МІНІСТРЕСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРНІГІВСТКИЙ НАЦІОНАЛЬНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

C# ТЕХНОЛОГІЇ ДЛЯ СТВОРЕННЯ WEB-ЗАСТОСУНКІВ

методичні вказівки до лабораторних робіт
з дисципліни
«Java та C# технології прикладного програмування»
для студентів
спеціальності 121 – «Інженерія програмного забезпечення»

Обговорено і рекомендованона
засіданні кафедри
інформаційних технологій та
програмної інженерії
Протокол №4 від 02.12.2019 р.

Чернігів - 2019

C # технології для створення web-застосунків. Методичні вказівки до лабораторних робіт з дисципліни «Java та C # технологій прикладного програмування» для студентів спеціальності 121 – «Інженерія програмного забезпечення» / Укл.: Задорожній А. О., Білоус І. В., Войцеховська М. М. Чернігів: ЧНТУ, 2019 .- 9 1 с.

Укладачі: Задорожний А. О., к.т.н., доцент кафедри ІТ та ПІ
Білоус І. В., к.т.н., доцент кафедри ІТ та ПІ
Войцеховська М. М., асистент кафедри ІТ та ПІ

Відповідальний
за випуск: Скітер Ігор Семенович, к.ф-м.н, доцент кафедри ІТ та ПІ

Рецензент: Литвинов В. В., д.т.н., професор, завідувач кафедри ІТ та ПІ

ЗМІСТ

Список умовних скорочень

ORM	-	Object Relation Mapping
LINQ	-	Language Integrated Query
VS	-	Visual Studio
DAO	-	Data Access Object
HTML	-	Hyper Text Markup Language
XML	-	Extensible Markup Language
SQL	-	Structured Query Language
CRUD	-	Create, Read, Update and Delete
ASP	-	Active Server Pages
MVC	-	Model View Controller

ВСТУП

Об’єктно-орієнтовані мови є ефективним інструментом для створення складних як настільних так і web-орієнтованих застосунків. З часу виникнення перших об’єктно-орієнтованих мов виробився ряд підходів, які значно спрощують створення застосунків. Так, наприклад, однотипні дані прийнято зберігати в масивах або колекціях, для роботи з базами даних прийнято використовувати драйвери баз даних, написані під певну мову програмування. Досвід, накопичений методом спроб і помилок, поступово переріс в технології, що значно спрошують розробку застосунків.

Дані методичні вказівки створені, щоб допомогти отримати практичний досвід по роботі з технологіями взаємодії з базами даних, web-технологіями і технологіями модульного тестування в платформі .NET.

В даних методичних вказівках розглядаються такі технології, як Fluent Hibernate – технологія для об’єктно-реляційного відображення, ASP.NET – технологія створення web-застосунків, Unit Testing Framework – технологія модульного тестування застосунків, а також розглядаються деякі аспекти використання базових механізмів платформи .NET, таких як колекції і рефлексія.

Деякі технології, описані в даних методичних вказівках, використовуються не тільки в платформі .NET, але і платформі Java. Наприклад, технологія Nhibernate використовується в Java під назвою Hibernate, а підходи, які застосовуються в технології ASP.NET дуже схожа на підходи, які застосовуються в Java-технології Java Server Faces. Таким чином, дані методичні вказівки повинні допомогти сформувати загальну картину не тільки для платформи .NET але і для платформи Java.

В деяких лабораторних роботах описані шаблони проектування, які доцільно застосовувати при вирішенні тої чи іншої задачі. Наприклад, в лабораторній роботі з об’єктно-реляційного відображення розглядається архітектурний патерн Data Acces Object, а також патерн Factory. Це повинно допомогти сформувати уявлення про використання архітектурних патернів і патернів проектування при створенні застосунків з використання об’єктно-орієнтованих мов програмування.

ЛАБОРАТОРНА РОБОТА №1. ПРОЕКТУВАННЯ І РОЗРОБКА ПРОШАРКУ ІНТЕГРАЦІЇ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ FLUENT HIBERNATE

1.1 Мета роботи

Вивчити можливості Fluent NHibernate, отримати практичні навички по роботі з NHibernate.

1.2 Теоретичні відомості

1.2.1 Технологія Fluent Hibernate

Для суміщення в одній системі переваг об'єктно-орієнтовного програмування і реляційних баз даних необхідно використовувати ORM (object relation mapping) – об'єктно-реляційне відображення. Об'єктно-реляційне відображення забезпечує можливість роботи з записами таблиці реляційної бази даних як з об'єктами.

Суть об'єктно-реляційного відображення в тому, щоб відділити зберігання даних від роботи з ними, або інакше – відділити бізнес-логіку і логіку зберігання даних. При розробці системи відображення до неї висуваються вимоги наступного плану – необхідно розробити систему таким чином, щоб при описі бізнес компонентів мінімум часу витрачалося на реалізацію необхідного для цієї системи відображення.

Важливою перевагою відображення є можливість розглянути окремі таблиці БД як колекції об'єктів відповідних класів.

В технології .NET використовується два підходи для реалізації відображення – з використанням атрибутів чи з використанням XML-файлів для зовнішнього співставлення.

Суть відображення на основі атрибутів базується на механізмі Reflection і полягає у наступному:

1. Визначаються атрибути, що описують, які таблиці зберігають дані відповідних класів полям відповідних таблиць. При цьому помічаються ключові поля.
2. При описі наступної сутності, що відображається, над класом розміщається атрибут, що вказує, в якій таблиці зберігаються сутності даного типу, а над полями вказуються атрибути, що ставлять їх у відповідність поля таблиці в базі даних.
3. Створюється відображувач (виконавець відображення), робота якого заснована на цих атрибутах. Інформацію про відображення поля цей відображувач отримує виключно з використанням Reflection. Значення полів також обираються і встановлюються через механізм Reflection.

З допомогою XML-файлів метадані для співставлення модна зберігати за межами коду застосування. Такий файл використовується для

співставлення між моделлю даних БД і об'єктною моделлю. Файл зовнішнього співставлення має наступні переваги:

1. Код співставлення можна зберігати за межами коду застосування. Цей підхід зменшує перевантаженість коду застосування.
2. Файл зовнішнього співставлення модна вважати подібним файлу конфігурації. Наприклад, можна змінити поведінку застосування після створення двійкових файлів, просто змінивши файл зовнішнього співставлення.

Популярними ORM для платформи .NET є LINQ to Entities, NHibernate, Fluent NHibernate. В лабораторних роботах даного навчального посібники найбільше уваги приділено ORM Fluent NHibernate – одному з найбільш зручних на даний момент.

Використання Fluent NHibernate дозволяє не задумуватись про структуру бази даних, а працювати з базою як з колекцією пов'язаних один з одним об'єктів. Така можливість забезпечується завдяки механізму відображення об'єкта на таблицю бази даних шляхом правил, що в NHibernate задаються map-класами. В map-класі вказуються правила, з допомогою яких поля об'єктів будуть відображатися на колонки бази даних, а також правила для зв'язку об'єктів один з одним. На етапі запуску застосування map-класи перетворюються в набори запитів, що дозволяють додавати, видаляти і модифікувати дані в таблицях бази даних, а також виконувати вибірку даних. Окрім map-файлів в форматі XML в біль пізніх версіях NHibernate правила відображення можна задавати з допомогою атрибутів, а в Fluent NHibernate з допомогою класів.

1.2.2 Зв'язки між об'єктами в Fluent NHibernate

Fluent NHibernate дозволяє задавати зв'язки один до одного, один до багатьох, багато до багатьох з використанням спеціальних конструкцій HasOne, HasMany, HasManyToMany.

Розглянемо приклад створення цих трьох типів зв'язків на прикладах.

1.2.3 Зв'язок один до одного

Для зв'язку один до одного візьмемо об'єкти «Студент» і «Залікова книжка» - один студент може мати лише одну залікову книжку. Для початку створимо класи студента і залікової книжки:

```
namespace Fluent.Domain
{
    //Домен студента
    public class Student
    {
        public virtual long Id { get; set; }

        public virtual string FirstName { get; set; }

        public virtual string LastName { get; set; }

        public virtual char Sex { get; set; }
```

```
        public virtual int Year { get; set; }

        //Посилання на залікову книжку
        public virtual RecordBook RecordBook { get; set; }

    }
}

namespace Fluent.Domain
{
    //Домен залікової книжки
    public class RecordBook
    {
        public virtual long Id { get; set; }

        public virtual string Number { get; set; }

        //Посилання на студента
        public virtual Student Student { get; set; }
    }
}
```

Тепер необхідно створити класи відображення (map-класси):

```
namespace Fluent.Mappings
{
    //Клас відображення залікової книжки
    public class RecordBookMap : ClassMap<RecordBook>
    {
        public RecordBookMap()
        {
            //Назва таблиці
            Table("RecordBooks");
            //Відображення ідентифікатора
            Id(x => x.Id).GeneratedBy.Native();
            //Відображення звичайного поля
            Map(x => x.Number);
            //Посилання на студента
            References(x => x.Student).Column("StudentId").Cascade.All();
        }
    }
}

namespace Fluent.Mappings
{
    //Клас відображення студента
    public class StudentMap : ClassMap<Student>
    {
        public StudentMap()
        {
            Table("Students");
            Id(x => x.Id).GeneratedBy.Native();
            Map(x => x.FirstName);
            Map(x => x.LastName);
            Map(x => x.Sex);
            Map(x => x.Year);
            // Зв'язок один до одного
            HasOne(x => x.RecordBook).ForeignKey("StudentId").Cascade.All();
        }
    }
}
```

Як видно із прикладу, кожний клас відображення містить конструкцію для зазначення назви таблиці в базі даних (Table), конструкцію для відображення ключового поля на таблицю бази даних (Id), конструкції для відображення полів на таблицю бази даних (Map), конструкції для зв’язування об’єктів.

В даному прикладі для того, щоб зв’язати об’єкти «Студент» і «Залікова книжка» зв’язком один до одного, в класі «Студент» необхідне посилання на об’єкт «Залікова книжка»:

```
public virtual RecordBook RecordBook { get; set; }
```

В класі «Залікова книжка» для того, щоб з об’єкта залікової книжки можна було отримати доступ до студента, необхідне посилання на об’єкт «Студент»:

```
public virtual Student Student { get; set; }
```

В класах відображення також повинні бути відповідні поля. Зі сторони студента:

```
HasOne(x => x.RecordBook).ForeignKey("StudentId").Cascade.All();
```

Зі сторони залікової книжки повинне бути поле:

```
References(x => x.Student).Column("StudentId").Cascade.All();
```

У відповідності до класів відображення в базі даних автоматично створюються таблиці, що представлені на рис 1.1 і 1.2.

id [PK] bigserial	number character var	studentid bigint

Рисунок 1.1. Таблиця залікової книжки

id [PK] bigserial	firstname character var	lastname character var	sex character(1)	year integer

Рисунок 1.2. Таблиця студента

Розглянемо тепер зв'язок один до багатьох.

1.2.4 Зв'язок один до багатьох

Для зв’язку один до багатьох візьмемо об’єкти «Група» і «Студент». В одній групі може бути багато студенті. Один студент може бути членом тільки одної групи.

Створимо класи доменів для групи і студента.

```
namespace Fluent.Domain
```

```
{
    //Домен групи
    public class Group
    {
        private IList<Student> studentList = new List<Student>();

        public virtual long Id { get; set; }

        public virtual string GroupName { get; set; }

        public virtual string CuratorName { get; set; }

        public virtual string HeadmanName { get; set; }

        public virtual IList<Student> StudentList
        {
            get { return studentList; }
            set { studentList = value; }
        }

    }
}

namespace Fluent.Domain
{
    //Домен студента
    public class Student
    {
        public virtual long Id { get; set; }

        public virtual string FirstName { get; set; }

        public virtual string LastName { get; set; }

        public virtual char Sex { get; set; }

        public virtual int Year { get; set; }

        public virtual Group Group { get; set; }

    }
}
```

Зверніть увагу на те, що зі сторони групи міститься посилання на колекцію для зберігання студентів.

Створимо класи відображення для груп і студентів:

```
using System;
using System.Collections.Generic;
using FluentNHibernate.Mapping;
using Fluent.Domain;

namespace Fluent.Mappings
{
    //Клас відобрадення групи
    public class GroupMap:ClassMap<Group>
    {
        public GroupMap()
        {
            Table("Groups");
            Id(x => x.Id).GeneratedBy.Native();
            Map(x => x.GroupName);
            Map(x => x.CuratorName);
```

```
            Map(x => x.HeadmanName);
            //Зв'язок один до багатьох
            HasMany(x => x.StudentList)
                .KeyColumns.Add("GroupId")
                .Inverse()
                .Cascade.All();
        }

    }
}

namespace Fluent.Mappings
{
    //Клас відображення групи
    public class StudentMap : ClassMap<Student>
    {
        public StudentMap()
        {
            Table("Students");
            Id(x => x.Id).GeneratedBy.Native();
            Map(x => x.FirstName);
            Map(x => x.LastName);
            Map(x => x.Sex);
            Map(x => x.Year);
            //Посилання на групу
            References(x => x.Group).Column("GroupId").Cascade.All();
        }
    }
}
```

На відміну від зв’язку один до одного, зі сторони один в класі відображення знаходиться конструкція:

```
HasMany(x => x.StudentList)
    .KeyColumns.Add("GroupId")
    .Inverse()
    .Cascade.All();
```

А зі сторони багатьох, як і при зв'язку один до одного знаходится конструкція:

```
References(x => x.Group).Column("GroupId").Cascade.All();
```

У відповідності з класами відображення в базі даних автоматично створюються таблиці, що зображені на рис. 1.3 і 1.4.

id [PK] bigserial	groupname character var	curatorname character var	headmannam character var

Рисунок 1.3. Таблиця групи

id [PK] bigserial	firstname character var	lastname character var	sex character(1)	year integer	groupid bigint

Рисунок 1.4. Таблиця студента

Розглянемо тепер зв'язок багато до багатьох.

1.2.5 Зв'язок багато до багатьох

Для зв’язку багато до багатьох оберемо об’єкти «Викладач» і «Предмет». Один викладач може вести багато предметів і один предмет може вести декілька викладачів.

Створимо класи домену для викладачів і для предметів:

```
namespace Fluent.Domain
{
    //Клас домена предмета
    public class Subject
    {
        private IList<Teacher> teacherList = new List<Teacher>();

        public virtual long Id { get; set; }

        public virtual string SubjectName { get; set; }

        public virtual int HoursNumber { get; set; }

        public virtual IList<Teacher> TeacherList
        {
            get { return teacherList; }
            set { teacherList = value; }
        }
    }
}

namespace Fluent.Domain
{
    //Клас домена викладача
    public class Teacher
    {
        private IList<Subject> subjectList = new List<Subject>();

        public virtual long Id { get; set; }

        public virtual string FirstName { get; set; }

        public virtual string LastName { get; set; }

        public virtual IList<Subject> SubjectList
        {
            get { return subjectList; }
            set { subjectList = value; }
        }
    }
}
```

Зверніть увагу, що клас предмета містить колекцію для зберігання викладачів, а клас викладача містить колекцію для зберігання предметів.

Створимо класи відображення для предмета і викладача:

```
namespace Fluent.Mappings
{
    //Клас відображення предмета
    public class SubjectMap:ClassMap<Subject>
```

```
    {
        public SubjectMap()
        {
            Table("Subjects");
            Id(x => x.Id).GeneratedBy.Native();
            Map(x => x.SubjectName);
            Map(x => x.HoursNumber);
            // Зв'язок багато до багатьох
            HasManyToMany(x => x.TeacherList)
                .Table("TeacherSubject")
                .ParentKeyColumn("SubjectId")
                .ChildKeyColumn("TeacherId");
        }
    }
}

namespace Fluent.Mappings
{
    //Клас відображення викладача
    public class TeacherMap:ClassMap<Teacher>
    {
        public TeacherMap()
        {
            Table("Teachers");
            Id(x => x.Id).GeneratedBy.Native();
            Map(x => x.FirstName);
            Map(x => x.LastName);
            Зв'язок багато до багатьох
            HasManyToMany(x => x.SubjectList)
                .Table("TeacherSubject")
                .ParentKeyColumn("TeacherId")
                .ChildKeyColumn("SubjectId");
        }
    }
}
```

Зверніть увагу, що в класі відображення з обох сторін вказується конструкція HasManyToMany.

Оскільки зв'язок багато до багатьох в реляційних базах даних здійснюється через окрему таблицю, то в базі даних буде автоматично створюватися три таблиці, що представлені на рис. 1.5, 1.6, 1.7.

id [PK] bigserial	subjectname character var	hoursnumber integer

Рисунок 1.5. Таблиця предмету

id [PK] bigserial	firstname character var	lastname character var

Рисунок 1.6. Таблиця викладача

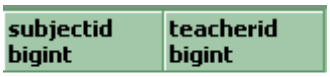

Рисунок 1.7. Проміжна таблиця викладача і предмету

Більш детально зі зв'язками між об'єктами, а також з параметрами зв'язків можна ознайомитись в книгах NHibernate in Action і NHibernate Cookbook.

1.2.6 Способи вибірки об'єктів в Fluent NHibernate

В Fluent NHibernate існує 4 способи вибірки об'єктів:

- Native SQL;
- запити по критерію;
- запити по прикладу;
- мова запитів HQL (Hibernate Query Language).

1.3 Порядок виконання роботи

В даній лабораторній роботі буде розроблено застосування, що буде працювати з даними з бази даних (проводити візуалізацію, додавати, віддаляти, редагувати дані). Доступ до бази даних буде здійснюватись з застосуванням технології ORM (object relation mapping) Fluent NHibernate. В якості СУБД була обрана PostgreSQL. Предметна область буде включати два об'єкти «Група» та «Студент», пов'язані відношенням один до багатьох.

Етап I – Підготовка

1. Встановіть PostgreSQL. Створіть базу даних university.
2. Створіть нове застосування Windows Forms.
3. Створіть каталоги domain, mapping і dao.
4. Додайте до каталогу з проектом бібліотеки для роботи з СУБД PostgreSQL, бібліотеки NHibernate і бібліотеки Fluent NHibernate. Ці бібліотеки можна завантажити за посиланнями:

Драйвер до PostgreSQL	http://www.postgresql.org/
Бібліотеки NHibernate	http://sourceforge.net/projects/nhibernate/
Бібліотеки Fluent NHibernate	http://fluentnhibernate.org/

В даному проекті були використані архіви:

- Npgsql2.0.10-bin-ms.net.zip
- NHibernate-2.1.2.GA-bin.zip
- fluentnhibernate-1.1.zip

Список усіх необхідних бібліотек приведений нижче:

Npgsql.dll
Mono.Security.dll

NHibernate.dll
Antlr3.Runtime.dll
Iesi.Collections.dll
log4net.dll
Castle.DynamicProxy2.dll
Castle.Core.dll
NHibernate.ByteCode.Castle.dll
FluentNHibernate.dll

Підключіть бібліотеки Npgsql.dll, NHibernate.dll, FluentNHibernate.dll і NHibernate.ByteCode.Castle до проекту (натисніть правою кнопкою на «Add reference», перейдіть на закладку Browse та виберіть необхідні файли).

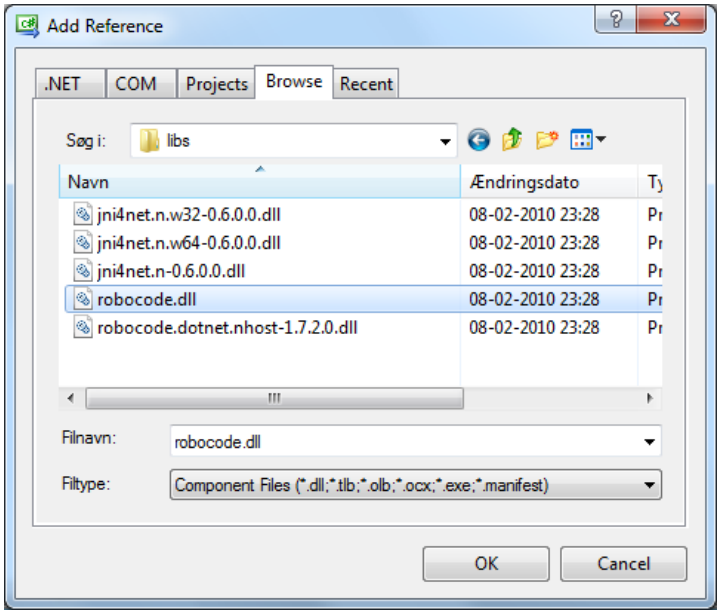

Рисунок 1.8. Вікно додання бібліотек до проекту

Тепер все готово для створення прошарку доступу до даних DAO. В лабораторній роботі можна використовувати NHibernate и Fluent Hibernate більш пізніх версій, наприклад NHibernate 3.

Етап II – Створення прошарку доступу до даних (DAO)

Будьте пильні при розробці прошарку доступу до даних, оскільки він буде використаний в наступних лабораторних роботах.

5. В каталозі domain створіть три класи-домени EntityBase, Group і Student. Тексти класів приведені нижче:

```
namespace lab4.domain
{
    //Клас базової сутності
    public abstract class  EntityBase
    {
        public virtual long Id { get; set; }
    }
}
```

```
using System.Collections.Generic;

namespace lab4.domain
{
    //Сутність группы
    public class Group:EntityBase
    {
        private IList<Student> studentList = new List<Student>();

        public virtual string GroupName { get; set; }

        public virtual string CuratorName { get; set; }

        public virtual string HeadmanName { get; set; }

        public virtual IList<Student> StudentList
        {
            get { return studentList; }
            set { studentList = value; }
        }
    }
}

namespace lab4.domain
{
    //Сутність студента
    public class Student:EntityBase
    {
        public virtual string FirstName { get; set; }

        public virtual string LastName { get; set; }

        public virtual char Sex { get; set; }

        public virtual int Year { get; set; }

        public virtual Group Group { get; set; }
    }
}
```

6. Створіть в каталозі dao інтерфейс IGenericDAO, в якому будуть описані всі необхідні для прошарку доступу до даних методи. Набір методів визначається на етапі проектування в залежності від потреб системи. Інтерфейс IGenericDAO приведений нижче:

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

namespace lab4.dao
{
    public interface IGenericDAO<T>
    {
        void SaveOrUpdate(T item);

        T GetById(long id);

        List<T> GetAll();
```

```
        void Delete(T item);
    }
}
```

7. Окрім стандартного набору методів, dao кожного об’єкта може містити свій власний додатковий список методів. Так, наприклад для об’єкта-групи таким методом буде getAllStudentOfGroup (отримати список студентів заданої групи). Тому для кожного об’єкта предметної області необхідно створити власні інтерфейси, що будуть розширювати IGenericDAO. Нижче приведені інтерфейси IGroupDAO та IStudentDAO:

```
using lab4.domain;
using System.Collections.Generic;

namespace lab4.dao
{
    public interface IGroupDAO:IGenericDAO<Group>
    {
        Group getGroupByName(string groupName);

        IList<Student> getAllStudentOfGroup(string groupName);

        void delGroupByName(string groupName);
    }
}
```

```
using lab4.domain;

namespace lab4.dao
{
    public interface IStudentDAO:IGenericDAO<Student>
    {
        Student getStudentByGroupFirstNameAndLastName(
            string groupName, string firstName, string LastName);

    }
}
```

8. Створіть в каталозі dao абстрактну фабрику, що буде повертати DAO для кожної сутності.

```
namespace lab4.dao
{
    abstract public  class DAOFactory
    {
        public abstract IStudentDAO getStudentDAO();

        public abstract IGroupDAO getGroupDAO();

    }
}
```

9. Тепер необхідно створити реалізацію інтерфейсів IGenericDAO, IGroupDAO, IStudentDAO і абстрактної фабрики для Fluent NHibernate. Нижче приведена реалізація IGenericDAO:

```
using System;
using System.Collections.Generic;
```

```
using NHibernate;

namespace lab4.dao
{
    public class GenericDAO<T>:IGenericDAO<T>
    {
        protected ISession session;

        public GenericDAO() { }

        public GenericDAO(ISession session)
        {
            this.session = session;
        }

        public void SaveOrUpdate(T item)
        {
            ITransaction transaction = session.BeginTransaction();
            session.SaveOrUpdate(item);
            transaction.Commit();
        }

        public T GetById(long id)
        {
            return session.Get<T>(id);
        }

        public List<T> GetAll()
        {
            return new List<T>(session.CreateCriteria(typeof(T)).List<T>());
        }

        public void Delete(T item)
        {
            ITransaction transaction = session.BeginTransaction();
            session.Delete(item);
            transaction.Commit();
        }
    }
}
```

10. Створіть в каталозі dao реалізацію інтерфейсів IGroupDAO, IStudentDAO:

```
using lab4.domain;
using NHibernate;
using System.Collections.Generic;
using NHibernate.Criterion;

namespace lab4.dao
{
    public class GroupDAO:GenericDAO<Group>, IGroupDAO
    {
        public GroupDAO(ISession session) : base(session) { }

        public Group getGroupByName(string groupName)
        {
            Group group = new Group();
            group.GroupName = groupName;
            ICriteria criteria = session.CreateCriteria(typeof(Group))
                .Add(Example.Create(group));
```

```
            IList<Group> list = criteria.List<Group>();
            group = list[0];
            return group;
        }

        public IList<Student> getAllStudentOfGroup(string groupName)
        {
            var list = session.CreateSQLQuery(
               "SELECT Students.* FROM Students JOIN Groups" +
               " ON Students.GroupId = Groups.Id" +
               " WHERE Groups.GroupName='" + groupName + "'")
               .AddEntity("Student", typeof(Student))
               .List<Student>();
            return list;
        }

        public void delGroupByName(string groupName)
        {
            Group group = getGroupByName(groupName);
            Delete(group);
        }
    }
}

using lab4.domain;
using NHibernate;

namespace lab4.dao
{
    public class StudentDAO:GenericDAO<Student>, IStudentDAO
    {

        public StudentDAO(ISession session): base(session) {  }

        public Student getStudentByGroupFirstNameAndLastName(
           string groupName, string firstName, string lastName)
        {
            var list = session.CreateSQLQuery(
               "SELECT Students.* FROM Students JOIN Groups" +
               " ON Students.GroupId = Groups.Id" +
               " WHERE Groups.GroupName='" + groupName + "'" +
               " and Students.FirstName='" + firstName + "'" +
               " and Students.LastName='" + lastName  + "'")
               .AddEntity("Student", typeof(Student))
               .List<Student>();
            Student student = list[0];
            return student;
        }
    }
}
```

11. Створіть реалізацію абстрактної фабрики NHibernateDAOFactory:

```
using System;
using FluentNHibernate;
using NHibernate;

namespace lab4.dao
{
    public class NHibernateDAOFactory:DAOFactory
    {
        /** NHibernate sessionFactory */
```

```
        protected ISession session = null;

        public NHibernateDAOFactory(ISession session)
        {
            this.session = session;
        }

        public override IStudentDAO getStudentDAO()
        {
            return new StudentDAO(session);
        }

        public override IGroupDAO getGroupDAO()
        {
            return new GroupDAO(session);
        }
    }
}
```

12. Створимо класи для відображення об’єктів предметної області на базу даних. Відповідно до mapping-класів в базі даних будуть створюватися таблиці і зв’язки між ними. При використанні NHibernate замість Fluent NHibernate замість класів використовуються XML-файли. Mapping-класи для об’єктів «Група» та «Студент» приведені нижче:

```
using FluentNHibernate.Mapping;
using lab4.domain;

namespace lab4.mapping
{
    public class GroupMap:ClassMap<Group>
    {
        public GroupMap()
        {
            Table("Groups");
            Id(x => x.Id).GeneratedBy.Native();
            Map(x => x.GroupName);
            Map(x => x.CuratorName);
            Map(x => x.HeadmanName);
            HasMany(x => x.StudentList)
                .Inverse()
                .Cascade.All()
                .KeyColumn("GroupId");
        }
    }
}

using FluentNHibernate.Mapping;
using lab4.domain;

namespace lab4.mapping
{
    public class StudentMap:ClassMap<Student>
    {
        public StudentMap()
        {
            Table("Students");
            Id(x => x.Id).GeneratedBy.Native();
            Map(x => x.FirstName);
            Map(x => x.LastName);
            Map(x => x.Sex);
```

```
            Map(x => x.Year);
            References(x => x.Group, "GroupId");
        }
    }
}
```

прошарку доступу до даних реалізовано.

Етап III – Реалізація прошарку візуалізації

Прошарок візуалізації в лабораторній роботі буде виконано з використанням Windows Forms [1, 2] (товстий клієнт). В наступній лабораторній роботі прошарок візуалізації буде реалізовано з використанням технології ASP.NET [3, 4] як тонкий клієнт.

13. Створіть три додаткові форми. Одна буде використовуватись для вводу налаштувань з'єднання з базою даних, друга - для додання і заміни групи, третя – для додання і заміни студента.

14. На головній формі розмістіть компонент MenuStrip, додайте до нього пункт меню «Data source», а до пункту «Data source» додайте пункти Connect та Disconnect. Створіть обробники натиснення на ці два пункти меню.

15. Встановіть на форму компонент SplitConteiner для того, щоб розділити робочу область форми на дві частини.

16. Встановіть в ліву і праву части компонент GoupBox. Встановіть у компонента GoupBox ліворуч властивість Text в значення «Groups», а у компонента праворуч властивість Text в значення – «Students», встановіть у обох GoupBox властивості Dock в значення Fill.

17. Розмістіть на компонентах GoupBox ліворуч і праворуч компоненти DataGridView. Встановіть в них властивість Dock в значення Fill. Компонент DataGridView ліворуч буде використовуватись для візуалізації списку груп, а компонент праворуч для візуалізації списку студентів в групі. Налаштуйте кожний із DataGridView наступним чином: ReadOnly – True, MultiSelect – False, RowHeadersVisible – False, SelectionMode – FullRowSelect, AllowUserToAddRows - False.

18. Додайте в перший DataGridView колонки «Group name», «CuratorcName», «Headman Name», а в інший додайте колонки «First Name», «Last name», «Sex», «Year». Додати колонки модна з використанням властивості Columns. В конструкторі колонок необхідно уважно слідкувати за властивістю Name колонки, через які поті відбувається звернення до колонки із програми.

19. Додайте два компоненти ContextMenuStrip, з'єднайте одне контекстне меню з компонентом DataGridView ліворуч, а інше з компонентом DataGridView праворуч з використанням властивості ContextMenuStrip. Додайте в кожне з меню три пункти «Add», «Delete» та «Change». До кожного пункту меню додайте обробник на натиснення.

20. Тепер необхідно реалізувати функцію підключення застосування до бази даних. Пов'язувати застосування з базою даних буде сесія

NHibernate. Розмістіть на другій формі (Form2) п’ять компонентів TextBox, кожний TextBox можна підписати з використанням компоненту Label. Перший TextBox буде використовуватись для вводу імені сервера БД, другий – для вводу порту, третій – для вводу імені бази даних, четвертий – для вводу імені користувача БД, а п’ятий для вводу пароля. На формі також необхідно розмістити кнопку і назвати її, наприклад, «Connect». Результат проектування форми підключення до бази представлений на рис. 1.9.

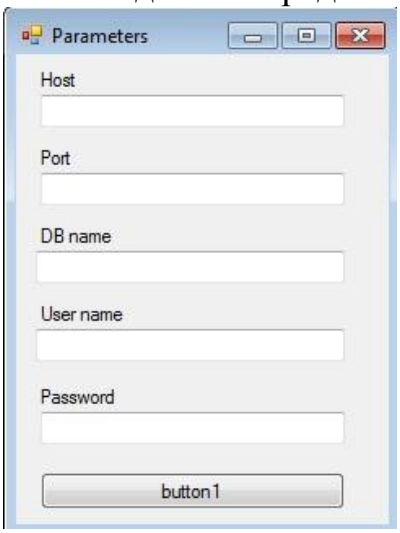

Рисунок 1.9. Форма підключення до бази

21. Кожна форма повинна мати доступ до батьківської форми (Form1), а також до сесії (ISession). Відкрийте форму Form2 з використанням редактору CSharp. Додайте над конструктором форми поля:

```
private ISessionFactory factory;
//Ссылка на сессию
private ISession session;
//Ссылка на родительскую форму
private Form1 perent;
```

22. Додайте метод для встановлення поля perent:

```
public void setPerent(Form1 perent)
{
      this.perent = perent;
}
```

23. Додайте методи для відкриття сесії:

```
//Метод відкриття сесії
private ISession openSession(String host, int port, String database,
     String user, String passwd)
{
     ISession session = null;
     //Получение ссылки на текущую сборку
     Assembly mappingsAssemly = Assembly.GetExecutingAssembly();
     if (factory == null)
     {
          //Конфигурирование фабрики сессий
```

```
            factory = Fluently.Configure()
               .Database(PostgreSQLConfiguration
               .PostgreSQL82.ConnectionString(c => c
               .Host(host)
               .Port(5432)
               .Database(database)
               .Username(user)
               .Password(passwd)))
               .Mappings(m => m.FluentMappings
               .AddFromAssembly(mappingsAssemly))
               .ExposeConfiguration(BuildSchema)
               .BuildSessionFactory();
      }
      //Открытие сессии
      session = factory.OpenSession();
      return session;
}

//Метод для автоматичного створення таблиці в базі даних
private static void BuildSchema(Configuration config)
{
      new SchemaExport(config).Create(false, true);
}
```

24. Додайте обробник натиснення на кнопку «Connect».

```
private void button1_Click(object sender, EventArgs e)
{
      session = openSession(this.textBox1.Text,
            Convert.ToInt32(this.textBox2.Text),
            this.textBox3.Text, this.textBox4.Text, this.textBox5.Text);
      //Скрытие формы
      this.Visible = false;
      //Передача открытой сессии открытой форме
      perent.setSession(session);
      //Вывод в таблицу всех групп
      perent.fillDataGridView1();
}
```

25. Тепер необхідно додати функції по підключенню і відключенню застосування від бази даних. Відкрийте Form1 в режимі редактору CSharp. Додайте над конструктором форми поля:

```
private ISession session;
private Form2 form2 = null;
private Form3 form3 = null;
private Form4 form4 = null;
```

Поля form3, і form4 будуть потрібні для форм додання і редагування груп і студентів. Також необхідно додати метод setSession(), щоб форма встановлення з'єднання могла передати сесію головній формі.

```
public void setSession(ISession session)
{
      this.session = session;
}
```

Для зручності роботи з формою додайте метод по створенню і ініціалізації форми:

```
private Form2 getForm2()
{
     if (form2 == null)
     {
          form2 = new Form2();
     }
     form2.setPerent(this);
     return form2;
}
```

26. Заповніть обробник натиснення на пункти меню «Connect» та «Disconnect».

```
private void присоединитьсяКБазеToolStripMenuItem_Click(
     object sender, EventArgs e)
{
     getForm2().Visible = true;
}

private void отключитьсяОтБазыToolStripMenuItem_Click(
     object sender, EventArgs e)
{
     session.Close();
     dataGridView1.Rows.Clear();
     dataGridView2.Rows.Clear();
}
```

27. Додайте методи по заповненню компонентів DataGridView даними про групи і студентів.

```
//Метод по заполнению dataGridView1
public void fillDataGridView1()
{
     dataGridView1.Rows.Clear();
     //Создание экземпляра фабрики NHibernate
     DAOFactory dao = new NHibernateDAOFactory(session);
     //Получение dao группы
     IGroupDAO groupDAO = dao.getGroupDAO();
     //Получение коллекции групп с базы
     List<Group> groupList = groupDAO.GetAll();
     //Вывод всех групп в таблицу
     foreach (Group g in groupList)
     {
          DataGridViewRow row = new DataGridViewRow();
          DataGridViewTextBoxCell cell1 = new DataGridViewTextBoxCell();
          DataGridViewTextBoxCell cell2 = new DataGridViewTextBoxCell();
          DataGridViewTextBoxCell cell3 = new DataGridViewTextBoxCell();
          cell1.ValueType = typeof(string);
          cell1.Value = g.GroupName;
          cell2.ValueType = typeof(string);
          cell2.Value = g.CuratorName;
          cell3.ValueType = typeof(string);
          cell3.Value = g.HeadmanName;
          row.Cells.Add(cell1);
          row.Cells.Add(cell2);
          row.Cells.Add(cell3);
          dataGridView1.Rows.Add(row);
     }
}
```

```
//Вывод всех студентов заданной группы в dataGridView2
public void fillDataGridView2(string key)
{
      dataGridView2.Rows.Clear();
      //Создание фабрики NHibernate
      DAOFactory dao = new NHibernateDAOFactory(session);
      //Получение dao группы
      IGroupDAO groupDAO = dao.getGroupDAO();
      //Получение студентов заданной группы
      IList<Student> studentList = groupDAO.getAllStudentOfGroup(key);
      foreach (Student s in studentList)
      {
            DataGridViewRow row = new DataGridViewRow();
            DataGridViewTextBoxCell cell1 = new DataGridViewTextBoxCell();
            DataGridViewTextBoxCell cell2 = new DataGridViewTextBoxCell();
            DataGridViewTextBoxCell cell3 = new DataGridViewTextBoxCell();
            DataGridViewTextBoxCell cell4 = new DataGridViewTextBoxCell();
            cell1.ValueType = typeof(string);
            cell1.Value = s.FirstName;
            System.Console.WriteLine(s.FirstName + "\n");
            cell2.ValueType = typeof(string);
            cell2.Value = s.LastName;
            cell3.ValueType = typeof(string);
            cell3.Value = s.Sex;
            cell4.ValueType = typeof(string);
            cell4.Value = s.Year;
            row.Cells.Add(cell1);
            row.Cells.Add(cell2);
            row.Cells.Add(cell3);
            row.Cells.Add(cell4);
            dataGridView2.Rows.Add(row);
      }
}
```

28. Розпочнемо реалізацію функцій додання, видалення і редагування групи. Відкрийте Form3 в режимі конструктора, додайте три компоненти TextBox. Перше поле вводу буде використовуватись для вводу назві груп, друге – для вводу ПІБ куратора, третє – вводу ПІБ старости. Поля вводу можна підписати з використанням компоненту Label. Додайте дві кнопки «Add» і «Change». Результат проектування форми групи представлений на рис. 1.10.

Рисунок 1.10. Форма додання і заміни групи

29. Відкрийте Form3 в режимі редактора CSharp. Додайте над конструктором класу поля:

```
private ISession session;
private Form1 perent;
private string key;
```

Перше поле зберігає в собі посилання на сесію, друге поле – посилання на головну форму, а третє – ключ, котрий нам буде необхідний при заміні групи. Для встановлення значень цих полів додайте методи:

```
public void setPerent(Form1 perent)
{
   this.perent = perent;
}

public void setSession(ISession session)
{
     this.session = session;
}

public void setKey(string key)
{
     this.key = key;
}
```

30. Для того, щоб отримати доступ до полів вводу і кнопок з головної форми, додайте наступні set-методи.

```
public void setTextBox1Text(string text)
{
     this.textBox1.Text = text;
}

public void setTextBox2Text(string text)
{
     this.textBox2.Text = text;
}

public void setTextBox3Text(string text)
{
     this.textBox3.Text = text;
}

public void setButton1Visible(bool visible)
{
     this.button1.Visible = visible;
}

public void setButton2Visible(bool visible)
{
     this.button2.Visible = visible;
}
```

31. Додайте обробники натиснення на кнопки «Add» та «Change»:

```
//Обробник на Add
private void button1_Click(object sender, EventArgs e)
{
```

```
        DAOFactory dao = new NHibernateDAOFactory(session);
        IGroupDAO groupDAO = dao.getGroupDAO();
        Group group = new Group();
        group.GroupName = textBox1.Text;
        group.HeadmanName = textBox2.Text;
        group.CuratorName = textBox3.Text;
        groupDAO.SaveOrUpdate(group);
        perent.fillDataGridView1();
        this.Visible = false;
}

//Обробник на Change
private void button2_Click(object sender, EventArgs e)
{
        DAOFactory dao = new NHibernateDAOFactory(session);
        IGroupDAO groupDAO = dao.getGroupDAO();
        Group group = groupDAO.getGroupByName(key);
        group.GroupName = textBox1.Text;
        group.CuratorName = textBox2.Text;
        group.HeadmanName = textBox3.Text;
        groupDAO.SaveOrUpdate(group);
        this.Visible = false;
        perent.fillDataGridView1();
}
```

32. Відкрийте Form1 в редакторі коду CSharp. Для зручності роботи з Form3 додайте метод:

```
private Form3 getForm3()
{
        if (form3 == null)
        {
                form3 = new Form3();
        }
        form3.setSession(session);
        form3.setPerent(this);
        return form3;
}
```

33. Додайте обробники «Add», «Delete», «Change» для контекстного меню групи:

```
//Натиснення на "Add" в контекстному меню групи
private void добавитьToolStripMenuItem_Click(object sender, EventArgs e)
{
        getForm3().Visible = true;
        getForm3().setTextBox1Text("");
        getForm3().setTextBox2Text("");
        getForm3().setTextBox3Text("");
        getForm3().setButton1Visible(true);
        getForm3().setButton2Visible(false);
}

//Натиснення на "Delete" в контекстному меню групи
private void удалитьToolStripMenuItem_Click(object sender, EventArgs e)
{
        DAOFactory dao = new NHibernateDAOFactory(session);
        IGroupDAO groupDAO = dao.getGroupDAO();
        int selectedRow = dataGridView1.SelectedCells[0].RowIndex;
        string key = (string)dataGridView1.Rows[selectedRow].Cells[0].Value;
        DialogResult dr = MessageBox.Show("Delete group?", "",
```

```
                        MessageBoxButtons.YesNo);
      if (dr == DialogResult.Yes)
      {
            try
            {
                  groupDAO.delGroupByName(key);
                  this.fillDataGridView1();
            }
            catch (Exception)
            {
            }
      }
}

//Натиснення на "Change" в контекстному меню групи
private void изменитьToolStripMenuItem_Click(object sender, EventArgs e)
{
      int selectedRow = dataGridView1.SelectedCells[0].RowIndex;
      string key = (string)dataGridView1.Rows[selectedRow].Cells[0].Value;
      DAOFactory dao = new NHibernateDAOFactory(session);
      IGroupDAO groupDAO = dao.getGroupDAO();
      Group group = groupDAO.getGroupByName(key);
      getForm3().Visible = true;
      getForm3().setKey(key);
      getForm3().setTextBox1Text(group.GroupName);
      getForm3().setTextBox2Text(group.CuratorName);
      getForm3().setTextBox3Text(group.HeadmanName);
      getForm3().setButton1Visible(false);
      getForm3().setButton2Visible(true);
}
```

34. Для того, щоб при натисненні на рядок DataGridView для групи в DataGridView для студентів виводився список студентів вибраної групи, додайте обробник натиснення на рядок DataGridView:

```
//Обработчик нажатия на строчку DataGridView группы
private void dataGridView1_CellClick(
      object sender, DataGridViewCellEventArgs e)
{
      int selectedRow = dataGridView1.SelectedCells[0].RowIndex;
      string key = (string)dataGridView1.Rows[selectedRow].Cells[0].Value;
      fillDataGridView2(key);
}
```

35. Розпочнемо реалізацію функції додання, видалення і заміни студентів. Відкрийте Form4 в режимі конструктора. Додайте 4 компоненти TextBox. Перше поле використовується для вводу імені студенту, друге – для вводу прізвища студента, третє - для вводу статі студента, четверте – для вводу року народження. Кожне поле вводу можна підписати з використанням компонента Label. Додайте на форму кнопки «Add» і «Change». Результат проектування форми студента представлена на рисунку 1.11.

Рисунок 1.11. Форма добавления и замены студента

36. Відкрийте Form4 в режимі редактора CSharp. Додайте над конструктором класу форми поля:

```
private ISession session;
private Form1 perent;
private string key1;
private string key2;
private string key3;
```

Поле key1 містить назву групи, поле key2 використовується для зберігання імені студента, поле key3 містить прізвище студента.

37. Додайте set-методи для встановлення вищеописаних змінних.

```
public void setSession(ISession session)
{
     this.session = session;
}

public void setPerent(Form1 perent)
{
     this.perent = perent;
}

public void setKey1(string key1)
{
     this.key1 = key1;
}

public void setKey2(string key2)
{
     this.key2 = key2;
}

public void setKey3(string key3)
{
     this.key3 = key3;
}
```

38. Для доступу до полів вводу і кнопок форми додайте наступні set-методи:

```
public void setTextBox1Text(string text)
{
     this.textBox1.Text = text;
}

public void setTextBox2Text(string text)
{
     this.textBox2.Text = text;
}

public void setTextBox3Text(string text)
{
     this.textBox3.Text = text;
}

public void setTextBox4Text(string text)
{
     this.textBox4.Text = text;
}

public void setButton1Visible(bool visible)
{
     this.button1.Visible = visible;
}

public void setButton2Visible(bool visible)
{
     this.button2.Visible = visible;
}
```

39. Створіть обробники натиснення на кнопки «Add» і «Change».

```
//Обробник натиснення на "Add"
private void button1_Click(object sender, EventArgs e)
{
     DAOFactory dao = new NHibernateDAOFactory(session);
     IGroupDAO groupDAO = dao.getGroupDAO();
     Group group = groupDAO.getGroupByName(key1);
     Student student = new Student();
     student.FirstName = textBox1.Text;
     student.LastName = textBox2.Text;
     student.Sex = textBox3.Text[0];
     student.Year = Int32.Parse(textBox4.Text);
     group.StudentList.Add(student);
     student.Group = group;
     groupDAO.SaveOrUpdate(group);
     perent.fillDataGridView2(key1);
     this.Visible = false;
}

//Обробник натиснення на "Change"
private void button2_Click(object sender, EventArgs e)
{
     DAOFactory dao = new NHibernateDAOFactory(session);
     IStudentDAO studentDAO = dao.getStudentDAO();
     Student student = studentDAO
          .getStudentByGroupFirstNameAndLastName(key1, key2, key3);
     student.FirstName = textBox1.Text;
```

```
        student.LastName = textBox2.Text;
        student.Sex = textBox3.Text[0];
        student.Year = Int32.Parse(textBox4.Text);
        studentDAO.SaveOrUpdate(student);
        this.Visible = false;
        perent.fillDataGridView2(key1);
}
```

40. Відкрийте Form1 з використанням редактора коду CSharp. Для зручності роботи з Form4 додайте метод:

```
private Form4 getForm4()
{
        if (form4 == null)
        {
                form4 = new Form4();
        }
        form4.setSession(session);
        form4.setPerent(this);
        return form4;
}
```

41. Заповніть обробник натиснення на «Add», «Delete», «Change» контекстного меню студена:

```
//Натиснення на "Add" в контекстному меню студента
private void добавитьToolStripMenuItem1_Click(object sender, EventArgs e)
{
        int selectedRow = dataGridView1.SelectedCells[0].RowIndex;
        string key = (string)dataGridView1.Rows[selectedRow].Cells[0].Value;
        getForm4().Visible = true;
        getForm4().setButton1Visible(true);
        getForm4().setButton2Visible(false);
        getForm4().setTextBox1Text("");
        getForm4().setTextBox2Text("");
        getForm4().setTextBox3Text("");
        getForm4().setTextBox4Text("");
        getForm4().setKey1(key);
}

//Натиснення на "Delete" в контекстному меню студента
private void удалитьToolStripMenuItem1_Click(object sender, EventArgs e)
{
        int selectedRow = dataGridView1.SelectedCells[0].RowIndex;
        string key1 = (string)dataGridView1.Rows[selectedRow].Cells[0].Value;
        selectedRow = dataGridView2.SelectedCells[0].RowIndex;
        string key2 = (string)dataGridView2.Rows[selectedRow].Cells[0].Value;
        selectedRow = dataGridView2.SelectedCells[0].RowIndex;
        string key3 = (string)dataGridView2.Rows[selectedRow].Cells[1].Value;
        DAOFactory dao = new NHibernateDAOFactory(session);
        IStudentDAO studentDAO = dao.getStudentDAO();
        Student student = studentDAO
                .getStudentByGroupFirstNameAndLastName(key1, key2, key3);
        student.Group.StudentList.Remove(student);
        studentDAO.Delete(student);
        fillDataGridView2(key1);
}

//Натитиснення на "Change" в контекстному меню студента
private void изменитьToolStripMenuItem1_Click(object sender, EventArgs e)
{
        int selectedRow = dataGridView1.SelectedCells[0].RowIndex;
```

```
        string key1 = (string)dataGridView1.Rows[selectedRow].Cells[0].Value;
        selectedRow = dataGridView2.SelectedCells[0].RowIndex;
        string key2 = (string)dataGridView2.Rows[selectedRow].Cells[0].Value;
        selectedRow = dataGridView2.SelectedCells[0].RowIndex;
        string key3 = (string)dataGridView2.Rows[selectedRow].Cells[1].Value;
        DAOFactory dao = new NHibernateDAOFactory(session);
        IStudentDAO studentDAO = dao.getStudentDAO();
        Student student = studentDAO
             .getStudentByGroupFirstNameAndLastName(key1, key2, key3);
        getForm4().Visible = true;
        getForm4().setTextBox1Text(student.FirstName);
        getForm4().setTextBox2Text(student.LastName);
        getForm4().setTextBox3Text(student.Sex.ToString());
        getForm4().setTextBox4Text(student.Year.ToString());
        getForm4().setKey1(key1);
        getForm4().setKey2(key2);
        getForm4().setKey3(key3);
        getForm4().setButton1Visible(false);
        getForm4().setButton2Visible(true);
}
```

42. Додайте обробник на зміну розмірів форми:

```
private void Form1_Resize(object sender, EventArgs e)
{
        this.dataGridView1.Columns["Column1"].Width =
             (this.dataGridView1.Width / 3) - 1;
        this.dataGridView1.Columns["Column2"].Width =
             (this.dataGridView1.Width / 3) - 1;
        this.dataGridView1.Columns["Column3"].Width =
             (this.dataGridView1.Width / 3) - 1;
        this.dataGridView2.Columns["Column4"].Width =
             System.Convert.ToInt32(this.dataGridView2.Width * 0.3) - 1;
        this.dataGridView2.Columns["Column5"].Width =
             System.Convert.ToInt32(this.dataGridView2.Width * 0.3) - 1;
        this.dataGridView2.Columns["Column6"].Width =
             System.Convert.ToInt32(this.dataGridView2.Width * 0.1) - 1;
        this.dataGridView2.Columns["Column7"].Width =
             System.Convert.ToInt32(this.dataGridView2.Width * 0.3) - 1;
}
```

1.4 Завдання на лабораторну роботу

Реалізувати застосування, описане в ході виконання лабораторної роботи, що може працювати з двома таблицями СУБД. На відміну від застосування, описаного в лабораторній роботі, предметну область необхідно вибрати з таблиці. Номер варіанту може бути вибраним відповідно до останньої цифри номеру залікової книжки.

Таблиця 1.1 – Варіанти завдань

Номер варіанта	Предметна область
1	Піцерія (Офіціант, Відвідувач)
2	Магазин (Поставщик, Товар)
3	Супермаркет (Продавець, Товар)

Номер варіанта	Предметна область
4	Відділення міліції (Міліціонер, Порушник)
5	Лікарня (Лікар, Пацієнт)
6	Кінотеатр (Кінозал, Відвідувач)
7	Корабель (Каюта, Пасажир)
8	Підприємство (Відділ, Співробітник)
9	Бібліотека (Книга, Читач)

1.5 Контрольні питання

1. Що таке ORM?
2. Чим відрізняється NHibernate від Fluent NHibernate?
3. Як в Fluent NHibernate відобразити поле класу на базу даних.
4. Як створити зв’язко між об’єктами в NHibernate?
5. Як задати зв’язки між об’єктами в Fluent NHibernate?
6. Зв'язок HasOne. Призначення і синтаксис.
7. Зв'язок HasMany. Призначення і синтаксис.
8. Зв'язок HasManyToMany. Призначення і синтаксис.

2 ЛАБОРАТОРНА РОБОТА №2. ПРОЕКТУВАННЯ І РОЗРОБКА ПРОШАРКУ ВІЗУАЛІЗАЦІЇ С З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ ASP.NET

2.1 Мета роботи

Вивчити можливості технології ASP.NET, отримати практичні навички по роботі з ASP.NET.

2.2 Теоретичні відомості

Основа всього web-застосування це сторінка (Page). Користувач, використовуючи браузер, переміщується між сторінками, іноді повертаючись до уже переглянутих сторінок, вводить дані в HTML-форми і отримує деякий результат.

В ASP.NET сторінка найчастіше представляє собою web-форму, що містить різні елементи управління, що реагують на події, що створює користувач.

ASP.NET 1.x дозволяє розділити код логіки від коду візуалізації, тобто помістити код програмної логіки сторінки в .cs або .vb файл, окремо від коду власне сторінки, розміщеної в .aspx файлі. Ця технологія має назву Code-Behind. Таким чином, дизайн сторінки може бути змінений , що дозволяє розділити відповідальність за зовнішній вигляд і роботу сторінки між дизайнером і програмістом. Для цього в .aspx файлі можна задати параметри директиви Page.

```
<%@ Page Language="c#" Src="User.aspx.cs" %>
```

Для підтримки редагування з допомогою VS .NET в ASP.NET сторінці необхідно вказати клас відповідний даній сторінці і файл, в якому знаходиться код цього класу. Для цього директива Page перетворюється з використанням кочового слова Codebenind і Inherits.

```
<%@ Page Language="c#" Codebehind="TheProject.User" Inherits="User.aspx.cs"
%>
```

В ASP.NET 2.0 сторінці використовується інший механізм розділення коду. В директиві Page при цьому необхідно використати інші ключові слова *CodeFile* і *Inherits.*

```
<%@ Page Language="c#" CodeFile="TheProject.User" Inherits="User.aspx.cs" %>
```

В цьому випадку код класу програмної логіки сторінки буде розміщений в файлі, вказаному в атрибуті CodeFile. Треба відмітити, що VS використовує часткові класи (partial classes).

```
public partial class Default : System.Web.UI.Page
{
     protected void Page_Load(object sender, EventArgs e){}
```

```
}
```

Тому розробники можуть помістити код класу в декілька файлів, але подібне розподілення коду робить застосування досить громіздким і вадким в підтримці і розробці.

Модель Code-Behind, що використовується в VS 2003, містить декілька досить вагомих недоліків. Передусім, використовуючи VS, розробнику необхідно компілювати проект перед публікацією, оскільки ASP.NET компілює сторінки тільки якщо вказаний атрибут Src, що не використовується в VS. При цьому, оскільки середовище ASP.NET виявляє зміну дати збірки, після кожної заміни старої збірки в каталозі bin проходить перезапуск домену застосування, що призводить до тимчасової загальмованості в роботі застосування.

VS використовує нові ключові слова,що підтримуються середовищем виконання ASP.NET 2.0, а середовище виконання, в свою чергу, використовує нову техніку компіляції сторінки. Це дозволяє вирішити проблему заміни збірки на нову версію.

Не зважаючи на це, VS досі дозволяє відмовитись від розділення коду і розмістити код програмної логіки в самому файлі сторінки, і використовувати теги *<script runat="server"></script>*. Більше того, за замовчуванням VS створює саме сторінки без розділення коду.

2.2.1 Компіляція сторінок за вимогою

Порівняємо дії, що виконують ASP.NET 2.0 і ASP.NET 1.0, коли користувач виконує запит файлу з розширенням .aspx.

В ASP.NET 1.x середовище виконання аналізує директиву Page сторінки, виконує пошук відповідного класу в збірці застосування, потім на основі коду сторінки створюється клас, похідний від класу програмної логіки сторінки. У випадку якщо збірка відсутня, здійснюється пошук файлу програмної логіки, вказаний в атрибуті Src директиви Page. Якщо файл знайдено, то відбувається компіляція застосування, якщо ні, то ASP.NET видає помилку.

В ASP.NET 2.0 середовище виконання також аналізує директиву Page, здійснює пошук збірки, що відповідає класу логіки сторінки, після чого створює клас сторінки. На відміну від ASP.NET 1.x, батьківським класом для класу сторінки є System.Web.UI.Page, оскільки динамічний клас, що створюється, є власне класом сторінки (використовуються батьківські класи для класу сторінки і класу програмної логіки), а не нащадки класу програмної логіки. Тому, якщо в ASP.NET 1.x клас web-форми міг називатись так само, як і сама web-форма.

```
<form id="frmDefault" runat="server"></form>
…
public class frmDefault : System.Web.UI.Page
{
     protected void Page_Load(object sender, EventArgs e){}
}
```

В ASP.NET 2.0 це недопустимо, оскільки елемент управління System.Web.UI.Form є елементом класу.

Основною перевагою ASP.NET є те, що в випадку відсутності збірки, необхідної для виконання сторінки, відбувається компіляція тільки файлу програмної логіки сторінки, без перекомпіляції всієї збірки. Оскільки існує динамічна компіляція, то необхідно забезпечити можливість створювати код, спільний для всіх сторінок застосування. Для цієї мети в ASP.NET 2.0 існують спеціальні директорії, що є дочірніми директоріями кореневої директорії застосування, одна з яких App_Code, слугує для зберігання файлів, загальний код для всіх сторінок. При виконанні динамічної компіляції, код із директорії App_Code компілюється і стає доступним для всіх сторінок web-застосування. При цьому VS підтримує код, що знаходиться в директорії App_Code, тому працює підсвічування синтаксису і IntelliSense.

Дослідити ASP.NET 2.0 файли, що сгенеровані середовищем, і розібратися в процесі компіляції можна вивчивши вміст директорії:

```
%WINDIT%\Microsoft.NET\Framework\версия\Temporary ASP.NET
Files\имя_приложения
```

куди ASP.NET 2.0 розміщує динамічно створені збірки. Або ж, викликавши помилку на ASP.NET сторінці в режимі від лагодження вибрати посилання Show Complete Compilation Source.

Якщо ж необхідно завчасно скомпілювати проект, перед розгортанням його на сервері в ASP.NET 2.0 існує можливість повної або часткової перекомпіляції web-застосування.

2.2.2 Сторінки ASP.NET 2.0

Для внесення нових можливостей в ASP.NET 2.0 треба було внести зміни і доповнення в клас сторінки Page. Оскільки для встановлення властивостей сторінки в design-time використовуються атрибути директиви Page, то тут будуть розглянуті нові атрибути , що з’явилися для реалізації механізму персоналізації, шаблонів дизайну, оформлення і асинхронної роботи сторінки.

Нижче коротко розглянуті нові атрибути директиви Page:

– Async. Вказує на те, який із інтерфейсів IHttpHandler чи IHttpAsyncHandler реалізує клас сторінки. Після становлення цього атрибуту в true, динамічний клас сторінки, що генерується, буде реалізовувати інтерфейс IHttpAsyncHandler, в іншому випадку клас буде реалізовувати інтерфейс IHttpHandler. Якщо клас сторінки реалізує інтерфейс IHttpAsyncHandler, то код сторінки може виконуватись асинхронно до настання нової події в життєвому циклі сторінки PreRender, до часу настання якого відбувається синхронізація і підготовка HTML-коду для відправки браузеру клієнта;

- AsyncTimeOut. Дозволяє встановити обмеження по часу, відведене для виконання асинхронної операції. За замовчуванням цей параметр рівний 45 секунд;
- Culture. Встановлює набір регіональних параметрів, що використовуються на сторінці;
- EnableTheming. Дозволяє включити підтримку тем оформлення, За замовчуванням ввімкнено;
- MasterPageFile. Вказує шлях до шаблону, який буде використовуватись для створення коду цієї сторінки;
- StyleSheetTheme. Дозволяє встановлювати ідентифікатор теми оформлення, що буде використовуватись для зміни встановленої теми оформлення. Таким чином можна встановити загальну тему оформлення для всього сайту, а з допомогою атрибуту StyleSheetThem вносити деякі зміни в загальне оформлення сторінки чи деяких елементів управління, що містяться на сторінці;
- Theme. Вказує назву теми оформлення, що буде використовуватись для оформлення коду даної сторінки;
- UICulture. Встановлює набір регіональних параметрів, що використовуються для інтерфейсу користувача сторінки.

Життєвий цикл сторінки ASP.NET починається з отримання і обробки web-сервісом IIS запиту до даної сторінки і передачі цього запиту середовищу виконання ASP.NET.

В момент отримання запиту, середовище виконання:

- завантажує клас сторінки, що викликається;
- встановлює властивості класу сторінки;
- вибудовує дерево елементів;
- заповнює властивості Request і Response;
- викликає метод IHttpHandler.ProcessRequest.

Після цього середовище виконання перевіряє, яким чином була викликана ця сторінка і, якщо сторінка викликана шляхом передачі даних з іншої сторінки, про що буде розповідатись далі, то середовище виконання встановлює властивість PreviousPage.

Під час проходження етапів життєвого циклу виникають події, підписавшись на які, розробник може виконувати свій власний код. Варто згадати атрибут AutoEventWireup директиви @Page. Якщо цей атрибут встановлено в true, то методи класу сторінки з назвою Page_НазваПодії автоматично стають обробниками відповідних подій життєвого циклу сторінки.

Для того, щоб від слідкувати життєвий цикл сторінки і послідовність виникнення подій, можна встановити атрибут Trace директиви @Page в true, а атрибут TraceMode в SortByTime. Тоді в розділі Trace Information можна знайти список подій, що відбулися.

Із усіх подій життєвого циклу сторінки розробник може підписатися тільки на п’ять, крім подій дочірніх елементів керування. Ці події PreInit, Init, Load, PreRender, Unload. Розглянемо варіанти використання цих подій.

PreInit. Під час цієї подій можна використовувати властивість IsPostBack, для того, щоб визначити чи викликана сторінка в перший раз чи в результаті постбеку. В плані управління сторінкою розробник може:

- створити динамічні елементи управління;
- динамічно встановлювати шаблон дизайну чи теми оформлення;
- зчитувати або встановлювати властивість об’єкту Profile.

Варто особливо відмітити, що на даному етапі, якщо сторінка була викликана в результаті постбеку, властивості елементів управління ще не встановлені. У випадку, якщо розробник самостійно встановить властивості на цьому етапі, на наступному встановлені значення можуть бути змінені.

Init. На цьому етапі розробник може зчитувати і ініціювати властивості елементів управління.

Load. На цьому етапі розробник може зчитувати і змінювати властивості елементів управління.

PreRender. Остання можливість внести зміни в зовнішній вигляд сторінки.

Unload. Вивільнення зайнятих ресурсів (закриття відкритих з’єднань з БД, завершення роботи з файлами і т.д.). Важливо, що на цьому етапі уже створено HTML код сторінки і спроба внести будь-які зміни приведе до помилки.

Для управління приведеними вище подіями і отримання інформації про результати їх виконання, у об’єкта Page в ASP.NET 2.0 з’явились наступні нові методи і властивості:

- ClientScript. Містить посилання на екземпляр об’єкта ClientScriptManager, що дозволяє працювати з клієнтськими сценаріями;
- EnableTheming. Властивість, що дозволяє відключити підтримку тем оформлення на сторінці, якщо застосування теми небажане;
- GetValidators. Метод, що повертає колекцію валідаторів даної сторінки;
- Header. Посилання на об’єкт HtmlHead, що дозволяє контролювати розділ <head> HTML сторінки при умові, що для елементу head встановлений атрибут runat="server";
- IsAsync. Властивість, що вказує на спосіб обробки сторінки – синхронний чи асинхронний;
- IsCrossPagePostBack. Властивість, що дозволяє визначити, чи буда сторінка запрошена у відповідь на відправку даних з іншої сторінки;
- Master. Посилання на екземпляр об’єкту шаблона сторінки;
- MasterPageFile. Властивість, що містить назву файлу шаблона сторінки;

– PageAdapter. Властивість, що повертає об’єкт PageAdapter, що управляє генерацією HTML сторінки. Конкретний PageAdapter встановлюється середовищем виконання в залежності від параметрів Requets. Якщо вибраний PageAdapter, то події життєвого циклу сторінки заміщаються подіями PageAdapter;
– PreviousPage. Посилання на екземпляр об’єкту сторінки, з якої було здійснено відправлення форми;
– SetFocus. Метод, що дозволяє встановити вибір на будь-який елемент управління;
– TestDeviceFilter. Метод, що перевіряє чи є поточний браузер браузером вказаним в якості аргументу типу;
– Title. Властивість, що дозволяє отримати і змінити заголовок сторінки.

2.2.3 Встановлення фокусу на елемент управління

Іноді буває необхідно, щоб після завантаження сторінки фокус був встановлений на деякий елемент управління (наприклад, в пошуковій формі логічно встановити фокус вводу на основне поле пошуку), для чого необхідно задати значення атрибуту defaultfocus тега form.

```
<form id="frmMain" runat="server" defaultfocus="txtFirstName">
    <asp:TextBox ID="txtFirstName" runat="server" />
    <asp:TextBox ID="txtSecondName" runat="server" />
    <asp:TextBox ID="txtLastName" runat="server" />
</form>
```

Встановленням можна управляти програмно, визиваючи метод SetFocus, передаючи в якості параметра унікальний ідентифікатор цього елементу:

```
protected void Page_Load(object sender, EventArgs e)
{ if(isPostBack)
  { SetFocus("txtLastName");
  }
  else
  { SetFocus("txtFirstText");
  }
}
```

2.2.4 Оновлення даних без перезавантаження сторінки

В ASP.NET 2.0 стало можливим оновлювати дані сторінки без відправки сторінки на сервер і її повного перезавантаження. Це стало можливим завдяки появі клієнтських сценаріїв з зворотнім викликом (callback scripts). Після того, як деяка подія викликає запит до сервера, оновлені дані передаються безпосередньо в клієнтський сценарій в якості аргументів функції.

Подібний спосіб оновлення даних на сторінці зручно і вигідно застосовувати, коли на оновлення всієї інформації потрібно багато ресурсів і

достатньо багато часу. При цьому частина цих даних оновлюється досить часто, а інша частина статична.

Тоді час на оновлення даних, наприклад, одного елементу користувача, враховуючи час запиту до сервера, обробки і отримання відповіді, буде значно менший , ніж час оновлення всієї сторінки.

Припустимо, існує сторінка, на якій знаходиться випадаючий список. Коли користувач вибирає деяке значення із списку, в деякий елемент управління завантажується значення , логічно пов’язане з вибраним із списку значенням.

При цьому задача така, що неможна зберігати всі дані на сторінці клієнта. В ASP.NET 1.x для вирішення цієї задачі необхідно пов’язати з подією зміни значення в випадаючому списку серверний метод. При цьому список повинен визивати відправку сторінки на сервер при кожній зміні значення (AutoPostBack="True").

```
protected void ddStatic_SelectedIndexChanged(object sender, EventArgs e)
{   // На основі значення
  ddStatic.Items[ddStatic.SelectedIndex].Value
    // метод встановлює властивості інших елементів управління
}
```

В ASP.NET 2.0, як вже було сказано вище, існує можливість не оновлювати всю сторінку цілком.

В ASP.NET 2.0, як уже було сказано вище, існує можливість не оновлювати всю сторінку цілком. В даному випадку краще оновити тільки необхідні дані, оскільки оновлювати всю сторінку тільки для того, щоб встановити одне значення занадто нераціонально.

Для реалізації механізму оновлення даних без перезавантаження сторінки необхідно створити клієнтську функцію зворотного виклику, що приймає передані з серверу параметри, серверну функцію, що приймає параметри від клієнта і повертає клієнту значення на основі отриманих параметрів і пов’язати ці дві функції. Механізм асинхронного виклику в ASP.NET 2.0дозволяє повертати результат асинхронно. Для цього в інтерфейсі ICallbackEventHandler визначено два методи: RaiseCallbackEvent – для отримання параметрів на сервері і GetCallbackResult – для повернення результату клієнту. Для реалізації функціональності попереднього прикладу в .aspx файлі розмістіть наступний код:

```
<script>
    function UpdateText(result, context)
    { dSpan.innerText = result;
    }
</script>
<asp:DropDownList ID="ddDynamic" runat="server" />
<br /><span id="dSpan" style="font-weight: bold;" />
```

Клас сторінки, що використовує функцію з зворотним викликом, повинен реалізовувати інтерфейс ICallbackEventHandler.

```
public partial class ScriptCallback_aspx :   System.Web.UI.Page,
System.Web.UI.ICallbackEventHandler { }
```

Сигнатури функцій, що підтримують зворотній виклик, мають наступний вигляд:

```
public virtual void PrepareCallbackEvent(string Аргументы)
public virtual string RenderCallbackResult()
private string EventArgument = "";
public void PrepareCallbackEvent(string eventArgument)
{ EventArgument = eventArgument;
}
public string RenderCallbackResult()
{ return EventArgument;
}
```

Останнім кроком до поставленої мети є зв'язування серверної і клієнтської функцій.

```
protected void Page_Load(object sender, EventArgs e)
{
  string callbackFunction =
   Page.ClientScript.GetCallbackEventReference  ( this,
    "document.all['ddDynamic'].value", "UpdateText",
    "null"
  );
  ddDynamic.Attributes["onchange"] = String.Format("javascript:{0}",
  callbackFunction);}
```

Метод GetCallbackEventReference об'єкта ClientScriptManager приймає в якості параметрів посилання на об'єкт сторінки, рядок, що вказує на значення, яке необхідно передати на сервер при зворотному виклику, назву методу на боці клієнта, що приймає відповідь від сервера. Подібний опис модна отримати з документації MSDN чи з допомогою інструменту VS - Object Browser.

Використовуючи цю технологію модна створити складні методи оновлення даних сторінки, що дозволяють отримати значний виграш в продуктивності. Для цього достатньо розібратись в тому, який код генерує середовище виконання ASP.NET, коли реалізує цю функціональність.

```
<select
name="ddStatic"onchange="javascript:setTimeout('__doPostBack(\'ddStati
c\',\'\')', 0)"id="ddStatic"></select>
<select name="ddDynamic"
id="ddDynamic"onchange="javascript:WebForm_DoCallback('__Page',document.all['
ddDynamic'].value,UpdateText,null,null,false)">
```

Стандартна функція doPostBack досить проста і слугує для зберігання даних про подію в скриті поля форми.

```
var theForm = document.forms['frmCallBack'];
function __doPostBack(eventTarget, eventArgument)
{ if (theForm.onsubmit == null || theForm.onsubmit())
  { theForm.__EVENTTARGET.value = eventTarget;
    theForm.__EVENTARGUMENT.value = eventArgument;
    theForm.submit();
```

```
  }
}
```

При використанні функцій з зворотнім викликом механізм значно складніший. Код функції WebForm_DoCallback набагато більший, ніж doPostBack, оскільки ця функція визначає тип об'єктної моделі браузера, після чого завантажує необхідний для передачі даних модуль.

Наприклад, для браузера, що підтримує DOM це буде Microsoft.XMLHTTP.

Подивитись код різних клієнтських функцій, що використовуються середовищем ASP.NET 2.0 можна, якщо зберегти сторінку, що підтримує функції з зворотним викликом на жорсткий диск і відкривши в текстовому редакторі файл WebResource.axd, посилання на який розміщене в HTML сторінці.

2.2.5 Відправлення даних форми іншій сторінці ASP.NET

Існує обмеження серверних форм в ASP.NET 1.0 – відсутність можливості безпосередньо передавати дані, що були введені в формі, іншій сторінці. Щоб відправити значення елементів на форму іншої сторінки, необхідно використовувати просту HTML форму і в атрибуті action вказати шлях до сторінки, що повинна отримати дані. При цьому втрачаються преваги використання серверних форм.

В ASP.NET 2.0 тег елементу управління може мати додатковий атрибут PostBackUrl, що дозволяє вказати якій сторінці система повинна передати web-форму, якщо відправлення даних на сервер ініційовано цим елементом управління.

```
<form id="frmTest" runat="server">
    <asp:textbox id="txtFirstName" runat="server" />
    <br /><asp:textbox id="txtLastName" runat="server" />
    <br /><asp:button id="btnSend" Text="Post Data"
PostBackUrl="crosspost.aspx"runat="server" />
</form>
```

Після натиснення на кнопку браузер користувача буде переадресований на сторінку crosspost.aspx, при цьому вся інформація про цей елемент управління форми, з якою сталась відправка даних, буде також передана.

Щоб реалізувати цю можливість, середовище ASP.NET 2.0 здійснює перевірку ASPX сторінок на предмет наявності елементів управління з заданим атрибутом PostBackUrl, і при наявності таких, створює на сторінці додаткове скрите поле __PREVIOUSPAGE, що і містить інформацію. Про стан елементів форми. Ця інформація доступна сторінці через властивість PreviousPage.

```
void Page_Load(object sender, EventArgs e)
{ if (PreviousPage != null)
 {
  TextBox txtFirstName =
```

```
            (TextBox)PreviousPage.FindControl("txtFirstName");
  TextBox txtLastName =
            (TextBox)PreviousPage.FindControl("txtLastName");
    txtInfo.Text = "Добрий день, " + txtFirstName.Text + "!";
 }
}
```

На рівні HTML коду, відправка даних іншої форми виглядає наступним чином.

```
<form method="post" action="GetValues.aspx" id="frmTest">
    <input type="submit" name="btnSend" value="Post
Data"onclick="javascript:WebForm_DoPostBackWithOptions(new
WebForm_PostBackOptions(&quot;btnSend&quot;, &quot;&quot;, false,
        &quot;&quot;, &quot;CrossPage.aspx&quot;, false, false))"
id="btnSend" />
</form>
```

Із чого можна зробити висновок, що дані форми не направляються з браузера безпосередньо сторінці CrossPage.aspx, а попередньо направляються все ті й же сторінці GetValues.aspx, що містить форму frmTest. Це необхідно для підтримки серверної перевірки введених даних з використанням елементів управління RequiredFieldValidation, RangeValidation і інших.

2.2.6 Шаблони дизайну сторінки

Що передусім відрізняє один web-сайт від іншого. Для простого користувача мережі Internet основна відмінність полягає в різноманітному дизайні сторінок. Перед тим як зустрітися з функціональними можливостями web-застосування, відвідувач сайту оцінює, чи подобається йому зовнішній вигляд сторінки. Тому дизайн сторінки є не менш важливим ніж загальна функціональність.

Задача розробника писати код, а задача дизайнера займатися зовнішнім виглядом і версткою HTML коду сторінки. При цьому в більшості випадків необхідно забезпечити можливість паралельної роботи над кодом застосування і HTML шаблоном.

З цим успішно справляється технологія розділення коду програмної логіки і HTML код сторінки Code-Behind. Але при цьому при роботі над кожною сторінкою , розробнику доводиться так або інакше працювати з розміткою сторінки.

Варто відмітити, що більшість сучасних web-сайтів мають схожий зовнішній вигляд усіх сторінок і кожна сторінка має загальні елементи дизайну. Для цього в ASP.NET 1.x загальні елементи дизайну сторінок заключали в користувацькі елементи управління, що додавались в кожну сторінку, або, навпаки, сторінки перетворювались в елементи управління, використовуючи одну сторінку в якості основи, в яку завантажували елементи управління в залежності від URL.

Підхід в ASP.NET 2.0 близький до другого з названих підходів, але заключає в собі суттєві переваги. Розробнику більше нема необхідності перетворювати сторінки в елементи управління користувача, що суттєво

спрощує розробку і від лагодження, а також суттєво спрощує роботу дизайнера, оскільки його задача зводиться лише до включення лише декількох ASP.NET елементів в код сторінки-шаблону.

2.2.7 Створення шаблону дизайну

Шаблон дизайну сторінки (Master pages) представляє собою звичайну ASP.NET сторінку, що включає декілька додаткових атрибутів і властивостей, а також містить один або декілька спеціальних елементі управління ContentPlaceHolder. Для того, щоб перетворити звичайну сторінку на шаблон, достатньо замінити директиву Page на директиву Master. Після цього необхідно вставити в розмітку сторінки серверні елементи управління ContentPlaceHolder в ті частини сторінки, де буде розміщуватись інформація, що не відноситься до загального дизайну.

```
<%@Master Language=”C#”
CodeFile=”MainMaster.master.cs” Inherits=”MainMaster_master” %>
<html>
    <head runat="server">
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Моя домашняя страница</title>
    </head>
    <body>
        <table width="100%">
          <tr>
            <span id="PageTitle" runat="server"></span>
          </tr>
          <tr>
            <table width="100%">
              <tr>
               <td>
                  <asp:contentplaceholder id="PageMenu" runat="server" />
               </td>
               <td>
                 <form runat="server">
                   <asp:contentplaceholder id="PageContent" runat="server" />
                 </form>
            </td>
             </tr>
           </table>
         </tr>
         <tr>
            <p align="right">Время: <span id="PageTime"
               runat="server"></span></p>
         </tr>
           </table>
    </body>
</html>
```

Також, як і для звичайної сторінки, в файлі програмної логіки можна працювати з наявними елементами управління, створювати і додавати нові, залежно від параметрів запиту змінювати сторінку і її поведінку.

Таким чином, можна винести логіку створення зовнішнього вигляду web-застосування практично цілком в шаблон сторінки.

Елемент управління ContentPlaceHolder дозволяє визначити вміст, що буде використовуватись в випадку, якщо сторінка, що використовує шаблон, не визначить вміст елемента управління.

```
<asp:contentplaceholder id="PageMenu" runat="server">
    <ul>
        <li><a href="Default.aspx">Головна сторінка</a></li>
        <li><a href="Contents.aspx">Зміст</a></li>
    </ul>
</asp:contentplaceholder>
```

Робота з елементами управління сторінки шаблону не відрізняються від роботи з елементами управління для звичайної сторінки. Таким же чином можна визначити методи і властивості, що будуть доступні всім сторінкам, що використовують даний шаблон.

```
protected void Page_Load(object sender, EventArgs e)
{ PageTime.InnerText = DateTime.Now.ToShortDateString();
}
public string Title
{ get
  { return PageTitle.InnerText;
  }
  Set
  { PageTitle.InnerText = value;
  }
}
```

2.2.8 Створення сторінки

Використання шаблонів дизайну накладає свої вимоги на сторінки. Оскільки шаблон містить елементи управління ContentPlaceHolder, то сторінка повинна містити елементи управління Content, що містять код розмітки і інші елементи управління, що будуть виводитись на результуючій сторінці. На сторінці не повинно буди ніяких серверних елементів управління чи коду розмітки поза елементом управління Content.

```
<%@Page Language=”C#” CodeFile=”Default.aspx.cs”
Inherits=”Default_aspx” MasterPageFile=”MainMaster.master”%>
    <asp:content runat="server" id="MyMenu" contentplaceholderID="PageMenu">
        <ul>
            <li><a href="Page1.aspx">Страница 1</a></li>
            <li><a href="Page2.aspx">Страница 2</a></li>
            <li><a href="Page3.aspx">Страница 3</a></li>
        </ul>
    </asp:content>
    <asp:content runat="server" ID="MyContent"
contentplaceholderID="PageContent">
        <asp:TextBox id="txtName"
runat="server"></asp:TextBox> <asp:Button id="btnShow" runat="server"
Text="Показать" OnClick="btnShow_Click" />
        <br /><asp:PlaceHolder ID="PlaceHolder"
runat="server"></asp:PlaceHolder>
 </asp:content>
```

Як і в випадку з шаблонами, код логіки сторінки створюється звичайним чином. Єдина відмінність в тому, що сторінка не має власних об'єктів типу HeadControl, тому треба використовувати посилання на сторінку шаблону через властивість Master.

```
protected void Page_Load(object sender, EventArgs e)
{ if (!Page.IsPostBack)
   Master.Page.Header.Title = "Домашня сторінка";
}
protected void btnShow_Click(object sender, EventArgs e)
{  PlaceHolder.Controls.Add(new LiteralControl("<script> alert('Добрий день,
" + txtName.Text + "'); </script>"));
  Master.Page.Header.Title = "Добрый день, " + txtName.Text;
}
```

Для того, щоб прив'язати сторінку до шаблону, використовується атрибут MasterPageFile директиви Page. Якщо ж необхідно прив'язати один і той же шаблон до всіх сторінок в директорії то нема необхідності вказувати атрибут MasterPageFile для кожної сторінки, достатньо задати базовий шаблон в файлі Web.config.

```
<?xml version="1.0"?>
    <configuration
xmlns="http://schemas.microsoft.com/.NETConfiguration/v2.0">
    <system.Web>
        <pages master="MainMaster.master" />
    </system.Web>
</configuration>
```

Крім цього, ASP.NET дозволяє встановлювати тему оформлення програмним чином. Як було вказано вище, завантаження і зв'язування з шаблоном оформлення відбувається під час підготовки сторінки до ініціалізації. Тому, якщо необхідно змішувати шаблони оформлення сторінки, необхідно робити це в обробнику події PreInit.

```
protected void Page_PreInit(object sender, EventArgs e)
{ Page.MasterPageFile = "AnotherMaster.master";
}
```

2.2.9 Обробка шаблону середовищем ASP.NET

Під час першого звернення до будь-якої сторінки, ASP.NET здійснює пошук і компіляцію збірок для всіх шаблонів в директорії.

Ця операція займає деякий час, що залежить від кількості сторінок шаблонів в директорії, але виконується тільки один раз.

Тому, якщо в директорії присутні шаблони, що не використовуються, це не призводить до страти продуктивності в процесі роботи застосування, єдина незручність – додатковий час на компіляцію непотрібних збірок для сторінок, що не використовуються.

При звернені до .aspx файлу сторінки з заданим шаблоном оформлення процес компіляції майже нічим не відрізняється від звичайного процесу

компіляції сторінки, за виключенням того, що створюється клас шаблону MasterPage, посилання на який доступне з властивості Page.Master.

Потім сторінка проходить всі ті ж кроки, що описані вище, в результаті чого генерується HTML код, що відправляється клієнту. В отриманому браузером HTML коді вже не можна визначити яка частина коду визначена на сторінці, оскільки елементи управління ContentPlaceHolder і Content не мають будь яких HTML відповідностей і не породжують додаткових тегів, крім свого вмісту.

```
<html>
  <head id="ctl00_Head1">
    <meta http-equiv="Content-Type" content="text/html;
                                            charset=utf-8">
    <title>Домашня сторінка</title>
  </head>
  <body>
    <table width="100%">
       <tr><span id="ctl00_PageTitle"></span></tr>
       <tr>
         <table width="100%">
          <tr>
             <td>
               <ul>
                 <li><a href="Page1.aspx">Сторінка 1</a></li>
                 <li><a href="Page2.aspx">Сторінка 2</a></li>
                 <li><a href="Page3.aspx">Сторінка 3</a></li>
               </ul>
             </td>
             <td>
               <form method="post" action="default.aspx"
                id="__aspnetForm">
               <div><input type="hidden" name="__VIEWSTATE" value=""
               /></div>
               <input name="ctl00$PageContent$txtName" type="text"
                      id="ctl00_PageContent_txtName" />
                
               <input type="submit" name="ctl00$PageContent$btnShow"
                    value="Показати" id="ctl00_PageContent_btnShow" />
               </form>
             </td>
          </tr>
         </table>
      </tr>
      <tr><p align="right">Час: <span
             id="ctl00_PageTime">20.03.2005</span></p></tr>
    </table>
    </body>
</html>
```

Оскільки шаблон є підмножиною сторінки, то є можливість створювати вкладені шаблони, вказуючи для шаблону в директиві Master шлях до іншого шаблону з допомогою атрибуту MasterPageFile. Для цього необхідно в основному шаблоні визначити елемент управління ContentPlaceHolder, а в дочірніх шаблонах на ряду с елементами управління ContentPlaceHolder визначити елементи управління Content для заміщення вмісту елементів ContentPlaceHolder базового шаблону.

2.3 Порядок виконання роботи

В даному порядку лабораторної роботи буде описане, створення web-застосування, в функції якого буде входити візуалізація, додання, видалення та редагування груп, візуалізація, додання, видалення та редагування студентів груп. В якості джерела даних, як і в попередній лабораторній роботі, буде використовуватися СУБД PostgreSQL і створений в попередній лабораторній роботі прошарок доступу до даних.

1. Створіть новий проект - web-застосування ASP.NET (натисніть «Файл», «Створити», «Проект», у вікні, що з'явилось, виберіть «Web», «Web-застосування ASP.NET», задайте каталог проекту і ім'я проекту, наприклад lab5).

2. Додайте до каталогу з проектом бібліотеки для роботи з СУБД PostgreSQL, бібліотеки NHibernate і бібліотеки Fluent NHibernate. Дані бібліотеки можна завантажити за посиланнями:

Драйвер до PostgreSQL	http://www.postgresql.org/
Бібліотеки NHibernate	http://sourceforge.net/projects/nhibernate/
Бібліотеки Fluent NHibernate	http://fluentnhibernate.org/

В даному проекті були використані архіви:

- Npgsql2.0.10-bin-ms.net.zip
- NHibernate-2.1.2.GA-bin.zip
- fluentnhibernate-1.1.zip

Список усіх необхідних бібліотек приведений нижче:

Npgsql.dll
Mono.Security.dll
NHibernate.dll
Antlr3.Runtime.dll
Iesi.Collections.dll
log4net.dll
Castle.DynamicProxy2.dll
Castle.Core.dll
NHibernate.ByteCode.Castle.dll
FluentNHibernate.dll

Підключіть бібліотеки Npgsql.dll, NHibernate.dll, FluentNHibernate.dll і NHibernate.ByteCode.Castle до проекту (натисніть правою кнопкою на «Reference», в дереві проекту, потім на «Add reference», перейдіть на закладку «Browse», виберіть необхідні файли).

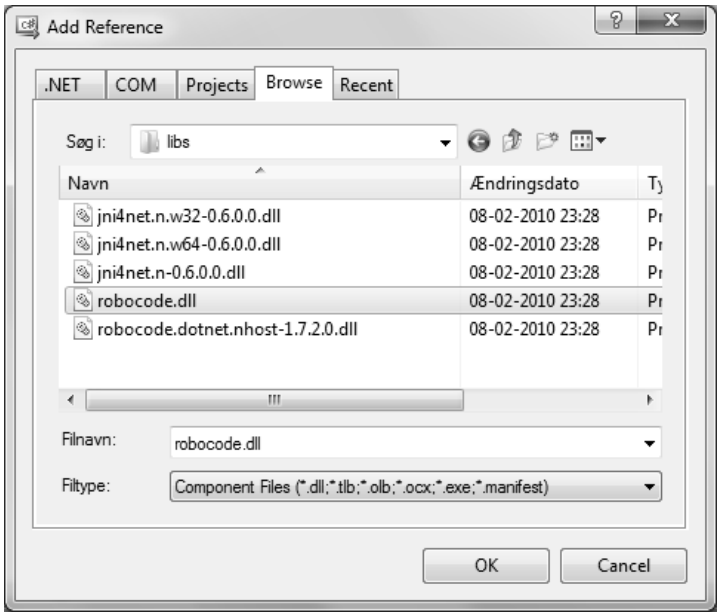

Рисунок 2.1. Вікно додання бібліотек до проекту

3. Додайте до проекту каталоги dao, domain, img, mapping, як показано на рис. 2.2. Каталоги dao, domain, mapping будуть вміщати класи прошарку доступу до даних DAO, реалізованого в попередній лабораторній роботі, а каталог img буде використовуватись для зберігання зображень застосування.

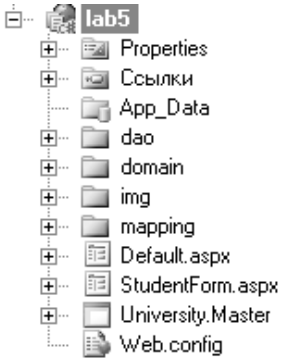

Рисунок 2.2. Каталоги dao, domain, img, mapping в дереві проекту

4. Скопіюйте класи прошарку доступу до даних, реалізовані в попередній лабораторній роботі у відповідні каталоги web-застосування. Це можна виконати, наприклад, з використанням будь-якого файлового менеджера. Відкрийте каталог dao з попередньої лабораторної роботи і скопіюйте всі класи в каталог dao проекту web-застосування. Результати копіювання представлені на рисунку 2.3.

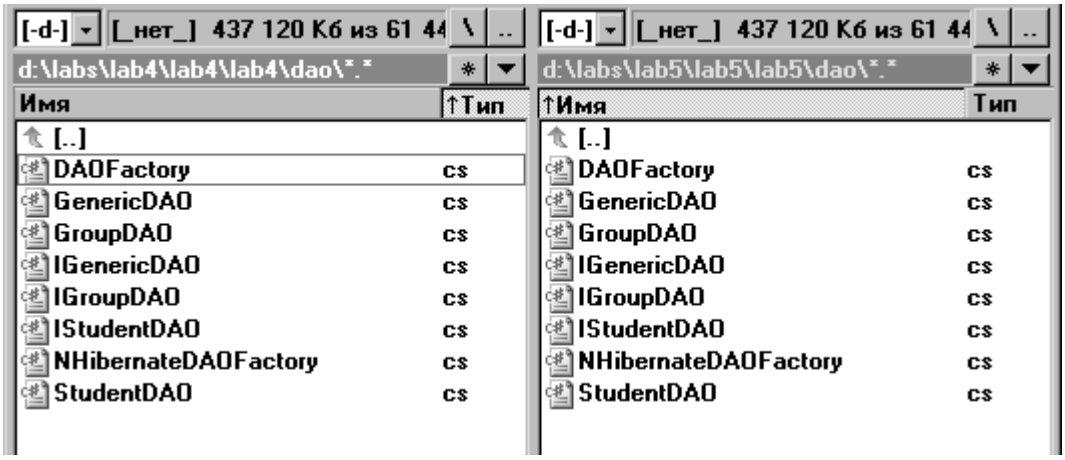

Рисунок 2.3. Результати копіювання класів з каталог dao попереднього проекту в каталог dao web-застосування

Класи каталогів domain і mapping таким же чином необхідно скопіювати у відповідні каталоги web-застосування.

Після копіювання класів в проекті вони не з’являться, тому їх необхідно додати до проекту (натисніть правою кнопкою миші на каталог dao і в контекстному меню виберіть пункт меню «Додати», «Існуючий елемент», і діалозі, що відкрився, виберіть всі класи каталогу dao web-застосування і натисніть «Додати»). Ті ж дії виконайте з класами каталогів domain и mapping. В проекті повинні з’явитися всі класи прошарку доступу до даних. Результати представлені на рис. 2.4.

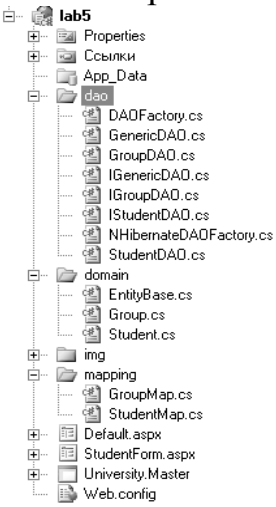

Рисунок 2.4. Результат додання класі прошарку доступу до даних

Таким чином, всі класи прошарку доступу до даних були перенесені в web-застосування і їх можна використовувати для візуалізації, додання, видалення і заміни даних.

5. Розпочнемо реалізацію інтерфейсу нашого web-застосування. Він у нас буде складатись з двох сторінок. На одній сторінці буде проходити

візуалізація списку груп, а по натисненню на групу на іншій сторінці буде проходити візуалізація списку студентів групи. Ці сторінки будуть однотипними, у кожної з них буде заголовок з назвою сторінки, нижня частина в меню ліворуч. В даному випадку сторінок всього дві, але в реальних проектах їх може бути набагато більше і ті частини, що є для декількох сторінок однотипними необхідно виносити в шаблони сторінок.

6. Створимо шаблон сторінки для web-застосування. Натисніть правою кнопкою миші на проект, натисніть «Додати», потім «Створити елемент», у діалоговому вікні, що з'явилось, виберіть «Web», потім «Головна сторінка», введіть ім'я шаблону сторінки, наприклад University і натисніть «Додати».

7. Відкрийте шаблон сторінки University з допомогою редактора головних сторінок (натисніть правою кнопкою на шаблон сторінки University, оберіть «Відкрити з допомогою», у діалоговому вікні, що відкрилося, оберіть «Редактор головних сторінок (За замовчуванням)»). Відкриється код шаблона головної сторінки, що зазвичай складається з HTML тегів і із тегів ASP.NET. Шаблон головної сторінки можна редагувати як у текстовому режимі, так і в графічному режимі, перетягуванням необхідних елементів з палітри компонентів. HTML-теги зазвичай використовуються для розмітки сторінки. Шаблон сторінки повинен містит хоча б один елемент:

```
<asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
</asp:ContentPlaceHolder>
```

Спроектуємо шаблон сторінки в графічному режимі.

8. Відкрийте шаблон сторінки University з використанням графічного редактора (натисніть два рази на шаблон сторінки і із трьох можливих варіантів «Конструктор», «Розділити», «Вихідний код» виберіть «Конструктор»). Для розмітки сторінки будемо використовувати таблиці.

9. Відкрийте палітру компонентів, розмістіть в верхній частині шаблона сторінки три компоненти Table з розділу HTML. Один компонент буде використовуватись для розміщення header (верхньої частини сторінки), другий компонент буде використовуватись для розміщення основної частини сторінки, а третій буде використовуватись для розміщення footer (нижньої частини сторінки). Таблиці за замовчування створюються з трьома рядками і трьома колонками і їх необхідно настроювати відповідним чином.

10. Налаштуємо header (верхню частину шаблона). Для верхньої таблиці встановіть кількість колонок рівним 2, а кількість рядків рівним 1. Нажаль, це не можна задати з використанням диспетчера властивостей, тому шаблон сторінки необхідно відкрити в текстовому режимі і видалити зайві теги <tr> и <td>. В результаті отримаємо код таблиці:

```
<table style="width:100%;">
    <tr>
        <td>
        </td>
```

```
            <td>
            </td>
        </tr>
    </table>
```

В колонку ліворуч верхньої таблиці розмістіть компонент Image з розділу «Стандартні», в колонку праворуч вставте компонент Label з того ж розділу палітри компонентів. Оберіть для компоненту Image зображення, розмістіть його каталозі img. Встановіть з допомогою диспетчера властивостей URL зображення (властивість ImageUrl). Встановіть властивість Text у компонента Label - це буде текст заголовка web-застосування. Остаточний варіант верхньої таблиці в текстовому вигляді представлений нижче.

```
<table style="width:100%;" cellpadding="0" cellspacing="0">
  <tr>
    <td width="200">
      <asp:Image ID="Image1" runat="server" ImageUrl="~/img/1.png" />
    </td>
    <td align="center" bgcolor="#8B0000">
      <asp:Label ID="Label1" runat="server" Font-Bold="True" Font-Size="20pt"
        Text="Черниговский государственный технологический университет"
        ForeColor="#EEC900">
      </asp:Label>
    </td>
  </tr>
</table>
```

Як видно із тексту таблиці, для першої колонки встановлюється ширина по розміру зображення, а для другої колонки встановлюється колір заливки і вирівнювання по центру.

11. Розпочнемо реалізацію центральної частини сторінки, що представлена центральною таблицею. Центральна таблиця також повинна містити один рядок і дві колонки. У лівій колонці буде розмішуватись меню застосування, але воно в даному прикладі використовуватися не буде, оскільки застосування складається лише з двох сторінок, тому ми лише розмістимо меню в лівій колонці, заповнимо назви пунктів меню , а адреси сторінок, на які необхідно переходити при натисненні на пункти меню, заповнювати не будемо. Як видалити зайві колонки розповідається у попередньому пункті даного керівництва. Розмістіть колонці ліворуч компонент Menu з розділу «Переходи». В колонку праворуч перетягніть компонент ContentPlaceHolder. Текст центральної таблиці представлений нижче:

```
<table style="width:100%;">
  <tr>
    <td width="200" valign="top">
      <table style="width:100%;">
        <tr>
          <td align="left">
            <asp:Label ID="Label2" runat="server" Text="МЕНЮ"
              Font-Size="16pt" ForeColor="Maroon">
            </asp:Label>
```

```
          </td>
        </tr>
        <tr>
          <td>
            <asp:Menu ID="Menu1" runat="server"
              Font-Size="16pt" ForeColor="Maroon">
              <Items>
                <asp:MenuItem Text="Групи" Value="Групи">
                </asp:MenuItem>
                <asp:MenuItem Text="Студенти" Value="Студенти">
                </asp:MenuItem>
              </Items>
            </asp:Menu>
          </td>
        </tr>
      </table>
    </td>
    <td>
      <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
      </asp:ContentPlaceHolder>
    </td>
  </tr>
</table>
```

Як видно із тексту таблиці, колонка ліворуч розбита для зручності на дві частини ще однією таблицею, у верхній частині розміщений компонент Label з текстом «Меню», в нижній частині розміщене меню.

12. Розпочнемо реалізацію нижньої частини сторінки, що представлена нижньою таблицею. Вона буде складатись з двох рядків і однієї колонки. В першому рядку буде розміщуватися лінія розділення нижньої частини від основної частини сторінки, в другому рядку буде розміщуватися текст. Текст нижньої сторінки приведений нижче:

```
<table style="width:100%;">
  <tr>
    <td>
      <hr style="color: #800000" />
    </td>
  </tr>
  <tr>
    <td align="center">
      <asp:Label ID="Label3" runat="server"
        Text="OrIoN (c)" ForeColor="Maroon">
      </asp:Label>
    </td>
  </tr>
</table>
```

Повний текст шаблона сторінки приведений нижче, а зовнішній вигляд шаблона сторінки приведений на рис. 2.5.

```
<%@ Master Language="C#" AutoEventWireup="true"
CodeBehind="University.master.cs" Inherits="lab5.University" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
  <head runat="server">
    <title>Чернігівський національний технологічний унівеситет</title>
  </head>
```

```
<body>
  <form id="form1" runat="server">
    <table style="width:100%;" cellpadding="0" cellspacing="0">
      <tr>
        <td width="200">
          <asp:Image ID="Image1" runat="server" ImageUrl="~/img/1.png" />
        </td>
        <td align="center" bgcolor="#8B0000">
          <asp:Label ID="Label1" runat="server"
            Font-Bold="True" Font-Size="20pt"
            Text="Чернігівський національний технологічний університет"
            ForeColor="#EEC900">
          </asp:Label>
        </td>
      </tr>
    </table>
    <table style="width:100%;">
      <tr>
        <td width="200" valign="top">
          <table style="width:100%;">
            <tr>
              <td align="left">
                <asp:Label ID="Label2" runat="server"
                  Text="МЕНЮ" Font-Size="16pt"
                  ForeColor="Maroon">
                </asp:Label>
              </td>
            </tr>
            <tr>
              <td>
                <asp:Menu ID="Menu1" runat="server"
                  Font-Size="16pt" ForeColor="Maroon">
                  <Items>
                    <asp:MenuItem Text="Групи" Value="Групи">
                    </asp:MenuItem>
                    <asp:MenuItem Text="Студенти"
                      Value="Студенти">
                    </asp:MenuItem>
                  </Items>
                </asp:Menu>
              </td>
            </tr>
          </table>
        </td>
        <td>
          <asp:ContentPlaceHolder ID="ContentPlaceHolder1"
            runat="server">
          </asp:ContentPlaceHolder>
        </td>
      </tr>
    </table>
    <table style="width:100%;">
      <tr>
        <td>
          <hr style="color: #800000" />
        </td>
      </tr>
      <tr>
        <td align="center">
          <asp:Label ID="Label3" runat="server" Text="OrIoN (c)"
            ForeColor="Maroon"></asp:Label>
        </td>
      </tr>
```

```
      </table>
    </form>
  </body>
</html>
```

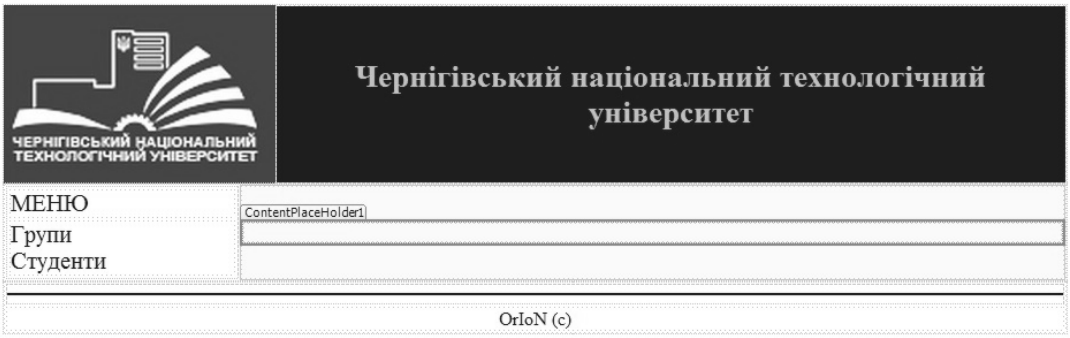

Рисунок 2.5. Зовнішній вигляд шаблона сторінки

13. Шаблон сторінки зручно використовувати для ініціалізації сесії NHibernate. Відкрийте C# код шаблону сторінки. Кожна сторінка, в тому числі і шаблон сторінки ASP.NET обов'язково містить C# код, відкрити який можна розкривши дерево, вузлом якого є сторінка. Якщо шаблон сторінки називається University.Master, то його C# код буде називатись University.Master. Додайте над методом Page_Load, який створився автоматично, поля:

```
private ISessionFactory factory;
private ISession session;
```

За методом додайте два допоміжні методи:

```
private ISession openSession(String host, int port, String database,
  String user, String passwd)
{
  ISession session = null;
  Assembly mappingsAssemly = Assembly.GetExecutingAssembly();
  if (factory == null)
  {
    //Конфігурування фабрики сесій
    factory = Fluently.Configure()
      .Database(PostgreSQLConfiguration
      .PostgreSQL82.ConnectionString(c => c
      .Host(host)
      .Port(port)
      .Database(database)
      .Username(user)
      .Password(passwd)))
      .Mappings(m => m.FluentMappings
      .AddFromAssembly(mappingsAssemly))
      .ExposeConfiguration(BuildSchema)
      .BuildSessionFactory();
  }
  //Відкриття сесії
  session = factory.OpenSession();
  return session;
}

//Метод для автоматичного створення таблиць в базі даних
private static void BuildSchema(Configuration config)
```

```
{
  new SchemaExport(config).Create(false, true);
}
```

Додайте обробник події Page_Init:

```
protected void Page_Init(object sender, EventArgs e)
{
  session = openSession("localhost", 5432,
    "university", "postgres", "111111");
  Session["hbmsession"] = session;
}
```

Обробник події Page_Initвикликається при ініціалізації сторінки, в цей момент і створюється сесія NHibernate, що розміщується в об'єкті Session. Об'єкт здатен зберігати необхідні дані, поки активна HTTP-сесія з застосуванням. Як видно з тексту події Page_Init, налаштування підключення до бази прописана в коді програми, що робити небажано. Всі налаштування пов'язані з базою даних повинні зберігатися в спеціальному файлі, наприклад Web.config. Наш шаблон реалізовано і ми можемо використовувати його ля оформлення наших сторінок.

14. Розпочнемо реалізацію головної сторінки, на якій буде розміщатися список груп. Необхідно буде реалізувати такі функції як додання, видалення і заміна даних про групи, а також візуалізація списку студентів обраної групи. Відкрийте Default.aspx в режимі вихідного коду. Видаліть весь вихідний код крім першого рядка:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="lab5._Default"%>
```

Вставте тег <asp:Content> и підключіть до сторінки Default.aspx шаблон сторінки:

```
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1"
  runat="server">
</asp:Content>
```

Початковий код сторінки повинен мати наступний вигляд:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="lab5._Default" MasterPageFile="~/University.Master"%>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1"
  runat="server">
</asp:Content>
```

Тегів <html>, <body>, <form> на сторінці бути не повинно. Якщо сторінка використовує шаблон сторінки, то всі ці теги вже присутні в шаблоні.

15. Переходимо до візуального проектування сторінки Default.aspx. Відкрийте сторінку в режимі вихідного коду. Якщо шаблон був підключений вірно, то сторінка буде складатись із компонентів, розміщених в шаблоні і із компонента, в я який можна розмістити вміст поточної сторінки. Розмітку сторінки, я к і у випадку з шаблоном сторінки, будемо виконувати з

використанням таблиць. Вставте в частину, що можна редагувати, таблицю, що складається з одної колонки і двох рядків. В першому рядку буде виводитись назва таблиці, а в другому рядку розмістіть компонент GridView з розділу «Дані». Цей компонент має ряд переваг і недоліків. Він, наприклад, зручний для редагування і видалення даних, але дуже не зручний для додання даних. Компонент GridView здатен оброблювати ряд подій, таких як:

RowEditing – виникає, коли рядок таблиці переводиться в режим редагування;

RowUpdating – виникає при натисненні на кнопку Update редагованого рядка;

RowDeleting – виникає при натисненні на кнопку Delete;

RowCancelingEdit – виникає при відміні редагування рядка;

RowCommand – ця подія виникає при натисненні кнопок, розміщених в дяках таблиці;

PageIndexChanging – виникає при зміні індексу сторінки GridView, якщо налаштований посторінковий вивід.

Вихідний код сторінки Default.aspx приведений нижче:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="lab5._Default" MasterPageFile="~/University.Master"%>

<asp:Content ID="Content1"
  ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
  <table align="center">
    <!—Рядок назви таблиці-->
    <tr align="center">
      <td>
        <asp:Label ID="Label4" runat="server" Text="Список групп"
          Font-Size="20pt" ForeColor="Maroon" Font-Bold="True"/>
      </td>
    </tr>
    <!--Рядок, що містить таблицю для візуалізації списку груп-->
    <tr>
      <td>
        <asp:GridView ID="GridView1" runat="server"
          AutoGenerateColumns="false"
          ShowFooter="true" ShowHeader="true"
          AllowPaging="true" PageSize="10"
          onrowdeleting="GridView1_RowDeleting" Font-Size="14pt"
          onrowediting="GridView1_RowEditing"
          onrowcancelingedit="GridView1_RowCancelingEdit"
          onpageindexchanging="GridView1_PageIndexChanging"
          HorizontalAlign="Center"
          onrowupdating="GridView1_RowUpdating"
          onrowcommand="GridView1_RowCommand" >
          <Columns>
            <!--Шаблон колонки для назви групи-->
            <asp:TemplateField HeaderText="Назва групи"
              ItemStyle-Width="200">
              <ItemTemplate>
                <asp:Label id="myLabel1" runat="server"
                  Text='<%# Bind("GroupName")%>' />
              </ItemTemplate>
              <EditItemTemplate>
```

```
      <asp:TextBox ID="myTextBox1" runat="server" Width="200"
        Text='<%# Bind("GroupName") %>'/>
    </EditItemTemplate>
    <FooterTemplate>
       <asp:TextBox ID="myFooterTextBox1" runat="server"
        Width="200" Text='<%# Bind("GroupName") %>' />
    </FooterTemplate>
  </asp:TemplateField>
  <!--Шаблон колонки для ПІБ куратора -->
  <asp:TemplateField HeaderText="ПІБ куратора"
    ItemStyle-Width="300">
    <ItemTemplate>
      <asp:Label id="myLabel2" runat="server"
        Text='<%# Bind("CuratorName")%>' />
    </ItemTemplate>
    <EditItemTemplate>
      <asp:TextBox ID="myTextBox2" runat="server" Width="300"
        Text='<%# Bind("CuratorName") %>'/>
    </EditItemTemplate>
    <FooterTemplate>
      <asp:TextBox ID="myFooterTextBox2" runat="server" Width="300"
        Text='<%# Bind("CuratorName") %>' />
    </FooterTemplate>
  </asp:TemplateField>
  <!--Шаблон колонки для ПІБ старости-->
  <asp:TemplateField HeaderText="ПІБ старости"
    ItemStyle-Width="300">
    <ItemTemplate>
      <asp:Label id="myLabel3" runat="server"
        Text='<%# Bind("HeadmanName")%>' />
    </ItemTemplate>
    <EditItemTemplate>
      <asp:TextBox ID="myTextBox3" runat="server" Width="300"
        Text='<%# Bind("HeadmanName") %>'/>
    </EditItemTemplate>
    <FooterTemplate>
      <asp:TextBox ID="myFooterTextBox3" runat="server"
        Width="300" Text='<%# Bind("HeadmanName") %>' />
    </FooterTemplate>
  </asp:TemplateField>
  <!--Шаблон командної колонки-->
  <asp:TemplateField HeaderText="Команды" ItemStyle-
    HorizontalAlign="Center" FooterStyle-HorizontalAlign="Center">
    <ItemTemplate>
      <asp:ImageButton ID="ibEdit" runat="server"
        CommandName="Edit" Text="Edit"
        ImageUrl="~/img/6.png" />
      <asp:ImageButton ID="ibDelete" runat="server"
        CommandName="Delete" Text="Delete"
        ImageUrl="~/img/3.png" />
      <asp:ImageButton ID="lbSelect" runat="server"
        CommandName="Select"
        ImageUrl="~/img/4.png"
        CommandArgument='<%# Container.DataItemIndex %>'/>
    </ItemTemplate>
    <EditItemTemplate>
      <asp:ImageButton ID="ibUpdate" runat="server"
        CommandName="Update" Text="Update"
        ImageUrl="~/img/5.png" />
      <asp:ImageButton ID="ibCancel" runat="server"
        CommandName="Cancel" Text="Cancel"
        ImageUrl="~/img/7.png" />
    </EditItemTemplate>
```

```
            <FooterTemplate>
              <asp:ImageButton ID="ibInsert" runat="server"
                CommandName="Insert" OnClick="ibInsert_Click"
                ImageUrl="~/img/2.png" />
            </FooterTemplate>
          </asp:TemplateField>
        </Columns>
        <!--Шаблон для візуалізації таблиці с порожніми даними-->
        <EmptyDataTemplate>
          <table border="1" cellpadding="0" cellspacing="0">
            <tr>
              <td width="200" align="center">Назва групи</td>
              <td width="300" align="center">ПІБ куратора</td>
              <td width="300" align="center">ПІБ старосты</td>
              <td>Команди</td>
            </tr>
            <tr>
              <td>
                <asp:TextBox ID="emptyGroupNameTextBox" runat="server"
                  Width="200"/>
              </td>
              <td>
                <asp:TextBox ID="emptyCuratorNameTextBox" runat="server"
                  Width="300"/>
              </td>
              <td>
                <asp:TextBox ID="emptyHeadmanNameTextBox" runat="server"
                  Width="300"/>
              </td>
              <td align="center">
                <asp:ImageButton ID="emptyImageButton" runat="server"
                  ImageUrl="~/img/2.png"
                  OnClick="ibInsertInEmpty_Click"/>
              </td>
            </tr>
          </table>
        </EmptyDataTemplate>
        <PagerStyle HorizontalAlign ="Center" />
      </asp:GridView>
    </td>
  </tr>
</table>
</asp:Content>
```

Таблиця складається з колонок, розміщених в тезі <Columns>. Для кожної колонки визначений шаблон із трьох частин. В першій частині шаблону описується, як буде виглядати елемент рядка таблиці в звичайному режимі. В другій частина шаблону описується, я к буде виглядати елемент рядка таблиці в режимі редагування. В третій частині шаблону описується, як буде виглядати останній рядок таблиці, призначений для додання даних.

Оскільки компонент GridView при відсутності даних в джерелі даних не відображається, то його зовнішній вигляд при відсутності даних необхідно описати з використанням тегу <EmptyDataTemplate>.

Для кожної кнопки таблиці необхідно вибрати зображення і помістити його в каталог img. Ладі необхідно реалізувати обробники подій по натисненню на кнопки компоненту GridView.

16. Відкрийте C# код сторінки Default.aspx. Оскільки вісі функції нашого застосування, котрі нам необхідно запрограмувати будуть використовувати сесію NHibernate, то в першу чергу необхідно отримати сесію NHibernate з об'єкта Session. Додайте обробник події Page_Prerender після порожнього обробника події событий Page_Load:

```
protected void Page_Prerender(object sender, EventArgs e)
{
  ISession session = (ISession)Session["hbmsession"];
  DAOFactory factory = new NHibernateDAOFactory(session);
  IGroupDAO groupDAO = factory.getGroupDAO();
  List<Group> groups = groupDAO.GetAll();
  GridView1.DataSource = groups;
  GridView1.DataBind();
}
```

17. Далі необхідно реалізувати функцію додання групи. Оскільки компонент GridView не відображається при порожньому джерелі даних і кнопка при порожньому и не порожньому джерелі даних – це дві різні кнопки, то необхідно реалізувати два різні обробники подій натиснення на кнопку додати. Нижче приведений код для обробників натиснення на кнопку «Додати» для порожнього і не порожнього джерела даних.

```
//Обробник натиснення на додати
protected void ibInsert_Click(object sender, EventArgs e)
{
  //Получаем значения полей
  string s1 =
    ((TextBox)GridView1.FooterRow.FindControl("MyFooterTextBox1")).Text;
  string s2 =
    ((TextBox)GridView1.FooterRow.FindControl("MyFooterTextBox2")).Text;
  string s3 =
    ((TextBox)GridView1.FooterRow.FindControl("MyFooterTextBox3")).Text;
  //Создаем группу
  Group group = new Group();
  group.GroupName = s1;
  group.CuratorName = s2;
  group.HeadmanName = s3;
  //Создаем DAO группы
  ISession session = (ISession)Session["hbmsession"];
  DAOFactory factory = new NHibernateDAOFactory(session);
  IGroupDAO groupDAO = factory.getGroupDAO();
  groupDAO.SaveOrUpdate(group);
  Response.Redirect(HttpContext.Current.Request.Url.ToString());
}

//Додання першого запису в порожній GridView
protected void ibInsertInEmpty_Click(object sender, EventArgs e)
{
  var parent = ((Control)sender).Parent;
  var groupNameTextBox = parent
    .FindControl("emptyGroupNameTextBox") as TextBox;
  var curatorNameTextBox = parent
    .FindControl("emptyCuratorNameTextBox") as TextBox;
  var headmanNameTextBox = parent
    .FindControl("emptyHeadmanNameTextBox") as TextBox;
  Group group = new Group();
  group.GroupName = groupNameTextBox.Text;
```

```
  group.CuratorName = curatorNameTextBox.Text;
  group.HeadmanName = headmanNameTextBox.Text;
  ISession session = (ISession)Session["hbmsession"];
  DAOFactory factory = new NHibernateDAOFactory(session);
  IGroupDAO groupDAO = factory.getGroupDAO();
  groupDAO.SaveOrUpdate(group);
  Response.Redirect(HttpContext.Current.Request.Url.ToString());
}
```

18. Тепер необхідно реалізувати функцію видалення групи Видалення групи відбувається по події RowDeleting, обробник якої приведено нижче:

```
protected void GridView1_RowDeleting(object sender,
  GridViewDeleteEventArgs e)
{
  int index = e.RowIndex;
  GridViewRow row = GridView1.Rows[index];
  //Отримання назви групи
  string key = ((Label)(row.Cells[0].FindControl("myLabel1"))).Text;
  ISession hbmSession = (ISession)Session["hbmsession"];
  DAOFactory factory = new NHibernateDAOFactory(hbmSession);
  IGroupDAO groupDAO = factory.getGroupDAO();
  Group group = groupDAO.getGroupByName(key);
  if (group != null)
  {
    groupDAO.Delete(group);
  }
  Response.Redirect(HttpContext.Current.Request.Url.ToString());
}
```

19. Тепер необхідно реалізувати функцію редагування групи. Рядок компонента GridView може знаходитись в звичайному стані і в режимі редагування. В звичайному стані в командній колонці знаходяться кнопки «Редагувати» та «Видалити» а в режимі редагування знаходяться кнопки «Оновити» і «Відмінити». В процесі редагування запису приймає участь три кнопки: «Редагувати», «Оновити» та «Відмінити», і необхідно реалізувати події натиснення на ці кнопки. Нижче приведений код цих обробників.

```
//Переведення рядка в режим редагування
protected void GridView1_RowEditing(object sender, GridViewEditEventArgs e)
{
  int index = e.NewEditIndex;
  GridViewRow row = GridView1.Rows[index];
  string oldGroupName = ((Label)(row.Cells[0].FindControl("myLabel1"))).Text;
  ViewState["oldGroupName"] = oldGroupName;
  GridView1.EditIndex = index;
  GridView1.ShowFooter = false;
  GridView1.DataBind();
}

//Відміна редагування запису
protected void GridView1_RowCancelingEdit(object sender,
  GridViewCancelEditEventArgs e)
{
  GridView1.EditIndex = -1;
  GridView1.ShowFooter = true;
  GridView1.DataBind();
}
```

```
//Редагування рядка
protected void GridView1_RowUpdating(object sender,
  GridViewUpdateEventArgs e)
{
  int index = e.RowIndex;
  GridViewRow row = GridView1.Rows[index];
  string newGroupName =
    ((TextBox)(row.Cells[0].FindControl("myTextBox1"))).Text;
  string newCuratorName =
    ((TextBox)(row.Cells[1].FindControl("myTextBox2"))).Text;
  string newHeadmanName =
    ((TextBox)(row.Cells[2].FindControl("myTextBox3"))).Text;
  string oldGroupName = (string)ViewState["oldGroupName"];
  ISession hbmSession = (ISession)Session["hbmsession"];
  DAOFactory factory = new NHibernateDAOFactory(hbmSession);
  IGroupDAO groupDAO = factory.getGroupDAO();
  Group group = groupDAO.getGroupByName(oldGroupName);
  group.GroupName = newGroupName;
  group.CuratorName = newCuratorName;
  group.HeadmanName = newHeadmanName;
  groupDAO.SaveOrUpdate(group);
  GridView1.EditIndex = -1;
  GridView1.ShowFooter = true;
  GridView1.DataBind();
}
```

20. Останньою функцією, яку необхідно реалізувати, буде візуалізація списку студентів по натисненню на кнопку «Студенти» в рядку групи. По натисненню на цю кнопку повинна завантажитися сторінка зі списком студентів заданої групи. Обробник натиснення на кнопку «Студенти» приведений нижче:

```
protected void GridView1_RowCommand(object sender,
  GridViewCommandEventArgs e)
{
  if (e.CommandName == "Select")
  {
    int index = Convert.ToInt32(e.CommandArgument);
    GridViewRow row = GridView1.Rows[index];
    string groupName = ((Label)(row.Cells[0].FindControl("myLabel1"))).Text;
    Session["keyGroupName"] = groupName;
    Response.Redirect("StudentForm.aspx");
  }
}
```

21. Наш компонент GridView підтримує посторінковий вивід даних, і подію перемикання між сторінками необхідно обробити, бо дані не будуть виводитись вірно. Нижче приведений код обробника події перемикання між сторінками.

```
protected void GridView1_PageIndexChanging(object sender,
  GridViewPageEventArgs e)
{
  GridView1.PageIndex = e.NewPageIndex;
  GridView1.EditIndex = -1;
  GridView1.ShowFooter = true;
  GridView1.DataBind();
}
```

Зовнішній вигляд головної web-форми в запущеному вигляді приведений на рис. 2.6.

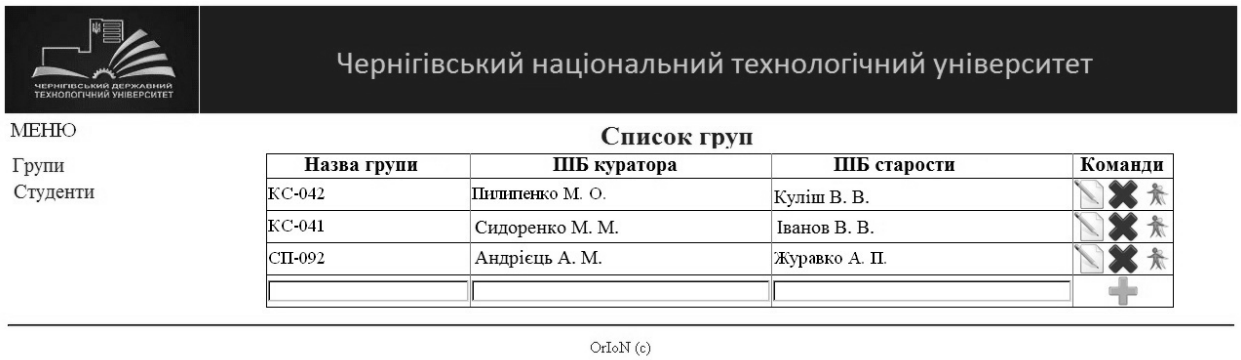

Рисунок 2.6. Зовнішній вигляд web-форми

Всі функції, що відносяться до роботи з групами, реалізовані. Розпочнемо реалізацію функцій, пов'язаних зі студентами.

22. В першу чергу необхідно творити нову сторінку web-форми. Для того, щоб створи нову web-форму, натисніть на проекті правою кнопкою миші, видеріть пункт меню «Додати», потім «Створити елемент», в розділі «Web» діалогу, що з'явився, необхідно вибрати «Форма Web Form». Введіть назву нової web-форми, наприклад StudentForm і натисніть «Додати».

23. Відкрийте нову web-форму StudentForm в режимі вихідного коду. Видаліть весь текст окрім першого рядка і підключіть до web-форми шаблон University. Вихідний код с StudentForm приведений нижче:

```
<%@ Page Language="C#" AutoEventWireup="true"
  CodeBehind="StudentForm.aspx.cs" Inherits="lab5.StudentForm"
  MasterPageFile="~/University.Master"%>

<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1"
  runat="server">

</asp:Content>
```

24. Розпочнемо візуальне проектування нової web-форми. Відкрийте форму в режимі конструктора. Для розмітки сторінки можна використати компонент Table. Розмістіть компонент Table з розділу палітри «HTML» в область сторінки StudentForm, яку можна редагувати. Залиште у таблиці одну колонку і два рядки. В першому рядку розмістіть два компоненти Label. Перший компонент Label буде використовуватись для виводу статичного тексту, а другий - для виводу назви групи. В другий рядок таблиці розмістіть компонент GridView.

Повний текст StudentForm приведений нижче:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="StudentForm.aspx.cs" Inherits="lab5.StudentForm"
MasterPageFile="~/University.Master"%>
```

```
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1"
  runat="server">
    <table style="width:100%;">
      <tr align="center">
        <td>
          <asp:Label ID="Label4" runat="server"
            Text="Список студентов группы  "
            Font-Bold="True" Font-Size="20pt" ForeColor="Maroon">
          </asp:Label>
          <asp:Label ID="Label5" runat="server" Font-Bold="True"
            Font-Size="20pt" ForeColor="Maroon">
          </asp:Label>
        </td>
      </tr>
      <tr>
        <td>
          <asp:GridView ID="GridView1" runat="server"
            AutoGenerateColumns="false"
            ShowFooter="true" ShowHeader="true"
            AllowPaging="true" PageSize="10"
            Font-Size="14pt" HorizontalAlign="Center"
            onrowcancelingedit="GridView1_RowCancelingEdit"
            onrowdeleting="GridView1_RowDeleting"
            onrowediting="GridView1_RowEditing"
            onrowupdating="GridView1_RowUpdating"
            onpageindexchanging="GridView1_PageIndexChanging">
            <Columns>
              <asp:TemplateField HeaderText="Ім'я студента"
                ItemStyle-Width="250">
                <ItemTemplate>
                  <asp:Label id="myLabel1" runat="server"
                    Text='<%# Bind("FirstName")%>' />
                </ItemTemplate>
                <EditItemTemplate>
                  <asp:TextBox ID="myTextBox1" runat="server" Width="250"
                    Text='<%# Bind("FirstName") %>'/>
                </EditItemTemplate>
                <FooterTemplate>
                  <asp:TextBox ID="myFooterTextBox1" runat="server"
                    Width="250" Text='<%# Bind("FirstName") %>' />
                </FooterTemplate>
              </asp:TemplateField>
              <asp:TemplateField HeaderText="Прізвище студента"
                ItemStyle-Width="250">
                <ItemTemplate>
                  <asp:Label id="myLabel2" runat="server"
                    Text='<%# Bind("LastName")%>' />
                </ItemTemplate>
                <EditItemTemplate>
                  <asp:TextBox ID="myTextBox2" runat="server" Width="250"
                    Text='<%# Bind("LastName") %>'/>
                </EditItemTemplate>
                <FooterTemplate>
                  <asp:TextBox ID="myFooterTextBox2" runat="server"
                    Width="250" Text='<%# Bind("LastName") %>' />
                </FooterTemplate>
              </asp:TemplateField>
              <asp:TemplateField HeaderText="Стать" ItemStyle-Width="50">
                <ItemTemplate>
                  <asp:Label id="myLabel3" runat="server"
                    Text='<%# Bind("Sex")%>' />
                </ItemTemplate>
                <EditItemTemplate>
```

```
        <asp:TextBox ID="myTextBox3" runat="server" Width="50"
          Text='<%# Bind("Sex") %>'/>
      </EditItemTemplate>
      <FooterTemplate>
        <asp:TextBox ID="myFooterTextBox3" runat="server"
          Width="50" Text='<%# Bind("Sex") %>' />
      </FooterTemplate>
    </asp:TemplateField>
    <asp:TemplateField HeaderText="Рік нарож."
      ItemStyle-Width="100">
      <ItemTemplate>
      <asp:Label id="myLabel4" runat="server"
        Text='<%# Bind("Year")%>' />
      </ItemTemplate>
      <EditItemTemplate>
        <asp:TextBox ID="myTextBox4" runat="server" Width="100"
          Text='<%# Bind("Year") %>'/>
      </EditItemTemplate>
      <FooterTemplate>
        <asp:TextBox ID="myFooterTextBox4" runat="server"
          Width="100" Text='<%# Bind("Year") %>' />
      </FooterTemplate>
    </asp:TemplateField>
    <asp:TemplateField HeaderText="Команди"
      ItemStyle-HorizontalAlign="Center"
      FooterStyle-HorizontalAlign="Center">
      <ItemTemplate>
        <asp:ImageButton ID="ibEdit" runat="server"
          CommandName="Edit" Text="Edit" ImageUrl="~/img/6.png"/>
        <asp:ImageButton ID="ibDelete" runat="server"
          CommandName="Delete" Text="Delete"
          ImageUrl="~/img/3.png" />
      </ItemTemplate>
      <EditItemTemplate>
        <asp:ImageButton ID="ibUpdate" runat="server"
          CommandName="Update" Text="Update"
          ImageUrl="~/img/5.png" />
        <asp:ImageButton ID="ibCancel" runat="server"
          CommandName="Cancel" Text="Cancel"
          ImageUrl="~/img/7.png" />
      </EditItemTemplate>
      <FooterTemplate>
        <asp:ImageButton ID="ibInsert" runat="server"
          CommandName="Insert" OnClick="ibInsert_Click"
          ImageUrl="~/img/2.png" />
      </FooterTemplate>
    </asp:TemplateField>
  </Columns>
  <EmptyDataTemplate>
    <table border="1" cellpadding="0" cellspacing="0">
      <tr>
        <td width="250" align="center">
          Ім’я студента
        </td>
        <td width="250" align="center">
          Прізвище студента
        </td>
        <td width="50" align="center">
          Стать
        </td>
        <td width="100" align="center">
          Рік народження
        </td>
```

```
                    <td>
                      Команди
                    </td>
                  </tr>
                  <tr>
                    <td>
                      <asp:TextBox ID="emptyFirstNameTextBox" runat="server"
                        Width="250"/>
                    </td>
                    <td>
                      <asp:TextBox ID="emptyLastNameTextBox" runat="server"
                        Width="250"/>
                    </td>
                    <td>
                      <asp:TextBox ID="emptySexTextBox" runat="server"
                        Width="50"/>
                    </td>
                    <td>
                      <asp:TextBox ID="emptyYearTextBox" runat="server"
                        Width="100"/>
                    </td>
                    <td align="center">
                      <asp:ImageButton ID="emptyImageButton" runat="server"
                        ImageUrl="~/img/2.png"
                        OnClick="ibInsertInEmpty_Click"/>
                    </td>
                  </tr>
                </table>
              </EmptyDataTemplate>
            <PagerStyle HorizontalAlign ="Center" />
          </asp:GridView>
        </td>
      </tr>
      </table>
</asp:Content>
```

На відміну від таблиці групи, таблиця студентів містить не три колонки, а чотири, а також відсутня кнопка для візуалізації списку студентів.

25. Далі необхідно запрограмувати функції додання, видалення, редагування студентів. Відкрийте C# код форми StudentForm. Як і в головній формі, спочатку необхідно реалізувати функцію виведення студентів заданої групи в таблиці. Це можна реалізувати в обробнику події Page_Prerender. Текст обробника цієї події приведений нижче:

```
protected void Page_Prerender(object sender, EventArgs e)
{
  string keyGroup = (string)Session["keyGroupName"];
  Label5.Text = keyGroup;
  ISession session = (ISession)Session["hbmsession"];
  DAOFactory factory = new NHibernateDAOFactory(session);
  IGroupDAO groupDAO = factory.getGroupDAO();
  IList<Student> students = groupDAO.getAllStudentOfGroup(keyGroup);
  GridView1.DataSource = students;
  GridView1.DataBind();
}
```

26. Далі необхідно реалізувати функцію додання записів про студентів. Як і для таблиці групи, додання представлене двома обробниками подій:

```
protected void ibInsert_Click(object sender, EventArgs e)
{
  string keyGroup = (string)Session["keyGroupName"];
  string s1 =
    ((TextBox)GridView1.FooterRow.FindControl("MyFooterTextBox1")).Text;
  string s2 =
    ((TextBox)GridView1.FooterRow.FindControl("MyFooterTextBox2")).Text;
  string s3 =
    ((TextBox)GridView1.FooterRow.FindControl("MyFooterTextBox3")).Text;
  string s4 =
    ((TextBox)GridView1.FooterRow.FindControl("MyFooterTextBox4")).Text;
  ISession session = (ISession)Session["hbmsession"];
  DAOFactory factory = new NHibernateDAOFactory(session);
  IGroupDAO groupDAO = factory.getGroupDAO();
  Group group = groupDAO.getGroupByName(keyGroup);
  IStudentDAO studentDAO = factory.getStudentDAO();
  Student student = new Student();
  student.FirstName = s1;
  student.LastName = s2;
  student.Sex = s3[0];
  student.Year = Convert.ToInt32(s4);
  student.Group = group;
  group.StudentList.Add(student);
  studentDAO.SaveOrUpdate(student);
  Response.Redirect(HttpContext.Current.Request.Url.ToString());
}

protected void ibInsertInEmpty_Click(object sender, EventArgs e)
{
  string keyGroup = (string)Session["keyGroupName"];
  var parent = ((Control)sender).Parent;
  var firstNameTextBox = parent
    .FindControl("emptyFirstNameTextBox") as TextBox;
  var lastNameTextBox = parent
    .FindControl("emptyLastNameTextBox") as TextBox;
  var sexTextBox = parent.FindControl("emptySexTextBox") as TextBox;
  var yearTextBox = parent.FindControl("emptyYearTextBox") as TextBox;
  ISession session = (ISession)Session["hbmsession"];
  DAOFactory factory = new NHibernateDAOFactory(session);
  IGroupDAO groupDAO = factory.getGroupDAO();
  Group group = groupDAO.getGroupByName(keyGroup);
  IStudentDAO studentDAO = factory.getStudentDAO();
  Student student = new Student();
  student.FirstName = firstNameTextBox.Text;
  student.LastName = lastNameTextBox.Text;
  student.Sex = sexTextBox.Text[0];
  student.Year = Convert.ToInt32(yearTextBox.Text);
  student.Group = group;
  group.StudentList.Add(student);
  studentDAO.SaveOrUpdate(student);
  Response.Redirect(HttpContext.Current.Request.Url.ToString());
}
```

27. Тепер необхідно реалізувати функцію видалення запису про студента. Це відбувається у обробнику події, текст якого приведений нижче:

```
protected void GridView1_RowDeleting(object sender,
  GridViewDeleteEventArgs e)
{
  string keyGroup = (string)Session["keyGroupName"];
  int index = e.RowIndex;
```

```
  GridViewRow row = GridView1.Rows[index];
  string firstName = ((Label)(row.Cells[0].FindControl("myLabel1"))).Text;
  string lastName = ((Label)(row.Cells[1].FindControl("myLabel2"))).Text;
  ISession hbmSession = (ISession)Session["hbmsession"];
  DAOFactory factory = new NHibernateDAOFactory(hbmSession);
  IStudentDAO studentDAO = factory.getStudentDAO();
  Student student =
    studentDAO.getStudentByGroupFirstNameAndLastName(keyGroup, firstName,
    lastName);
  if (student != null)
  {
    student.Group.StudentList.Remove(student);
    studentDAO.Delete(student);
  }
  Response.Redirect(HttpContext.Current.Request.Url.ToString());
}
```

28. І остання функція, яку необхідно реалізувати, це заміна студента, яка, як і в таблиці групи представлена трьома обробниками подій:

```
protected void GridView1_RowEditing(object sender, GridViewEditEventArgs e)
{
  int index = e.NewEditIndex;
  GridViewRow row = GridView1.Rows[index];
  string oldFirstName = ((Label)(row.Cells[0].FindControl("myLabel1"))).Text;
  string oldLastName = ((Label)(row.Cells[1].FindControl("myLabel2"))).Text;
  ViewState["oldFirstName"] = oldFirstName;
  ViewState["oldLastName"] = oldLastName;
  GridView1.EditIndex = index;
  GridView1.ShowFooter = false;
  GridView1.DataBind();
}


protected void GridView1_RowCancelingEdit(object sender,
  GridViewCancelEditEventArgs e)
{
  GridView1.EditIndex = -1;
  GridView1.ShowFooter = true;
  GridView1.DataBind();
}


protected void GridView1_RowUpdating(object sender,
  GridViewUpdateEventArgs e)
{
  string keyGroup = (string)Session["keyGroupName"];
  int index = e.RowIndex;
  GridViewRow row = GridView1.Rows[index];
  string newFirstName =
    ((TextBox)(row.Cells[0].FindControl("myTextBox1"))).Text;
  string newLastName =
    ((TextBox)(row.Cells[1].FindControl("myTextBox2"))).Text;
  string newSex = ((TextBox)(row.Cells[2].FindControl("myTextBox3"))).Text;
  string newYear = ((TextBox)(row.Cells[2].FindControl("myTextBox4"))).Text;
  string oldFirstName = (string)ViewState["oldFirstName"];
  string oldLastName = (string)ViewState["oldLastName"];
  ISession hbmSession = (ISession)Session["hbmsession"];
  DAOFactory factory = new NHibernateDAOFactory(hbmSession);
  IStudentDAO studentDAO = factory.getStudentDAO();
  Student student =
    studentDAO.getStudentByGroupFirstNameAndLastName(keyGroup, oldFirstName,
```

```
    oldLastName);
  student.FirstName = newFirstName;
  student.LastName = newLastName;
  student.Sex = newSex[0];
  student.Year = Convert.ToInt32(newYear);
  studentDAO.SaveOrUpdate(student);
  GridView1.EditIndex = -1;
  GridView1.ShowFooter = true;
  GridView1.DataBind();
}
```

29. Оскільки GridView підтримує посторінковий вивід даних, то необхідно обробити подію переходу на наступну сторінку:

```
protected void GridView1_PageIndexChanging(object sender,
  GridViewPageEventArgs e)
{
  GridView1.PageIndex = e.NewPageIndex;
  GridView1.EditIndex = -1;
  GridView1.ShowFooter = true;
  GridView1.DataBind();
}
```

Форма студентів представлена на рисунку 2.7.

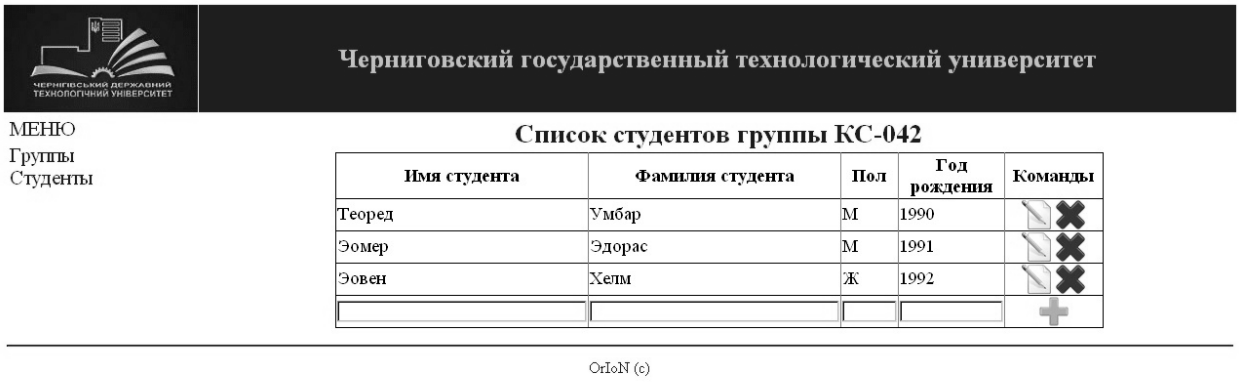

Рисунок 2.7. Зовнішній вигляд web-форми студентів

2.4 Завдання на лабораторну роботу

Створіть web-застосування, описане в ході виконання лабораторної роботи, але об’єкти предметної області необхідно вибрати з таблиці. Номер варіанта можна вибрати, наприклад, за останньою цифрою номера залікової книжки. Варіанти цілком відповідають варіантам завдань з попередньої лабораторної роботи.

Таблиця 2.1. Варіанти завдань

Номер варіанту	Предметна область
1	Піцерія (Офіціант, Відвідувач)
2	Крамниця (Постачальник, Товар)
3	Супермаркет (Продавець, Товар)

4	Відділення міліції (Міліціонер, Порушник)
5	Лікарня (Лікар, Пацієнт)
6	Кінотеатр (Кінозал, Відвідувач)
7	Корабель (Каюта, Пасажир)
8	Підприємство (Відділ, Співробітник)
9	Бібліотека (Книга, Читач)

2.5 Контрольні питання

1. Що таке шаблон сторінки?
2. Для чого використовуються шаблони сторінки?
3. Які етапи життєвого циклу сторінки ASP.NET?
4. Як відбувається генерація коду сторінки на основі шаблону сторінки?
5. Для чого необхідна тришарова архітектура?

3 ЛАБОРАТОРНА РОБОТА №3. ТЕСТУВАННЯ

3.1 Мета роботи

Вивчити технології модульного тестування. Отримати практичні навички по роботі з Testing Framework від Microsoft.

3.2 Теоретичні відомості

Модульне тестування, або unit-тестування (англ. unit testing) - процес у програмуванні, що дозволяє перевірити на коректність окремі модулі вихідного коду програми.

Ідея полягає в тому, щоб писати тести для кожної нетривіальною функції або методу. Це дозволяє досить швидко перевірити, чи не призвело чергове зміна коду до регресії, тобто до появи помилок у вже протестованих місцях програми, а також полегшує виявлення та усунення таких помилок. Мета модульного тестування - ізолювати окремі частини програми і показати, що окремо ці частини працездатні.

Як і будь-яка технологія тестування, модульне тестування не дозволяє відловити всі помилки програми. Справді, це випливає з практичної неможливості трасування всіх можливих шляхів виконання програми, за винятком найпростіших випадків. Крім того, відбувається тестування кожного з модулів окремо. Це означає, що помилки інтеграції, системного рівня, функцій, виконуваних в декількох модулях не будуть визначені. Крім того, дана технологія марна для проведення тестів на продуктивність. Таким чином, модульне тестування більш ефективно при використанні в поєднанні з іншими методиками тестування.

Для отримання вигоди від модульного тестування потрібно строго слідувати технології тестування у всьому процесі розробки програмного забезпечення. Потрібно зберігати не тільки записи про всі проведених тестах, але і про всі зміни вихідного коду у всіх модулях. З цією метою слід використовувати систему контролю версій ПЗ. Таким чином, якщо пізніша версія ПЗ не проходить тест, який був успішно пройдений раніше, буде нескладним звірити вихідний код варіантів і усунути помилку. Також необхідно переконатися в незмінному відстеженні і аналізі невдалих тестів. Ігнорування цієї вимоги призведе до лавиноподібного збільшення невдалих тестових результатів.

3.2.1 Бібліотеки модульного тестування в .NET

На даний момент найбільш поширеними інструментами модульного тестування платформи. NET є бібліотека NUnit і Microsoft Unit Testing Framework.

NUnit - відкрите середовище unit-тестування програм для. NET. Вона була перенесена з мови Java (бібліотека JUnit). Перші версії NUnit були

написані на J #, але потім весь код був переписаний на C # з використанням таких нововведень. NET, як атрибути.

Існують також відомі розширення оригінального пакета NUnit, велика частина з них також з відкритим вихідним кодом. NUnit.Forms доповнює NUnit засобами тестування елементів користувацького інтерфейсу Windows Forms. NUnit.ASP виконує ту ж задачу для елементів інтерфейсу в ASP.NET.

Приклад модульного тесту з використанням бібліотеки NUnit наведено нижче:

```
using NUnit.Framework;

[TestFixture]
public class ExampleTestOfNUnit
{
    [Test]
    public void TestMultiplication()
    {
        Assert.AreEqual(4, 2 * 2, "Умножение");
    }
}
```

Як видно з прикладу модульного тесту NUnit, тестовий клас позначений атрибутом [TestFixture] який вказує, що в даному класі будуть міститися тестові методи. Тестові методи позначаються атрибутом [Test], а в середині тестового методу використовується статичний клас Assert з набором спеціальних методів, для порівняння результатів, що повертаються тестувальним методом з необхідними результатами.

Одним з недоліків бібліотеки є те, що вона не вбудована в середовище розробки Visual Studio.

Існує вбудований в середу розробки Visual Studio інструмент модульного тестування - Microsoft Unit Testing Framework.

3.2.2 Unit Testing Framework от Microsoft

Visual Studio Unit Testing Framework - інструмент модульного тестування для платформи. NET, вбудований в середу розробки Visual Studio 2005 і вище. Щоб визначити, що клас є тестувальним, необхідно позначити його атрибутом [TestClass]. Якщо клас позначений цим атрибутом, то він може містити в собі тестові методи. Зазвичай тестувальний клас називають так само, як і тестований, тільки з префіксом Test.

У тестувальному класі можуть міститися тестувальні методи і зазвичай для всіх методів тестованого класу, які повертають значення, створюється окремий тестувальний метод. Тестувальний метод зазвичай називають, так само як і тестований, тільки з префіксом Test.

Крім тестувальних методів у тестувальному класі можуть бути методи ініціалізації та очищення. Метод ініціалізації позначається атрибутом [TestInitialize] і дозволяє ініціалізувати необхідні змінні перед виконанням методу-тесту. Метод очищення позначається атрибутом [TestCleanup] і дозволяє очистити результати виконання тесту, наприклад, очистити файл,

видалити зайві записи з бази даних, привласнити змінним значення за замовчуванням.

Крім методів ініціалізації та очищення на рівні тесту, в тестуючому класі можуть бути присутніми методи ініціалізації та очищення рівня класу. Ці методи викликаються один раз. Методи ініціалізації рівня класу викликається один раз перед викликом першого тесту, а метод очищення рівня класу викликається після виконання останнього тесту. Метод ініціалізації рівня класу позначається атрибутом [ClassInitialize], а метод очищення рівня класу позначається атрибутом [ClassCleanup]. Нижче наведено приклад тестового класу.

```
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace Test
{
  [TestClass]
  public class TestClass
  {
    public TestClass(){}

    //Тестовый контекст
    private TestContext testContextInstance;

    public TestContext getTestContext(){
      return testContextInstance;
    }

    public void setTestContext(TestContext testContext){
      this.testContextInstance = testContextInstance;
    }

    [ClassInitialize]
    public static void TestClassInitialize(TestContext testContext) {
      /*Код будет выполняться до вызова первого теста*/
    }

    [ClassCleanup]
    public static void TestCClassCleanup(){
      /*Код будет выполняться после вызова последнего теста*/
    }

    [TestInitialize]
    public void TestInitialize(){
      /*Код будет выполняться перед выполнением каждого теста*/
    }

    [TestCleanup]
    public void TestCleanup(){
      /*Код будет выполняться после выполнением каждого теста*/
    }

    [TestMethod]
    public void TestMethod(){
      /*Код теста*/
    }
  }
}
```

У тестовому класі для перевірки значень, що повертаються тестованими методами, використовуються статичні методи класу Assert.

Список всіх методів класу Assert наведено в таблиці 3.1.

Таблиця 3.1. Методи класу Assert для тестування

Назва методу	Опис
Assert.AreEqual	Перевіряє, чи рівні задані значення. Помилка виникає, якщо значення нерівні.
Assert.AreNotEqual	Перевіряє, щоб задані значення небилиці рівні. Помилка виникає якщо значення, якщо значення рівні.
Assert.AreSame	Перевіряє, чи вказують посилання на один і той же об'єкт. Помилка виникає, якщо посилання вказують на різні об'єкти.
Assert.AreNotSame	Перевіряє, щоб посилання не вказували на один і той же об'єкт. Помилка виникає, якщо посилання вказують на один і той же об'єкт.
Assert.Fail	Помилка виникає без будь-яких перевірок.
Assert.IsTrue	Перевіряє, чи є передане значення істинним. Помилка виникає якщо значення хибне.
Assert.IsFalse	Перевіряє, чи є передане значення хибним. Помилка виникає якщо значення істинно.
Assert.IsNull	Перевіряє, чи є передане значення null. Помилка виникає, якщо значення не null.
Assert.IsNotNull	Перевіряє, щоб передане значення не було null. Помилка виникає, якщо передане значення null.
Assert.IsInstanceOfType	Перевіряє, чи є переданий об'єкт екземпляром заданого класу. Помилка виникає, якщо значення не є екземпляром заданого типу.
Assert.IsNotInstanceOfType	Перевіряє, щоб переданий об'єкт не був екземпляром заданого класу. Помилка виникає, якщо об'єкт є екземпляром заданого класу.

3.3 Порядок виконання роботи

В даному порядку виконання лабораторної роботи буде створено тести для методів прошарку доступу до даних DAO, що були розроблені і реалізовані раніше. Для тестування будемо використовувати Unit Testing Framework, що включений в Visual Studio 2008.

1. Створіть консольне застосування Windows (натисніть «Файл», в меню виберіть «Створити», потім «Проект», в діалоговому вікні виберіть «Windows», «Консольне застосування», введіть ім’я проекту, наприклад lab6 і виберіть каталог, в якому буде зберігатися проект).
2. Створіть в проекті каталоги dao, domain, mapping. Перенесіть в ці каталоги будь-яким доступним способом класи прошарку доступу до даних,

що були реалізовані в попередніх лабораторних роботах. Відкрийте каталог dao з попередніх лабораторних робіт і скопіюйте всі класи в каталог dao проекту консольного застосування. Результати копіювання представлені на рисунку 3.1.

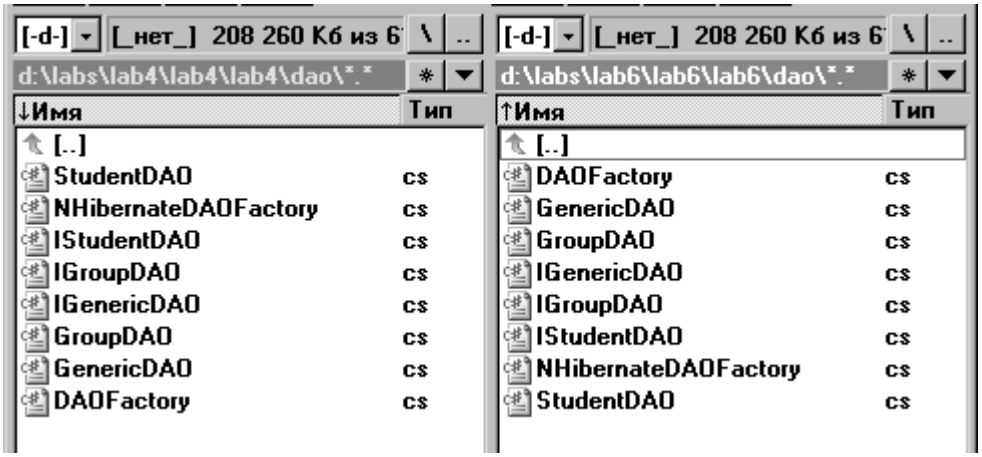

Рисунок 3.1. Результати копіювання класів з каталогу dao попереднього проекту в каталог dao тестового проекту

Класи каталогів domain і mapping таким же чином необхідно скопіювати в відповідні каталоги консольного застосування.

Після копіювання класів в проекті вони не з'являться, їх потрібно додати до проекту (натисніть правою кнопкою мині на каталог dao і в контекстному меню оберіть пункт «Додати», «Існуючий елемент», в діалоговому вікні, що відкрилося оберіть всі класи каталогу dao консольного застосування і натисніть «Додати»). Такими же чином виконайте додання класів каталогів domain і mapping. В проекті повинні з'явитися всі класи прошарку доступу до даних.

3. Підключіть до проекту всі необхідні бібліотеки. Список бібліотек приведений нижче:

Npgsql.dll
Mono.Security.dll
NHibernate.dll
Antlr3.Runtime.dll
Iesi.Collections.dll
log4net.dll
Castle.DynamicProxy2.dll
Castle.Core.dll
NHibernate.ByteCode.Castle.dll
FluentNHibernate.dll

Змініть namespace кожного з класів з lab4 на lab6 як показано в прикладі:

```
namespace lab4.dao
{
  abstract public  class DAOFactory
  {
```

```
namespace lab6.dao
{
  abstract public  class DAOFactory
  {
```

```
    public abstract IStudentDAO
      getStudentDAO();

    public abstract IGroupDAO
      getGroupDAO();
  }
}
```

```
    public abstract IStudentDAO
      getStudentDAO();

    public abstract IGroupDAO
      getGroupDAO();
  }
}
```

Виправте всі помилки в проекті і от компілюйте проект з консольним застосуванням, він повинен відкомпілюватися без помилок. Даний проект можна було й не створювати, а використовувати проекти з попередніх лабораторних робіт, в яких були присутні класи прошарку доступу до даних DAO.

4. Створіть тестовий проект (натисніть «Файл», «Створити проект», в діалоговому вікні виберіть «Тест», «Тестовий проект», введіть ім'я проекту, яв що ім'я проекту, що тестується lab6, то прийнято називати тестовий проект lab6Test, в полі «Рішення», виберіть «Додати рішення»). В тестовому проекті будуть розміщуватись модульні тести класів для прошарку доступу до даних. Таким чином, у нас в одному рішенні міститься два проекти, тестовий проекти (проект з тестами) і проект, що тестується, класи якого необхідно перевірити на працездатність. На даний момент в проекті міститься один клас - UnitTest1, в якому міститься один модульний тест TestMethod1. Якщо запустити процес тестування (натиснути «Тест», «Виконати», «Всі тести рішення»), то один єдиний тест проекту буде успішно пройдений, оскільки в ньому не міститься перевірок. Видаліть клас тестування, створений за замовченням.

5. Підключіть до тестового проекту проект lab6 і всі бібліотеки проекту lab6.

6. В тестовому проекті необхідно створити набір класів, методи яких протестували б коректність методів класів прошарку доступу до даних. Необхідно протестувати клас GroupDAO і клас StudentDAO, але оскільки ці класи є спадкоємцями шаблонного класу GenericDAO, то логічно буде також винести тестові методи GenericDAO в шаблонний клас. Створіть в тестовому проекті три класи: TestGenericDAO, TestGroupDAO, TestStudentDAO. В класі TestGenericDAO буде проводитись тестування методів класу GenericDAO, в класі TestGroupDAO буде проводитись тестування методів класу GroupDAO, а в класі TestStudentDAO буде проводитись тестування методів класу StudentDAO. Тестовий клас зазвичай складається із набору методі-тестів помічених атрибутом [TestMethod]. Також може міститись метод ініціалізації тесту, який викликається през запуском метода-тесту помічений атрибутом [TestInitialize], і метод очистки тесту, який викликається кожного разу по завершенню методу-тесту і помічений атрибутом [TestCleanup]. Також можуть бути присутні методи ініціалізації і очистки рівня класу, котрі викликаються перед запуском першого тесту і після запуску останнього тесту відповідно.

7. Текст тестового класу TestGenericDAO приведений нижче:

```
using System;
```

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using NHibernate;
using NHibernate.Criterion;
using System.Collections.Generic;
using FluentNHibernate;
using FluentNHibernate.Cfg;
using FluentNHibernate.Cfg.Db;
using FluentNHibernate.Automapping;
using NHibernate.Tool.hbm2ddl;
using System.Reflection;
using FluentNHibernate.Mapping;
using NHibernate.Cfg;
using lab6.mapping;
using lab6.dao;
using lab6.domain;

namespace lab6
{
  [TestClass()]
  public abstract class TestGenericDAO<T> where T:EntityBase
  {

    protected static ISessionFactory factory;
    protected static ISession session;
    protected DAOFactory daoFactory;
    protected TestContext testContextInstance;
    /** DAO that will be tested */
    protected IGenericDAO<T> dao = null;

    /** First entity that will be used in tests */
    protected T entity1 = null;
    /** Second entity that will be used in tests */
    protected T entity2 = null;
    /** Third entity that will be used in tests */
    protected T entity3 = null;

    public TestGenericDAO()
    {
      session = openSession("localhost", 5432, "university",
        "postgres", "111111");
    }

    public TestContext TestContext
    {
      get
      {
        return testContextInstance;
      }
      set
      {
        testContextInstance = value;
      }
    }

    /*Getting dao this test case works with*/
    public IGenericDAO<T> getDAO()
    {
      return dao;
    }

    /*Setting dao this test case will work with*/
    public void setDAO(IGenericDAO<T> dao)
    {
```

```
  this.dao = dao;
}

[ClassCleanup]
public static void ClassCleanup()
{
  session.Close();
}

[TestInitialize]
public void TestInitialize()
{
  Assert.IsNotNull(dao,
    "Please, provide IGenericDAO implementation in constructor");
  createEntities();
  Assert.IsNotNull(entity1, "Please, create object for entity1");
  Assert.IsNotNull(entity2, "Please, create object for entity2");
  Assert.IsNotNull(entity3, "Please, create object for entity3");
  checkAllPropertiesDiffer(entity1, entity2);
  checkAllPropertiesDiffer(entity1, entity3);
  checkAllPropertiesDiffer(entity2, entity3);
  saveEntitiesGeneric();
}

[TestCleanup]
public void TestCleanup()
{
  try {
    if ((entity1 = dao.GetById(entity1.Id)) != null)
    dao.Delete(entity1);
  } catch (Exception){
    Assert.Fail("Problem in cleanup method");
  }
  try {
    if ((entity2 = dao.GetById(entity2.Id)) != null)
      dao.Delete(entity2);
  } catch (Exception) {
    Assert.Fail("Problem in cleanup method");
  }
  try {
    if ((entity3 = dao.GetById(entity3.Id)) != null)
      dao.Delete(entity3);
  } catch (Exception) {
    Assert.Fail("Problem in cleanup method");
  }
  entity1 = null;
  entity2 = null;
  entity3 = null;
}

[TestMethod]
public void TestGetByIdGeneric()
{
  T foundObject = null;
  // Should not find with inexistent id
  try
  {
    long id = DateTime.Now.ToFileTime();
    foundObject = dao.GetById(id);
    Assert.IsNull(foundObject, "Should return null if id is inexistent");
  }
  catch (Exception)
  {
```

```
      Assert.Fail("Should return null if object not found");
    }
    // Getting all three entities
    getEntityGeneric(entity1.Id, entity1);
    getEntityGeneric(entity2.Id, entity2);
    getEntityGeneric(entity3.Id, entity3);
  }

  [TestMethod]
  public void TestGetAllGeneric()
  {
    List<T> list = getListOfAllEntities();
    Assert.IsTrue(list.Contains(entity1),
      "After dao method GetAll list should contain entity1");
    Assert.IsTrue(list.Contains(entity2),
      "After dao method GetAll list should contain entity2");
    Assert.IsTrue(list.Contains(entity3),
      "After dao method GetAll list should contain entity3");
  }

  [TestMethod]
  public void TestDeleteGeneric()
  {
    try
    {
      dao.Delete((T)null);
      Assert.Fail("Should not delete entity will null id");
    }
    catch (Exception)
    {
    }
    // Deleting second entity
    try
    {
      dao.Delete(entity2);
    }
    catch (Exception)
    {
      Assert.Fail("Deletion should be successful of entity2");
    }

    // Checking if other two entities can be still found
    getEntityGeneric(entity1.Id, entity1);
    getEntityGeneric(entity3.Id, entity3);

    // Checking if entity2 can not be found
    try
    {
      T foundEntity = null;
      foundEntity = dao.GetById(entity2.Id);
      Assert.IsNull(foundEntity,
       "After deletion entity should not be found with id " + entity2.Id);
    }
    catch (Exception)
    {
      Assert.Fail("Should return null if finding the deleted entity");
    }

    // Checking if other two entities can still be found in getAll list
    List<T> list = getListOfAllEntities();
    Assert.IsTrue(list.Contains(entity1),
      "After dao method GetAll list should contain entity1");
    Assert.IsTrue(list.Contains(entity3),
```

```
    "After dao method GetAll list should contain entity3");
}

protected abstract void createEntities();

protected abstract void checkAllPropertiesDiffer(T entityToCheck1,
  T entityToCheck2);

protected abstract void checkAllPropertiesEqual(T entityToCheck1,
  T entityToCheck2);

protected void saveEntitiesGeneric()
{
  T savedObject = null;
  try
  {
    dao.SaveOrUpdate(entity1);
    savedObject = getPersistentObject(entity1);
    Assert.IsNotNull(savedObject,
      "DAO method saveOrUpdate should return entity if successfull");
    checkAllPropertiesEqual(savedObject, entity1);
    entity1 = savedObject;
  }
  catch (Exception)
  {
    Assert.Fail("Fail to save entity1");
  }
  try
  {
    dao.SaveOrUpdate(entity2);
    savedObject = getPersistentObject(entity2);
    Assert.IsNotNull(savedObject,
      "DAO method saveOrUpdate should return entity if successfull");
    checkAllPropertiesEqual(savedObject, entity2);
    entity2 = savedObject;
  }
  catch (Exception)
  {
    Assert.Fail("Fail to save entity2");
  }

  try
  {
    dao.SaveOrUpdate(entity3);
    savedObject = getPersistentObject(entity3);
    Assert.IsNotNull(savedObject,
      "DAO method saveOrUpdate should return entity if successfull");
    checkAllPropertiesEqual(savedObject, entity3);
  }
  catch (Exception)
  {
    Assert.Fail("Fail to save entity3");
  }
}

protected T getPersistentObject(T nonPersistentObject)
{
  ICriteria criteria = session.CreateCriteria(typeof(T))
    .Add(Example.Create(nonPersistentObject));
  IList<T> list = criteria.List<T>();
  Assert.IsTrue(list.Count >= 1,
    "Count of grups must be equal or more than 1");
  return list[0];
```

```
}

protected void getEntityGeneric(long id, T entity)
{
  T foundEntity = null;
  try
  {
    foundEntity = dao.GetById(id);
    Assert.IsNotNull(foundEntity,
      "Service method getEntity should return entity if successfull");
    checkAllPropertiesEqual(foundEntity, entity);

  }
  catch (Exception)
  {
    Assert.Fail("Failed to get entity with id " + id);
  }
}

protected List<T> getListOfAllEntities()
{
  List<T> list = null;

  // Should get not null and not empty list
  try
  {
    list = dao.GetAll();
  }
  catch (Exception)
  {
    Assert.Fail(
      "Should be able to get all entities that were added before");
  }
  Assert.IsNotNull(list,
    "DAO method GetAll should return list of entities if successfull");
  Assert.IsFalse(list.Count == 0,
   "DAO method should return not empty list if successfull");
  return list;
}

//Метод открытия сессии
public static ISession openSession(String host, int port,
  String database, String user, String passwd)
{
  ISession session = null;
  if (factory == null)
  {
    FluentConfiguration configuration = Fluently.Configure()
      .Database(PostgreSQLConfiguration
      .PostgreSQL82.ConnectionString(c => c
        .Host(host)
        .Port(port)
        .Database(database)
        .Username(user)
        .Password(passwd)))
       .Mappings(m => m.FluentMappings.Add<StudentMap>().Add<GroupMap>())
       .ExposeConfiguration(BuildSchema);
    factory = configuration.BuildSessionFactory();
  }
  //Открытие сессии
  session = factory.OpenSession();
  return session;
}
```

```
    //Метод для автоматического создания таблиц в базе данных
    private static void BuildSchema(Configuration config)
    {
      new SchemaExport(config).Create(false, true);
    }
  }
}
```

8. Текст тестового класу TestGroupDAO приведений нижче:

```
using System;
using System.Collections.Generic;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using lab6.domain;
using lab6.dao;

namespace lab6
{
  [TestClass]
  public class TestGroupDAO:TestGenericDAO<Group>
  {

    protected IGroupDAO groupDAO = null;
    protected Student student1 = null;
    protected Student student2 = null;
    protected Student student3 = null;

    public TestGroupDAO():base()
    {
      DAOFactory daoFactory = new NHibernateDAOFactory(session);
      groupDAO = daoFactory.getGroupDAO();
      setDAO(groupDAO);
    }

    protected override void createEntities()
    {
      entity1 = new Group();
      entity1.GroupName = "KS-091";
      entity1.CuratorName = "Pronin P. P.";
      entity1.HeadmanName = "Volosniy R. R.";

      entity2 = new Group();
      entity2.GroupName = "KS-092";
      entity2.CuratorName = "Eroshenko G. L.";
      entity2.HeadmanName = "Kruglenko T. R.";

      entity3 = new Group();
      entity3.GroupName = "KS-093";
      entity3.CuratorName = "Grab E. E.";
      entity3.HeadmanName = "Stecenko Q R.";
    }

    protected override void checkAllPropertiesDiffer(Group entityToCheck1,
      Group entityToCheck2)
    {
      Assert.AreNotEqual(entityToCheck1.GroupName, entityToCheck2.GroupName,
        "Values must be different");
      Assert.AreNotEqual(entityToCheck1.CuratorName,
        entityToCheck2.CuratorName, "Values must be different");
      Assert.AreNotEqual(entityToCheck1.HeadmanName,
        entityToCheck2.HeadmanName, "Values must be different");
```

```
}

protected override void checkAllPropertiesEqual(Group entityToCheck1,
  Group entityToCheck2)
{
  Assert.AreEqual(entityToCheck1.GroupName, entityToCheck2.GroupName,
    "Values must be equal");
  Assert.AreEqual(entityToCheck1.CuratorName, entityToCheck2.CuratorName,
    "Values must be equal");
  Assert.AreEqual(entityToCheck1.HeadmanName, entityToCheck2.HeadmanName,
    "Values must be equal");
}

[TestMethod]
public void TestGetByIdGroup()
{
  base.TestGetByIdGeneric();
}

[TestMethod]
public void TestGetAllGroup()
{
  base.TestGetAllGeneric();
}

[TestMethod]
public void TestDeleteGroup()
{
  base.TestDeleteGeneric();
}

[TestMethod]
public void TestGetGroupByName()
{
  Group group1 = groupDAO.getGroupByName(entity1.GroupName);
  Assert.IsNotNull(group1,
    "Service method getGroupByName should return group if successfull");
  Group group2 = groupDAO.getGroupByName(entity2.GroupName);
  Assert.IsNotNull(group2,
    "Service method getGroupByName should return group if successfull");
  Group group3 = groupDAO.getGroupByName(entity3.GroupName);
  Assert.IsNotNull(group3,
    "Service method getGroupByName should return group if successfull");
  checkAllPropertiesEqual(group1, entity1);
  checkAllPropertiesEqual(group2, entity2);
  checkAllPropertiesEqual(group3, entity3);
}

[TestMethod]
public void TestGetAllStudentOfGroup()
{
  createEntitiesForStudent();
  Assert.IsNotNull(student1, "Please, create object for student1");
  Assert.IsNotNull(student2, "Please, create object for student2");
  Assert.IsNotNull(student3, "Please, create object for student3");

  entity1.StudentList.Add(student1);
  student1.Group = entity1;
  entity1.StudentList.Add(student2);
  student2.Group = entity1;
  entity1.StudentList.Add(student3);
  student3.Group = entity1;
```

```
  Group savedObject = null;
  try
  {
    dao.SaveOrUpdate(entity1);
    savedObject = getPersistentObject(entity1);
    Assert.IsNotNull(savedObject,
      "DAO method saveOrUpdate should return entity if successfull");
    checkAllPropertiesEqual(savedObject, entity1);
    entity1 = savedObject;
  }
  catch (Exception)
  {
    Assert.Fail("Fail to save entity1");
  }

  IList<Student> studentList =
    groupDAO.getAllStudentOfGroup(entity1.GroupName);
  Assert.IsNotNull(studentList, "List can't be null");
  Assert.IsTrue(studentList.Count == 3,
    "Count of students in the list must be 3");
  checkAllPropertiesEqualForStudent(studentList[0], student1);
  checkAllPropertiesEqualForStudent(studentList[1], student2);
  checkAllPropertiesEqualForStudent(studentList[2], student3);
}

[TestMethod]
public void TestDelGroupByName()
{
  try
  {
    groupDAO.delGroupByName(entity2.GroupName);
  }
  catch (Exception)
  {
    Assert.Fail("Deletion should be successful of entity2");
  }

  // Checking if other two entities can be still found
  getEntityGeneric(entity1.Id, entity1);
  getEntityGeneric(entity3.Id, entity3);

  // Checking if entity2 can not be found
  try
  {
    Group foundGroup = null;
    foundGroup = dao.GetById(entity2.Id);
    Assert.IsNull(foundGroup,
      "After deletion entity should not be found with groupName " +
      entity2.GroupName);
  }
  catch (Exception)
  {
    Assert.Fail("Should return null if finding the deleted entity");
  }

  // Checking if other two entities can still be found in getAll list
  List<Group> list = getListOfAllEntities();
  Assert.IsTrue(list.Contains(entity1),
    "After dao method GetAll list should contain entity1");
  Assert.IsTrue(list.Contains(entity3),
    "After dao method GetAll list should contain entity3");
}
```

```
    protected void createEntitiesForStudent()
    {
      student1 = new Student();
      student1.FirstName = "Ivanov";
      student1.LastName = "Kiril";
      student1.Sex = 'M';
      student1.Year = 1995;

      student2 = new Student();
      student2.FirstName = "Petrenko";
      student2.LastName = "Ivan";
      student2.Sex = 'M';
      student2.Year = 1997;

      student3 = new Student();
      student3.FirstName = "Karpov";
      student3.LastName = "Danil";
      student3.Sex = 'M';
      student3.Year = 2000;
    }

    protected void checkAllPropertiesEqualForStudent(Student entityToCheck1,
      Student entityToCheck2)
    {
      Assert.AreEqual(entityToCheck1.FirstName, entityToCheck2.FirstName,
        "Values must be equal");
      Assert.AreEqual(entityToCheck1.LastName, entityToCheck2.LastName,
        "Values must be equal");
      Assert.AreEqual(entityToCheck1.Sex, entityToCheck2.Sex,
        "Values must be equal");
      Assert.AreEqual(entityToCheck1.Year, entityToCheck2.Year,
        "Values must be equal");
    }
  }
}
```

9. Текст тестового класу StudentDAO приведений нижче:

```
using System;
using System.Collections.Generic;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using lab6.domain;
using lab6.dao;
using NHibernate.Criterion;
using NHibernate;

namespace lab6
{
  [TestClass]
  public class TestStudentDAO:TestGenericDAO<Student>
  {
    protected IStudentDAO studentDAO = null;
    protected IGroupDAO groupDAO = null;
    protected Group group = null;

    public TestStudentDAO():base()
    {
      DAOFactory daoFactory = new NHibernateDAOFactory(session);
      studentDAO  = daoFactory.getStudentDAO();
      groupDAO = daoFactory.getGroupDAO();
      setDAO(studentDAO);
    }
```

```
protected override void createEntities()
{
  entity1 = new Student();
  entity1.FirstName = "Ivanov";
  entity1.LastName = "Kiril";
  entity1.Sex = 'M';
  entity1.Year = 1995;

  entity2 = new Student();
  entity2.FirstName = "Petrenko";
  entity2.LastName = "Ivan";
  entity2.Sex = 'M';
  entity2.Year = 1997;

  entity3 = new Student();
  entity3.FirstName = "Karpov";
  entity3.LastName = "Danil";
  entity3.Sex = 'M';
  entity3.Year = 2000;
}

protected override void checkAllPropertiesDiffer(Student entityToCheck1,
  Student entityToCheck2)
{
  Assert.AreNotEqual(entityToCheck1.FirstName, entityToCheck2.FirstName,
    "Values must be different");
  Assert.AreNotEqual(entityToCheck1.LastName, entityToCheck2.LastName,
    "Values must be different");
  Assert.AreNotEqual(entityToCheck1.Year, entityToCheck2.Year,
    "Values must be different");
}

protected override void checkAllPropertiesEqual(Student entityToCheck1,
  Student entityToCheck2)
{
  Assert.AreEqual(entityToCheck1.FirstName, entityToCheck2.FirstName,
    "Values must be equal");
  Assert.AreEqual(entityToCheck1.LastName, entityToCheck2.LastName,
    "Values must be equal");
  Assert.AreEqual(entityToCheck1.Sex, entityToCheck2.Sex,
    "Values must be equal");
  Assert.AreEqual(entityToCheck1.Year, entityToCheck2.Year,
    "Values must be equal");
}

[TestMethod]
public void TestGetByIdStudent()
{
  base.TestGetByIdGeneric();
}

[TestMethod]
public void TestGetAllStudent()
{
  base.TestGetAllGeneric();
}

[TestMethod]
public void TestDeleteStudent()
{
  base.TestDeleteGeneric();
}
```

```
[TestMethod]
public void TestGetStudentByGroupFirstNameAndLastName()
{
  group = new Group();
  group.GroupName = "KS-091";
  group.CuratorName = "Pronin P. P.";
  group.HeadmanName = "Volosniy R. R.";
  group.StudentList.Add(entity1);
  entity1.Group = group;
  group.StudentList.Add(entity2);
  entity2.Group = group;
  group.StudentList.Add(entity3);
  entity3.Group = group;
  Group savedGroup = null;
  try
  {
    groupDAO.SaveOrUpdate(group);
    savedGroup = getPersistentGroup(group);
    Assert.IsNotNull(savedGroup,
      "DAO method saveOrUpdate should return group if successfull");
    checkAllPropertiesEqualGroup(savedGroup, group);
    group = savedGroup;
  }
  catch (Exception)
  {
    Assert.Fail("Fail to save group");
  }
  getStudentByGroupFirstNameAndLastName(entity1, group.GroupName,
    entity1.FirstName, entity1.LastName);
  getStudentByGroupFirstNameAndLastName(entity2, group.GroupName,
    entity2.FirstName, entity2.LastName);
  getStudentByGroupFirstNameAndLastName(entity3, group.GroupName,
    entity3.FirstName, entity3.LastName);
  group.StudentList.Remove(entity1);
  group.StudentList.Remove(entity2);
  group.StudentList.Remove(entity3);
  entity1.Group = null;
  entity2.Group = null;
  entity3.Group = null;
  groupDAO.Delete(group);
}

protected void getStudentByGroupFirstNameAndLastName(Student student,
  string groupName, string firstName, string lastName)
{
  Student foundStudent = null;
  try
  {
    foundStudent = studentDAO.getStudentByGroupFirstNameAndLastName(
      groupName, firstName, lastName);
    Assert.IsNotNull(studentDAO,
      "Service method should return student if successfull");
    checkAllPropertiesEqual(foundStudent, student);
  }
  catch (Exception)
  {
    Assert.Fail("Failed to get student with  groupName " +
      groupName + " firstName " + firstName+" and lastName " + lastName);
  }
}

protected Group getPersistentGroup(Group nonPersistentGroup)
```

```
    {
      ICriteria criteria = session.CreateCriteria(typeof(Group))
        .Add(Example.Create(nonPersistentGroup));
      IList<Group> list = criteria.List<Group>();
      Assert.IsTrue(list.Count >= 1,
        "Count of grups must be equal or more than 1");
      return list[0];
    }

    protected void checkAllPropertiesEqualGroup(Group entityToCheck1,
      Group entityToCheck2)
    {
      Assert.AreEqual(entityToCheck1.GroupName, entityToCheck2.GroupName,
       "Values must be equal");
      Assert.AreEqual(entityToCheck1.CuratorName, entityToCheck2.CuratorName,
        "Values must be equal");
      Assert.AreEqual(entityToCheck1.HeadmanName, entityToCheck2.HeadmanName,
        "Values must be equal");
    }
  }
}
```

10. Виконайте тестування класів прошарку доступу до даних. Всі тести повинні пройти успішно.

3.4 Завдання на лабораторну роботу

Створіть тестові класи для прошарку доступу до даних, розробленого і реалізованого в попередніх лабораторних роботах. Варіанти завдань необхідно вибрати з таблиці 3.2.

Номер варіанту можна вибрати за останньою цифрою номера залікової книжки. Варіанти завдань цілком відповідають варіантам завдань з попередніх лабораторних робіт.

Таблиця 3.2. Варіанти завдань

Номер варіанту	Предметна область
1	Піцерія (Офіціант, Відвідувач)
2	Крамниця (Постачальник, Товар)
3	Супермаркет (Продавець, Товар)
4	Відділення міліції (Міліціонер, Порушник)
5	Лікарня (Лікар, Пацієнт)
6	Кінотеатр (Кінозал, Відвідувач)
7	Корабель (Каюта, Пасажир)
8	Підприємство (Відділ, Співробітник)
9	Бібліотека (Книга, Читач)

3.5 Контрольні питання

1. Що таке модульне тестування?
2. Які бібліотеки модульного тестування ви знаєте?

3. Яка структура модульного тесту в Testing Framework?
4. Як перевірити, що значення , що повертається методом, відповідає необхідному значенню?
5. Які атрибути використовуються в Unit Testing Framework для визначення тестового класу, тестового методу, методів ініціалізації і очистки класу і методів тестування?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Джесс Либерти. Программирование на C# (2-е издание).- СПб.: Символ, 2003.-688с.
2. Эндрю Троелсен. Язык программирования C# 2005 и платформа .NET 2.0, 3-е издание.: Пер. с англ.- М.: «Вильямс», 2007.-1168с.
3. Эспозито Д. Microsoft ASP.NET 2.0. Базовый курс. Мастер-класс / Пер. с англ. - М.: Издательство «Русская редакция» ; Питер, 2007.- 688с.
4. Нэш Трей C# 2010: ускоренный курс для профессионалов.: Пер. с англ. – М.: ООО «И. Д. Вильямс», 2010.-592с.