ГОСУДАРСТВЕННОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение информационных технологий»

СОВРЕМЕННЫЕ СИСТЕМЫ ПРОГРАММИРОВАНИЯ

Методические рекомендации к лабораторным работам для студентов специальности 1-53 01 02 «Автоматизированные системы обработки информации» дневной и заочной форм обучения

Могилев 2017

УДК 004.4
ББК 32.97
С 56

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Программное обеспечение информационных технологий» «25» октября 2017 г., протокол № 3

Составители: ст. преподаватель Н. В. Выговская;
ст. преподаватель Л. А. Козлова

Рецензент канд. техн. наук, доц. Д. М. Свирепа

Приведены методические рекомендации к лабораторным работам для студентов специальности 1-53 01 02 «Автоматизированные системы обработки информации» дневной и заочной форм обучения по дисциплине «Современные системы программирования».

Учебно-методическое издание

СОВРЕМЕННЫЕ СИСТЕМЫ ПРОГРАММИРОВАНИЯ

Ответственный за выпуск	К. В. Овсянников
Технический редактор	А. А. Подошевко
Компьютерная верстка	Е. С. Лустенкова

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 31 экз. Заказ №

Издатель и полиграфическое исполнение:
Государственное учреждение высшего профессионального образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 24.01.2014.
Пр. Мира, 43, 212000, Могилев.

Содержание

Введение

Цель методических рекомендаций к выполнению лабораторных работ по дисциплине «Современные системы программирования» заключается в овладении студентами практическими навыками разработки приложений по различным современным технологиям: Windows Forms с дополнительными библиотеками, WPF с использованием 3D-графики и с базой данных, а также Web-приложений по технологии ASP.NET в среде Microsoft Visual Studio.

Цель изучения дисциплины – подготовка студентов к использованию современных технологий и программных средств как профессионального инструмента для решения научных и практических задач.

Дисциплина «Современные системы программирования» является неотъемлемой частью знаний инженера-программиста и связана с разработкой современных программных систем и комплексов.

Рассматриваемый в данных методических рекомендациях материал актуален для специалистов по разработке приложений в среде Windows, а также при разработке интерфейсов различных приложений.

Методические рекомендации предназначены для выполнения лабораторных работ в компьютерном классе, а также для получения практических навыков разработки графических приложений с использованием обширных возможностей WPF и ASP.NET. Выполнение рассмотренных примеров позволит студентам выработать практические навыки в создании таких приложений.

Полученные при изучении дисциплины знания и навыки могут быть востребованы при курсовом и дипломном проектировании, при разработке интерфейсов приложений.

1 Лабораторная работа № 1. Общий подход MS Forms. Приложение с графиком. Внешние библиотеки

Цель работы: изучение возможностей построения графических зависимостей с типом приложения MS Forms. Изучение возможностей компонента ZedGraph и подключение его как внешней библиотеки.

1.1 Краткие теоретические сведения

Для построения графических зависимостей используется компонент ZedGraph. Пример построения графической зависимости:

```
double[] x = new double[100];

double[] y = new double[100];

for (int i = 0; i < 100; i++)
{
    x[i] = 0.1 * i;

    y[i] = x[i] * x[i] * Math.Sin(x[i]);
}

zedGraphControl1.GraphPane.CurveList.Clear();//удаление старых графиков

zedGraphControl1.GraphPane.AddCurve("x^2sin",x, y,
                    Color.Red,SymbolType.Plus);//добавление нового графика

zedGraphControl1.AxisChange();// изменение диапазона осей графика

zedGraphControl1.Invalidate ();//обновление графика
```


1.2 Выполнение работы

1.2.1 Подключение ZedGraph к проекту в VS 2010–2013.

1 Поместить файл ZedGraph.dll из архива, приложенного к лабораторной работе, в папку с проектом.

2 В меню «Solution Explorer»(«Обозреватель решений»):

– (клик правой)References > Add Reference… > Browse > Выбрать нужную нам dll из корня проекта*;

– * (клик правой) Ссылки > Добавить ссылку… > Обзор > Выбрать нужную нам dll из корня проекта для русскоязычной VS.

3 В меню «Toolbox»(«Панель элементов»):

– (клик правой) > Choose Items… > Browse > Выбрать нужную dll из корня проекта*.

– * (клик правой) > Выбрать элементы… > Обзор > Выбрать нужную dll из корня проекта для русскоязычной VS.

1.3 Общая часть задания

Разработать приложение, выводящее график соответствующей функции в диапазоне [*a*,*b*] с шагом *h* (параметры *a*, *b*, *h* вводить во время выполнения приложения). Если в заданной математической функции присутствуют параметры, также обеспечить их ввод. Предусмотреть возможность удаления графиков с экрана. Вычисленные данные для построения графиков выводить на экран. Задания для выполнения (по вариантам) представлены в таблице 1.1.

Таблица 1.1 – Задания для выполнения по вариантам

Номер варианта	Виды функций для построения графиков
1	$f(x)=\begin{cases}-x, & x<0\\ a\cdot x, & 10\geq x\geq 0\\ \lvert x\rvert, & x>10\end{cases}$
2	$f(x)=\begin{cases}x^2, & x<0\\ x^3, & a\geq x\geq 0\\ b, & x>a\end{cases}$
3	$f(x)=\begin{cases}\sin(x), & x<0\\ \cos(x), & 20\geq x\geq 0\\ a\cdot\cos(x), & x>20\end{cases}$
4	$f(x)=\begin{cases}x^2, & x<0\\ \cos(ax), & a\geq x\geq 0\\ \sqrt{x}, & x>a\end{cases}$
5	$f(x)=\begin{cases}\sin(x), & x<0\\ \sqrt{x}, & a\geq x\geq 0\\ \sqrt[5]{x}, & x>a\end{cases}$
6	$f(x)=\begin{cases}x^2, & x<0\\ x^3, & a\geq x\geq 0\\ b\cdot\cos(x), & x>a\end{cases}$
7	$f(x)=\begin{cases}x, & x<0\\ \sqrt{x}, & a\geq x\geq 0\\ b\cdot\sin(x), & x>a\end{cases}$
8	$f(x)=\begin{cases}e^x, & x<0\\ b/x, & a\geq x\geq 0\\ c, & x>a\end{cases}$
9	$f(x)=\begin{cases}x^2, & x<0\\ c\cdot x^3, & a\geq x\geq 0\\ b\cdot\cos(x), & x>a\end{cases}$
10	$f(x)=\begin{cases}x\cdot(x^2+c), & x<0\\ \sqrt{x}, & a\geq x\geq 0\\ b\cdot\sin(x), & x>a\end{cases}$

Контрольные вопросы

1 Как подключить ZedGraph к проекту?
2 Как построить график по точкам?
3 Как включить отображение сетки и изменить ее внешний вид?
4 Для чего используется метод Invalidate объекта ZedGraph ?

2 Лабораторная работа № 2. Общий подход WPF Forms. Приложение с формами WPF

Цель работы: изучение возможностей построения приложения WPF с формами (окнами) с использованием элементов управления содержимым WPF и меню.

2.1 Краткие теоретические сведения

2.1.1 WPF и XAML.

Графическая система Windows Presentation Foundation предназначена для создания пользовательских интерфейсов, 2D- и 3D-графики. Преимущества WPF заключаются в том, что 2D-графика строится в векторном виде, а это значит, что интерфейсы будут максимально независимы от разрешения экрана и размера окна.

Они будут легко масштабироваться без потери качества и быстро работать благодаря максимальному использованию возможностей современных графических ускорителей. WPF объединяет документы, формы и мультимедийное содержание в пакет, состоящий из языка разметки и процедурного языка программирования.

Для создания и инициализации объектов в WPF используется язык разметки XAML Extensible.

Application Markup Language (расширяемый язык разметки приложений). XAML использует основные четыре категории элементов:

1) панели размещения;
2) элементы управления;
3) элементы, связанные с документом;
4) графические фигуры.

XAML выступает диалектом языка XML. Файл XAML содержит ровно одну корневую вершину и является деревом отображения. На вершине иерархии находится один из контейнерных объектов. Внутри этих объектов располагаются элементы управления и другие контейнеры. В XAML названия элементов чувствительны к регистру и совпадают с именами классов, доступных в кодовой части WPF.

2.1.2 Создание приложения.

2.1.2.1 Добавление проекта в Visual studio.

Выбрать пункт меню «Файл» (File) > «Новый» (New) > «Проект» (Project) и далее – Приложение WPF (рисунок 2.1). Результат показан на рисунке 2.2.

Выбрать шаблон «Приложение WPF», ввести имя проекта, выбрать папку сохранения проекта и нажать кнопку OK.

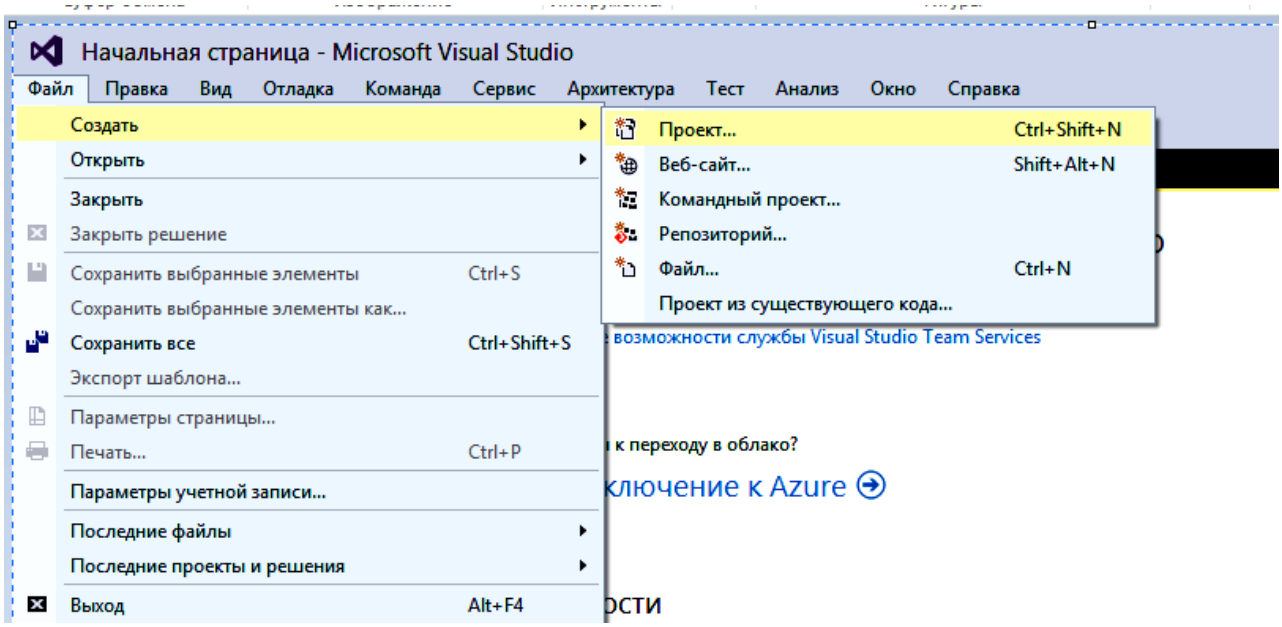

Рисунок 2.1 – Создание проекта

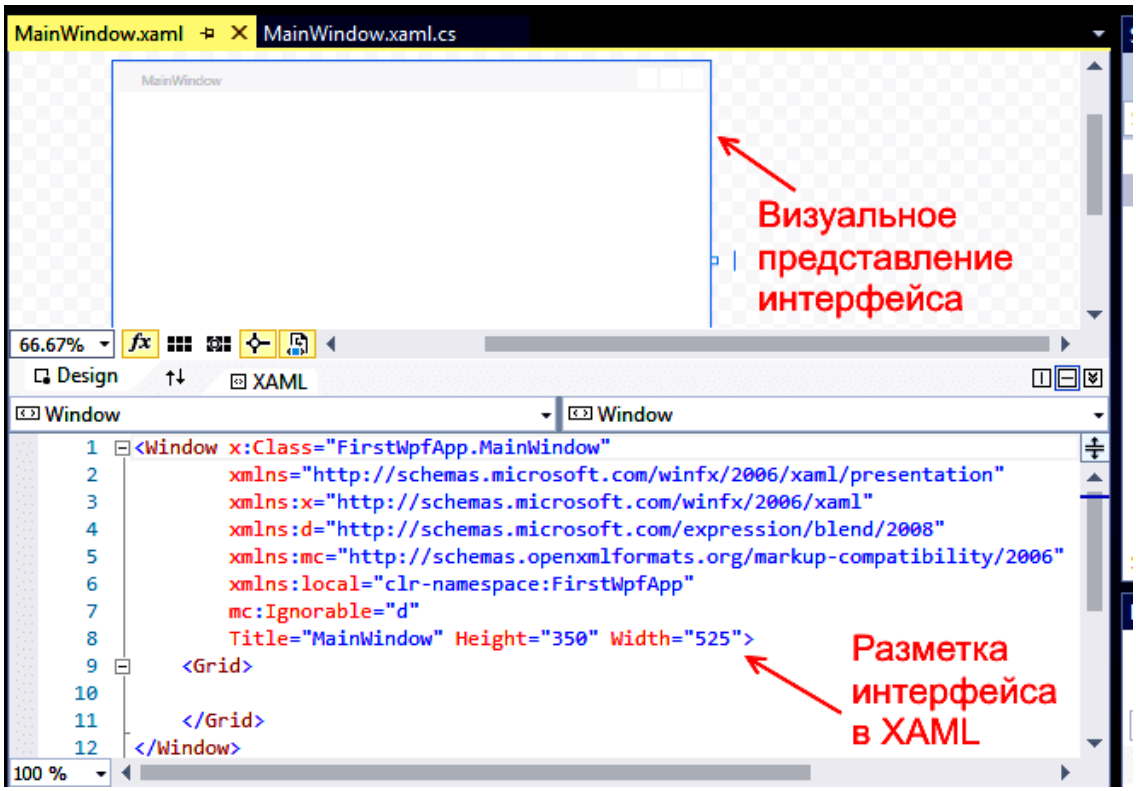

Рисунок 2.2 – Вид окна при создании проекта

2.1.2.2 Добавление картинки к проекту.

После команды Добавить в появившемся окне (рисунок 2.3) выбрать тип файла и далее – сам файл.

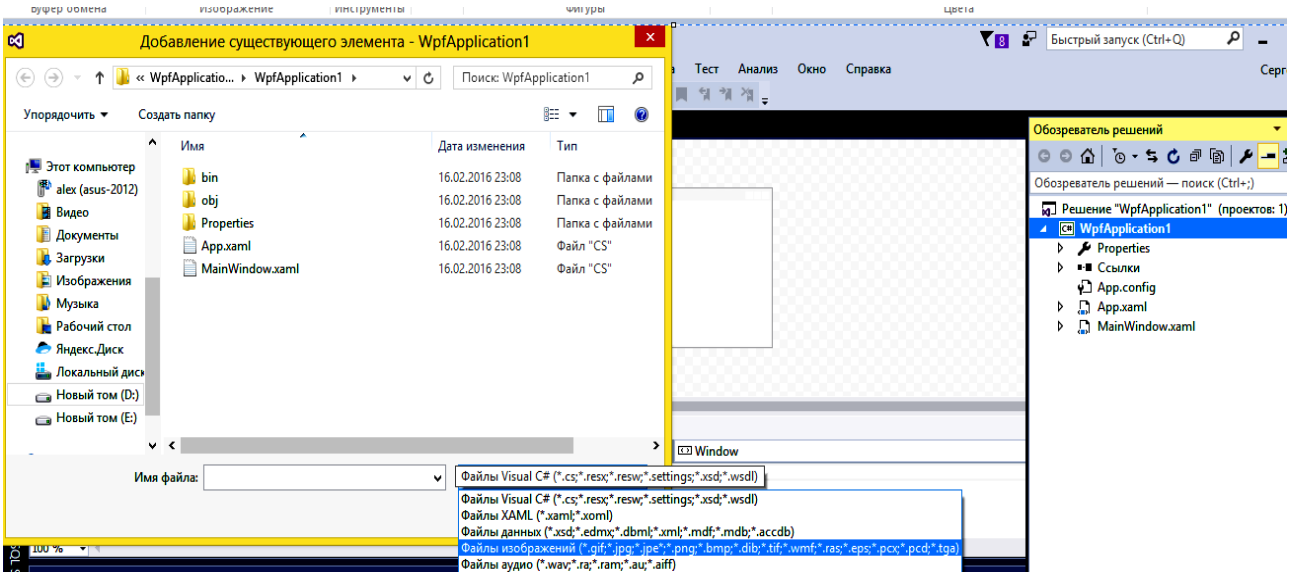

Рисунок 2.3 – Добавление картинки к проекту

В результате всех операций изображение должно появиться в проекте (рисунок 2.4).

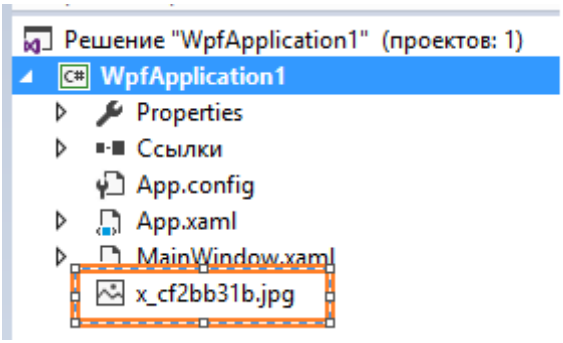

Рисунок 2.4 – Результат добавления

2.1.2.3 Создание главного окна MainWindow.xaml.

Для получения такого вида меню, как показано на рисунке 2.5, необходимо добавить к нашей разметке следующий код для добавления меню:

```
<Menu x:Name="menu" HorizontalAlignment="Left" Height="18" VerticalAlignment="Top" Width="294">
            <MenuItem Header="Input" Name="miInput" />
            <MenuItem Header="Calc" Name="miCalc" />
            <MenuItem Header="Exit" />
        </Menu>
```

Рисунок 2.5 – Образец окна приложения с меню

Затем нужно разработать логику приложения согласно выбранному варианту.

2.2 Выполнение работы

2.2.1 Варианты заданий.

Вариант 1

Создать меню с командами Input**,** Calc и Exit.

При выборе команды Input открывается диалоговое окно, содержащее:

– три поля типа TextBox для ввода длин трех сторон треугольника;

– группу из двух флажков (Периметр и Площадь) типа CheckBox;

– кнопку типа Button.

Обеспечить возможность:

– ввода длин трех сторон треугольника;

– выбора режима с помощью флажков: подсчет периметра и/или площади треугольника.

При выборе команды Calc открывается диалоговое окно с результатами. При выборе команды Exit приложение завершается.

Вариант 2

Создать меню с командами Size, Color, Paint, Quit.

Команда Paint недоступна. При выборе команды Quit приложение завершается. При выборе команды Size открывается диалоговое окно, содержащее:

– два поля типа TextBox для ввода длин сторон прямоугольника;

– группу из трех флажков (Red, Green, Blue) типа CheckBox;

– кнопку типа Button.

Обеспечить возможность:

– ввода длин сторон прямоугольника в пикселах в поля ввода;

– выбора его цвета с помощью флажков.

После задания параметров команда Paint становится доступной.

При выборе команды Paint в главном окне приложения выводится

прямоугольник заданного размера и сочетания цветов или выдается сообщение, если введенные размеры превышают размер окна.

Вариант 3

Создать меню с командами Input, Work, Exit.

При выборе команды Exit приложение завершает работу. При выборе команды Input открывается диалоговое окно, содержащее:

– три поля ввода типа TextBox с метками Radius, Height, Density;

– группу из двух флажков (Volume, Mass) типа CheckBox;

– кнопку типа Button.

Обеспечить возможность:

– ввода радиуса, высоты и плотности конуса;

– выбора режима с помощью флажков: подсчет объема и/или массы конуса.

При выборе команды Work открывается окно сообщений с результатами.

Вариант 4

Создать меню с командами Input, Calc, Draw, Exit.

При выборе команды Exit приложение завершает работу. При выборе команды Input открывается диалоговое окно, содержащее:

– поле ввода типа TextBox с меткой Radius;

– группу из двух флажков (Square, Length) типа CheckBox;

– кнопку типа Button.

Обеспечить возможность:

– ввода радиуса окружности;

– выбора режима с помощью флажков: подсчет площади круга (Square) и/или длины окружности (Length).

При выборе команды Calc открывается окно сообщений с результатами. При выборе команды Draw в центре главного окна выводится круг введенного радиуса или выдается сообщение, что рисование невозможно (если диаметр превышает размеры рабочей области).

Вариант 5

Создать меню с командами Input, Calc, About.

При выборе команды About открывается окно с информацией о разработчике. При выборе команды Input открывается диалоговое окно, содержащее:

– три поля ввода типа TextBox: Number 1, Number 2, Number 3;

– группу из двух флажков (Summ, Least multiple) типа CheckBox;

– кнопку типа Button.

Обеспечить возможность ввода трех чисел и выбора режима вычислений с помощью флажков: подсчет суммы трех чисел (Summ) и/или наименьшего общего кратного двух первых чисел (Least multiple). При выборе команды Calc открывается диалоговое окно с результатами.

Вариант 6

Создать меню с командами Input, Calc, Quit.

Команда Calc недоступна. При выборе команды Quit приложение завершается. При выборе команды Input открывается диалоговое окно, содержащее:

– два поля ввода типа TextBox с метками Number 1, Number 2;

– группу из трех флажков (Summa, Max divisor, Multiply) типа CheckBox;

– кнопку типа Button.

Обеспечить возможность:

– ввода двух чисел;

– выбора режима вычислений с помощью флажков (можно вычислять в любой комбинации такие величины, как сумма, наибольший общий делитель и произведение двух чисел).

При выборе команды Calc открывается окно сообщений с результатами.

Вариант 7

Создать меню с командами Begin, Help, About.

При выборе команды About открывается окно с информацией о разработчике. При выборе команды Begin открывается диалоговое окно, содержащее:

– поле ввода типа TextBox с меткой input;

– метку типа Label для вывода результата;

– группу из трех переключателей (2, 8, 16) типа RadioButton;

– две кнопки типа Button – Do и OK.

Обеспечить возможность:

– ввода числа в десятичной системе в поле input;

– выбора режима преобразования с помощью переключателей: перевод в двоичную, восьмеричную или шестнадцатеричную систему счисления.

При щелчке на кнопке Do должен появляться результат перевода.

Вариант 8

Создать меню с командами Input color, Change, Exit, Help.

При выборе команды Exit приложение завершает работу. При выборе команды Input color открывается диалоговое окно, содержащее:

– три поля ввода типа TextBox с метками Red, Green, Blue;

– группу из двух флажков (Left, Right) типа CheckBox;

– кнопку типа Button.

Обеспечить возможность ввода RGB-составляющих цвета. При выборе команды Change цвет главного окна изменяется на заданный (левая, правая или обе половины окна в зависимости от установки флажков).

Вариант 9

Создать меню с командами Input size, Choose, Change, Exit.

При выборе команды Exit приложение завершает работу. Команда Change недоступна. При выборе команды Input size открывается диалоговое окно,

содержащее:

– два поля ввода типа TextBox с метками Size x, Size y;

– кнопку типа Button.

При выборе команды Choose открывается диалоговое окно, содержащее:

– группу из двух переключателей (Increase, Decrease) типа RadioButton;

– кнопку типа Button.

Обеспечить возможность ввода значений в поля Size x и Size y. Значения интерпретируются как количество пикселов, на которое надо изменить размеры главного окна (увеличить или уменьшить в зависимости от положения переключателей).

После ввода значений команда Change становится доступной. При выборе этой команды размеры главного окна увеличиваются или уменьшаются на введенное количество пикселов.

Вариант 10

Создать меню с командами Begin, Work, About.

При выборе команды About открывается окно с информацией о разработчике. При выборе команды Begin открывается диалоговое окно, содержащее:

– поле ввода типа TextBox с меткой Input word;

– группу из двух переключателей (Upper case, Lower case) типа RadioButton;

– кнопку типа Button.

Обеспечить возможность ввода слова и выбора режима перевода в верхний или нижний регистр в зависимости от положения переключателей. При выборе команды Work открывается диалоговое окно с результатом перевода.

Контрольные вопросы

1 Как создать проект WPF в Visual Studio и какие файлы он включает в себя?

2 Для чего используется язык XAML?

3 Как создать меню?

4 Как добавить и вызвать новое окно в приложении WPF?

5 Какие элементы контроля используются для перехода между страницами приложения?

3 Лабораторная работа № 3. Разработка WPF-приложения с 3D-графикой

Цель работы: получение практических навыков разработки приложений с использованием 3D-графики в среде Microsoft Visual Studio по технологии WPF.

3.1 Краткие теоретические сведения

3.1.1 Координаты трехмерного пространства.

Начало системы координат WPF для графики 2D отсчитывается от левого верхнего угла области рисования (обычно экрана). В системе 2D положительные значения оси *x* откладываются вправо, а положительные значения оси *y* – сверху вниз. Однако в системе координат 3D начало располагается в центре отрисовываемой области, положительные значения оси *x* откладываются вправо, оси *y* – снизу-вверх, а оси *z* – из центра к наблюдателю. Направления осей отрисовки показаны на рисунке 3.1.

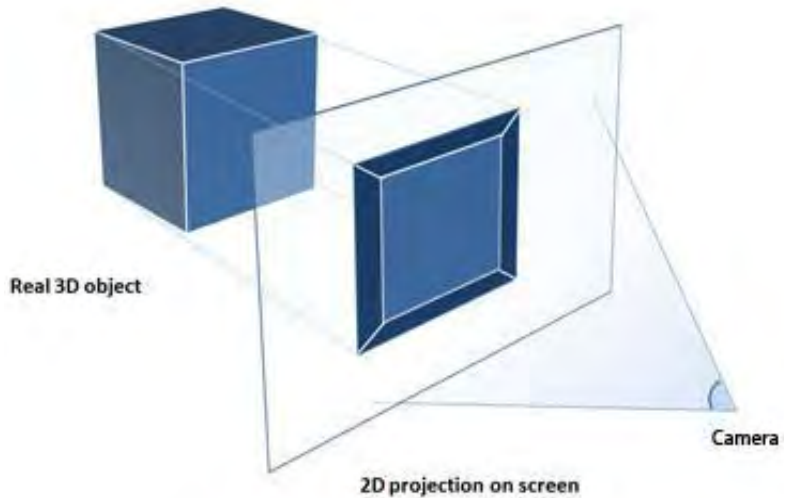

Рисунок 3.1 – 3D-модель

Пространство, определяемое этими осями, является стационарной системой отсчета координат для объектов 3D в приложении WPF. При построении моделей в этом пространстве и создании источников света и камер для их отображения необходимо отличать стационарную систему отсчета координат (или «мировую систему координат») от локальной системы отсчета, которая создается для каждой модели при применении к ней преобразований [1].

Необходимо помнить, что в зависимости от настройки освещения и камеры объекты в мировой системе координат могут выглядеть совсем по-другому или вообще быть невидимыми, но положение камеры не изменяет расположения объектов в мировой системе координат.

3.1.2 Камеры и проекции.

Разработчики, работающие в координатах 2D, привыкли к размещению графических примитивов на двухмерном экране. При создании сцены 3D важно помнить, что фактически создается представление 2D объектов 3D. Поскольку сцена 3D выглядит по-разному в зависимости от точки наблюдения, необходимо указать эту точку наблюдения. Указать эту точку наблюдения для сцены 3D позволяет класс Camera [1].

Другой способ понимания того, как представляется сцена 3D на поверхности 2D, – это описание сцены как проекции на поверхность просмотра. Камера ProjectionCamera позволяет указать различные проекции и их свойства для изменения того, как наблюдатель видит модели 3D.

Камера PerspectiveCamera указывает проекцию сцены в перспективе (рисунок 3.1). Другими словами, камера PerspectiveCamera предоставляет точку схода перспективы. Можно указать положение камеры в пространстве координат сцены, направление и поле зрения камеры и вектор, определяющий направление «вверх» в сцене. Следующая схема иллюстрирует проекции PerspectiveCamera.

Свойства NearPlaneDistance и FarPlaneDistance камеры ProjectionCamera ограничивают диапазон ее проекции. Поскольку камеры могут быть расположены в любом месте сцены, фактически можно расположить камеру внутри модели или очень близко от нее, что усложняет правильное распознавание объекта. Свойство NearPlaneDistance позволяет определить минимальное расстояние от камеры, за которым не будут располагаться объекты. И наоборот, свойство FarPlaneDistance позволяет задать расстояние от камеры, дальше которого объекты не будут нарисованы; это гарантирует, что объекты, расположенные слишком далеко для распознавания, не будут включены в сцену.

3.1.3 Пример создания трехмерной буквы.

Выполнить пример создания графического приложения в системе Windows с использованием технологии WPF. Пример буквы «К» изображен на рисунке 3.2. На рисунках 3.3 и 3.4 показано последовательное создание такой буквы.

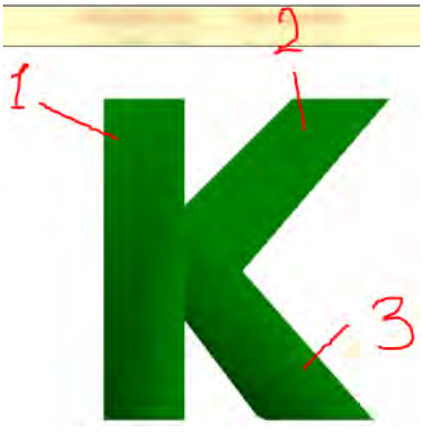

Рисунок 3.2 – Графическое представление буквы «К»

```
        // Добавляем вершины сетки (x, y, z).
        // Средняя часть буквы «К» (1).
        mesh.Positions.Add(new Point3D(-1, -1, 1)); // 0.
        mesh.Positions.Add(new Point3D(0.5, -1, 1)); // 1.
        mesh.Positions.Add(new Point3D(0.5, 5, 1)); // 2.
        mesh.Positions.Add(new Point3D(-1, 5, 1)); // 3.
        mesh.Positions.Add(new Point3D(-1, -1, -1)); // 4.
        mesh.Positions.Add(new Point3D(0.5, -1, -1)); // 5.
        mesh.Positions.Add(new Point3D(0.5, 5, -1)); // 6.
        mesh.Positions.Add(new Point3D(-1, 5, -1)); // 7.

        // верхняя часть (2)
        mesh.Positions.Add(new Point3D(1.5, 1.8, 1)); // 8.
        mesh.Positions.Add(new Point3D(0.5, 3, 1)); // 9.
        mesh.Positions.Add(new Point3D(2.5, 5, 1)); // 10.
        mesh.Positions.Add(new Point3D(4.3, 5, 1)); // 11.
        mesh.Positions.Add(new Point3D(1.5, 1.8, -1)); // 12.
        mesh.Positions.Add(new Point3D(0.5, 3, -1)); // 13.
        mesh.Positions.Add(new Point3D(2.5, 5, -1)); // 14.
        mesh.Positions.Add(new Point3D(4.3, 5, -1)); // 15.

        // нижняя часть (3)
        mesh.Positions.Add(new Point3D(0.5, 1, 1)); // 16.
        mesh.Positions.Add(new Point3D(0.5, 3, 1)); // 17.
        mesh.Positions.Add(new Point3D(0.5, 1, -1)); // 18.
        mesh.Positions.Add(new Point3D(0.5, 3, -1)); // 19.
        mesh.Positions.Add(new Point3D(2, -1, 1)); // 20.
        mesh.Positions.Add(new Point3D(4, -1, 1)); // 21.
        mesh.Positions.Add(new Point3D(2, -1, -1)); // 22.
        mesh.Positions.Add(new Point3D(4, -1, -1)); // 23.

        // Соединяем точки (x, y, z) между собой против часовой стрелки, тем самым
получая треугольник.
        // Создаем стороны средней части буквы треугольниками.
        mesh.TriangleIndices.Add(0); mesh.TriangleIndices.Add(1);
mesh.TriangleIndices.Add(2); // Спереди.
        mesh.TriangleIndices.Add(2); mesh.TriangleIndices.Add(3);
mesh.TriangleIndices.Add(0);
        mesh.TriangleIndices.Add(1); mesh.TriangleIndices.Add(5);
mesh.TriangleIndices.Add(6); // Справа.
        mesh.TriangleIndices.Add(1); mesh.TriangleIndices.Add(6);
mesh.TriangleIndices.Add(2);
        mesh.TriangleIndices.Add(3); mesh.TriangleIndices.Add(2);
mesh.TriangleIndices.Add(6); // Сверху.
        mesh.TriangleIndices.Add(3); mesh.TriangleIndices.Add(6);
mesh.TriangleIndices.Add(7);
        mesh.TriangleIndices.Add(0); mesh.TriangleIndices.Add(5);
mesh.TriangleIndices.Add(1); // Снизу.
        mesh.TriangleIndices.Add(0); mesh.TriangleIndices.Add(4);
mesh.TriangleIndices.Add(5);
        mesh.TriangleIndices.Add(0); mesh.TriangleIndices.Add(3);
mesh.TriangleIndices.Add(7); // Слева.
```

```
mesh.TriangleIndices.Add(0); mesh.TriangleIndices.Add(7);
mesh.TriangleIndices.Add(4);
mesh.TriangleIndices.Add(7); mesh.TriangleIndices.Add(6);
mesh.TriangleIndices.Add(5); // Сзади.
mesh.TriangleIndices.Add(7); mesh.TriangleIndices.Add(5);
mesh.TriangleIndices.Add(4);
```

Рисунок 3.3 – Создание средней части буквы «К»

```
// Создаем стороны верхней части буквы треугольниками.
mesh.TriangleIndices.Add(8); mesh.TriangleIndices.Add(11);
mesh.TriangleIndices.Add(10); // Спереди.
mesh.TriangleIndices.Add(10); mesh.TriangleIndices.Add(9);
mesh.TriangleIndices.Add(8);
mesh.TriangleIndices.Add(11); mesh.TriangleIndices.Add(15);
mesh.TriangleIndices.Add(14); // Справа.
mesh.TriangleIndices.Add(14); mesh.TriangleIndices.Add(10);
mesh.TriangleIndices.Add(11);
mesh.TriangleIndices.Add(10); mesh.TriangleIndices.Add(14);
mesh.TriangleIndices.Add(13); // Сверху.
mesh.TriangleIndices.Add(13); mesh.TriangleIndices.Add(9);
mesh.TriangleIndices.Add(10);
mesh.TriangleIndices.Add(20); mesh.TriangleIndices.Add(22);
mesh.TriangleIndices.Add(23); // Справа.
mesh.TriangleIndices.Add(23); mesh.TriangleIndices.Add(21);
mesh.TriangleIndices.Add(20);
mesh.TriangleIndices.Add(17); mesh.TriangleIndices.Add(21);
mesh.TriangleIndices.Add(23); // Сверху.
mesh.TriangleIndices.Add(23); mesh.TriangleIndices.Add(19);
mesh.TriangleIndices.Add(17);
mesh.TriangleIndices.Add(16); mesh.TriangleIndices.Add(18);
mesh.TriangleIndices.Add(22); // Снизу.
mesh.TriangleIndices.Add(22); mesh.TriangleIndices.Add(20);
mesh.TriangleIndices.Add(16);
mesh.TriangleIndices.Add(18); mesh.TriangleIndices.Add(19);
mesh.TriangleIndices.Add(23); // Сзади.
mesh.TriangleIndices.Add(23); mesh.TriangleIndices.Add(22);
mesh.TriangleIndices.Add(18);
```

Рисунок 3.4 – Создание нижней части буквы «К»

3.2 Выполнение работы

1 Выполнить рассмотренные в [1] и в методических рекомендациях примеры формирования 3D-изображений.

2 Создать приложение для формирования 3D-изображения первых букв своего имени и фамилии с возможностью вращения в трехмерном пространстве. В случае совпадения букв, следует взять первую букву отчества.

Контрольные вопросы

1 Назвать состав и направление осей в 3D-графике WPF.

2 Дать характеристику камерам ProjectionCamera и PerspectiveCamera.

3 Какие свойства объекта камеры позволяют задать минимальное и максимальное расстояния от камеры до первого видимого объекта?

4 Как можно организовать вращение объекта в трехмерном пространстве?

4 Лабораторная работа № 4. Разработка корпоративного приложения в системе Windows с использованием технологий Windows Presentation foundation

Цель работы: освоение основных приемов разработки WPF-приложений на базе страничной организации приложения, создания меню, панели команд, табличных элементов управления и системы команд для выполнения задач приложения.

4.1 Краткие теоретические сведения

Как правило, корпоративное приложение представляет собой программу, реализующую определенную бизнес-задачу (бизнес-функцию). Приложение должно взаимодействовать с данными, которые располагаются в базе данных информационной системы.

Архитектура приложения обычно включает слой представления, бизнес-логики и данных.

Так, слой представления обеспечивает интерфейс пользователя с системой. Интерфейс может быть создан с использованием окон Windows и страниц WPF, которые наполняются различными визуальными элементами контроля.

Слой бизнес-логики приложения должен обеспечивать основную функциональность приложения: формировать бизнес-классы, реализовывать алгоритмы обработки данных, обеспечивать соединение с данными и их кэширование. Реализация данного слоя приложения может быть построена на базе классов, реализующих бизнес-логику, методами классов интерфейсных элементов или методами классов модели данных.

Слой данных должен обеспечить взаимодействие приложения с данными системы управления базами данных. В корпоративных приложениях для этого наиболее целесообразно использовать платформу *ADO.NET Entity Framework* и модель *EDM* (*Entity Data Model*). Модель *EDM* описывает структуру данных независимо от формы хранения.

Для изучения вопросов проектирования корпоративных приложений рассмотрим основные подходы при разработке отдельной функции информационной системы, которая обеспечивает обработку данных по сотрудникам компании.

4.2 Выполнение работы

Для учебного примера используется база данных *TitlePresonal* с небольшим набором таблиц и полей, а функциональность приложения предполагает обеспечение ввода, корректировки и удаление данных о сотрудниках компании. Разрабатываемое приложение должно обеспечивать хранение и обработку следующих данных по сотрудникам компании: фамилия; имя; отчество; должность; дата рождения; телефон; адрес электронной почты.

Функции приложения:

– просмотр данных по сотрудникам;
– ввод данных по новому сотруднику;
– редактирование данных по сотруднику;
– удаление данных по сотруднику;
– поиск данных по сотруднику.

4.2.1 Создание проекта в соответствии с шаблоном «Приложение WPF» и разработка интерфейса пользователя.

Для разработки приложения необходимо создать WPF-проект. В окне «Создать проект» нужно проверить установку библиотеки .NET Framework 4, выбрать шаблоны Windows, а среди имеющихся шаблонов задать приложение WPF и в поле «Имя» ввести имя проекта WpfApplProject.

После нажатия кнопки ОК будет сформирован шаблон проекта. При этом инструментальная система сгенерирует следующий XAML-документ:

```
<Window x:Class="WpfApplProject.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:WpfApplProject"
mc:Ignorable="d"
Title="Информационная система ВУ" Height="450" Width="825">
<Grid>
<Frame x:Name="frame" Content="Frame" Margin="3" Source="/WpfApplProject;component/PageMain.xaml" NavigationUIVisibility="Hidden">
<Frame.Background>
<LinearGradientBrush>
<LinearGradientBrush.GradientStops>
<GradientStop Offset="0.00" Color="#FFFF8686"/>
<GradientStop Offset="1.00" Color="#FFA5B1FF"/>
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>

</Frame.Background>
</Frame>
</Grid>
</Window>
```

Для вставки страницы внутрь окна будем использовать класс Frame. Автоматически будет сгенерирован экземпляр класса Frame с фиксированными границами (рисунки 4.1 и 4.2).

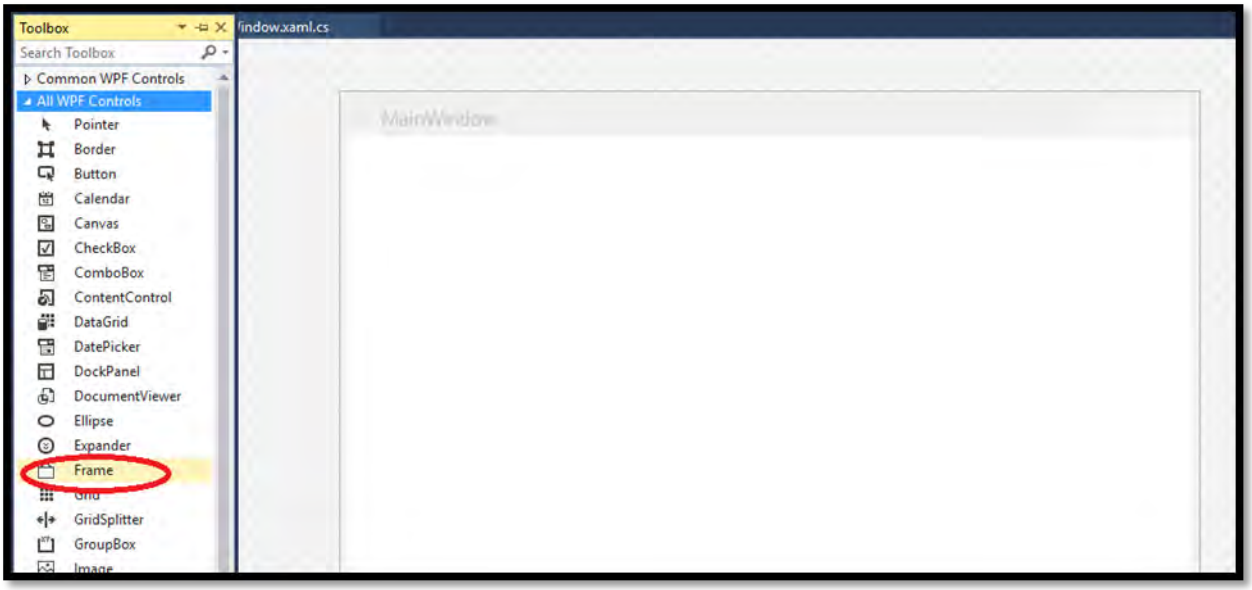

Рисунок 4.1 – Выбор класса Frame в панели инструментов

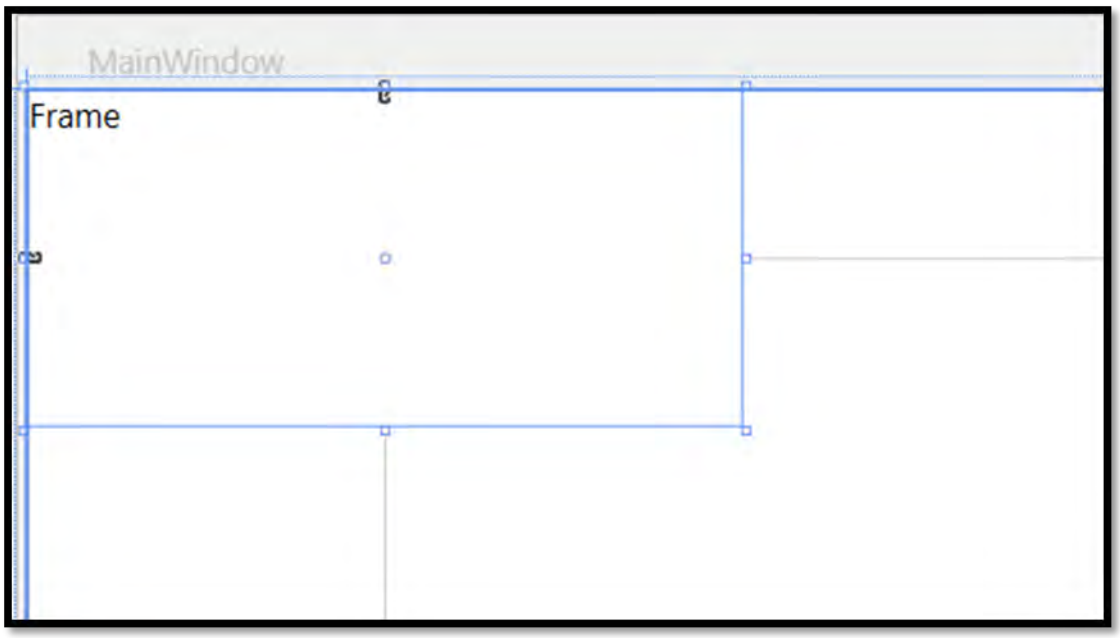

Рисунок 4.2 – Фрейм с фиксированными границами

В XAML-документ проекта будет добавлена следующая строка:

```
<Frame Height="100" HorizontalAlignment="Left"
Name="frame1" VerticalAlignment="Top" Width="200" />
```

С учетом того, что создается страничное приложение, размеры фрейма не нужно фиксировать, поэтому изменим описание свойств фрейма:

```
<Frame Name="frame1" Margin="3" />
```

В результате фрейм заполнит всё окно.

Созданный фрейм необходим для размещения в нем страницы WPF-приложения. Класс Page допускает наличие только одного вложенного элемента. Он не является элементом управления содержимым и наследуется от класса FrameworkElement. Класс Page имеет небольшой набор дополнительных свойств, которые позволяют настраивать его внешний вид, взаимодействовать с контейнером и использовать навигацию. Для перехода на другую страницу необходимо применять навигацию.

Чтобы добавить в проект начальную страницу, нужно в обозревателе решений щелкнуть правой кнопкой мыши на проекте WpfApplProject. В контекстном меню выбрать пункт «Добавить» (1), а в раскрывающемся меню – пункт «Страница» (2) (рисунок 4.3).

В окне «Добавление нового элемента» необходимо выбрать шаблон «Страница (WPF)» (1) и задать имя страницы PageMain (2) (рисунок 4.4).

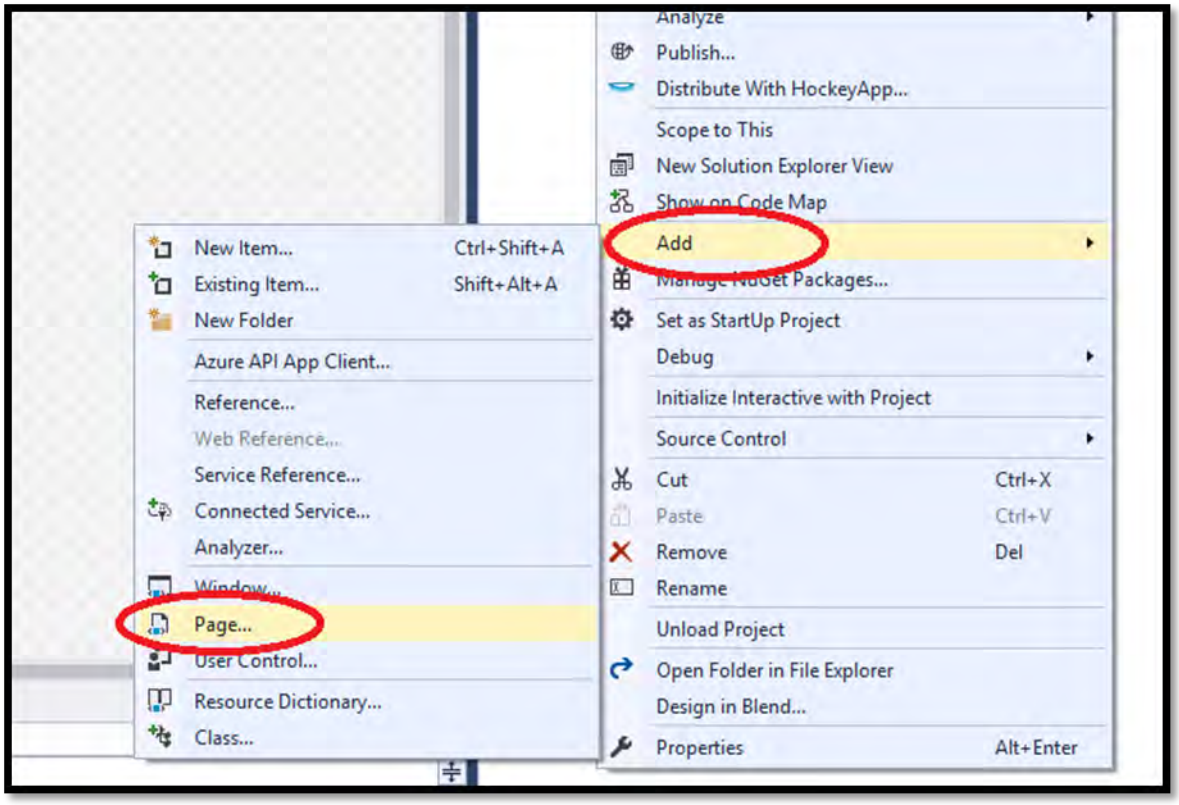

Рисунок 4.3 – Меню создания страницы

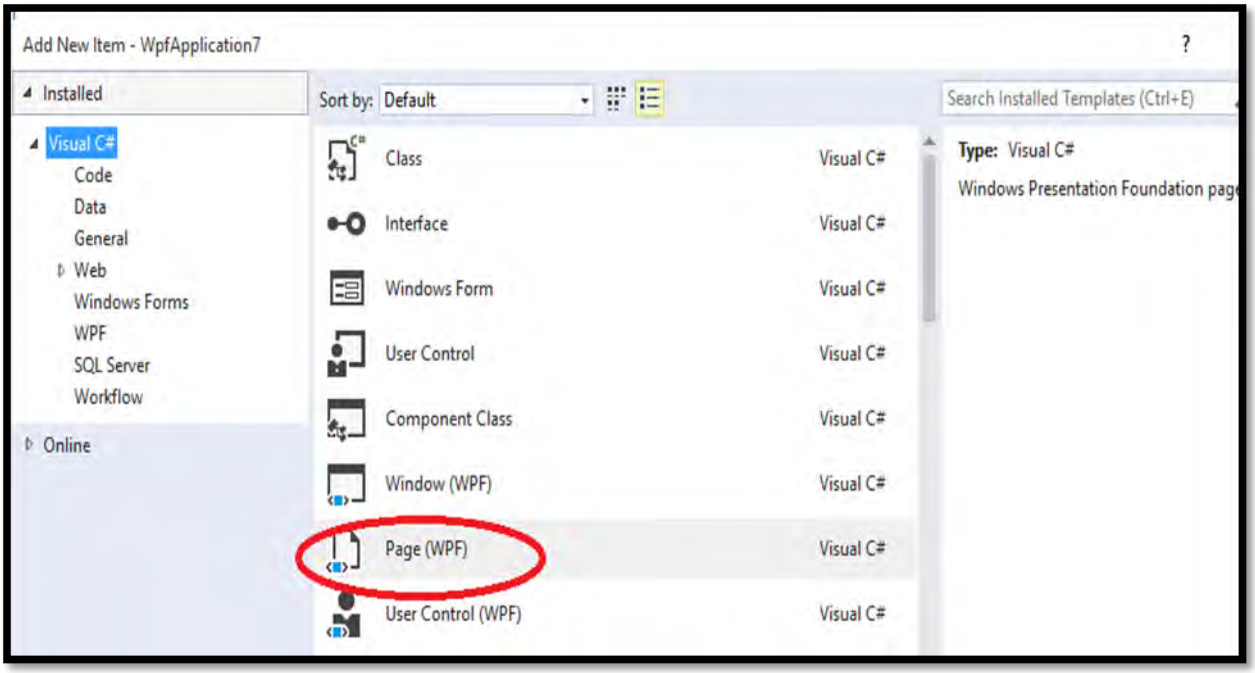

Рисунок 4.4 – Окно добавления нового элемента

В дизайнере проекта сгенерируется страница PageMain.xaml.

В сгенерированной странице в качестве контейнера верхнего уровня используется Grid. Следует заменить Grid на контейнер StackPanel.

Главная страница приложения в дизайнере представлена на рисунке 4.5.

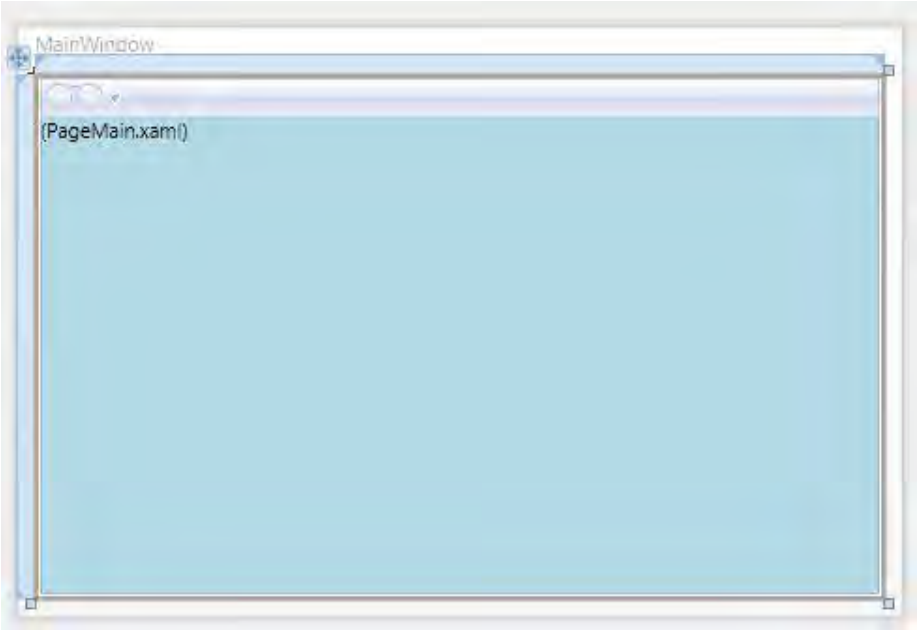

Рисунок 4.5 – Главная страница WPF-приложения в дизайнере

Необходимо изменить свойство Title окна класса Window в соответствии с вариантом лабораторной работы.

Основная страница должна обеспечивать переход на другие страницы и выход из системы. Для перехода на другие страницы нужно использовать гиперссылки. Элемент гиперссылки, соответствующий объекту класса Hyperlink, определяется следующей строкой:

```
<Hyperlink >Текст Гиперссылки</Hyperlink>
```

Класс Hyperlink имеет свойство NavigateUri. Данное свойство определяет, на какую страницу будет переходить приложение при щелчке на соответствующей гиперссылке. Например, NavigateUri="Page2.xaml".

В WPF гиперссылки являются не отдельными, а внутристроковыми потоковыми элементами, которые должны размещаться внутри другого поддерживающего их элемента. Это можно сделать, например, в элементе TextBlock, который для гиперссылки является контейнером.

На первой странице следует создать гиперссылки для перехода на страницы приложения в соответствии с заданием (пример на рисунке 4.6).

Необходимо использовать StackPanel для размещения в нём Textblock.

Создание основного меню. Основное меню создается с помощью класса Menu, который представляет Windows элементы управления меню, позволяющие иерархически организовать элементы, связанные с командами и обработчиками событий. Меню формируют из объектов MenuItem (имя пункта меню) и Separator (разделитель). Класс MenuItem имеет свойство Header, которое определяет текст элемента меню. Данный класс может хранить коллекцию объектов MenuItem, которая соответствует подпунктам меню. Класс Separator отображает горизонтальную линию, разделяющую пункты меню.

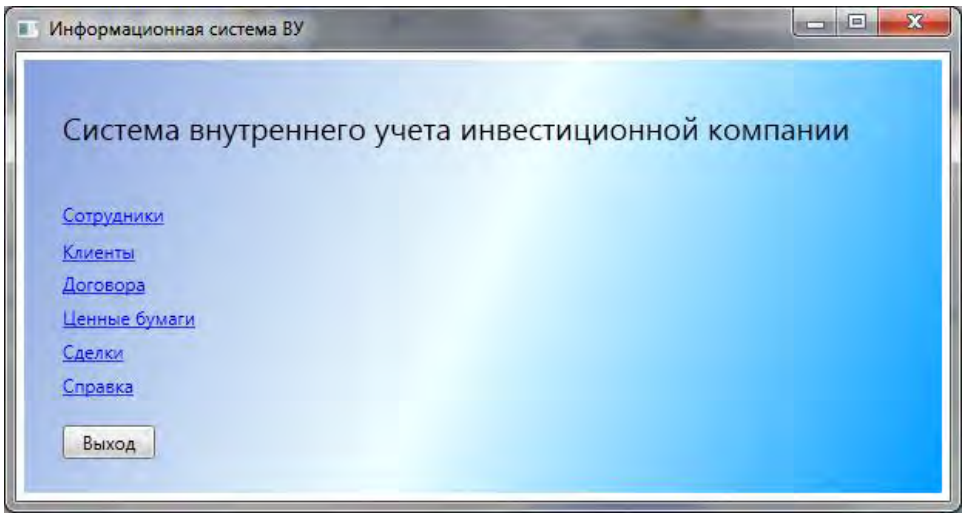

Рисунок 4.6 – Начальная страница WPF-приложения с гиперссылками

Нужно добавить в StackPanel страницы приложения XAML-описание меню, которое на верхнем уровне будет содержать, например, два пункта: «Действие» и «Отчет». Пункт «Действие» включает подпункты «Отменить», «Создать», «Редактировать», «Сохранить», «Найти» и «Удалить». Между пунктами «Отменить», «Создать» и пунктами «Найти», «Удалить» необходимо добавить разделительные линии.

```
<Menu FontSize="14" FontFamily="Times New Roman">
<MenuItem Header="Действие" >
<MenuItem Header="Отменить"></MenuItem>
<Separator></Separator>
<MenuItem Header="Создать"></MenuItem>
<MenuItem Header="Редактировать"></MenuItem>
<MenuItem Header="Сохранить"></MenuItem>
<Separator></Separator>
<MenuItem Header="Удалить"></MenuItem>
</MenuItem>
<MenuItem Header="Отчет"></MenuItem>
</Menu>
```

Создание панели инструментов. Панель инструментов представляет специализированный контейнер для хранения коллекции элементов, обычно кнопок. Следует расположить в панели инструментов кнопки, функциональное назначение которых будет соответствовать подпунктам меню «Действие», то есть «Отменить», «Создать», «Редактировать», «Сохранить», «Найти» и «Удалить».

На лицевой стороне кнопок нужно поместить графическое изображение соответствующего действия. Для этого необходимо добавить в файл проекта папку Images и в неё включить графические объекты, которые можно найти в библиотеке графических объектов Visual studio (папка VS2010 ImageLibrary).

После добавления графических файлов в проект они будут отображены в обозревателе решений (рисунок 4.7).

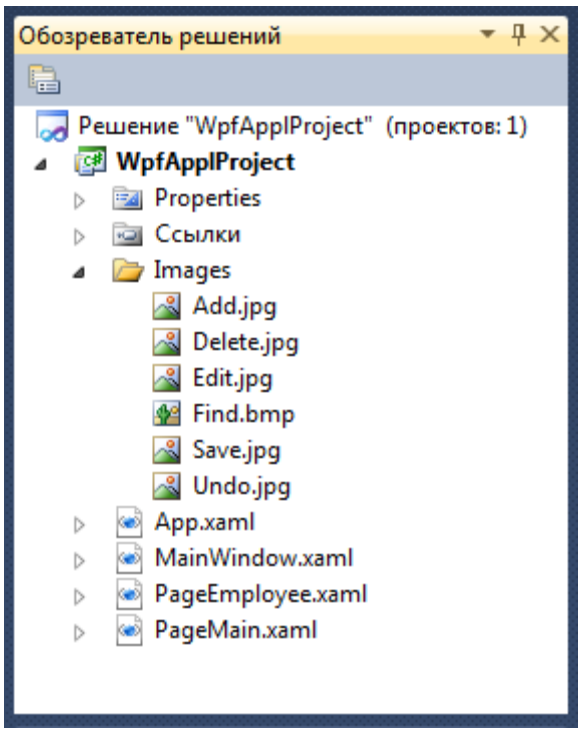

Рисунок 4.7 – Включение в проект папки Images с файлами рисунков

Для каждой кнопки нужно задать свойство «Name» – имя объекта в программе и свойство «ToolTip» с текстом всплывающей подсказки при наведении указателя мыши на кнопку.

Свойство «Margin» определяет внешние отступы для кнопки. Задание графического объекта для кнопки осуществляется определением для объекта Image источника Source, который должен соответствовать полному пути к графическому файлу. Результат представлен на рисунке 4.8.

```
<ToolBar Name="ToolBar1" Margin="3">
<Button Name="btnUndo" ToolTip="Отменить редактирование/создание"
Margin="5,2,5,2" Width="30" Height="30">
<Button.OpacityMask>
<ImageBrush Stretch="Uniform" ImageSource="icons/undo.png"/>
</Button.OpacityMask>
<Button.Background>
<ImageBrush Stretch="Uniform" ImageSource="icons/undo.png"/>
</Button.Background>
…
</Button>
</ToolBar>
```

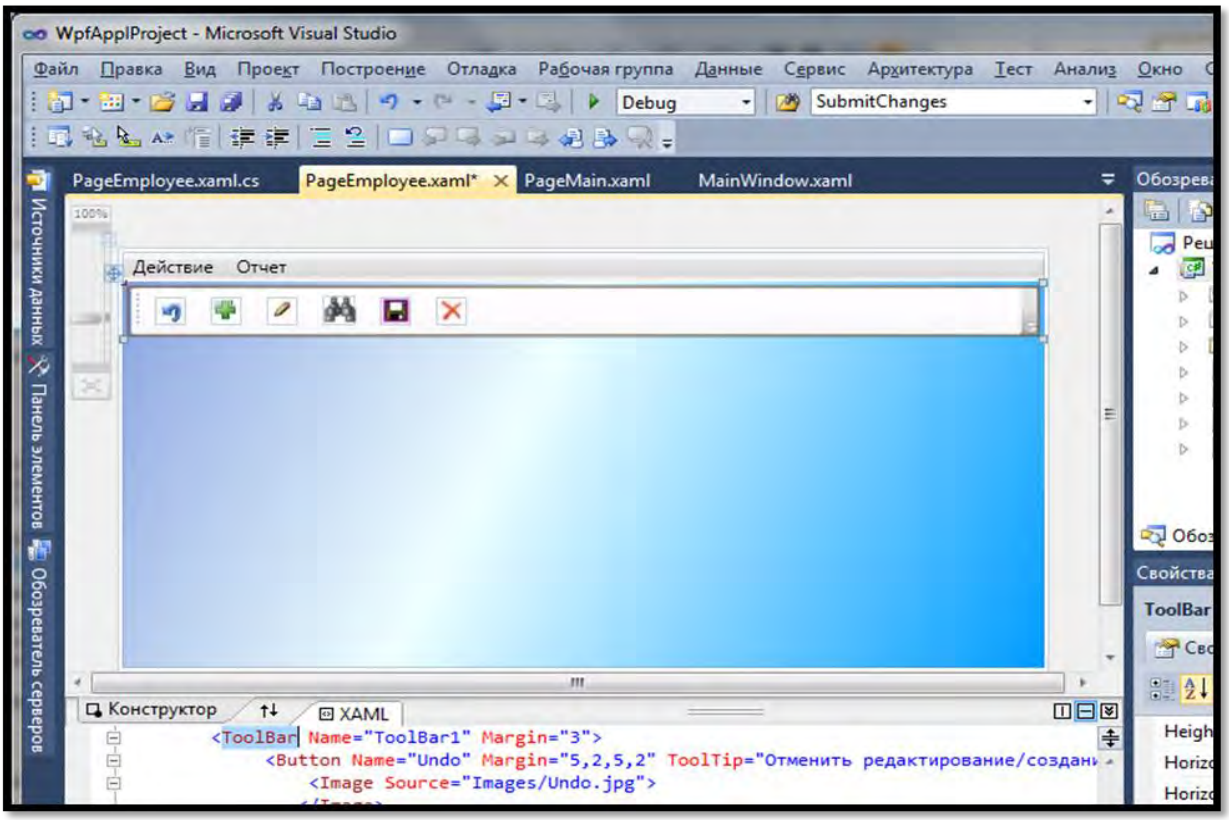

Рисунок 4.8 – Страница PageEmployee с панелью инструментов

При проектировании интерфейсных элементов для страницы приложения используются элементы контроля ListBox, ListView, TextBox, TextBlock, ComboBox, DataGrid и другие.

Далее разрабатывается отдельно каждая страница WPF.

4.3 Варианты заданий

Вариант 1. Информационная подсистема ведения счетов клиентов.

БД включает таблицы: Account, Agreement, Bank, TypeAccount.

Назначение подсистемы ведения счетов клиентов – поддержание в актуальном состоянии инвестиционных счетов клиентов.

Назначение атрибутов таблицы Счет – Account:

– ID – суррогатный ключ;
– TypeID – внешний ключ для связи с таблицей Type;
– BankID – внешний ключ для связи с таблицей Bank;
– AgreementID – внешний ключ для связи с таблицей Agreement;
– Account – номер инвестиционного счета.

Назначение атрибутов таблицы Тип счета – TypeAccount:

– ID – суррогатный ключ;
– TypeAccount – тип счета.

Назначение атрибутов таблицы Банк – Bank:

– ID – суррогатный ключ;
– NameFull – полное наименование банка;

– NameShort – краткое наименование банка;
– Inn – ИНН банка;
– Bik – БИК банка;
– CorAccount – номер корсчета;
– Account – номер счета;
– City – город.

Назначение атрибутов таблицы Договор – Agreement:
– ID – суррогатный ключ;
– PersonID – внешний ключ для связи с таблицей Person;
– TypeID – внешний ключ для связи с таблицей Type;
– StatusID – внешний ключ для связи с таблицей Status;
– Number – номер договора;
– DataOpen- – дата заключения договора;
– DataClouse – дата закрытия договора;
– Note – пояснения.

Вариант 2. Информационная подсистема ведения адресов клиентов.

БД включает таблицы: City, Region, Address, Country.

Назначение подсистемы ведения адресов клиентов – поддержание в актуальном состоянии адресов клиентов.

Назначение атрибутов таблицы Адрес – Address:
– ID – суррогатный ключ;
– Index Address – адресный индекс;
– PersonID – внешний ключ для связи с таблицей Person;
– CountryID – внешний ключ для связи с таблицей Country;
– RegionID – внешний ключ для связи с таблицей Region;
– CityID – внешний ключ для связи с таблицей City;
– Street – наименование улицы;
– Bulding – номер строения, дома;
– Office – номер офиса.

Назначение атрибутов таблицы Город – City:
– ID – суррогатный ключ;
– RegionID – внешний ключ для связи с таблицей Region;
– CountryID – внешний ключ для связи с таблицей Country;
– City – город.

Назначение атрибутов таблицы Регион – Region:
– ID – суррогатный ключ;
– CountryID – внешний ключ для связи с таблицей Country;
– Region – регион.
– Назначение атрибутов таблицы Страна – Country:
– ID – суррогатный ключ;
– CountryFull – полное наименование страны;
– CountryShort – краткое наименование страны.

Вариант 3. Информационная подсистема ведения договоров клиентов.

БД включает таблицы: Person, Agreement, StatusAgreement, TypeAgreement.

Назначение подсистемы ведения договоров клиентов – поддержание в актуальном состоянии договоров клиентов.

Назначение атрибутов таблицы Договор –Agreement:

– ID – суррогатный ключ;
– PersonID – внешний ключ для связи с таблицей Person;
– TypeID – внешний ключ для связи с таблицей Type;
– StatusID – внешний ключ для связи с таблицей Status;
– Number – номер договора;
– DataOpen – дата заключения договора;
– DataClouse – дата закрытия договора;
– Note – пояснения.

Назначение атрибутов таблицы Статус договор – StatusAgrement:

– ID – суррогатный ключ;
– Status – статус договора.

Назначение атрибутов таблицы Клиент – Person:

– ID – суррогатный ключ;
– OrgLicenseID – внешний ключ для связи с таблицей OrgLicense;
– VerietyID – внешний ключ для связи с таблицей Veriety;
– StatusID – внешний ключ для связи с таблицей Status;
– Inn – ИНН клиента;
– Type – тип клиента;
– Shifer – шифр клиента;
– Data – дата регистрации клиента.

Вариант 4. Информационная подсистема ведения клиентов – физических лиц.

БД включает таблицы: Person, Citizen, Document.

Назначение подсистемы ведения клиентов – поддержание в актуальном состоянии информации по клиентам – физическим лицам.

Назначение атрибутов таблицы Клиент – Person:

– ID – суррогатный ключ;
– OrgLicenseID – внешний ключ для связи с таблицей OrgLicense;
– VerietyID – внешний ключ для связи с таблицей Veriety;
– StatusID – внешний ключ для связи с таблицей Status;
– Inn – ИНН клиента;
– Type – тип клиента;
– Shifer – шифр клиента;
– Data – дата регистрации клиента.

Назначение атрибутов таблицы Физическое лицо – Citizen:

– ID – суррогатный ключ;
– DocumentID – внешний ключ для связи с таблицей Document;
– SurName – фамилия клиента;
– Name – имя клиента;

– Patronic – отчество клиента;
– Number – номер документа, удостоверяющего личность;
– Seriy – серия документа, удостоверяющего личность;
– Organ – орган, выдавший документ, удостоверяющий личность;
– Data – дата выдачи документа, удостоверяющего личность.

Назначение атрибутов таблицы Документ – Document:
– ID – суррогатный ключ;
– Document – наименование документа, удостоверяющего личность.

Контрольные вопросы

1 Сколько можно вложить элементов в класс Page?
2 Пояснить назначение свойства Content класса Page.
3 В каких контейнерах можно размещать страницы WPF?
4 Из каких объектов можно сформировать меню приложения?
5 Какие элементы контроля используются для перехода между страницами приложения?

5 Лабораторная работа № 5. Разработка ASP.Net-приложения со стандартными элементами управления

Цель работы: изучение различных элементов управления в приложениях ASP.NET. Разработка простого приложения со стандартными элементами управления на базе технологии ASP.NET.

5.1 Краткие теоретические сведения

Одна из самых важных задач Web-разработчика – получение и обработка данных, введенных пользователем. Информация посылается серверу через форму. Форма содержит элементы управления, которые позволяют различными способами вводить информацию.

Формы ASP.NET отличаются от обычных форм наличием свойства *runat = "server"*. Они обрабатываются ASP.NET на сервере. На странице находятся элементы управления. ASP.NET позволяет запоминать состояние элементов управления, то есть текст, который был введен, или выбранный переключатель, передавать его на сервер и обратно на страницу после ее обновления.

Элемент управления, для которого использован атрибут runat="server", становится доступным из программного кода в файле codebehind, а на события этого элемента управления реагирует сервер. Если данный атрибут убрать, элемент управления станет обычным элементом управления HTML.

Второй обязательный атрибут — это его идентификатор, или ID. Он нужен, чтобы обращаться к элементу в программе.

Существуют две группы элементов управления.

1 Элементы управления HTML.

Элементы управления HTML являются наследниками класса System.Web.UI.HtmlControls.HtmlControl. Они непосредственно отображаются в виде элементов разметки HTML. Их отображение не зависит от типа браузера. Свойства таких элементов полностью соответствуют атрибутам тегов HTML.

2 Стандартные или серверные элементы управления.

Серверные элементы мощнее, они привязаны не к разметке, а к функциональности, которую нужно обеспечить. Многие элементы не имеют аналогов в HTML, например, календарь. Их отрисовка полностью контролируется ASP.NET. Перехватывая события PreRender, Init, Load, можно вмешаться в этот процесс. Объявления серверного элемента управления начинаются с блока <asp:тип> и заканчиваются </asp:тип>.

В таблице 5.1 приведены элементы управления, которые имеют пару среди тегов HTML. Некоторые элементы генерируют не только HTML-код, но и JavaScript.

Таблица 5.1 – Соответствие некоторых *серверных элементов управления* тегам HTML

Элемент управления ASP.NET	Соответствующий тег HTML	Назначение
<asp:Label>	<Span>	Отобразить текст
<asp:ListBox>	<Select>	Список выбора
<asp:DropDownList>	<Select>	Выпадающий список
<asp:TextBox>	<Input Type="Text"> <Input Type="Password"> <Textarea>	Строка редактирования Поле редактирования
<asp:HiddenField>	<Input Type="Hidden">	Невидимое поле
<asp:RadioButton>,		
<asp:RadioButtonList>	<Input Type="Radio">	Переключатель, список переключателей
<asp:CheckBox>,		
<asp:CheckBoxList>	<Input Type="CheckBox">	Флажок, список флажков
<asp:Button>	<Input Type="button"> <Input Type="submit">	Командная кнопка
<asp:Image>	<img>	Изображение
<asp:ImageButton>	<input type="image">	Кнопка-изображение
<asp:Table>	<Table>	Таблица
<asp:Panel>	<Div>	Контейнер
<asp:HyperLink>	<A Href>	Гиперссылка

5.2 Выполнение работы

Создать Web-форму, аналогичную представленной на рисунке 5.1. При нажатии на кнопку «Посчитать» должны рассчитываться проценты, которые должны быть уплачены по кредиту. В данном случае будем считать, что проценты рассчитываются по формуле

(Сумма кредита) · ((Процентная ставка / 100) · ((Количество дней) / 360).

Пользователь должен выбирать только из стандартных ставок по кредиту в 12, 15 и 20 % годовых (в ниспадающем списке).

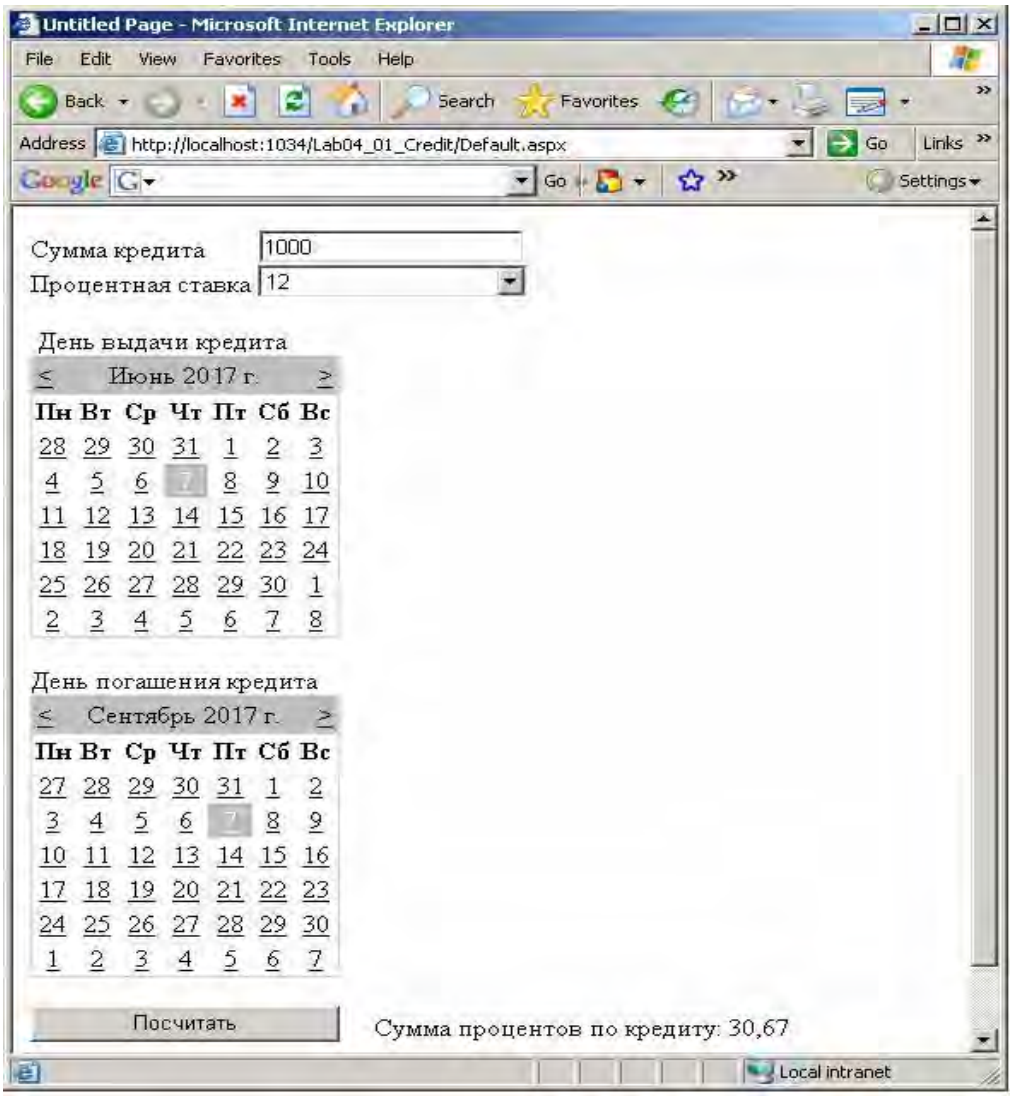

Рисунок 5.1 – Пример Web-формы

Контрольные вопросы

1 Чем отличаются серверные элементы управления от стандартных?
2 Чем отличаются приложения Web Forms и Windows Forms?
3 Какие события жизненного цикла Web-приложения вы знаете?
4 Какие существуют три типа событий серверных элементов управления?
5 Какие этапы можно выделить в жизненном цикле страницы ASP.NET?

6 Лабораторная работа № 6. Разработка ASP.Net-приложения с валидаторами

Цель работы: изучение различных способов валидации, применяемых в ASP.NET. Создание Web-приложения с использованием различных валидаторов.

6.1 Краткие теоретические сведения

Валидация – это процесс проверки данных на соответствие различным критериям.

Существуют два вида валидации данных, введенных пользователем:

1) клиентская валидация;

2) серверная валидация.

Клиентская валидация производится в браузере на стороне клиента. Как правило, логика валидации на стороне клиента реализуется посредством сценариев JavaScript, которые запускаются внутри браузера.

Серверная валидация работает в рамках программного кода, размещенного на стороне сервера. Здесь проверяются всевозможные случаи, в том числе те, которые уже были проверены на стороне клиента. Кроме тривиальных проверок, на стороне сервера могут работать более сложные алгоритмы.

При разработке приложений на базе ASP.NET Web Forms валидация данных, введенных пользователем, осуществляется при помощи специальных элементов управления – валидаторов. Валидаторы полностью реализуют логику проверки достоверности введенных пользователем данных. Каждый валидатор проверяет одно атомарное условие. Если требуется определить более сложную логику проверки, то на странице используется несколько валидаторов одновременно.

В составе ASP.NET Web Forms доступны следующие элементы управления для проверки пользовательского ввода:

— RequiredFieldValidator позволяет проверить, ввел ли пользователь данные в поле ввода или нет;

— RangeValidator позволяет проверить значение поля ввода на заданный диапазон. Для задания границ диапазона используются свойства MinimumValue и MaximumValue. При этом объект RangeValidator способен работать с различными типами данных;

— CompareValidator позволяет сравнивать значение одного элемента управления с константой или значением другого элемента управления;

— RegularExpressionValidator позволяет проверять текст путем сопоставления с образцом, определенным в регулярном выражении;

— CustomValidator позволяет выполнять специальные процедуры проверки как на стороне клиента, так и на стороне сервера.

6.2 Выполнение работы

Задание 1. Создание простых валидаторов

1 Создать новое приложение ASP.NET Web Forms.

2 Создать новую страницу в рамках приложения.

3 Добавить на страницу элемент управления TextBox.

4 Добавить на страницу элемент управления Button.

5 Добавить на страницу сразу за TextBox валидатор RequiredFieldValidator.

6 Установить свойство ControlToValidate валидатора RequiredFieldValidator равным имени элемента управления TextBox.

7 Задать сообщение об ошибке, которое будет выводиться при срабатывании валидатора. Для этого следует использовать свойство ErrorMessage.

8 Код валидатора в итоге должен выглядеть следующим образом:

```
<asp:TextBox ID="TextBox1" runat="server"/>
    <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
      ErrorMessage="Поле не должно быть пустым" ControlToValidate="TextBox1" />
    <br />
    <asp:Button ID="Button1" runat="server" Text="Button" />
```

9 Запустить приложение и убедиться, что при нажатии на кнопку выдается сообщение об ошибке, если поле ввода остается пустым.

10 Добавить на страницу валидатор RangeValidator.

11 Установить свойство ControlToValidate валидатора RangeValidator равным имени элемента управления TextBox.

12 Задать сообщение об ошибке, которое будет выводиться при срабатывании валидатора. Для этого следует использовать свойство ErrorMessage.

13 Указать тип проверяемого значения валидатора RangeValidator равным целочисленным значениям (свойство Type следует установить в значение «Integer»).

14 Код валидатора в итоге должен выглядеть следующим образом:

```
<asp:TextBox ID="TextBox1" runat="server"/>
    <asp:RangeValidator ID="RangeValidator1" runat="server"
ControlToValidate="TextBox1"
      ErrorMessage="Значение должно быть в диапазоне от 0 до 5"
       MaximumValue="5" MinimumValue="0"></asp:RangeValidator>
    <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
      ErrorMessage="Поле не должно быть пустым" ControlToValidate="TextBox1" />

    <br />
    <asp:Button ID="Button1" runat="server" Text="Button" />
```

15 Запустить приложение и убедиться, что при нажатии на кнопку выдается сообщение об ошибке, если поле ввода содержит значение, не входящее в диапазон от 0 до 10.

Задание 2. Использование валидаторов сравнения

1 Создать новое приложение ASP.NET Web Forms.

2 Создать новую страницу в рамках приложения.

3 Добавить на страницу элемент управления TextBox.

4 Добавить на страницу элемент управления Button.

5 Добавить на страницу валидатор CompareValidator.

6 Установить свойство ControlToValidate валидатора CompareValidator равным имени элемента управления TextBox.

7 Указать тип проверяемого значения валидатора CompareValidator равным целочисленным значениям (свойство Type следует установить в значение «Integer»).

8 Установить операцию сравнения (свойство Operator) равным операции «больше чем» (значение GreaterThan).

9 Установить значение для сравнения (ValueToCompare) равным 10.

10 Код валидатора в итоге должен выглядеть следующим образом:

```
<asp:TextBox ID="TextBox1" runat="server"/>
<asp:CompareValidator ID="CompareValidator1" runat="server"
ControlToValidate="TextBox1"
ErrorMessage="CompareValidator" Operator="GreaterThan"
ValueToCompare="10"
</asp:CompareValidator>

<br />
<asp:Button ID="Button1" runat="server" Text="Button" />
```

11 Запустить приложение и убедиться в том, что валидатор выдает ошибку в случае, если значение меньше или равно 10.

12 Добавить на страницу элемент управления TextBox.

13 Удалить значение для сравнения из свойства ValueToCompare.

14 Установить значение ControlToCompare валидатора CompareValidator равным имени поля ввода, созданного на шаге № 12.

15 Код валидатора в итоге должен выглядеть следующим образом:

```
<asp:TextBox ID="TextBox1" runat="server"/>
<asp:TextBox ID="TextBox2" runat="server"/>
<asp:CompareValidator ID="CompareValidator1" runat="server"
ControlToValidate="TextBox1"
ErrorMessage="CompareValidator" Operator="GreaterThan"
ValueToCompare="TextBox2">

</asp:CompareValidator>

<br />
<asp:Button ID="Button1" runat="server" Text="Button" />
```

16 Запустить приложение и убедиться в том, что валидатор сообщает об ошибке, если значение одного валидатора меньше, чем значение другого.

Задание 3. Работа с валидаторами со сложной логикой (CustomValidator)

1 Создать новое приложение ASP.NET Web Forms.

2 Создать новую страницу в рамках приложения.

3 Добавить на страницу элемент управления TextBox.

4 Добавить на страницу элемент управления Button.

5 Добавить на страницу валидатор CustomValidator.

6 Установить свойство ControlToValidate валидатора CustomValidator равным имени элемента управления TextBox.

7 Определить обработчик события валидации для собственного валидатора (путем двойного нажатия на валидатор в редакторе Visual Studio).

```
protected void CustomValidator1_ServerValidate(object source, ServerValidateEventArgs args)
        {

        }
```

8 В этом обработчике следует определить логику валидации. Для получения значения из поля ввода, а также установления результата необходимо использовать параметр args. Для получения значения для проверки нужно воспользоваться свойством Value. Для установки результата проверки следует установить значение свойства IsValid. Далее определяется логика, согласно которой проверяется, чтобы строка была не длиннее 10 символов.

```
protected void CustomValidator1_ServerValidate(object source, ServerValidateEventArgs args)
        {
           if (args.Value.Length < 10)
           { args.IsValid = true;
           }
           else
           {
              args.IsValid = false;
           }
        }
```

9 Запустить приложение и убедиться в том, что валидатор возвращает ошибку при значении поля ввода, длина которого больше 10.

Задание 4. Разработать форму для ввода и редактирования информации

Разработать форму, представленную на рисунке 6.1.

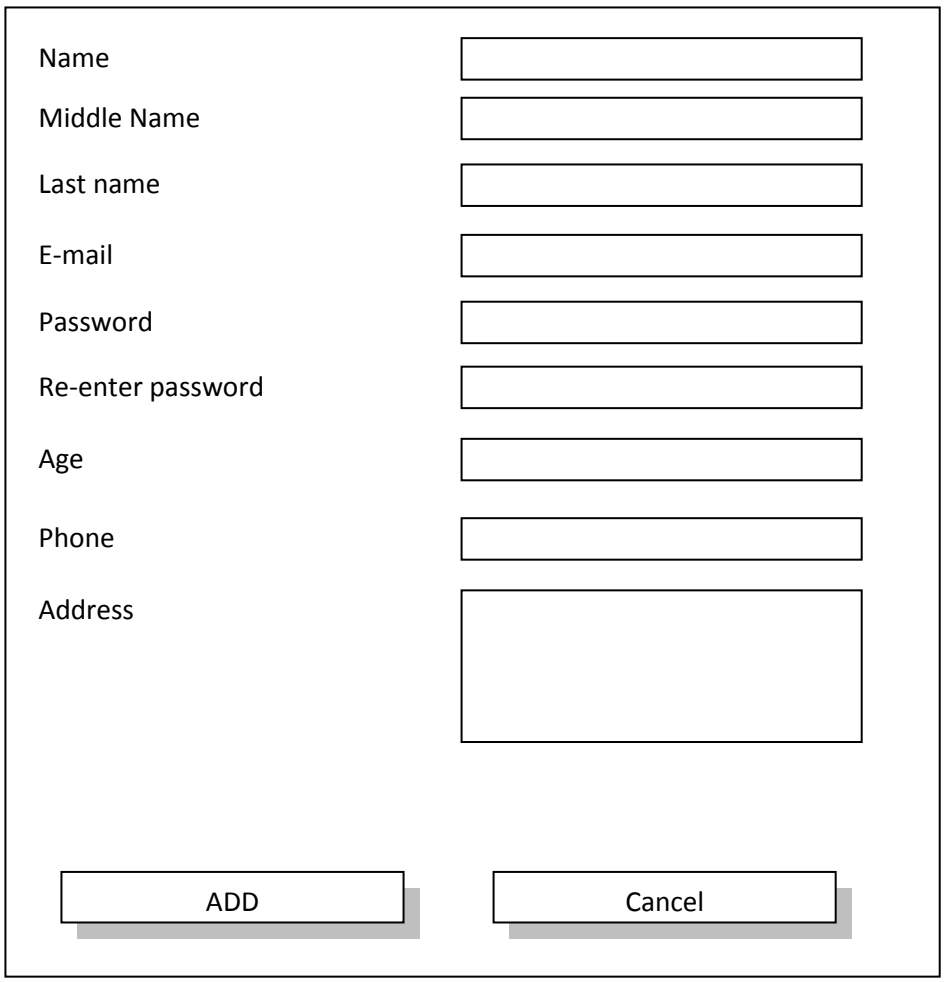

Рисунок 6.1 – Приблизительный вид формы

К разработанной форме добавить валидацию введенных данных.

1 Все поля обязательны, кроме поля Middle Name.

2 E-mail должен быть введен в соответствующем формате.

3 Поле password должно содержать комбинацию букв и цифр длиной не менее 6 (присутствие цифр в пароле обязательно).

4 Поля password и re-enter password должны содержать одинаковый текст.

5 Поле age должно содержать число в диапазоне от 18 до 99.

6 Поле phone должно содержать номер телефона в формате +(nnn)-nn-nnn-nn-nn.

7 Поле Address должно содержать не менее 20 символов.

Контрольные вопросы

1 Что такое валидация данных?

2 Почему необходимо выполнять проверку данных, введенных пользователем?

3 Какие виды валидации существуют для Web-приложений?

4 Чем серверная валидация отличается от клиентской?

7 Лабораторная работа № 7. Разработка ASP.Net-приложения с Master Page

Цель работы: изучение работы MasterPage. Создание Web-страницы с помощью MasterPage. Создание панели навигации по сайту.

7.1 Краткие теоретические сведения

7.1.1 Мастер-страницы.

Мастер-страницы представляют собой шаблоны Web-страниц, которые могут определять фиксированное содержимое и объявлять часть Web-страницы, куда можно помещать нестандартное содержимое.

В ASP.NET определены два новых типа страниц: мастер-страницы и страницы содержимого.

Мастер-страница представляет собой шаблон страницы. Как и обычная Web-страница ASP.NET, она может содержать любую комбинацию HTML, Web-элементов управления и даже кода.

Различие между ними заключается в том, что Web-формы начинаются с директивы Page, а мастер-страница – с директивы Master:

```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile= "SiteTemplate.master.cs" Inherits="SiteTemplate" %>
```

Другое различие между мастер-страницами и обычными Web-формами состоит в том, что мастер-страницы могут использовать элемент управления *ContentPlaceHolder,* который является частью страницы, в которую страница содержимого может вставлять содержимое.

Каждая *страница содержимого* ссылается на одну мастер-страницу и получает ее компоновку и содержимое. Также страница содержимого может добавлять характерное для страницы содержимое в любые заполнители.

Чтобы предоставить содержимое для элемента управления *ContentPlaceHolder,* используется другой специализированный элемент управления, называемый *Content*. Элемент управления ContentPlaceHolder и Content связаны отношением «один к одному». Для каждого элемента управления ContentPlaceHolder в мастер-странице страница содержимого предоставляет соответствующий элемент управления Content.

7.1.2 Навигация по страницам (карта сайта).

Карты сайта целесообразно использовать в случае, когда Web-приложение содержит большое количество страниц. Карты сайта предлагают удобный механизм определения структуры страниц приложения, а также ее отображение с помощью нескольких элементов управления. Эти элементы управления расположены в разделе *Navigation* панели *Toolbox*. Основных элементов управления три – *SiteMapPath, Menu, TreeView.* Для использования любого из перечисленных компонентов необходимо определить структуру приложения,

которая хранится отдельно.

В качестве хранилища структуры обычно выступает XML-файл. В VisualStudio существует уже определенная структура XML-файла, предназначенная для хранения структуры приложения, – это файлы типа .sitemap.

Карта сайта должна начинаться с корневого узла <siteMap>. Элементы структуры описываются в тегах <siteMapNode>. С помощью данных тегов можно указывать иерархию элементов Web-приложения: для этого их просто необходимо расположить внутри соответствующего тега <siteMapNode>. Каждому элементу соответствуют три свойства: *url, title, description.* Их назначение очевидно: *url* используется для указания интернет-адреса страницы, которой соответствует этот элемент, *title* задает наименование элемента, отображаемое элементом управления, *description* – описание элемента, которое отображается в виде всплывающей подсказки при наведении указателя мыши на соответствующий элемент.

В качестве примера простейшей карты сайта создадим иерархию страниц, состоящую из шести позиций. Самым верхним элементом иерархии будет домашняя страница, ссылающаяся на файл Default.aspx. Все остальные элементы будут вложены в него. Кроме того, в элемент «Страница 4» должны быть вложены еще два элемента. Пример карты сайта, описывающей данную структуру, приведен далее.

```
<?xmlversion="1.0" encoding="utf-8" ?>
<siteMapxmlns="http://schemas.microsoft.com/AspNet/
 SiteMap-File-1.0" >
<siteMapNodeurl="Default.aspx" title="Домашняя" description="">
<siteMapNodeurl="Default2.aspx" title="Страница 2"
description="Перейти к странице 2" />
<siteMapNodeurl="Default3.aspx" title="Страница 3"
    description="" />
<siteMapNodeurl="Default4.aspx" title="Страница 4">
<siteMapNodeurl="Default5.aspx" title="Страница 5"/>
<siteMapNodeurl="Default6.aspx" title="Страница 6"/>
</siteMapNode>
</siteMapNode>
</siteMap>
```

После добавления элемента навигации на форму необходимо задать источник данных. Для этого нужно в свойстве *ChooseDataSource* выбрать пункт *<New data source...>* и в открывшемся окне выбрать SiteMap. При этом будет создан и добавлен на страницу источник данных с заданным именем.

7.2 Выполнение работы

1 Создать мастер-страницу по классической схеме: верхний колонтитул, нижний колонтитул, слева – панель навигации, справа – будет размещать свой контент страница содержимого.

2 Создать пять страниц содержимого со следующей вложенностью: домашняя страница, в нее вложены первая и вторая страницы, третья и

четвертая будут вложены в первую страницу.

3 Создать навигацию для перемещения по страницам, используя TreeView или Menu, а также элемент SiteMapPath.

4 Создать еще одну мастер-страницу с идентичным содержимым, но отличающуюся стилем.

5 Создать на обеих мастер-страницах кнопку для изменения стиля. Добавить функционал, чтобы по нажатию кнопки подключалась другая мастер-страница.

7.3 Варианты заданий

1 Новостной портал.
2 Турфирма.
3 Интернет-магазин одежды.
4 Кулинарный сайт.
5 Книжный сайт.
6 Сайт университета.
7 Сайт школы.
8 Интернет-магазин продуктов.
9 Сайт о домашних животных.
10 Автомобильный сайт.

Контрольные вопросы

1 Для чего применяются мастер-страницы?

2 Как создать карту сайта?

3 Как организовать доступ к элементам управления, расположенным на мастер-странице?

4 Какие существуют элементы управления для создания навигации по страницам и в чем их отличия?

5 Как организовать динамическое изменение мастер-страницы?

8 Лабораторная работа № 8. Разработка ASP.Net-приложения с базой данных

Цель работы: изучение возможности разработки приложений для работы с базами данных с помощью технологии Microsoft ASP.NET. Создание приложения ведения базы данных и ее визуализации.

8.1 Краткие теоретические сведения

Для работы с данными используются системы управления базами данных (СУБД). Основные функции СУБД – это определение данных, обработка данных и управление данными. Любая СУБД позволяет выполнять следующие операции с данными:

– добавлять в таблицу одну или несколько записей;

– удалять из таблицы одну или несколько записей;

– обновлять значения некоторых полей в одной или нескольких записях;

– находить одну или несколько записей, удовлетворяющих заданному условию.

Для выполнения этих операций применяется механизм запросов. Запросы к базе формируются на специально созданном языке, который так и называется «язык структурированных запросов» (SQL – Structured Query Language).

8.2 Пример создания приложения

База данных хранит информацию гостевой книги в базе данных SQL. База данных состоит из одной таблицы с именем Messages.

Выполним построение графического интерфейса и настройку привязки данных между GridView и базой данных.

Шаг 1. Создание Web-сайта.

Создать Web-приложение Guestbook по шаблону *Empty Web Site.* Добавить в проект Web-форму с именем *Guestbook.aspx.* Задать свойству *Title* документа значение «*Guestbook*». Чтобы страница *Guestbook.aspx* первой загружалась при запуске приложения, щелкнуть на ней правой кнопкой мыши в окне Solution Explorer и выбрать команду *Set As Start Page.*

Шаг 2. Создание формы для ввода данных

В режиме конструктора добавить на страницу текст *Please leave a message in our guest book*, вставить таблицу из четырех строк и двух столбцов. Поместить соответствующий текст (рисунок 8.1).

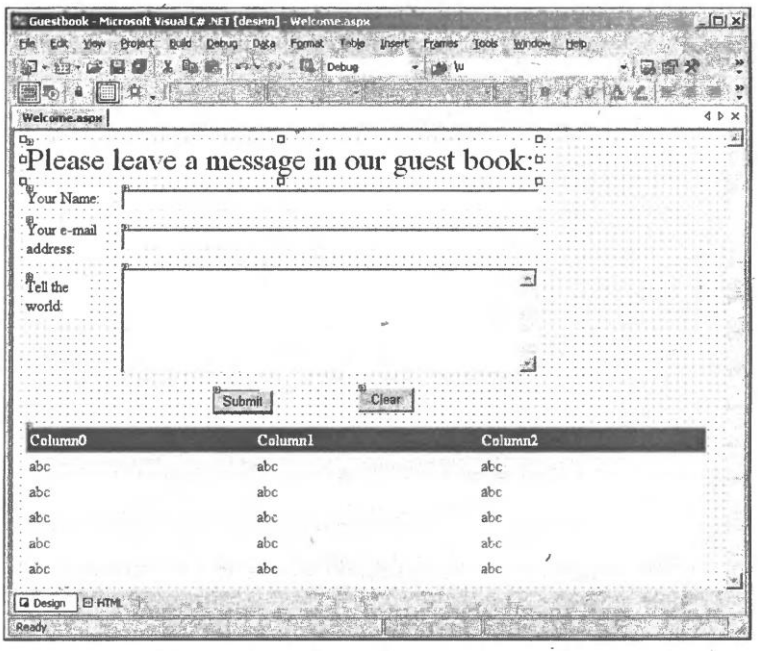

Рисунок 8.1 – GUI-приложения гостевой книги

Шаг 3. Добавление элемента управления GridView на Web-форму.

Добавить элемент управления *GridView* с именем *messagesGridview* для вывода записей гостевой книги. Этот элемент находится в разделе *Data* панели элементов. Для задания цвета GridView воспользоваться ссылкой *Auto Format...* меню смарт-тега, которое открывается при размещении *GridView* на странице. Щелчок на ссылке открывает диалоговое окно *AutoFormat* с несколькими вариантами. В данном примере выбран вариант *Professional*. Далее будет рассмотрено, как задать источник данных *GridView* (то есть источник, из которого элемент берет данные для вывода в строках и столбцах).

Шаг 4. Создание модели данных: сущностей.

Далее в проект включается модель данных сущностей. Выполнить следующие действия.

1 Щелкнуть правой кнопкой мыши на имени проекта в окне Solution Explorer, выбрать команду Add ► Add New Item....

2 Выбрать шаблон ADO.NET Entity Data Model, изменить имя на *Guest*, щелкнуть на кнопке Add. Откроется диалоговое окно, в нем щелкнуть Yes. IDE создаст папку APP_Code и поместит в нее классы модели данных сущностей.

3 Откроется диалоговое окно Entity data model wizard. Убедиться в том, что выбран режим *Generate from database*, и щелкнуть на кнопку Next.

4 На шаге *Choose your Data Connection* мастера Entity Data Mode! Wizard выбрать команду *New Connection...*, а в диалоговом окне *Connection Properties* – файл базы данных (находится в папке databases примеров этой главы). Щелкнуть на кнопку ОК, чтобы создать подключение, а затем завершить шаг *Choose Your Data Connection* кнопкой *Next >*.

5 Откроется диалоговое окно с предложением скопировать файл базы данных в проект. Щелкнуть на кнопке Yes. IDE создаст папку *App_Data* и поместит в нее файл *Guestbook.mdf*. Как и папка App_Code, эта папка доступна только для Web-приложения на сервере.

6 На шаге Choose Your Database Objects and Settings мастера Entity Data Model Wizard выбрать из базы данных таблицу *Messages*. По умолчанию IDE присваивает модели имя GuestbookModel. Убедиться в том, что флажок *Pluralize or singularize generated object names* установлен. Оставить другие настройки без изменений и щелкнуть на кнопке Finish. IDE отображает модель *GuestbookModel* в редакторе, где можно увидеть, что класс Message содержит свойства *MessagelD, Date, Name, Email и Messagel*. Свойство Message было переименовано в Messagel средой разработки для предотвращения конфликтов с классом Message модели данных сущностей.

7 Выполнить команду BUILD ► Build Solution, чтобы откомпилировать классы модели данных сущностей.

Шаг 5. Привязка элемента управления GridView к таблице Messages базы данных Guestbook.

Теперь нужно настроить *GridView* для отображения информации из базы данных.

1 В меню смарт-тега *GridView* выбрать вариант *<New data source...>*

в поле *Choose Data Source*. На экране появится окно мастера Data Source Configuration Wizard.

2 В этом примере используется элемент управления *EntityDataSource* для взаимодействия приложения с базой данных Guestbook.mdf. Выбрать значок *Entrity*, ввести в поле *Specify an ID for the data source* строку messagesEntityDataSource и щелкнуть на кнопке ОК, чтобы запустить мастер Configure Data Source.

3 На шаге Configure ObjectContext выбрать в поле *Named Connection* вариант *GuestbookEntities* и щелкнуть на кнопке *Next >*.

4 В окне Configure Data Selection можно указать, какие данные объекта *EnitityDataSource* должен получить из контекста данных.

5 Щелкнуть на кнопке *Finish*, чтобы завершить работу мастера. На Web-форме прямо под *GridView* появится элемент управления с именем *messagesEntityDataSouruce*. В режиме конструктора он представлен серым прямоугольником с указанием типа и значения. На Web-странице он не отображается — серый прямоугольник всего лишь позволяет визуально взаимодействовать с элементом управления в режиме конструктора (по аналогии с объектами в области компонентой в изложениях Windows Forms).

Шаг 6. Настройка столбцов источника данных, отображаемых в GridView.

Если необходимо настроить GridView так, чтобы исключить, например, столбец с первичным ключом из отображения на Web-форме, то понадобится команда Edit *Columns*. Она находится в меню смарт-тега GridView *Tasks*.

Шаг 7. Изменение файла программной логики приложения Guestbook.

После построения Web-формы и настройки элементов данных, используемых в данном примере, сделать двойной щелчок на кнопках *Submit* и *Clear* конструктора, чтобы создать соответствующие обработчики события *Click* в программной логике. IDE генерирует пустые обработчики событий, поэтому в них необходимо добавить код, обеспечивающий работу этих кнопок. Обработчик события *clearButton* (строки 39–44) очищает текстовое поле, его свойству *Text* присваивается пустая строка. Форма инициализируется для отправки записи гостевой книги.

```
1  // Guestbook.aspx.cs
2  // Файл программной логики с обработчиками событий гостевой книги
3  using System;
4
5  Public partial class Guestbook : System.Web.UI.Page
6  {
7  // Кнопка Submit добавляет записи и очищает форму.
8
9   Protected void submitButton_Click ( object sender, EventArgs e )
10 {
11 // Использование DbContext для добавления сообщения
12 using ( GuestbookEntities dbcontext • new GuestbookEntities() )
```

```
13 {
14 //  Создание нового сообщения для добавления в базу данных}
15 // Message - класс модели данных, представляющий строку таблиц
16 Message message = new Message();
17
18 // Задание свойств нового объекта Message
19  message.Date = DateTime,Now.ToShortDateString()
20 message.Name = nameTextBox.Text;
21 message.Email = emailTextBox.Text;
22 message.Message= messageTextBox.Text;
23
24 // Добавление нового объекта Message в DbContext
25 dbcontext.Messages.Add( message );
26 dbcontext.SaveChanges(); // Сохранение изменений в базе данных
27  } // Конец команды using
28
29 // Очистка текстовых полей
30 nameTextBox.Text = String.Empty;
31 emailTextBox.Text = String.Empty;
32 messageTextBox.Text = String.Empty;
33
34 // Обновление Gridview содержимым таблицы базы данных
35 messagesGridView.DataBindO;
36 }  // submitButton_Click
37
38 // Кнопка Clear очищает текстовые поля веб-формы
39 protected void clearButton_Click( object sender, EventArgs e )
40 {
41 nameTextBox.Text = String.Empty;
42 emailTextBox.Text = String.Empty;
43 messageTextBox.Text = String.Empty;
44 }  // clearButton_Click
45 // Конец класса Guestbook
```

Строки 9–36 содержат код обработки *submitButton*, который добавляет информацию пользователя в таблицу *Messages* базы данных Guestbook. Команда *using* в строках 12–27 начинается с создания объекта *GuestbookEntities* для взаимодействия с базой данных. Команда *using* вызовет *Dispose* для объекта *GuestbookEntities* после завершения. Так рекомендуется поступать при запросе Web-страниц ASP.NET, чтобы не удерживать подключение к базе данных после обработки запроса.

В строке 16 создается объект модели данных сущностей класса *Message*, представляющего строку таблицы базы данных *Messages*. Строки 19–22 задают свойствам объекта *Message* значения, которые должны быть сохранены в базе данных. Строка 25 вызывает метод Add свойства *Messages*

объекта *GuestbookEntities,* представляющего таблицу *Messages* базы данных. Метод добавляет новую запись в представление таблицы в модели данных сущностей. Строка 26 сохраняет изменения в базе данных.

8.3 Выполнение работы

Согласно исходным данным для вашего варианта создать приложение для работы с базой данных. В приложении обязательно должна быть страница авторизации.

Для создания базы использовать только SQl или Access. В базе должно быть не менее пяти таблиц. В каждой таблице обязательно должен быть определен первичный ключ.

8.4 Варианты заданий

1 Автошкола.
2 Заправка.
3 Гостиница.
4 Магазин обуви.
5 Аптека.
6 Поле чудес.
7 Автовокзал.
8 Кинотеатр.
9 Детский сад.
10 Библиотека.

Контрольные вопросы

1 Как реализуется взаимодействие с базой данных в ASP.Net?
2 Что такое база данных?
3 Для чего используется строка подключения?
4 Основное значение объекта DataSet.

Список литературы

1 **Мак-Дональд, М.** WPF: Windows Presentation Foundation в .NET 3.5 с примерами на C# 2008 для профессионалов / М. Мак-Дональд. – 2-е изд. – Москва: И.Д. Вильямс, 2008. – 928 с.

2 **Петцольд, Ч.** Microsoft Windows Presentation Foundation. Базовый курс / Ч. Петцольд. – Санкт-Петербург: Питер, 2012. – 945 с.

3 **Эспозито, Д.** Microsoft ASP.NET 2.0. Углубленное изучение : пер. с англ. / Д. Эспозито. – Санкт-Петербург : Питер, 2007. – 592 с.

4 **Гробов, И. Д.** ASP.NET 2.0: теория и практика / И. Д. Гробов. – Москва : Диалог-МИФИ, 2007. – 608 с.

5 **Дейтел, П.** Как программировать на C# 2012 / П. Дейтел, Х. Дейтел. – Санкт-Петербург : Питер, 2014. – 864 с.