfg.CreateMap<StudentViewModel, Student>());
            // Отображение объекта StudentViewModel на объект Student
            var stud = Mapper.Map<Student>(student);
            // Определение цены для студента
            stud.IndividualPrice = student.HasDiscount == true
                ? group.BasePrice * (decimal)0.8
                : group.BasePrice;
            // Добавить студента
            group.Students.Add(stud);
        }
    }
}
```

```

        // Сохранить изменения
        dataBase.Save();
    }

    public void CreateGroup(GroupViewModel group)
    {
        throw new NotImplementedException();
    }

    public void DeleteGroup(int groupId)
    {
        throw new NotImplementedException();
    }

    public GroupViewModel Get(int id)
    {
        throw new NotImplementedException();
    }

    public ObservableCollection<GroupViewModel> GetAll()
    {
        // Конфигурирование AutoMapper
        Mapper.Initialize(cfg => {
            cfg.CreateMap<Group, GroupViewModel>();
            cfg.CreateMap<Student, StudentViewModel>();
        });
        // Отображение List<Group> на
        ObservableCollection<GroupViewModel>
        var groups =
        Mapper.Map<ObservableCollection<GroupViewModel>>(dataBase.Groups.GetAll());
        return groups;
    }

    public void RemoveStudentFromGroup(int groupId, int studentId)
    {
        throw new NotImplementedException();
    }

    public void UpdateGroup(GroupViewModel group)
    {
        throw new NotImplementedException();
    }
}

```

Окончательный вид проекта бизнес-логики см. Рисунок 7

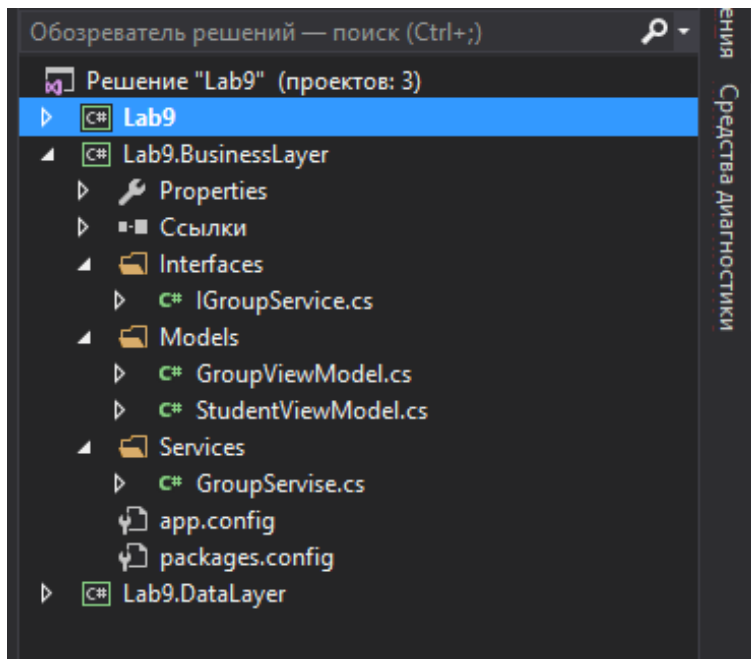


Рисунок 7 Структура проекта уровня бизнес-логики

## 5.4 Главное окно приложения

Добавьте в проект ссылку на библиотеку **Lab9.BusinessLayer**.

Пример разметки главного окна:

```
<Grid Margin="10" >
  <Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
    <RowDefinition/>
    <RowDefinition/>
  </Grid.RowDefinitions>
  <StackPanel>
    <TextBlock>Выберите группу:</TextBlock>
    <ComboBox x:Name="cBoxGroup"
      Margin="0,10,10,10"
      DisplayMemberPath="CourseName"
      ItemsSource="{Binding}"
      SelectedValuePath="GroupId"
      SelectedIndex="0"/>
    <Button Margin="10" Content="Добавить студента"
      Click="Button_Click"/>
  </StackPanel>
  <Grid Grid.Column="1">
    <Grid.RowDefinitions>
      <RowDefinition/>
      <RowDefinition/>
    </Grid.RowDefinitions>
  </Grid>
</Grid>
```

Список групп

```

</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="auto"/>
    <ColumnDefinition Width="*/>
</Grid.ColumnDefinitions>
<TextBlock Margin="10,30,10,10">Начало
занятий:< TextBlock>
    <TextBlock Grid.Column="1"
        Text="{Binding ElementName=cBoxGroup,
            Path=SelectedItem.Commence,
            StringFormat={}{0:dd.MM.yyyy}}"/>
        Margin="0,30,0,0"/>
    <TextBlock Margin="10,0,0,0"
Grid.Row="1">Стоимость:</TextBlock>
    <TextBlock Grid.Row="1" Grid.Column="1"
        Text="{Binding ElementName=cBoxGroup,
            Path=SelectedItem.BasePrice,
            StringFormat={}{0} py6.}"/>
</Grid>
<ScrollView Grid.Row="1" Grid.ColumnSpan="2">
    <ListBox DataContext="{Binding ElementName=cBoxGroup,
        Path=SelectedItem}"
        ItemsSource="{Binding Path=Students}">
        <ListBox.ItemTemplate>
            <DataTemplate>
                <Grid Margin="10">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="auto"/>
                        <ColumnDefinition Width="*/>
                    </Grid.ColumnDefinitions>
                    <Image Width="50"
                        Source="{Binding Path=FileName}"/>
                    <StackPanel Grid.Column="1" >
                        <TextBlock Text="{Binding
                            Path=FullName}" Margin="30,0,0,0"/>
                        <TextBlock Text="{Binding
                            Path=DateOfBirth,
                            StringFormat={}{0:dd.MM.yyyy г.р.}}"
                            Margin="30,0,0,0"/>
                    </StackPanel>
                </Grid>
            </DataTemplate>
        </ListBox.ItemTemplate>
    </ListBox>
</ScrollView>
</Grid>

```

Информация о группе

Список студентов

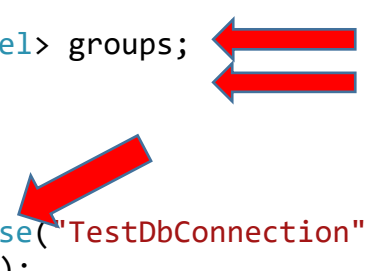


Привязка подробной информации о группе и списка студентов осуществляется к свойству SelectedItem элемента ComboBox.

В коде окна:

```
using Lab9.BusinessLayer.Interfaces;
using Lab9.BusinessLayer.Models;
using Lab9.BusinessLayer.Services;
using System.Collections.ObjectModel;
using Lab9.Dialogs;

namespace Lab9
{
    /// <summary>
    /// Логика взаимодействия для MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        ObservableCollection<GroupViewModel> groups;
        IGroupService groupService;
        public MainWindow()
        {
            InitializeComponent();
            groupService = new GroupService("TestDbConnection");
            groups = groupService.GetAll();
            cBoxGroup.DataContext = groups;
        }
    }
}
```



В файл App.config добавлен раздел:

```
<startup>
  <supportedRuntime version="v4.0"
    sku=".NETFramework,Version=v4.5.2" />
</startup>
<connectionStrings>
  <add name="TestDbConnection" connectionString="Data
Source=(localdb)\mssqllocaldb;Initial Catalog=Lab9Db;Integrated
Security=True" providerName="System.Data.SqlClient" />
</connectionStrings>
```

В папку **bin/bebug** (или **bin/release**) добавьте папку **images**, содержащую файлы изображений.

Можно запустить проект и проверить результат.

В списке студентов отсутствуют изображения. Дело в том, что путь к файлу привязан к свойству FileName. Но свойство FileName – это только имя файла, а не путь к нему. Необходимо создать конвертор значений.

Добавьте в проект папку Infrastructure и создайте в ней класс ImageSourceConverter:

```
using System;
using System.Globalization;
using System.IO;
using System.Windows.Data;

namespace Lab9.Infrastructure
{
    class ImageSourceConverter : IValueConverter
    {
        string imageDirectory = Directory.GetCurrentDirectory();
        string ImageDirectory { get
        {
            return Path.Combine(imageDirectory, "images");
        } }
        public object Convert(object value,
                              Type targetType,
                              object parameter, CultureInfo culture)
        {
            return Path.Combine(ImageDirectory, (string)value);
        }

        public object ConvertBack(object value,
                                   Type targetType,
                                   object parameter, CultureInfo culture)
        {
            throw new NotImplementedException();
        }
    }
}
```

Зарегистрируйте конвертер в качестве ресурса Window:

```
<Window x:Class="Lab9.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
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:local="clr-namespace:Lab9"
xmlns:inf="clr-namespace:Lab9.Infrastructure"
mc:Ignorable="d"
Title="MainWindow" Height="350" Width="525">
<Window.Resources>
    <inf:ImageSourceConverter x:Key="ImageConverter"/>
</Window.Resources>
. . .
```

</Window>

Укажите конвертер в привязке изображения:

```
<Image Width="50" Source="{Binding Path=FileName,  
Converter={StaticResource ImageConverter}}"/>
```

Запустите проект и проверьте результат

Пример реализации добавления студента в список:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    var student = new StudentViewModel();
    student.DateOfBirth = new DateTime(1990, 01, 01);
    var dialog = new EditStudent(student);
    var result = dialog.ShowDialog();
    if (result==true)
    {
        var group = (GroupViewModel)cBoxGroup.SelectedItem;
        group.Students.Add(student);

        groupService.AddStudentToGroup(group.GroupId, student);
        dialog.Close();
    }
}
```

Пример окна EditStudent приведен на Рисунок 8.

Рисунок 8 Окно добавления/редактирования студента

Окончательная структура проекта приведена на Рисунок 9

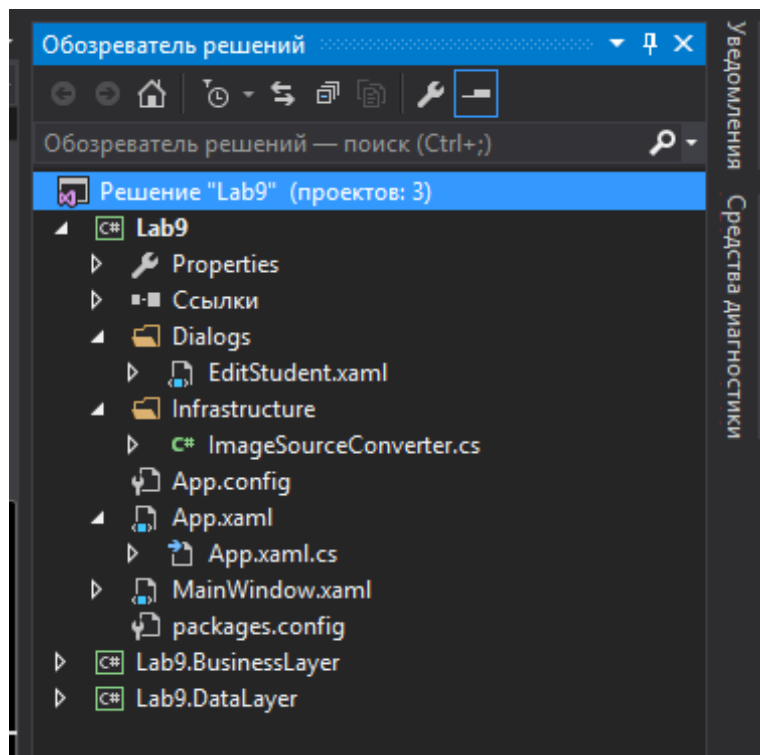


Рисунок 9 Структура проекта Lab9.

Схема полученной многослойной архитектуры – см. Рисунок 10

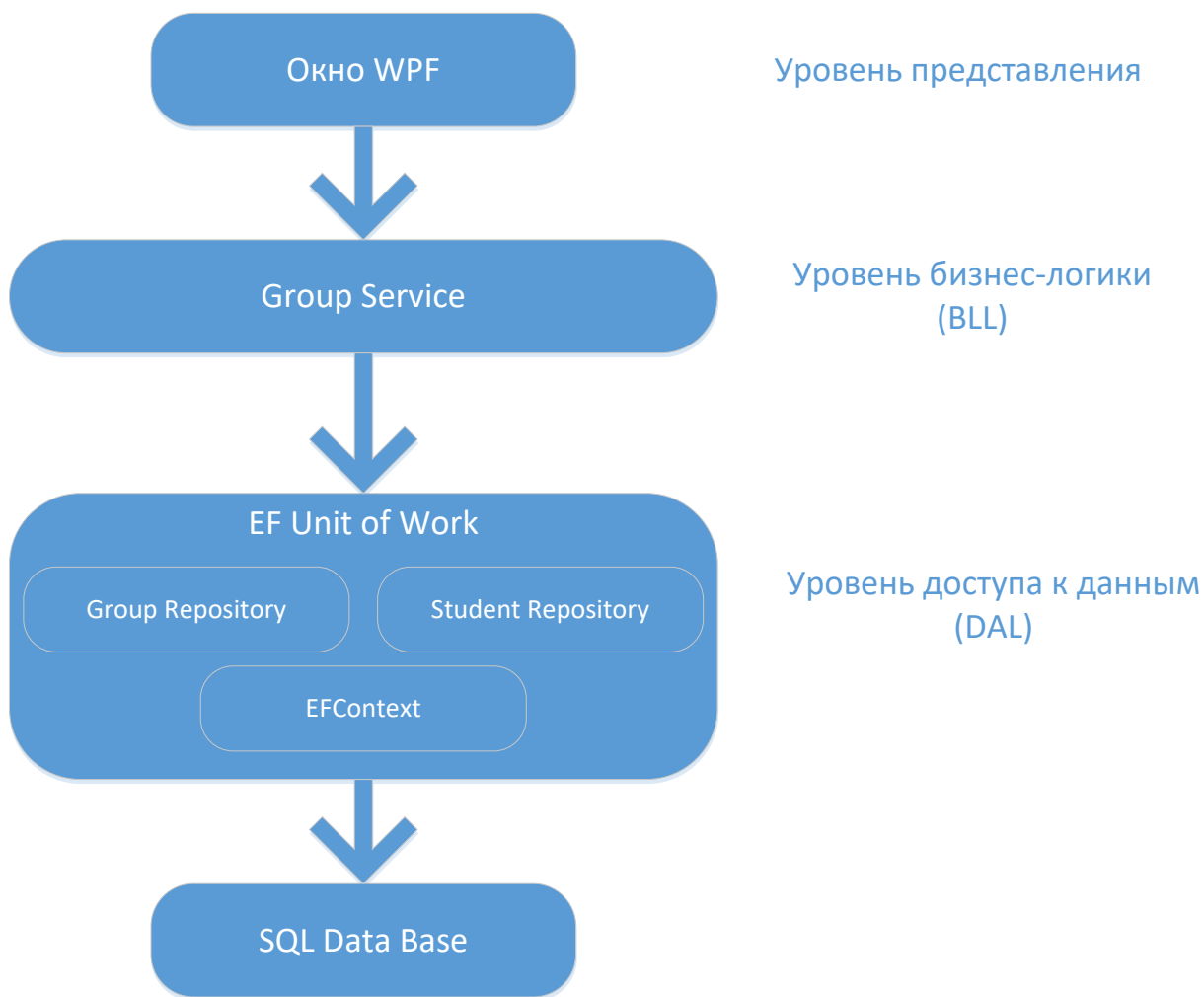


Рисунок 10 Многослойная архитектура проекта