

Министерство образования Российской Федерации

Владимирский государственный университет

Кафедра информационных систем и
программной инженерии

ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

Методические указания к лабораторным работам

Составители:
Вершинин В.В.,
Штых А.Д.
Ефремов Д.В.

Владимир, 2017

ОГЛАВЛЕНИЕ

ОГЛАВЛЕНИЕ	2
ЛАБОРАТОРНАЯ РАБОТА №1	3
НАЧАЛЬНОЕ ЗНАКОМСТВО С ПРОСТЕЙШИМИ WEB-ПРИЛОЖЕНИЯМИ (ASP.NET)	3
ЛАБОРАТОРНАЯ РАБОТА №2	10
РАБОТА С ДАННЫМИ В ASP.NET 4.0	10
ЛАБОРАТОРНАЯ РАБОТА №3	19
ДОСТУП К ДАННЫМ В .NET. ТЕХНОЛОГИЯ ADO.NET	19
ЛАБОРАТОРНАЯ РАБОТА №4	25
WEB-СЕРВИСЫ (СЛУЖБЫ)	25
ЛАБОРАТОРНАЯ РАБОТА №5	37
ШАБЛОН ПРОЕКТИРОВАНИЯ MVC В ASP.NET	37
ЛАБОРАТОРНАЯ РАБОТА №6	56
РЕАЛИЗАЦИЯ АУТЕНТИФИКАЦИИ В MVC ASP.NET	56
ЛАБОРАТОРНАЯ РАБОТА №7	61
ЯЗЫК ИНТЕГРИРОВАННЫХ ЗАПРОСОВ LINQ TO ENTITIES	61
ЛАБОРАТОРНАЯ РАБОТА №3	77
ПОСТРОЕНИЕ ИНТЕРАКТИВНЫХ ПОЛЬЗОВАТЕЛЬСКИХ ИНТЕРФЕЙСОВ С ИСПОЛЬЗОВАНИЕМ AJAX	77

Лабораторная работа №1

НАЧАЛЬНОЕ ЗНАКОМСТВО С ПРОСТЕЙШИМИ WEB-ПРИЛОЖЕНИЯМИ (ASP.NET)

Цель работы

Изучить базовые принципы построения и функционирования простейших веб-приложений на платформе ASP.NET 4.0.

Общие сведения

Технология ASP.NET

ASP .NET — это элемент технологии .NET, используемый для написания клиент-серверных интернет-приложений. Она позволяет создавать динамические страницы HTML. ASP .NET возникла в результате объединения более старой технологии ASP (активные серверные страницы) и .NET Framework. Она содержит множество готовых элементов управления, применяя которые, можно быстро создавать интерактивные web-сайты, которые также называются web-приложениями.

Что такое динамические страницы HTML и чем они отличаются от статических? Статическая страница содержит код на языке гипертекстовой разметки HTML. Когда автор страницы пишет ее, он определяет, как будет выглядеть страница для всех пользователей. Содержание страницы будет всегда одинаковым, независимо от того, кто и когда решит ее просмотреть. Языка HTML вполне достаточно для отображения информации, которая редко изменяется и не зависит от того, кто ее просматривает.

Динамическими принято называть web-страницы, которые перед отправкой клиенту проходят цикл обработки на сервере (или целиком генерируются сервером). В самом простом случае это может быть некоторая программа, которая модифицирует запрашиваемые клиентом статические страницы, используя параметры полученного запроса и некоторое хранилище данных.

Существует несколько вариантов построения веб-приложений:

1. Вариант на основе "чистого" HTML (приводится в качестве общей демонстрации):

```
<html >
<body>
  <form>
    <i nput type="text" name="op1" />
    +
    <i nput type="text" name="op2" />
    <i nput type="submi t" val ue=" = " />
```

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

Приведенный выше пример содержит форму, в которой находятся 2 текстовых поля ввода и одна кнопка. По нажатию кнопки все данные из элементов формы (тэг <form>...</form> и все его содержимое) запросом POST протокола HTTP будут отправлены на сервер, однако обработаны они не будут. Другими словами, в ответ сервер вернет исходную HTML-страницу.

2. Вариант на основе CGI (Common Gateway Interface – общий шлюзовый интерфейс). Здесь запрос отправляется на сервер также с формы статической HTML-страницы, однако в ответ специальные программы на сервере (исполняемые программы, написанные на языках C/C++, скрипты на языке Perl) сгенерирует страницу с учетом полученных им данных формы. Особенности CGI является то, что: они представляют низкоуровневый программный интерфейс; для обработки каждого запроса на сервере запускается свой процесс (следствие – коллизии, взаимные блокировки, нарушение целостности данных и т.п.); низкая скорость. Такой подход в основном используется на unix-платформах.

3. Вариант на основе ISAPI. В отличие от CGI вместо исполняемых файлов или скриптов на сервере используются DLL-библиотеки. Код DLL находится в памяти все время и для каждого запроса создает не процессы, а потоки (Threads) исполнения. Все потоки используют один и тот же программный код. ISAPI-приложение выполняется в процессе веб-сервера. Это позволяет повысить производительность и масштабируемость.

4. Скриптовые языки (PHP, ASP, отчасти JSP). Здесь код, исполняемый на сервере внедрен в текст страницы в виде специальных тэгов. Когда пользователь запрашивает страницу, на сервере выполняется код, внедренный в текст страницы. В задачу этого кода, как правило, входит извлечение каких-либо данных из БД (сообщений ветки форума, информации о погоде и т.п.) и представление этих данных в виде HTML. В результате пользователь получает страницу, структура (шаблон) которой неизменен, а содержимое генерируется сервером.

Этот вариант наиболее распространен, однако имеет существенный недостаток – HTML-код и код, выполняемый на сервере «свалены в кучу», что сильно затрудняет работу как дизайнеров, так и программистов и делает неудобным сопровождение веб-приложения.

5. ASP.NET позволяет избавиться от недостатков, присущих скриптовым языкам. В ASP.NET HTML-код страницы и программный код,

выполняемый на сервере (серверный код), выделены в разные файлы. Содержимое ASP.NET-страниц помимо тэгов HTML включает в себя так называемые **серверные элементы управления** – специальные тэги, которые имеют объектное представление в серверном коде.

Так, например, элемент Label на ASP.NET-странице выглядит следующим образом:

```
<asp:Label id="Label1" runat="server" Text="" />
```

а в серверном коде он же представлен в виде объекта класса Label с именем Label1, и к свойству Text у него можно обратиться следующим образом:

```
Label1.Text = "Hello!";
```

Очевидно, что на сгенерированной сервером странице вместо тэга, приведенного выше, будет красоваться надпись Hello!

Первый проект

Первый проект создадим с использованием программной среды Visual Studio for Web 2015 или ей подобных. При создании необходимо задать единый каталог проекта. После запуска среды выберите пункт меню File → New Web Site. Появится диалоговое окно (рис.1).

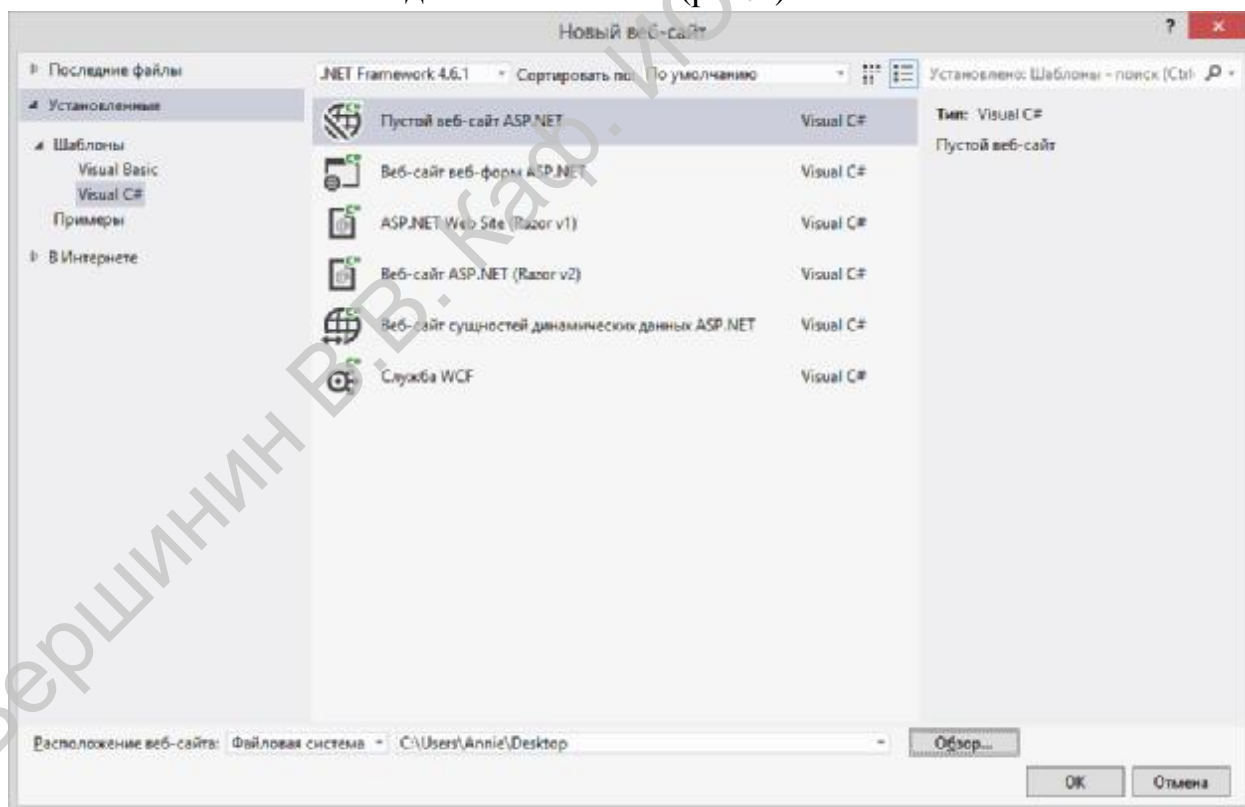


Рисунок 1. Создание нового веб-сайта.

Выберите язык C#, “Пустой веб-сайт ASP.NET” и назначьте в нем имя проекта.

По умолчанию проект создается в файловой системе. По желанию его можно создать на HTTP или FTP-сервере. Из файловой системы проект

всегда можно скопировать на сервер нажатием одной кнопки в заголовке Solution Explorer.

Для создания страницы ASP.NET необходимо создать новую форму. Для этого выберите вкладку Веб-Сайт а Добавить новый элемент. В диалоговом окне выберите язык C# и элемент Форма Web Form. В обозревателе вы можете увидеть новую страницу default.aspx. Выберите ее, и появится окно редактирования с закладками Конструктор и Исходный код. Не меняя ничего, щелкните на кнопке со стрелкой, чтобы просмотреть страницу в браузере.

Откроется браузер, показывающий страницу по адресу *http://localhost:номер_порта/default.aspx*. "Localhost" обозначает сервер, работающий на вашем компьютере. Встроенный сервер, используемый по умолчанию, сам назначает себе номер порта – для каждого проекта он разный. Сервер IIS обычно работает через порт 80 (или 8080, если порт 80 занят), и для него номер порта указывать не нужно. При этом ваша страница будет скомпилирована.

Пока что страница в браузере пустая. Но исходный код этой страницы не пустой. Среда автоматически сгенерировала код страницы.

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

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<meta http-equiv="Content-Type" content="text/html; charset=utf-
8"/>
<title></title>
</head>
<body>
<form id="form1" runat="server">
</form>
</body>
</html>
```

Разберем содержимое этой страницы:

`<%@ Page Language="C#" %>` – тег `<%` всегда предназначается для интерпретации ASP-кода. Директива `Page` всегда присутствует на странице `aspx`. Ее атрибут `Language` — это указание, что в скриптах данной страницы будет использоваться C#, а могли бы VB, C++ или J#.

`CodeFile` — имя файла с отделенным кодом (code-behind), который должен выполняться на сервере.

Inherits — класс, определенный в том файле, от которого наследуется класс страницы.

Одновременно будет создан и файл Default.aspx.cs. Подход разделения кода называется **фоновый код**. Сама форма находится в файле Default.aspx, а в файле Default.aspx.cs находится класс страницы на языке C#. Таким образом, дизайн страницы может быть изменен не затрагивая кода страницы, что позволяет разделить ответственность за внешний вид и работу страницы между дизайнером и программистом.

<form runat="server"> – тег дает указание компилятору обрабатывать элементы управления страницы. Обратите внимание на то, что данный тег имеет свойство runat, для которого установлено значение "server" (других значений не бывает). При использовании этого свойства элемент управления обрабатывается компилятором, а не передается браузеру "как есть".

Вставьте в Default.aspx между тегами <form> и </form> тег, задающий элемент управления:

```
<asp:Label id="Time" runat="server" Text="Сервер находится в  
Москве. Московское время: "/>
```

Серверный элемент управления Label является средством размещения на странице текста, который может содержать теги HTML. Изменяя значения свойств этого элемента управления в коде, можно динамически изменять текст на странице. В asp:Label компилятору сообщается, с каким объектом ведется работа (в рассматриваемом случае с элементом управления Label).

Далее задаются различные свойства элемента управления. В первую очередь определяется его имя id="time" и атрибут "runat", а также текст.

В файле Default.aspx.cs должен содержаться такой текст:

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

Ключевое слово partial появилось в C# 2.0, и позволяет разбить текст определения класса между разными файлами. Класс System.Web.UI.Page является базовым для всех страниц ASP.NET.

Вставьте в эту функцию строку

```
Time.Text += DateTime.Now.ToString();
```

что означает получение системного времени. Полученное значение присваивается свойству Text объекта time (time элемент управления типа Label (метка)). Время на часах клиента и сервера может не совпадать, если

они находятся в разных точках земного шара. Метод Page_Load похож на обычный обработчик события формы. Как можно легко догадаться, эта функция вызывается каждый раз, когда загружается форма.

Запустите страницу на просмотр кнопкой F5 или нажав на кнопку со стрелкой на панели инструментов. В браузере должна открыться страница, на которой будет написано текущее время. Откройте исходный текст страницы. Никакого кода на C# или элементов управления ASP.NET там не будет:

```
<!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><title>Время, вперед</title>
</head>
<body>
    <form name="form1" method="post" action="Default.aspx"
id="form1">
    <div>
    <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value=" /wEPDwUJODExMDE5NzY5D2QWAgIDD2QWAgIBDw8WAh4EVGv4dAUS
MDguMDY
uMjAwNiA0OjU2OjQ3ZGRkkEMgqXmKC0v9vwAwh999lefuIOw=" />
    </div>
    <div>
        <span id="Time">Сервер находится в Москве. Московское
время:
08.06.2006 4:56:47</span>
    </div>
    </form>
</body>
</html>
```

Обновив страницу, можно увидеть новое значение времени на сервере.

3. Порядок выполнения работы

1. Создать веб-приложение, как это описано в п. 2.2.
2. С помощью панели инструментов Toolbox (меню View → Toolbox или Вид → Панель элементов) добавьте в вашу веб-страницу элементы управления TextBox и Button.
3. С использованием вышеперечисленных элементов создайте веб-приложение "Калькулятор", реализующее операции сложения, вычитания, умножения и деления, а также проверку вводимых значений.

4. Содержание отчета

1. Тексты всех разработанных модулей приложения.
2. Таблицы с исходными данными.
3. Результаты выполнения запросов.
4. Выводы по работе.

5. Контрольные вопросы

1. Что такое WEB-приложение, и каковы его основные особенности?
2. Что такое серверный элемент управления? Чем он отличается от HYML элемента управления?

6. Варианты индивидуальных заданий

Выберите вариант индивидуального задания в соответствии с интересующей вас предметной областью, чтобы использовать его в последующих лабораторных работах.

Вершинин В.В. Каф. ИСПИ, ВлГУ 2017

Лабораторная работа №2

РАБОТА С ДАННЫМИ В ASP.NET 4.0

1. Цель работы

Познакомиться с рядом средств ASP.NET 4.0 для представления и отображения данных.

2. Общие сведения

Привязка к данным

При создании веб-приложений часто решается задача отображения на клиентской стороне некоторых данных, которые на стороне сервера могут быть представлены либо в объектном (экземпляры классов с данными, списочные типы или коллекции), либо в реляционном виде (таблицы данных, представляющие собой выборки из БД).

Платформа ASP.NET предоставляет универсальный и достаточно простой способ отображения данных на стороне клиента. Данные (в каком бы виде они не существовали на сервере) могут быть представлены на клиентской стороне с помощью специальных элементов управления. Задача этих элементов управления – отображение данных на веб-странице в удобной для пользователя форме:

- в виде таблицы (так, например, может быть представлен прайс-лист);
- в виде множества одинаковых по структуре (но имеющих разное содержимое) HTML-фрагментов; в таком виде, например, представляются результаты поискового запроса в Гугле (и не только), страницы тем форумов, ленты новостей на крупных информационных порталах;
- иные способы представления данных.

Рассмотрим механизм отображения данных на элементарнейшем прототипе системы управления контентом сайта: в нашем примере реализована страница добавления новостей на сайт и отображения только что добавленных новостей. Например, сущность "Новость" на стороне сервера представлена следующим классом:

```
/// <summary>
/// Класс, описывающий бизнес-объект - элемент новости
/// </summary>
public class NewsItem {
    private string title;
    private string text;
    private string author;
    private int articleRating;
```

```

public string Title {
    get { return title; }
    set { title = value; }
}

public string Text {
    get { return text; }
    set { text = value; }
}

public string Author {
    get { return author; }
    set { author = value; }
}

public int ArticleRating {
    get { return articleRating; }
    set { articleRating = value; }
}

public NewsItem() {
    Random rnd = new Random();
    this.ArticleRating = rnd.Next(5);
}
}

```

Как видно из примера, сущность "Новость" содержит 4 свойства – Title (заголовок), Text (текст новостной статьи), Author (e-mail автора) и ArticleRating (рейтинг статьи), который в данном примере может принимать значения от 0 до 5 и выставляется случайным образом.

Все множество новостей представим типизированным динамическим списком «`protected List<NewsItem> newsList;`», где каждый элемент списка имеет приведенный выше тип.

Каким образом данные привязываются к серверным элементам управления?

С помощью специальных элементов управления:

- списки (DataList),
- таблицы (GridView),
- репитеры или повторители (Repeater)

Все они имеют свойство **DataSource**, которое отвечает за привязку к данным. Тип этого свойства — **object**, то есть он может быть любым, но должен реализовывать интерфейс **IEnumerable**. Часто значениями этого свойства назначают коллекции.

Свойство DataSource может быть "привязано" к коллекциям, поддерживающим интерфейсы IEnumerable, ICollection или IListSource. Источником данных также могут быть XML-файлы, таблицы баз данных, массивы. Вызовом метода DataBind() данные непосредственно привязываются к серверному элементу управления. Метод Page.DataBind() вызывает привязку данных у всех элементов на странице.

Так, в рассматриваемом примере для отображения новостей используется серверный элемент управления **Repeater** (повторитель). Для привязки к повторителю всего множества новостей, представленных списком **newsList**, на сервере достаточно вызвать следующий код:

```
this.repNews.DataSource = newsList;
```

```
this.repNews.DataBind();
```

repNews здесь – это ID элемента типа Repeater нашей ASP страницы.

Как повторитель "выглядит" в коде рассматриваемого примера?

```
<%!-- Это, собственно, тэг повторителя. Ниже рассмотрено его содержимое --%>
```

```
<asp:Repeater ID="repNews" runat="server">
```

```
<%!-- Повторитель содержит HTML-шаблоны (template) для следующих своих "частей":
```

- шаблон заголовка (HeaderTemplate)
- шаблон повторяемой части (ItemTemplate)
- шаблон завершающей строки (FooterTemplate)

Заголовок и завершающая строка будут присутствовать на сгенерированной с использованием повторителя странице однократно. Повторяемая же часть будет повторена столько раз, сколько элементов в коллекции у связанного с повторителем бизнес-объекта: т.е. столько, сколько строк у DataTable, (в случае, если DataTable привязана к повторителю) или столько, сколько элементов у массива или списка...

В рассматриваемом нами примере мы "плодим" тэг <tr> обычной HTML-таблицы, наполняя его при этом данными из бизнес-объекта

```
--%>
<HeaderTemplate>
<table width="100%">
<tr>
<td style="background-color: #ebebeb"><h3>Лента
новостей:</h3></td>
</tr>
</HeaderTemplate>
```

```
<ItemTemplate>
<tr>
<td style="background-color: #ebdada">
<h4>
```

<%!-- Ниже, в рамках тэга <%# ... %>– "серверный" C#-код, вписанный непосредственно в код веб-страницы. Этот код отвечает за отображение в данном конкретном месте значение соответствующего поля конкретного элемента коллекции. В данном случае – будет отображено значение поля Title объекта класса NewsItem, находящегося в "привязанной" к повторителю коллекции.

-- %>

```
<%# DataBinder.Eval(Container.DataItem, "Title") %>
</h4>
<div>
<%# DataBinder.Eval(Container.DataItem, "Text") %>
</div>
<div>Рейтинг статьи: &nbsp;
<%# DataBinder.Eval(Container.DataItem, "ArticleRating") %>
</div>
<div style="text-align: right">
<%# DataBinder.Eval(Container.DataItem, "Author") %>
</div>
</td>
</tr>
</ItemTemplate>

<FooterTemplate>
<tr><td style="background-color: #daebeb">
Здесь могла быть Ваша реклама!</td></tr>
</table>
</FooterTemplate>
</asp:Repeater>
```

Как непосредственно осуществляется привязывание бизнес-объектов к серверным элементам управления, отображающим данные из этих бизнес-объектов, продемонстрировано выше, в комментариях к коду повторителя.

Компоненты GridView и DataList имеют практически полностью аналогичную структуру, однако имеют значительно больше свойств и опций, позволяющих настраивать их вид согласно дизайну разрабатываемого сайта.

Например, DataList:

```
<asp:DataList ID="NewList" runat="server">
<ItemStyle BackColor="#99ccff" HorizontalAlign="Center"
/> - этот тэг определяет стиль всех нечетных строк списка.
<AlternatingItemStyle BorderStyle="Double"
BackColor="White" HorizontalAlign="Left"/> - этот тэг определяет стиль
всех четных строк списка.
```

```
<ItemTemplate>
<%# DataBinder.Eval(Container.DataItem, "Country")
%>
```

Внутри этого тэга происходит привязка тех данных, которые нужно показывать в нечетных строках.

```
</ItemTemplate>
<AlternatingItemTemplate >
<%# DataBinder.Eval(Container.DataItem, "Duration")
%>
<%# DataBinder.Eval(Container.DataItem, "When") %>
```

//Внутри этого тэга происходит привязка тех данных, которые нужно показывать в четных строках.

```
</AlternatingItemTemplate>  
</asp: DataLi st>
```

Или GridView:

```
<asp: GridVi ew ID="NewGrid" runat="server">  
    <AlternatingRowStyle BackCol or="White"  
ForeCol or="#284775" /> //- определение стиля четных строк.  
    <Col umns>  
        <asp: ButtonFi el d Text="выбрать"  
ButtonType="Image" />  
        //В GridView так же можно вставлять  
дополнительные столбцы различного типа. Например, столбец с кнопками.  
    </Col umns>  
</asp: GridVi ew>
```

Заметьте, что GridView автоматически определяет место отображения данных после их привязки. Столбцы называются как атрибуты класса, а их порядок идентичен порядку полей в классе.

Проверка вводимых данных

Вводимые на веб-страницах данные как правило записываются в базы данных, а, следовательно, и тип информации в них должен соответствовать типу и длине данных в соответствующих полях таблиц баз данных. Кроме того, иногда нужно вводить взаимосвязанные данные, например, пароль во время регистрации нужно вводить 2 раза, и он в обоих полях должен совпадать. Или вводимые данные должны соответствовать определенному формату.

В рассматриваемом нами примере, прежде чем работать с данными, нужно убедиться, что:

- в поле "Заголовок новости" введен текст;
- текст в поле "E-mail автора" имеет форму электронного адреса (с @ и с точкой).

Проверка может происходить и на стороне клиента, и на сервере. При проверке (валидации) на стороне клиента в страницу автоматически встраивается код на Javascript. Если данные в форме не проходят проверку, страница просто не будет отправлена на сервер. Это позволит избежать лишнего трафика и не загружать канал связи с сервером.

С другой стороны, валидация на стороне сервера более надежна. Javascript-код может быть легко просмотрен и изменен, наконец, Javascript можно просто выключить в настройках браузера. При валидации на стороне сервера данные проверяются программой на полноценном языке. Ее код

пользователю неизвестен. В результате проверки генерируется новая страница с сообщениями об ошибках, если они возникли. Самая разумная стратегия — применять комбинацию этих методов. Предварительная проверка у клиента защитит от опечаток, а серьезная проверка на сервере — от злонамеренного взлома.

Существует целый ряд серверных элементов управления, которые не занимаются выводом информации, а проверяют данные, введенные пользователем. ASP.NET 4.0 сам определяет тип браузера и генерирует наиболее подходящий для данного случая код. Если браузер поддерживает Javascript-код, который он может послать, то валидация или ее часть происходит на стороне клиента. Если браузер не поддерживает Javascript, то вся валидация происходит на сервере.

Получить доступ к валидаторам просто — они находятся в панели Toolbox на вкладке "Validation" («Проверка»). Классы валидаторов образуют иерархию (рис.2), во главе которой стоит абстрактный класс `BaseValidator`.

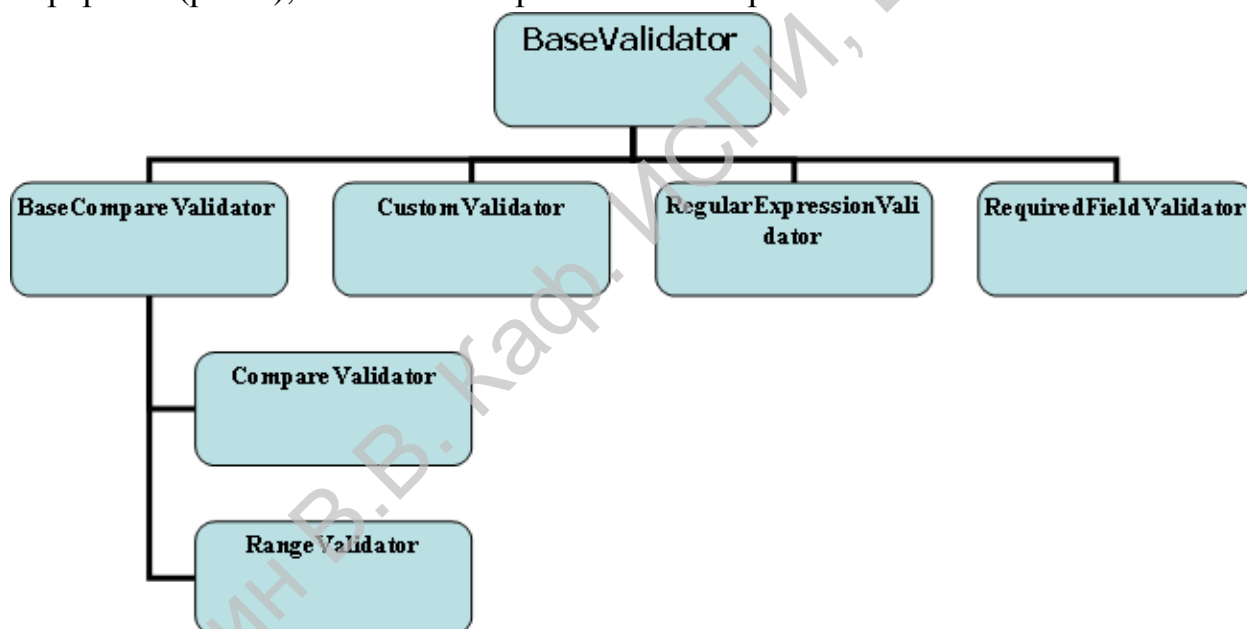


Рисунок 2. Иерархия классов валидаторов.

RequiredFieldValidator проверяет ввел ли пользователь данные. Если поле ввода не содержит данных, выводится ошибка.

RangeValidator проверяет, попадает ли введенное значение в заданный диапазон. Для этого существуют поля `MinimumValue` и `MaximumValue`, задающие границы диапазона. С помощью поля `Type` можно определить тип вводимых данных. Это могут быть строки, целые числа, числа с плавающей запятой, дата/время и валюта.

CompareValidator проверяет, совпадают ли введенные данные с заданной константой (`ValueToCompare`) или с данными в другом элементе управления (`ControlToCompare`). Свойство `Operator` определяет операцию

сравнения (равны, не равны, больше, меньше и т.д.). Среди этих операций есть операция сравнения типов – `DataTypeCheck`. Тип, с которым надо сравнить введенные данные, хранится в поле `Type`.

RegularExpressionValidator проверяет соответствие введенных данных заданному регулярному выражению. Само выражение хранится в поле `ValidationExpression`. Регулярные выражения используются там, где данные должны вводиться по определенному регламенту. Например, email должен содержать символ “@”, а после него точку. Этот валидатор не проверяет пустые строки.

Базовый класс валидаторов сам наследник класса `Label`, так что по существу все валидаторы — метки, текст в которых становится видимым, когда не выполняются заданные нами условия проверки. По умолчанию текст в валидаторах — красный.

Все валидаторы имеют свойство `ControlToValidate`. Оно задает тот элемент управления, содержимое которого проверяются данным валидатором. Этот элемент должен находиться в одной форме с валидатором.

Таблица 1. Свойства валидаторов

Общие свойства валидаторов	
<code>Display</code>	Предоставлять ли место статически или динамически
<code>EnableClientScript</code>	Разрешать ли генерировать клиентский javascript-код
<code>ErrorMessage</code>	Текст сообщения об ошибке
<code>IsValid</code>	Прошел ли валидацию связанный с валидатором элемент управления

Проверка данных всегда инициируется каким-либо событием. Обычно это щелчок на кнопках `Button`, `ImageButton`, `LinkButton`, в которые по умолчанию свойство `CausesValidation` установлено в `True`. Можно убрать это свойство для некоторых кнопок, которым оно не нужно, например, для кнопки `Cancel`. В нашем примере на веб-странице имеется несколько валидаторов. Проверка заголовка:

```
<asp:RequiredFieldValidator ID="rfvTitle" runat="server"
ErrorMessage="RequiredFieldValidator" ControlToValidate="txtTitle">
Заголовок не может быть пустым! </asp:RequiredFieldValidator>
```

Класс `RequiredFieldValidator` проверяет, было ли изменено значение в связанном с ним элементе управления. Если, как в данном случае, это элемент ввода текста — требуется, чтобы пользователь ввел туда что-нибудь. Если выбор не был сделан, но кнопка (см. пример) была нажата, валидация проваливается и выводится текст, заданный в `ErrorMessage` или в `Text`. Валидаторы отображают текст, указанный внутри тэга.

Для проверки корректности ввода электронного адреса используется класс `RegularExpressionValidator`:

```
<asp:RegularExpressionValidator ID="revAuthor" runat="server"
ErrorMessage="RegularExpressionValidator">
```


`ControlToValidate="txtAuthor" ValidationExpression="\w+([-+.']\w+)*@\w+([-.\w+)*\.\w+([-.\w+)*"> Нужно ввести e-mail
</asp:RegularExpressionValidator>`

Класс `ValidationExpression` – регулярное выражение, на соответствие которому проходит проверку значение текстового поля. В Visual Studio 2015 предоставляет несколько готовых шаблонов регулярных выражений, которые можно выбрать в окне свойств, – телефонных номеров разных стран, адресов, и, самое полезное, шаблоны электронной почты и адреса в Интернете.

С одним элементом управления может быть связано несколько валидаторов. Например, электронный адрес проверяется и на соответствие шаблону, и на обязательное заполнение.

Свойство `Page.IsValid` позволяет определить, прошла ли вся страница валидацию. Для браузеров, которые поддерживают DHTML, проверка происходит на стороне клиента. Для этого автоматически генерируется JavaScript-код. Таким образом экономятся ресурсы сервера и трафик, который бы пришлось потратить на передачу неправильных данных.

Замечание: при выполнении задания может возникнуть ошибка «В WebForms для режима `UnobtrusiveValidationMode` требуется сопоставление `ScriptResourceMapping` для "jquery". Добавьте сопоставление `ScriptResourceMapping` с именем jquery (с учетом регистра)». В этом случае подключите библиотеки jQuery самостоятельно или в каждом валидаторе запретите клиентский скрипт.

3. Порядок выполнения работы

1. Ознакомиться с примером, который предоставлен Вам преподавателем.
2. Исследовать следующие вопросы, касающиеся представленного примера:
 - а. как данные представлены на стороне сервера;
 - б. каким образом данные представляются на стороне клиента;
 - в. каким образом клиентское представление связывается с серверным;
 - г. как осуществляется проверка вводимых пользователем данных;
 - е. как используется и работает компонент **Repeater**.
3. Составить свой пример, используя вместо компонента **Repeater** компоненты **DataList** или **GridView**.
4. Реализовать проверку данных на своем сайте.

4. Содержание отчета

1. Цель работы.
2. Описание механизма работы разработанного приложения с хранилищами данных и комментариями по тексту программы.
3. Выводы по работе

5. Контрольные вопросы

1. Каким образом осуществляется связывание данных и представления?
2. Какие типы объектов могут быть присвоены свойству DataSource?
3. Как (и на какой стороне – на сервере или на клиенте) осуществляется валидация данных?
4. Каким образом валидация осуществляется на клиенте?
5. Есть две кнопки – OK и Cancel. Мы хотим, чтобы по нажатию OK валидация выполнялась, а по нажатию Cancel – нет.
6. Как узнать (на стороне сервера), прошла ли страница валидацию?
7. Как валидатор "привязывается" к элементу управления, который он проверяет?

Лабораторная работа №3

ДОСТУП К ДАННЫМ В .NET. ТЕХНОЛОГИЯ ADO.NET

Цель работы

Познакомиться с технологией доступа к данным ADO.NET.

Теоретические сведения:

В настоящей лабораторной работе технология ADO.NET рассматривается на примере использования ASP.NET-приложением данных из БД под управлением СУБД Microsoft SQL Server 2005 Express, поставляемой вместе с Visual C# Express 2005 и Visual Web Developer Express 2005.

Создание базы данных

СУБД Microsoft SQL Server в настоящей работе подробно не рассматривается, поскольку в рамках одной работы невозможно даже поверхностно ознакомиться и с сотой долей возможностей СУБД. Стоит лишь отметить, что MS SQL Server – многоцелевая промышленная СУБД, используемая в приложениях самого разного назначения (от настольных до распределенных кластерных систем).

В настоящей работе рассматривается бесплатно распространяемая Express-версия SQL Server, имеющая ограничения по аппаратной конфигурации сервера (может использоваться на системах с 1 процессором, и максимум 4 Гб оперативной памяти). SQL Server Express подходит для настольных и малых "офисных" систем.

Microsoft Visual Studio 2015 (любая редакция, в т.ч. Express) предоставляет в себя набор средств, необходимых для создания и выполнения несложных типовых операций с БД. Рассмотрим их ниже.

Для создания БД в среде разработки Visual Studio (Visual C# или Web Developer) нужно:


1. Убедиться, что SQL Server Express установлен и запущен (выбрать в контекстном меню значка "Мой Компьютер", пункт "Управление", узел "Службы и приложения\Службы", убедиться, что служба SQL Server (SQLEXPRESS) – собственно, сервер – присутствует в списке и запущена).
2. Открыть панель Server Explorer (меню View → Server Explorer).
3. В контекстном меню узла Data Connections выбрать пункт Добавить подключение. Источник данных - Microsoft SQL Server (SqlClient)
4. В открывшемся окне в поле Server name указать имя (сетевой адрес) сервера: LOCALHOST\SQLEXPRESS, а в поле New Database name –

указать имя создаваемой Вами БД и нажать ОК. База с указанным Вами именем создана и доступна.

5. После этого в панели Server Explorer можно будет увидеть только что созданную БД и все ее элементы, представленные в виде дерева.
6. Для добавления в БД таблиц в контекстном меню узла Tables следует выбрать пункт Add New Table.
7. Для просмотра и "ручного" редактирования содержимого созданной Вами таблицы (таблиц) можно воспользоваться ее контекстным меню.

В качестве примера создадим базу данных **Test**, содержащую единственную таблицу **tb_News** (таблица для хранения новостей) со следующими полями:

1. **ID**, имеющее тип int,
2. **Text**, имеющее тип nvarchar(max),
3. **Author**, тип nvarchar(50),
4. **Rating**, тип int,
5. **Date**, тип DateTime.

Для колонки ID в редакторе ее свойств установим значение параметра Identity Specification  Is Identity в true. Это будет означать, что при каждой вставке новой строки в данную таблицу значение поля ID будет автоматически увеличиваться. У Вас должна получиться таблица, изображенная на рис.3.

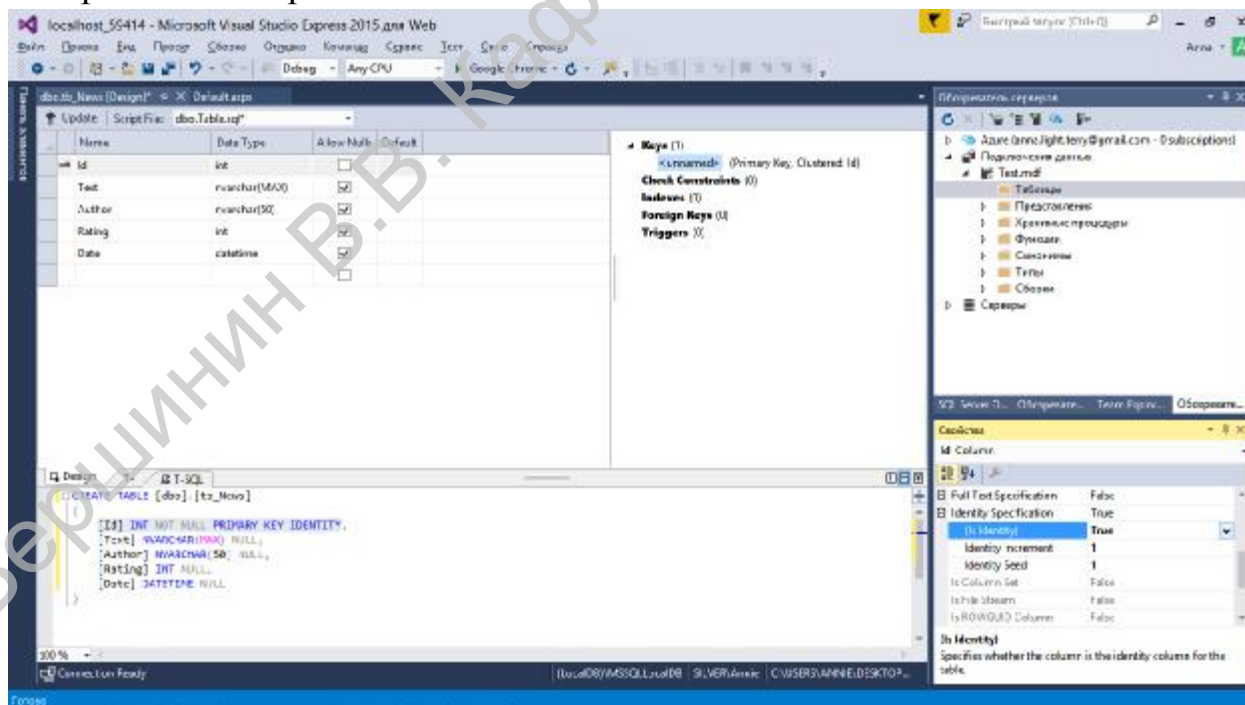


Рисунок 3. Создание таблицы БД.

Для добавления таблицы в базу данных нажмите кнопку Update в левом верхнем углу таблицы. Обновите базу данных в обозревателе.

Доступ к данным SQL Server в C# с использованием ADO.NET

Технология ADO.NET фактически представляет собой множество пространств имен и классов для организации соединения с СУБД и для управления данными. ADO.NET может использоваться в любых приложениях – от консольных до ASP.NET без каких-либо ограничений.

ADO.NET реализует **единый принцип доступа к данным** вне зависимости от используемой СУБД. Так, в качестве СУБД (и в качестве источника данных вообще) может использоваться как MS SQL Server, так и Oracle, и MySQL и т.д., а также COM-компоненты, предоставляющие доступ к данным из Excel-файлов, таблиц MS Access и пр.

Принцип организации доступа к данным рассмотрим на примере MS SQL Server Express.

Для работы с данными необходимо подключить следующие пространства имен:

```
using System.Data;  
using System.Data.SqlClient;  
using System.Data.SqlTypes;
```

Принцип доступа к данным опирается на следующие понятия:

- **соединение (Connection)** – "канал" доступа к БД (и СУБД), по которому осуществляется все взаимодействие с базой данных; по сути соединение предоставляет доступ к источнику данных, которым является база данных. Для SQL Server – используется класс **SqlConnection**.
- **команда (Command)** – объект, несущий в себе информацию о том, какие действия необходимо выполнить СУБД над данными; все взаимодействие с БД – выборка данных, изменение данных, создание объектов БД – осуществляется посредством команд. Для SQL Server следует использовать класс **SqlCommand**.
- **метод передачи/сохранения или извлечения данных**; таких методов реализовано много, в рамках настоящей работы мы рассмотрим простейшие из них. Для работы с SQL Server используются классы **SqlDataReader**, **SqlDataAdapter** и пр.
- **"потребитель" данных** – объект на стороне клиента, в который данные должны быть извлечены. В качестве потребителей данных могут быть использованы **DataTable**, а также **DataSet** и прочие объекты.

Рассмотрим, как осуществляется взаимодействие с СУБД.

Первым делом необходимо установить соединение с БД. Для определения параметров соединения используется строка соединения (**connection string**), которую в реальных приложениях, как правило, хранят в конфигурационном файле приложения.

Мы сделаем строку соединения атрибутом класса, в котором соединение будет использоваться:

```
private string connectionString = @"Initial Catalog=<имя БД>; " +
    @"Data Source=<имя (адрес) сервера БД>; " +
    @"User ID=<имя пользователя>; " +
    @"Password=<пароль>;";
```

В данном случае знак @ означает, что следующая за ним строка (то есть все то, что находится в кавычках) серверу следует воспринимать точно, без модификаций. Исключение – парные кавычки.

Например, для созданной нами БД (п. 2.1) строка соединения будет выглядеть следующим образом:

```
"Data Source=LOCALHOST\SQLEXPRESS;
Initial Catalog=Test;
Integrated Security=True;
Pooling=False"
```

Параметр «Integrated Security=True;» заменяет имя пользователя и пароль и указывает, что для получения доступа к SQL Server используется встроенная безопасность Windows. Фактически, Вы "войдете" на сервер под тем же логином и паролем, под которыми Вы вошли в Windows.

Далее рассмотрим пример простейшей программы, прилегающий к настоящей лабораторной работе. Следующий код:

```
SqlConnection connection;
try {
    connection = new SqlConnection(connectionString);
    connection.Open();
}
catch (SqlException ex) {
    Console.WriteLine("Ошибка при установлении соединения: ",
ex.Message);
    Console.ReadKey();
    return;
}
```

создает и открывает соединение с БД, используя строку соединения. При открытии соединения может возникнуть множество различных исключительных ситуаций (Exception): сервер СУБД недоступен, или на сервере отсутствует БД с указанным именем, или доступ для указанного пользователя запрещен...

Все исключительные ситуации, относящиеся к соединению с БД, имеют тип **SqlException** и перехватываются в предлагаемом примере.

Далее рассматриваются примеры вставки данных в таблицу и выборки данных из таблицы. Для удобства изучения программы эти примеры вынесены в отдельные функции. Функции принимают в качестве параметра уже открытое соединение с БД. Следующий код осуществляет вставку записи в таблицу **tb_News** (см. пример п. 2.1):

```
try {
    SqlCommand command1 = new SqlCommand("INSERT INTO tb_News (Text,
Author, Rating, Date) " +
```

```

        "VALUES ('Another one article', 'Somebody', 10,
09/30/2008)", connection);
        command1.ExecuteNonQuery();
    }
    catch (SqlException ex) {
        Console.WriteLine("Ошибка при вставке данных: ", ex.Message);
        Console.ReadKey();
    }
}

```

Здесь создается экземпляр класса SqlCommand, в конструктор которого в качестве параметров передается SQL-запрос на вставку данных, а также созданное выше соединение.

Выполняется команда на сервере при вызове метода ExecuteNonQuery(). Важно отметить, что этот метод следует вызывать, когда нужно лишь передать команду SQL Server'у, и нет необходимости извлекать какие-либо данные из таблицы БД.

При выполнении команды так-же возможны исключительные ситуации, которые перехватываются в блоке catch { ... }.

Для извлечения данных из таблицы у класса SqlCommand существуют другие методы. Представленный ниже пример иллюстрирует один из способов извлечения данных из БД.

```

SqlCommand command2 = new SqlCommand("SELECT * FROM tb_News",
connection);
SqlDataReader reader = command2.ExecuteReader();
while (reader.Read()) {
    Console.WriteLine("ID: {0}, Автор статьи: {1}", reader[0],
reader["Author"]);
}
reader.Close();

```

Здесь для извлечения данных используется класс SqlDataReader, который осуществляет строчное чтение **набора данных**, которые вернул запрос **SELECT * FROM tb_News**. При этом весь набор данных хранится на сервере БД. Вызов метода Read() инициирует передачу ровно одной строки данных в наше клиентское приложение и переходит на следующую строку набора данных. К элементам строки можно обратиться с как по индексу, так и по имени.

Порядок выполнения работы

1. Создать пример БД в соответствии с вариантом, как описано в п. 2.1, и изучить работу примера приложения, которое предоставлено преподавателем.
2. Изучить способ извлечения данных из БД с помощью класса SqlDataAdapter (см. документацию на этот класс в MSDN).
3. Модифицировать веб-сайт, разработанный Вами на предыдущей работе так, чтобы он использовал данные из БД.

Содержание отчета

1. Цель работы.
2. Вариант индивидуального задания.
3. Описание схемы БД.
4. Описание механизмов доступа к данным с использованием технологии доступа ADO.NET по коду программы.
5. Выводы по работе

Контрольные вопросы

1. Основные особенности использования технологии ADO.NET и ее возможности.
2. Основные классы, используемые для организации слоя доступа к данным, их особенности.
3. Как извлекаются и обрабатываются данные на уровне приложения.

Варианты индивидуальных заданий

Для выбранной ранее предметной области и разработанного в предыдущей работе приложения, организовать хранилище данных и доработать приложение так, чтобы доступ к хранилищу производился через ADO.NET.

Лабораторная работа №4

WEB-СЕРВИСЫ (СЛУЖБЫ)

Цель работы

Познакомиться с технологией создания и использования веб-сервисов.

Теоретические сведения

Web-сервисы

В процессе эволюции Интернета появилась необходимость в создании распределенных приложений. Приложения, установленные на компьютере, обычно используют библиотеки, размещенные на нем. Одну библиотеку могут использовать несколько программ. Можно ли размещать аналоги библиотек в Интернете, чтобы разные сайты могли их вызывать? Оказывается, да.

Предприятия на своих страницах предоставляют разнообразную информацию. Например, со своего сайта www.Ford.com компания "Форд" публикует информацию о моделях и ценах. Дилер этой компании хотел бы иметь эту информацию и на своем сайте. Web-служба позволяет сайту-потребителю получать информацию с сайта-поставщика. Сайт-потребитель показывает эту информацию на своих страницах. Код для генерации этой информации написан один раз, но может использоваться многими потребителями. Данные представлены в простом текстовом виде, поэтому ими можно пользоваться независимо от платформы.

Web-сервисы широко используются и в настольных, и в Интернет-приложениях. Они сами по себе являются не приложениями, а источниками данных для приложений. У них отсутствует пользовательский интерфейс. Web-сервисами необязательно пользоваться через сеть — они могут быть частью проекта, в котором используются.

Web-сервисы — это функциональность и данные, предоставляемые для использования внешними приложениями, которые работают с сервисами посредством стандартных протоколов и форматов данных. Web-сервисы полностью независимы от языка и платформы реализации. Технология Web-сервисов является краеугольным камнем программной модели Microsoft .NET.

Это дальнейшее развитие компонентного программирования CORBA и DCOM. Однако для использования таких компонентов необходимо регистрировать их в системе потребителя. Для web-служб это не требуется. Компоненты хорошо работают в локальных сетях. Протокол HTTP не

приспособлен для вызова удаленных процедур (RPC). Даже в одной организации часто используются разные операционные системы, которые могут взаимодействовать только через HTTP.

Было предпринято несколько попыток создания языка коммуникации между разнородными системами – DCOM, CORBA, RMI, IIOP. Они не получили всеобщего распространения, так как каждый из них продвигался разными производителями и поэтому был привязан к технологиям конкретного изготовителя. Никто не хотел принимать чужой стандарт. Чтобы выйти из этой дилеммы, несколько компаний договорились выработать независимый от производителя стандарт обмена сообщениями через HTTP. В мае 2000 года компании IBM, Microsoft, HP, Lotus, SAP, UserLand и другие обратились к W3C и выдвинули SOAP в качестве кандидата на такой стандарт. SOAP произвел революцию в области разработки приложений, объединив два стандарта Интернета — HTTP и XML.

SOAP

Для взаимодействия с web-сервисами применяется протокол SOAP, основанный на XML. Можно было бы использовать просто XML, но это слишком свободный формат, в нем каждый документ фактически создает свой язык. SOAP — это соглашение о формате XML-документа, о наличии в нем определенных элементов и пространств имен.

SOAP позволяет публиковать и потреблять сложные структуры данных, например, DataSet. В то же самое время его легко изучить. Текущая версия SOAP 1.2.

SOAP – это Простой Протокол Обмена Данными (Simple Object Access Protocol). SOAP создан для того, чтобы облегчать взаимодействие приложениям через HTTP. Это особый независимый от платформы формат обмена сообщениями через Интернет. Сообщение SOAP – это обычный XML-документ. Стандарт SOAP разрабатывает консорциум W3C.

SOAP-сообщение состоит из конверта (envelope), заголовка (header) и тела (body). Элементы body и envelope должны присутствовать всегда, а header необязателен. Элемент envelope – корневой. Элемент header может содержать специфичную для данного приложения информацию. В документе, сгенерированном web-сервисом, может присутствовать элемент fault, который описывает ошибку работы.

```
POST /WebSite5/WebService.asmx HTTP/1.1
Host: localhost
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length
<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```

xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <HelloWorld xmlns="http://localhost/webservices" />
  </soap12:Body>
</soap12:Envelope>

```

В ASP .NET можно легко как создать web-службу, так и пользоваться готовой.

Пользование web-службой

В Интернете существует множество готовых web-служб. Сайты <http://uddi.microsoft.com>, <http://www.webservicelist.com/> – каталоги различных сервисов. Чтобы получить информацию от web-службы, нужно только послать HTTP-запрос, в теле которого находится SOAP-сообщение. Запрос к службе <http://www.webserviceX.net/globalweather.asmx> на получение прогноза погоды в Москве выглядит так:

```

POST /globalweather.asmx HTTP/1.1
Host: www.webserviceX.net
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://www.webserviceX.NET/GetWeather"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetWeather xmlns="http://www.webserviceX.NET">
      <CityName>Moscow</CityName>
      <CountryName>Russian</CountryName>
    </GetWeather>
  </soap:Body>
</soap:Envelope>

```

Заголовок запроса отличается от запросов, которые обычно посылают браузеры, прежде всего полем Content-Type — text/xml; а не text/html; В теле запроса находится SOAP-сообщение. Сервис в ответ отправляет XML-документ:

```

<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://www.webserviceX.NET">
<?xml version="1.0" encoding="utf-16"?>
  <CurrentWeather>
    <Location>Moscow / Vnukovo , Russia (UUWW) 55-39N 037-
16E</Location>
    <Time>Aug 07, 2006 - 04:30 AM EDT / 2006.08.07 0830 UTC</Time>
    <Wind> from the E (080 degrees) at 11 MPH (10 KT):0</Wind>
    <Visibility> greater than 7 mile(s):0</Visibility>
    <SkyConditions> overcast</SkyConditions>
    <Temperature> 66 F (19 C)</Temperature>
    <DewPoint> 55 F (13 C)</DewPoint>

```

```

<RelativeHumidity> 68%</RelativeHumidity>
<Pressure> 29.85 in. Hg (1011 hPa)</Pressure>
<Status>Success</Status>
</CurrentWeather></string>

```

Чтобы использовать web-сервис, первым делом в проекте надо создать web-ссылку на удаленный объект — web-сервис. Выберите в меню Website пункт Add Web Reference (Добавить ссылку на службу). Появится диалоговое окно (рис.4).

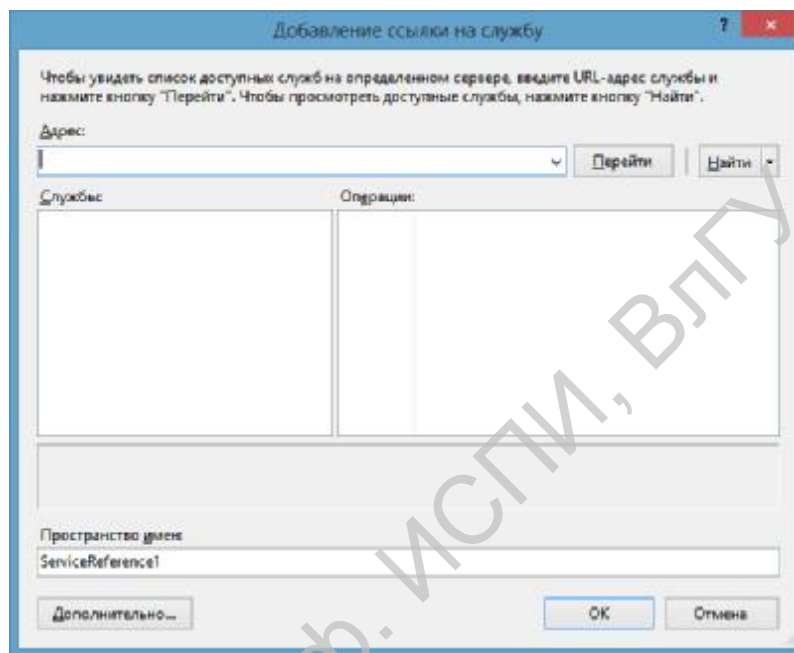


Рисунок 4. Диалоговое окно Add Web Reference

Введите URL web-сервиса с параметром в текстовое поле <http://www.cbr.ru/dailyinfowebser/dailyinfo.asmx>. В файл web.config добавляется настройка приложения:

```

<configuration>
...
  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding name="DailyInfoSoap"/>
      </basicHttpBinding>
    </bindings>
    <client>
      <endpoint
address="http://www.cbr.ru/dailyinfowebser/dailyinfo.asmx"
binding="basicHttpBinding"
bindingConfiguration="DailyInfoSoap"
contract="ServiceReference2.DailyInfoSoap" name="DailyInfoSoap"/>
    </client>
  </system.serviceModel>
</configuration>

```

Чтобы определить доступные по этому адресу web-сервисы, используется алгоритм DISCO. При этом создается файл dailyinfo.wsdl. WSDL (Web Service Discovery Language) — это язык описания web-сервисов. Это еще один тип XML-документов.

Чтобы облегчить работу с web-сервисами, используют прокси-классы. Они предоставляют разработчикам удобные функции и берут на себя преобразование их параметров в элементы XML, после чего посылают запрос web-сервису через Интернет. Утилита wsdl в составе .NET Framework поможет преобразовать его в прокси-класс: `wsdl dailyinfo.wsdl`.

Найдите эту утилиту в папке *C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\Bin* (версии могут варьироваться), скопируйте в папку, в которой находится файл dailyinfo.wsdl. С помощью командной строки или перетаскиванием откройте файл dailyinfo.wsdl утилитой wsdl. В этой же папке появится файл прокси-класса.

Прокси-класс необходимо поместить в папку App_Code, после чего объекты этого класса можно создавать в коде любой страницы. Программу wsdl можно запустить и удаленно. В командной строке: `wsdl http://www.cbr.ru/dailyinfowebsew/dailyinfo.asmx?wsdl`

В созданном файле объявлен класс DailyInfo, наследник System.Web.Services.Protocols.SoapHttpClientProtocol. Функции этого класса предназначены как для синхронного, так и для асинхронного вызова. Синхронные методы выполняются в том же потоке, что и запущены, а асинхронные запускают отдельный поток. Например, синхронная функция GetCursOnDate(DateTime On_Date) запрашивает параметры даты и возвращает узел с XML-документа.

Это текст можно использовать на странице:

```
...
<body>
<form id="form1" runat="server">
<div>

        <asp:Button ID="Button1" runat="server"
OnClick="Button1_Click" Text="Коды валют разных стран" />
        <asp:Label ID="Label1" runat="server"></asp:Label>
</div>

        <asp:GridView ID="GridView1" runat="server">
</asp:GridView>
</form>
</body>
...
```

```

protected void Button1_Click(object sender, EventArgs e)
{
    try
    {
        // создание экземпляра прокси-класса
        DailyInfo di = new DailyInfo();

        // запрос функции web-сервиса
        DataSet ds = di.EnumReutersValutes();

        //привязка источника данных к GridView
        GridView1.DataSource = ds;
        GridView1.DataBind();

        //получение XML-узла, содержащего информацию о
валютках

        XmlNode node = di.EnumReutersValutesXML();
        //вывод текста узла
        Label1.Text = node.InnerText;
    }
    catch(Exception ex) { Label1.Text = "Что-то пошло не так
: (" + ex.Message; }
        //обработка исключений
    }
}

```

Создание web-сервиса

Создание web-сервиса немногим отличается от создания обычной страницы. Есть два варианта: можно создать отдельный проект или вставить web-сервис в существующий проект. В первом случае другие проекты должны создавать web-ссылку, чтобы обращаться к сервисам этого проекта. Файл web-сервиса имеет расширение asmx. Файл web-сервиса должен начинаться с директивы WebService. Класс web-сервиса может быть потомком класса System.Web.Services.WebService.

Если при объявлении web-сервиса вы породили его от класса System.Web.Services.WebService, то вы автоматически получаете доступ к глобальным объектам приложения ASP .NET Application, Context, Session, Server и User. Если же вы создавали класс web-сервиса как-то иначе — ничего страшного. Вы все равно можете получить доступ к вышеперечисленным свойствам с помощью соответствующих свойств статического HttpContext.Current.

Методы, которые сервис предоставляет для вызова с помощью SOAP-запросов, должны иметь атрибут WebMethod. У атрибута WebMethod существует 6 свойств, влияющих на работу метода.

Создадим новое приложение в Visual Studio и добавим к нему файл web-службы Customers.asmx (см. рис.5).

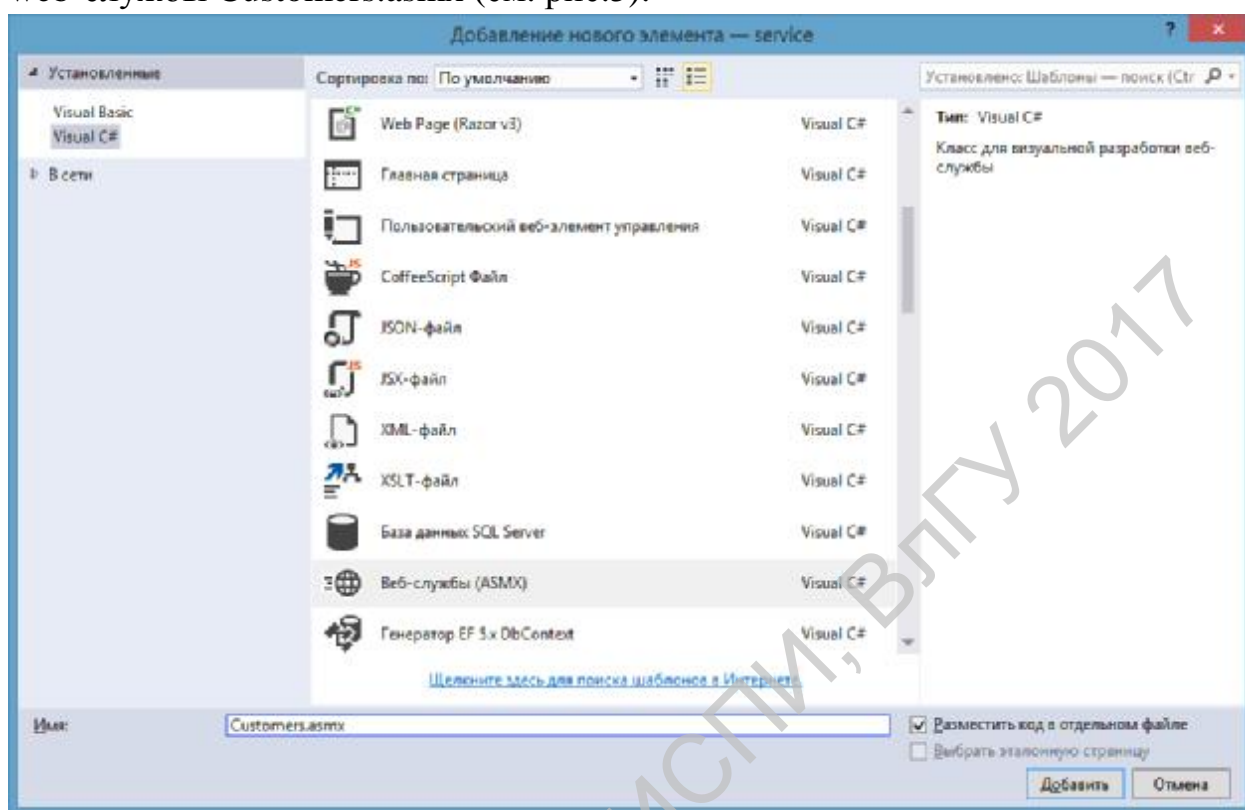


Рисунок 5. Добавление новой Web-службы

Файл Customers.asmx содержит единственную строку – директиву WebService, которая утверждает, что этот файл – web-сервис.

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/Customers.cs"
Class="Customers" %>
```

У директивы WebService всего 4 возможных атрибута. CodeBehind, который раньше был атрибутом и директивы Page, определяет физический путь к файлу отделенного (фонового) кода. Атрибут Class обязателен и определяет класс, который реализует функциональность web-сервиса. Debug и Language аналогичны тем же атрибутам директивы Page:

Файл с расширением .asmx — точка входа создаваемого web-сервиса. Класс System.Web.Services.WebService, который обычно наследуется класс сервиса, предоставляет доступ к глобальным объектам Application и ViewState.

Весь код web-сервиса будет располагаться в codebehind-файле Service.asmx.cs. Изначально этот файл (созданный в Visual Studio .NET и располагающийся, как Вы, наверное, догадались в папке App_Code данного проекта) имеет следующий вид:

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

```

using System. Web;
using System. Web. Servi ces;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles. BasicProfile1_1)]
// Чтобы разрешить вызывать веб-службу из скрипта с помощью
ASP. NET AJAX, раскомментируйте следующую строку.
// [System. Web. Script. Servi ces. ScriptService]
public class Customers : System. Web. Servi ces. WebService
{
    public Customers()
    {
        //Раскомментируйте следующую строку в случае
использования сконструированных компонентов
        //InitializeComponent();
    }

    [WebMethod]
    public string HelloWorld()
    {
        return "Привет всем!";
    }
}

```

Атрибут `WebServiceBinding` удостоверяет соответствие откликов web-сервиса WS-I Basic Profile 1.0 release требованиям организации WS-I (Web Services Interoperability organization), которая занимается вопросами межплатформенной совместимости web-сервисов.

Метод `HelloWorld` создан Visual Studio в качестве примера начинающим разработчикам.

Web-сервис может состоять из множества классов. Однако только один класс в web-сервисе может иметь методы, помеченные атрибутом `WebMethod` (которые можно вызывать через SOAP-запросы).

Когда страница web-сервиса запрашивается в браузере, он возвращает автоматически генерируемую страницу помощи по данному сервису. Откроем в браузере страницу `Customers.asmx`. В браузере появится страница, как на рис.6.

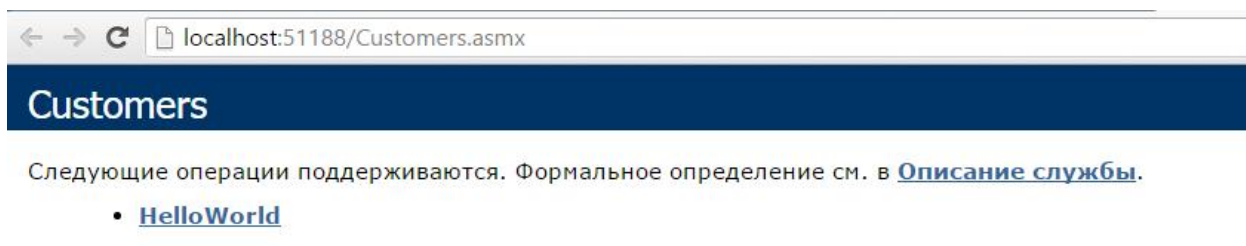


Рисунок 6. Внешний вид окна Customers

ASP .NET для отображения web-сервиса использует файл шаблона DefaultWsdHelpGenerator.aspx, расположенный в папке `<%SYSTEM_ROOT%\Microsoft.NET\Framework\<version>\CONFIG`. На выводимой странице web-сервиса есть название web-сервиса, ссылка на описание сервиса и список web-методов, объявленных в web-сервисе.

Остальная часть страницы посвящена тому, что необходимо изменить пространство имен по умолчанию для web-сервиса `http://tempuri.org/` на собственное. При публикации службы нужно изменить параметр Namespace атрибута WebService класса web-сервиса: `[WebService(Namespace = "...")]` на уникальное. Например,

```
[WebService(Namespace = "http://lab3.i stx05. org/")] .
```

Кроме того, в атрибуте WebService можно задать свойства Name и Description, и они появятся на странице помощи по web-сервису:

```
[WebService(Namespace =  
"http://lab3.i stx05. org/",  
Name = "Мой веб-сервис",  
Description = "Здесь следует сказать немного слов о вашем  
сервисе")]
```

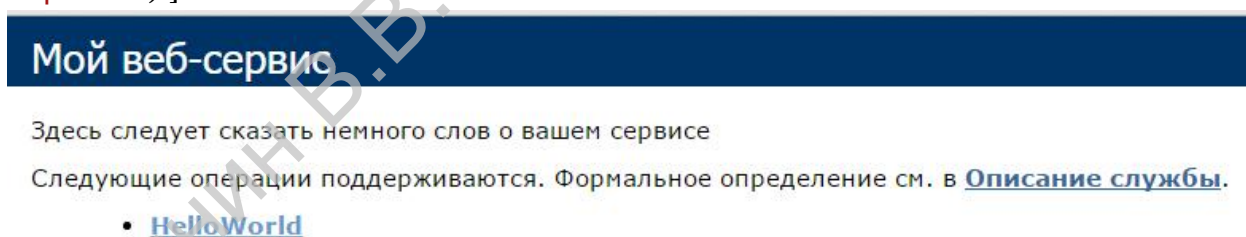


Рисунок 7. Название и описание сервиса

Итак, перейдем к странице описания web-метода HelloWorld (просто кликните по ссылке HelloWorld на странице описания web-сервиса).

Как видите, на ней также присутствует название web-сервиса, ссылка на первую страницу web-сервиса, название web-метода. Кроме этого, на странице расположена кнопка Invoke (Запуск), предназначенная для вызова web-метода через GET-запрос (данное правило не выполняется, если web-метод не может быть вызван таким образом), и примеры запросов для вызова данного web-метода с помощью SOAP, HTTP POST (если такой вызов

возможен) и HTTP GET (если такой вызов возможен). Также представлены примеры ответов вызова web-метода. Протестируем нашу работу с помощью страницы web-сервиса, нажав кнопку Запуск. В ответ получаем:

```
<string xmlns="http://lab3.istx05.org/">Привет всем!</string>
```

От такого web-сервиса нет особенной пользы, поэтому создадим новый метод, который создает и возвращает объект класса List:

```
[WebMethod(Description = "Этот метод расскажет вам о всех клиентах в базе.")]
```

```
public List<Customer> GetCustomersInfo()
{
    List<Customer> data = new List<Customer>();
    data.Add(new Customer("Иван Иванов", DateTime.Now));
    data.Add(new Customer("Петр Петров", DateTime.Today));
    return data;
}
```

Как видите, здесь используется класс Customer, который необходимо создать в каталоге App_Code:

```
public class Customer
{
    public Guid ID { get; set; }
    public string FIO { get; set; }
    public DateTime CreationDate { get; set; }
    public Customer(string FIO, DateTime cr)
    {
        ID = Guid.NewGuid();
        //Guid - глобальный уникальный идентификатор длиной 128 бит
        //метод Guid.NewGuid() создает новую последовательность бит,
        //отличную от предыдущих
        this.FIO = FIO;
        CreationDate = cr;
    }
}
```

Попробуйте самостоятельно использовать List<Customer> в качестве источника данных для GridView в Вашем веб-приложении.

При разработке необходимо учитывать, что в веб-сервисах, которые могут использоваться приложениями-клиентами на различных языках, не стоит использовать DataSet'ы. В некоторых языках они просто не поддерживаются (например, в Java).

Другие аргументы атрибута WebMethod:

- `CacheDuration` определяет промежуток времени в секундах, на которое кэшируется web-сервис. По умолчанию равно 0, что значит, что кэширование отключено;
- `Description` — описание метода, которое выводится на странице сервиса под ссылкой на страницу метода;
- `EnableSession` позволяет включить поддержку сессий. По умолчанию поддержка сессий в web-сервисах отключена. Чтобы включить ее, определите web-метод следующим образом:
[WebMethod(EnableSession=true)];
- `MessageName` определяет уникальное имя метода. Позволяет использовать перегруженные функции с одним именем, но разными сигнатурами.

`TransactionOption` управляет поддержкой транзакций. По умолчанию она отключена. Web-сервисы ограниченно поддерживают транзакции, то есть веб-сервис может порождать транзакцию, но при этом не может быть участником другой транзакции. Возможные значения аргумента — `Disabled`, `NotSupported`, `Supported`, `Required`, и `RequiresNew`. Если вызывается web-метод с `TransactionOption`, установленным в `Required` или `RequiresNew`, а в нем вызывается другой web-метод с такими же установками, каждый из этих методов инициирует свою транзакцию.

Метод `GetCustomersInfo` возвращает тип `List`. Посмотрим, как он помещается в SOAP:

```
<ArrayOfCustomer
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://lab3.istx05.org/">
  <Customer>
    <ID>6e59e4f1-8f1f-484f-b5ea-846239d6e8cb</ID>
    <FIO>Иван Иванов</FIO>
    <CreationDate>2016-05-
19T21:00:14.7061316+03:00</CreationDate>
  </Customer>
  <Customer>
    <ID>cca64d4c-afef-4aa8-91e3-40afe898e9d8</ID>
    <FIO>Петр Петров</FIO>
    <CreationDate>2016-05-19T00:00:00+03:00</CreationDate>
  </Customer>
</ArrayOfCustomer>
```

Это почти готовая схема `xsd`. Из нее можно, например, сгенерировать класс бизнес-логики. Использовать созданный сервис можно и удаленно, и из приложения, в котором он находится. Стать потребителем этого сервиса

могло бы и обычное Windows-приложение, написанное на C++, C# или Visual Basic. Web-сервисы тоже могут пользоваться услугами других web-сервисов.

Порядок выполнения работы

1. Создать веб-сервис на основе примера в п.2.4, который возвращал бы данные, извлеченные из БД (см работу №6).
2. Модифицировать веб-приложение, разработанное в предыдущей работе таким образом, чтобы оно получало данные из сервиса, созданного в предыдущем пункте.
3. Ознакомиться со службой DailyInfo (<http://www.cbr.ru/dailyinfowebsew/dailyinfo.aspx>). Создать настольное WindowsForms-приложение, которое использовало бы одну из функций веб-сервиса.
4. Взять вариант индивидуального задания и создать свой сервис и использующее его веб-приложение в соответствии с ним (веб-приложение и сервис лучше расположить в разных проектах).

Содержание отчета

1. Цель работы.
2. Вариант индивидуального задания.
3. Словесное описание сервиса.
4. Исходный код разработанного сервиса и примеры работы с ним.
5. Выводы по работе

Контрольные вопросы

1. Что такое WEB-сервис? Основные особенности.
2. Каковы основные элементы типового WEB-сервиса?

Варианты индивидуальных заданий

Вариант работы взять в соответствии с интересующей предметной областью или из темы курсовой работы (если таковая существует), предварительно согласовав его с преподавателем. Реализовать функции системы в виде Web-методов соответствующего сервиса.

Лабораторная работа №5

ШАБЛОН ПРОЕКТИРОВАНИЯ MVC В ASP.NET

Цель работы

Познакомиться с технологией создания и использования шаблона проектирования MVC на платформе ASP.NET.

Общие сведения

Model-View-Controller (MVC)

Model-view-controller (MVC, «Модель-вид-поведение», «Модель-представление-контроллер») — архитектура программного обеспечения, в которой модель данных приложения, пользовательский интерфейс и управляющая логика разделены на три отдельных компонента, так, что модификация одного из компонентов оказывает минимальное воздействие на другие компоненты. Шаблон MVC позволяет разделить данные, представление и обработку действий пользователя на три отдельных компонента (рис.8).

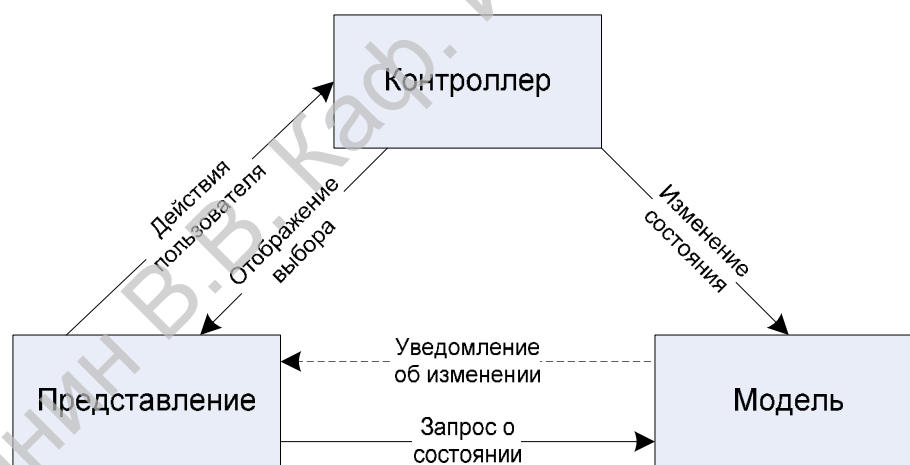


Рисунок 8. Структурная схема шаблона MVC

- 1) **Модель (Model)**. Модель представляет собой данные, которыми оперирует приложение. Она предоставляет данные для View, а также реагирует на запросы от контроллера, изменяя свое состояние.
- 2) **Представление (View)**. Представляет собой компонент системы для отображения состояния модели в понятном человеку представлении. В нашем случае это aspx (cshtml) – страницы. Представление не изменяет данные напрямую (режим «Только чтение»), данные изменяются при помощи контроллера.

- 3) **Контроллер (Controller).** Является средством, при помощи которого пользователи взаимодействуют с системой, а также является управляющим элементом для обмена данными между представлением и моделью.

Важное свойство MVC: модель не зависит ни от представления, ни от контроллера, что позволяет одновременно строить различные интерфейсы пользователя для взаимодействия с одной и той же моделью данных.



Рисунок 9. Структурная схема шаблона MVC для веб-приложения

Жизненный цикл запроса.

На следующей диаграмме показан жизненный цикл запроса в ASP.NET MVC Framework. Необходимо отметить, что большей частью данный ЖЦ характерен и для большинства реализация MVC на других платформах и не является самобытным.

При обращении к веб-приложению пользователь запрашивает какой-либо URL-адрес. Далее запрос направляется в таблицу маршрутов (Global.asax.cs), где устанавливается соответствие между запрашиваемым URL-адресом и контроллером. Далее таблица маршрутов выбирает контроллер + его действие и отправляет запрос туда. Действие может работать с данными или ничего не делать, т.е. быть пустым. Далее действие производит возврат представления, которое отправляется пользователю в виде разметки (HTML или XML).

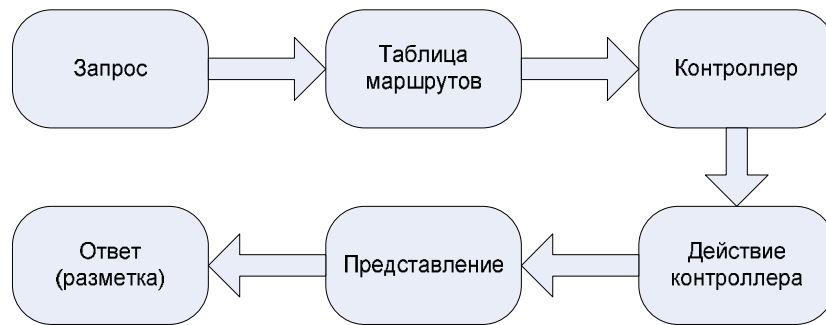


Рисунок 10. Жизненный цикл запроса

Работа с веб-формами

На следующем рисунке показана диаграмма взаимодействий процесса вывода веб-формы в MVC. Когда пользователь запрашивает страницу (через GET), на которой присутствует веб-форма, его запрос передается контроллеру, который возвращает нужное представление. Пользователь заполняет веб-форму, и отправляет данные с неё на сервер (через POST). Контроллер получает данные и проводит их валидацию. Если валидация прошла успешно, то он передает эти данные в модель, получает данные для вывода и отправляет в представление.

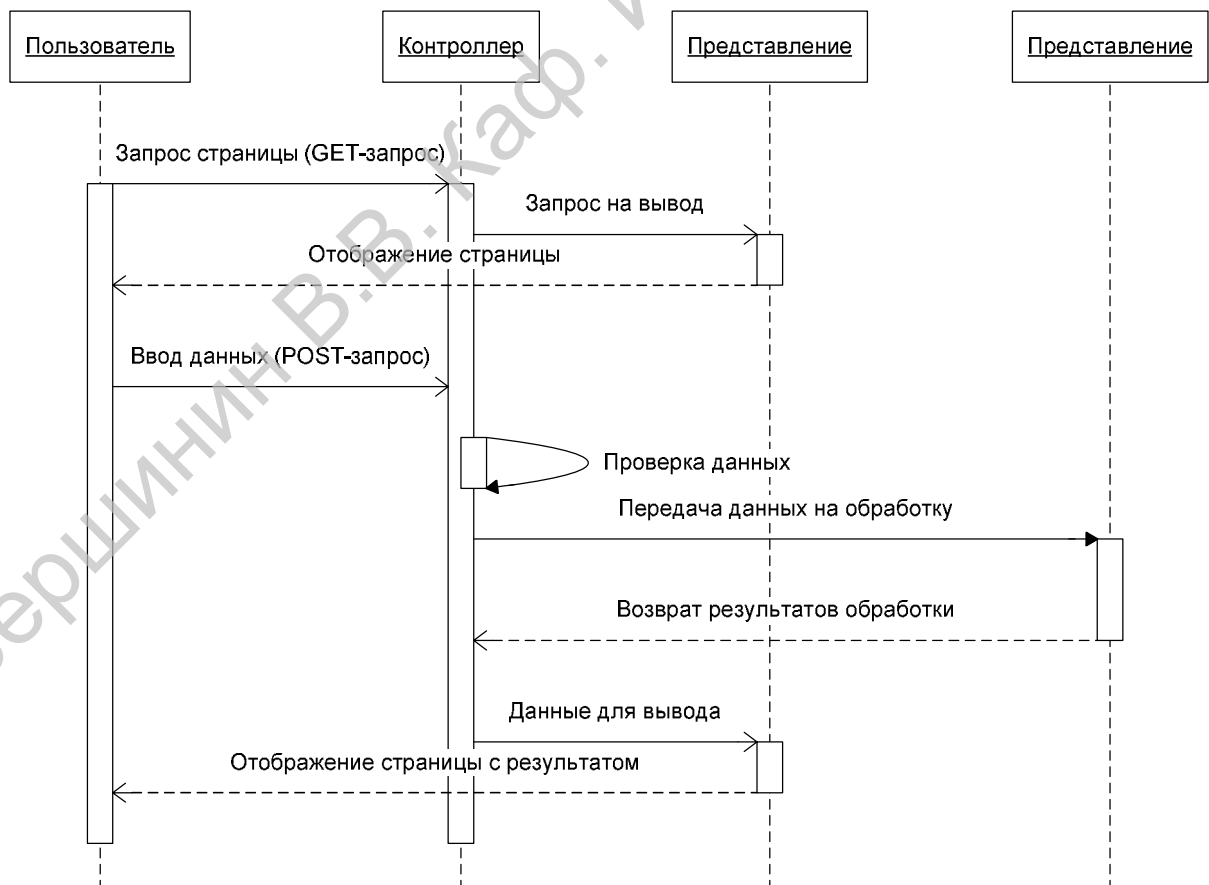


Рисунок 11. Диаграмма последовательностей процесса вывода веб-формы в MVC

Необходимый инструментарий

- NET Framework 4.0 SP1
- Microsoft Visual Studio 2015 Express for Web
- ASP.NET MVC Framework

Ход работы

Создание шаблона MVC

Чтобы создать шаблон MVC в Visual Studio выбираем **File / New / Project...**

В списке выбираем **Visual C# / Web** и указываем **ASP.NET Web Application**, создаем веб-приложение ASP. В открывшемся окне выбираем шаблон MVC (рис. 12):

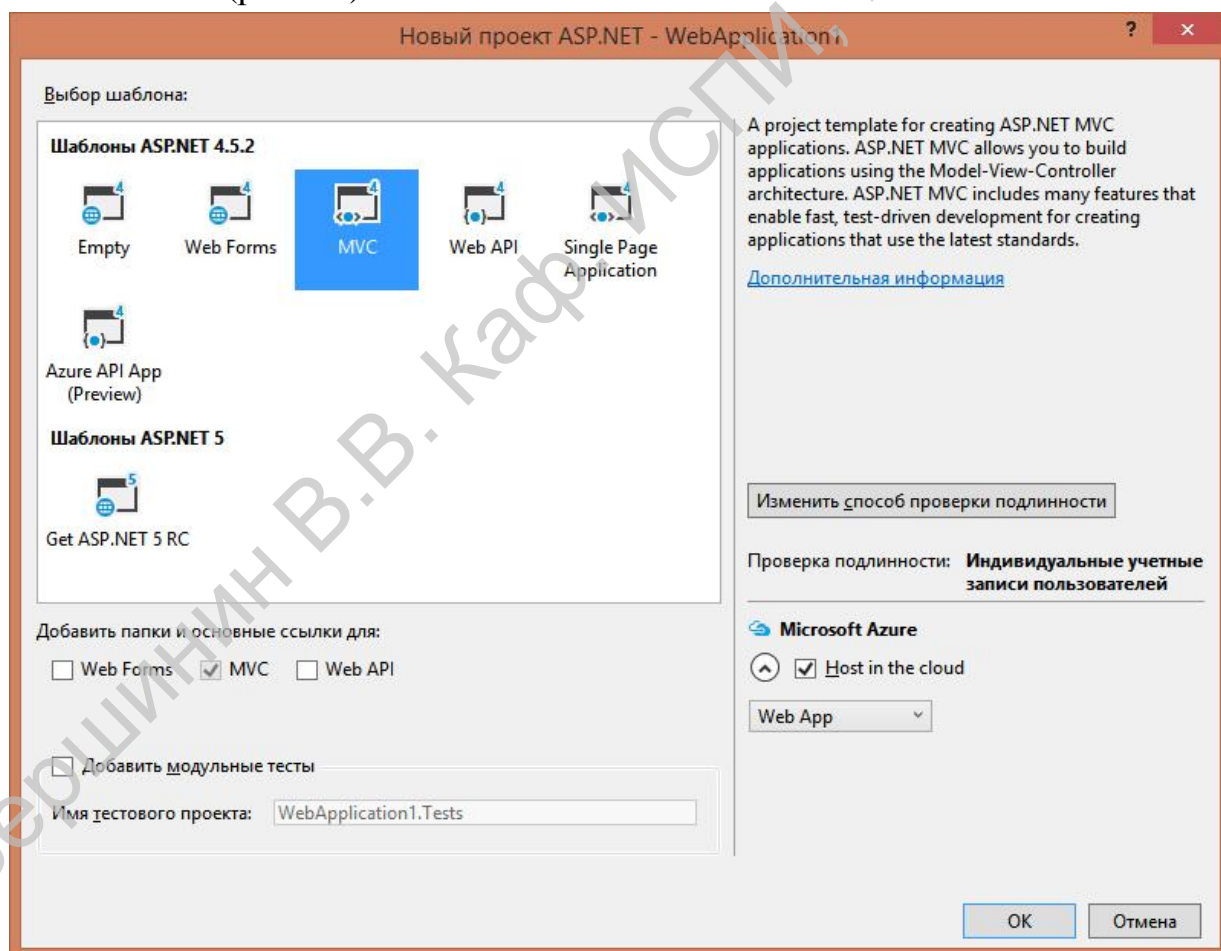


Рисунок 12. Диалог создания проекта

Перед созданием шаблона студия предложит создать к нему Unit-тесты (модульные тесты). Поставим галочку «Добавить модульные тесты» (см. рис. 12).

В результате в окне Solution Explorer появится структура проекта, содержащего шаблон MVC:

- /Properties — стандартная директория для WebApplication, содержит настройки проекта.
- /References (/Ссылки) — ссылки на другие сборки, проекты и т.д.
- /App_Data — специальная директория для хранения данных.
- /Content — в этой директории должны храниться изображения, стили и весь статический контент веб-приложения.
- /Controllers — классы, отвечающие за компоненты Controller.
- /Models — классы-компоненты Models.
- /Scripts — содержит библиотеки для работы в Ajax.
- /Views — непосредственно UI (*user interface*) приложения. В этой директории создаются директории для каждого контроллера (/Views/Home в нашем случае). И уже в них по aspx(chtml)-странице для каждого из методов контроллера.
- /View/Shared — содержит то, что может пригодиться для всех контроллеров, например MasterPages и UserControls.

Удаление из шаблона примеров кода

По умолчанию в шаблоне созданы представления и контроллеры для раздела Home. Для того, чтобы можно было наполнить этот раздел другой логикой, необходимо удалить в Solution Explorer следующие файлы:

- \Controllers\HomeController.cs (и все другие контроллеры)
- Все представления из папки Views

А также удалить методы Index() и About() (или другие подобные) из проекта MvcApplication.Tests файла-контроллера HomeControllerTest.cs.

Создание (подключение) БД

ASP.NET MVC Framework может работать с любой современной СУБД, включая Microsoft SQL Server, Oracle, MySQL и пр. В данной работе будем использовать БД, созданную в работе №3. Необходимо убедиться, что таблица имеет первичный ключ и содержит какие-либо данные.

В рамках данного примера рассматривается файловая БД MS SQL Express с единственной таблицей GuestBook.

Создание модели данных

ASP.NET MVC Framework включает в себя 3 главных компонента: модель представление и контроллер. Создадим первый компонент шаблона MVC, который будет представлять модель данных. В Visual Studio нажимаем правой кнопкой мыши на папке Model, выбираем **Add, New Item...** (**Добавить -> Создать элемент**) и создаем следующий класс Records (листинг 1):

Листинг 1 – Models\Records.cs

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

```
namespace MvcApplication.Models
{
```

```
    public class Records
    {
```

```
        public int ID { get; set; }
        public string Title { get; set; }
        public string Author { get; set; }
        public DateTime PublicationDate { get; set; }
```

```
        public List<Records> RecordsList { get; set; }
```

/* КСТАТИ: Конструкция вида public int ID { get; set; } - это auto property.

* Она эквивалентна следующему фрагменту кода:

```
*
* private int id;
* public int ID
* {
*     get { return id; }
*     set { id = value; }
* }
*/
```

```
    }
}
```

Фактически структура модели повторяет структуру в таблицы в БД (плюс некоторые дополнительные методы) – она позволяет из программы работать не с полями таблицы, а с объектом класса модели. Для каждой таблицы в БД создается отдельный класс в модели.

Реализация шаблона проектирования Data Access Object

Шаблон проектирования **Объект доступа к данным (Data Access Object – DAO)** используется для инкапсуляции доступа к источнику данных. Через данный объект в нашем случае будет обеспечиваться взаимодействие приложения и базы данных. Data Access Object всегда выносится в один или

несколько дополнительных классов, работающих с источником данных, и не является частью триады Модель-Представление-Контроллер. Для каждой таблицы в БД необходимо создавать отдельный класс в DAO. На Листинге 2 показан класс DAO, реализующий соединение с базой данных. На Листинге 3 представлен RecordsDAO – наследник от DAO – класс, реализующий работу с данными в таблице GuestBook. Создайте папку DAO и внесите эти классы в приложение.

Листинг 2 – Класс, реализующий соединение с БД – DAO\DAO.cs

```
using System.Data.SqlClient;

namespace mvcApplication.DAO
{
    public class DAO
    {
        private const string ConnectionString =
            "Data Source=.\SQLEXPRESS;AttachDbFilename=\"D: \\Work\\Университет\\Семестр
            #9 (сен-дек 2009)\\Test\\mvcApplication\\App_Data\\Test.mdf\";
            Integrated Security=True; User Instance=True";
        private SqlConnection Connection { get; set; }

        public void Connect()
        {
            Connection = new SqlConnection(ConnectionString);
            Connection.Open();
        }

        public void Disconnect()
        {
            Connection.Close();
        }
    }
}
```

Листинг 3 – Класс, реализующий взаимодействие с таблицей GuestBook – DAO\RecordsDAO.cs

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using mvcApplication.Models;

namespace mvcApplication.DAO
{
    public class RecordsDAO : DAO
    {
        public List<Records> GetAllRecords()
        {

```

```

Connect();
List<Records> recordList = new List<Records>();

try
{
    SqlCommand command = new SqlCommand("SELECT *
FROM GuestBook", Connection);
    SqlDataReader reader = command.ExecuteReader();
    while (reader.Read())
    {
        Records record = new Records();
        record.ID = Convert.ToInt32(reader["ID"]);
        record.Author
Convert.ToString(reader["Author"]);
        record.Title
Convert.ToString(reader["Text"]);
        record.PublicationDate
Convert.ToDateTime(reader["PublicationDate"]);
        recordList.Add(record);
    }
    reader.Close();
}
catch (Exception)
{
    // Не обрабатывать исключительные ситуации -
дурной тон
}
finally
{
    Disconnect();
}
return recordList;
}
}
}

```

Создание контроллера

Контроллер (Controller). Является средством, при помощи которого пользователи взаимодействуют с системой, а также является управляющим элементом для обмена данными между представлением и моделью.

Для создания контроллера нажимаем правой кнопкой мыши на папке **Controllers** и выбираем **Add/Controller...** В появившемся окне выбираем пункт **MVC 5 Controller with read/write actions** – это позволит создать несколько action-методов, которые будут реагировать на запросы от представления (рис. 13).

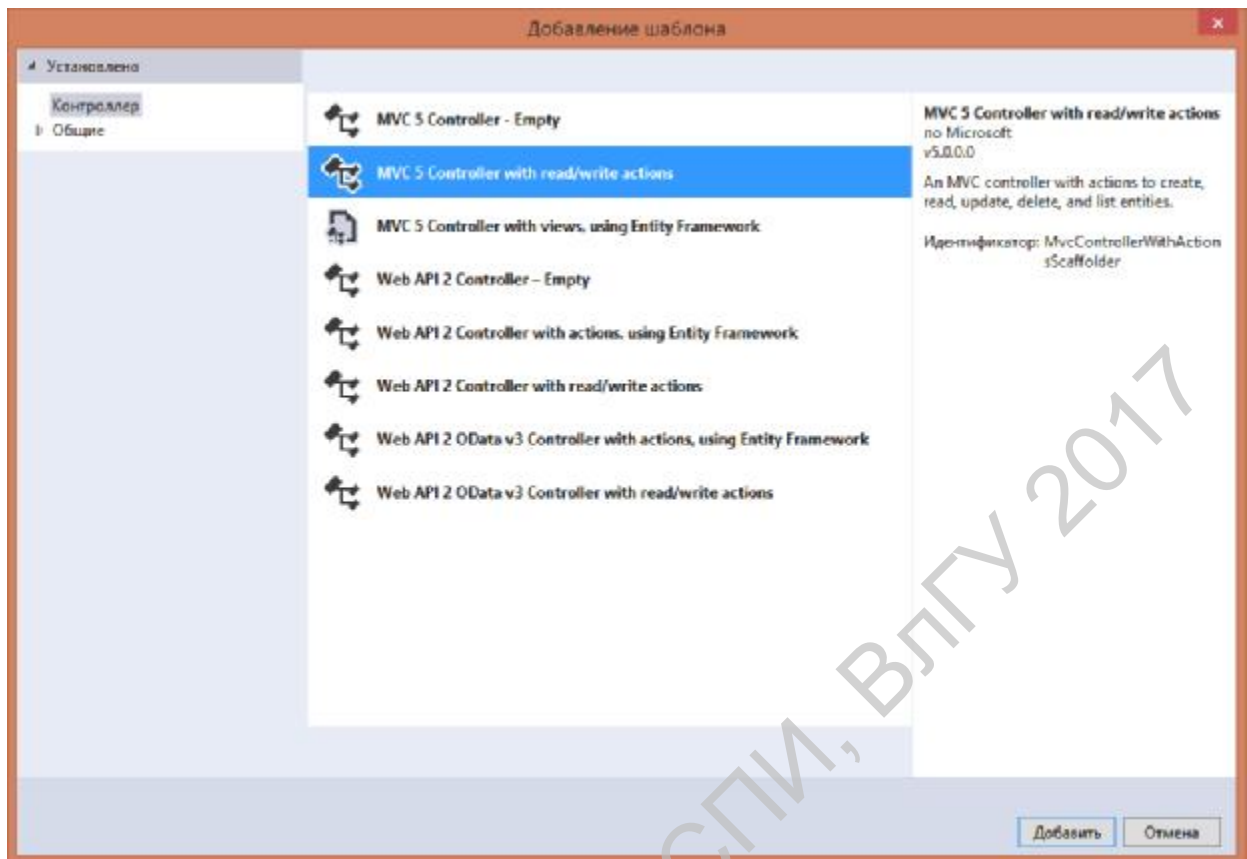


Рисунок 13. Добавление контроллера

В следующем окне задайте имя контроллера – HomeController.

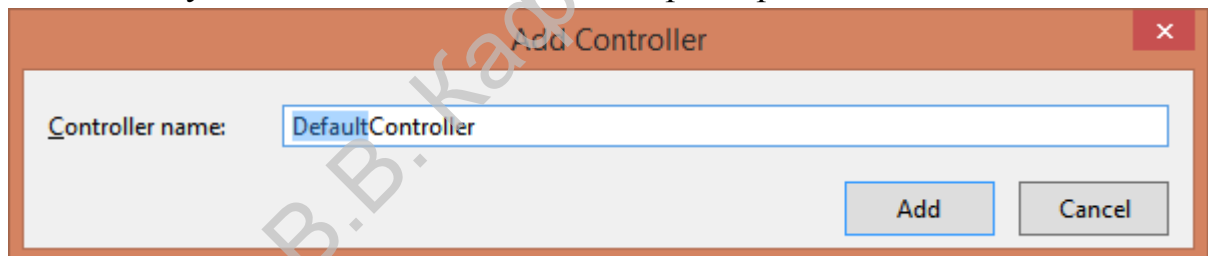


Рисунок 14. Имя контроллера

В результате создания Home контроллера система сгенерирует следующий класс (Листинг 4):

Листинг 4 – Controllers\HomeController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace WebApplication1.Controllers
{
    public class ToursController : Controller
```

```

{
    // GET: Tours
    public ActionResult Index()
    {
        return View();
    }

    // GET: Tours/Details/5
    public ActionResult Details(int id)
    {
        return View();
    }

    // GET: Tours/Create
    public ActionResult Create()
    {
        return View();
    }

    // POST: Tours/Create
    [HttpPost]
    public ActionResult Create(FormCollection collection)
    {
        try
        {
            // TODO: Add insert logic here

            return RedirectToAction("Index");
        }
        catch
        {
            return View();
        }
    }

    // GET: Tours/Edit/5
    public ActionResult Edit(int id)
    {
        return View();
    }

    // POST: Tours/Edit/5
    [HttpPost]

```

```

collection) public ActionResult Edit(int id, FormCollection
collection)
{
    try
    {
        // TODO: Add update logic here

        return RedirectToAction("Index");
    }
    catch
    {
        return View();
    }
}

// GET: Tours/Delete/5
public ActionResult Delete(int id)
{
    return View();
}

// POST: Tours/Delete/5
[HttpPost]
collection) public ActionResult Delete(int id, FormCollection
collection)
{
    try
    {
        // TODO: Add delete logic here

        return RedirectToAction("Index");
    }
    catch
    {
        return View();
    }
}
}
}

```

Чтобы можно было передавать в представление данные из модели на страницу (сделаем вывод всех контактов на странице), нужно модифицировать Home контроллер следующим образом:

Листинг 5 – Модификация Controllers\HomeController.cs

```

public class HomeController : Controller
{
    RecordsDAO recordsDAO = new RecordsDAO();

    //
    // GET: /Home/
    public ActionResult Index()
    {
        return View(recordsDAO.GetAllRecords());
    }
}

```

Также добавим пространства имен:

```

using mvcApplication.Models;
using mvcApplication.DAO;

```

Сборка проекта

Данное действие выделено в отдельный пункт, т.к. для генерации представлений необходимо иметь собранные Models и Controllers. В Visual Studio выполняем команду из меню **Build / Build Solution (Сборка -> Собрать решение)**.

Создание представлений

После сборки проекта можно приступить к заключительной части создания триады шаблона MVC – добавлению представлений (aspx (cshtml)-страниц).

В файле **Controllers\HomeController.cs** над методом **public ActionResult Index()** либо внутри него щелкаем правой кнопкой мыши и выбираем **Add View...** (либо в Solution Explorer на папке **Views\Home** и выбираем **Add / View...** и добавляем одноименное методу представление).

В диалоге Add View нажимаем Create a strongly-typed view. Выберем класс для представления **mvcApplication.Models.Records** и View content типа **List**.

В после нажатия кнопки Add получим файл следующего содержания:

Листинг 6 – Views\Home\Index.cshtml

```

<%@
    Page
    Title=""
    Language="C#"
    MasterPageFile="~/Views/Shared/Site.Master"
    Inherits="System.Web.Mvc.ViewPage<IEnumerable<mvcApplication.Models.Records>>" %>
    <asp:Content ID="Content1" ContentPlaceHolderID="TitleContent"
    runat="server">
        Index
    </asp:Content>
    <asp:Content ID="Content2" ContentPlaceHolderID="MainContent"
    runat="server">

```



```

<h2>Index</h2>
<table>
  <tr>
    <th></th>
    <th>
      ID
    </th>
    <th>
      Title
    </th>
    <th>
      Author
    </th>
    <th>
      PublicationDate
    </th>
  </tr>
  <% foreach (var item in Model) { %>
    <tr>
      <td>
        <%= Html.ActionLink("Edit", "Edit", new { /*
id=item.PrimaryKey */ }) %> |
        <%= Html.ActionLink("Details", "Details", new {
/* id=item.PrimaryKey */ }) %>
      </td>
      <td>
        <%= Html.Encode(item.ID) %>
      </td>
      <td>
        <%= Html.Encode(item.Title) %>
      </td>
      <td>
        <%= Html.Encode(item.Author) %>
      </td>
      <td>
        <%=
item.PublicationDate)) %>
        Html.Encode(String.Format("{0:g}",
      </td>
    </tr>
  <% } %>
</table>
<p>
  <%= Html.ActionLink("Create New", "Create") %>
</p>
</asp:Content>

```

Необходимо изменить это представление – заменить строки:

```

<%= Html.ActionLink("Edit", "Edit", new { /* id=item.PrimaryKey */ }) %> |
<%= Html.ActionLink("Details", "Details", new { /* id=item.PrimaryKey */ }) %>

```

на следующие:

```
<%= Html.ActionLink("Edit", "Edit", new { ID = item.ID }) %> |  
<%= Html.ActionLink("Details", "Details", new { ID = item.ID }) %>
```

После создания представления можно приступить к самому главному – запустить проект, нажав в меню на **Debug / Start Debugging** или клавишу F5.

Представление Index будет вызвано по умолчанию. В нем будут выведены все данные, которые находятся в таблице GuestBook (рис. 15).

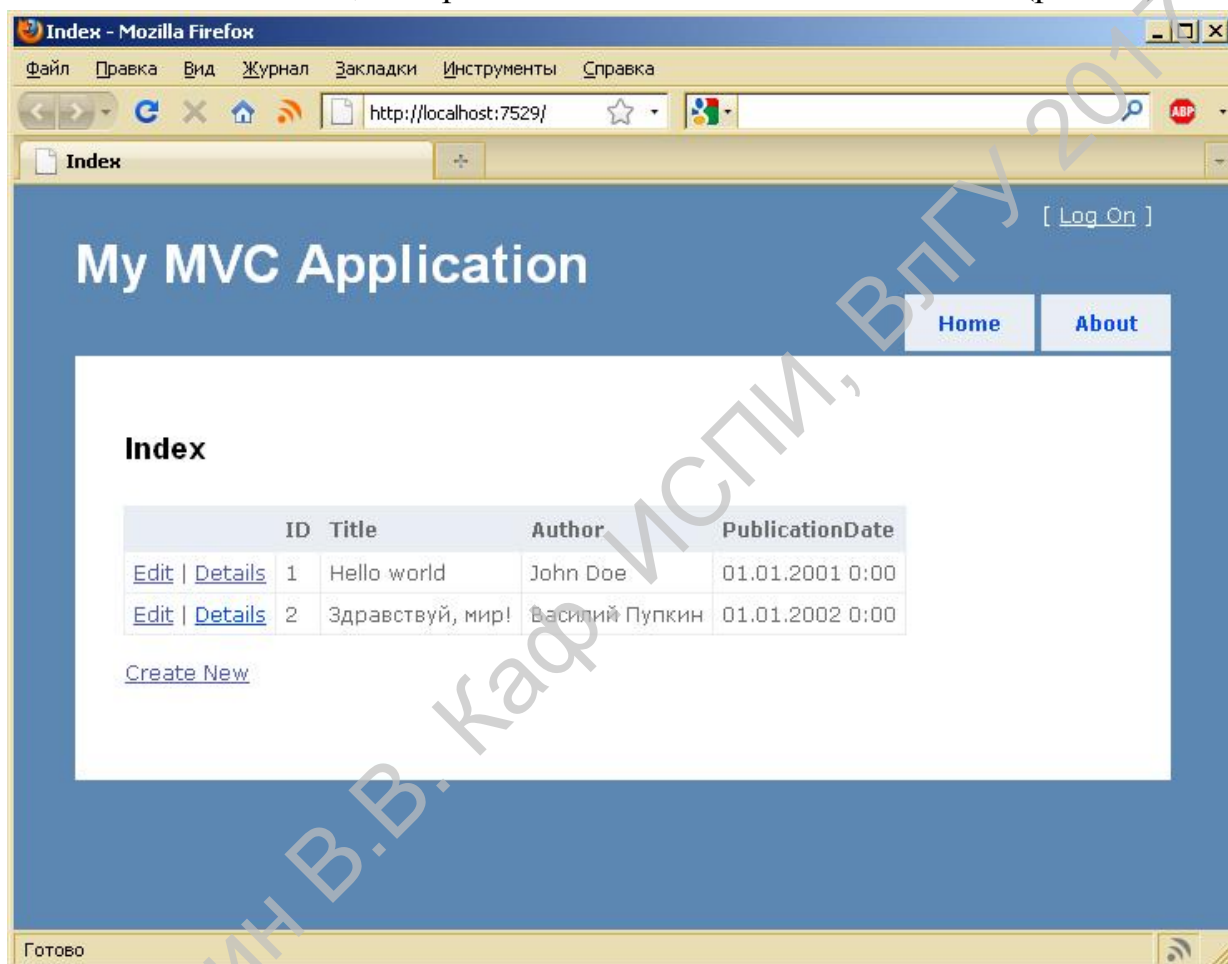


Рисунок 15. Index-представление

Данное представление содержит ссылку **Create New**, по нажатию на которую будем переходить к добавлению нового контакта. Следующий пункт посвящен именно этому.

Добавление новой записи в таблицу GuestBook из веб-приложения

Чтобы можно было добавлять новые записи в таблицу GuestBook, необходимо:

1. добавить метод в класс RecordsDAO, который будет добавлять запись в БД;
2. добавить 2 метода Create() в Home-контроллер;
3. добавить Create-представление.

Добавим в класс RecordsDAO следующий метод, который будет добавлять новую запись в таблицу GuestBook:

Листинг 7 – Добавление метода в DAO\RecordsDAO.cs

```
public bool AddRecord(Records records)
{
    bool result = true;
    Connect();

    try
    {
        SqlCommand cmd = new SqlCommand(
            "INSERT INTO GuestBook (Title, Author,
PublicationDate) " +
            "VALUES (@Title, @Author, @Date)",
            Connection);

        cmd.Parameters.Add(new SqlParameter("@Title",
records.Title));
        cmd.Parameters.Add(new SqlParameter("@Author",
records.Author));
        cmd.Parameters.Add(new SqlParameter("@Date",
records.PublicationDate));

        cmd.ExecuteNonQuery();
    }
    catch (Exception)
    {
        result = false;
    }
    finally
    {
        Disconnect();
    }
    return result;
}
```

Если страница, которую мы создаем, содержит веб-форму, то в контроллере обязательно будет 2 метода. Первый будет срабатывать при простом обращении к странице (так называемый GET-запрос) через URL. Он будет просто возвращать представление, которое содержит веб-форму. Второй, помеченный атрибутом `[AcceptVerbs(HttpVerbs.Post)]`, будет срабатывать при POST-запросе к странице, при котором будет происходить передача данных с формы на сервер. Через него и будет добавляться новый контакт в базу данных.

Обратите внимание на атрибут `[Bind(Exclude = "Id")]` – он исключает из принимаемых от клиента данных Id элемента. Дело в том, что

Id является в таблице GuestBook первичным ключем, и в тоже время автоинкрементом. Чтобы программа потенциально не могла принять от пользователя данное поле (оно проставляется автоматически внутри СУБД), оно с помощью данного атрибута исключается из принимаемого объекта.

Листинг 8 – Добавление методов в Controllers\HomeController.cs

```
//
// GET: /Home/Create
public ActionResult Create()
{
    return View();
}

//
// POST: /Home/Create
[AcceptVerbs(HttpVerbs.Post)]
public ActionResult Create([Bind(Exclude = "ID")] Records
records)
{
    try
    {
        if (recordsDAO.AddRecord(records))
            return RedirectToAction("Index");
        else
            return View("Create");
    }
    catch
    {
        return View("Create");
    }
}
```

Теперь добавим представление. Для этого нажмем правой кнопкой мыши на любом из методов Create() и выберем **Add View...** В диалоге **Add View** нажимаем **Create a strongly-typed view**. Выберем класс для представления **mvcApplication.Models.Records** и **View content** типа **Create**. После нажатия кнопки Add получим файл следующего содержания:

Листинг 9 – Views\Home\Create.cshtml

```
<%@
    Page
    Title=""
    Language="C#"
MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage<mvcApplication.Models.Records>" %>
    <asp:Content ID="Content1" ContentPlaceHolderID="TitleContent"
runat="server">
        Create
    </asp:Content>
```

```

        <asp:Content ID="Content2" ContentPlaceHolderID="MainContent"
runat="server">
        <h2>Create</h2>
        <%= Html.ValidationSummary("Create was unsuccessful. Please
correct the errors and try again.") %>
        <% using (Html.BeginForm()) {%>
        <fieldset>
        <legend>Fields</legend>
        <p>
        <label for="Title">Title: </label>
        <%= Html.TextBox("Title") %>
        <%= Html.ValidationMessage("Title", "*") %>
        </p>
        <p>
        <label for="Author">Author: </label>
        <%= Html.TextBox("Author") %>
        <%= Html.ValidationMessage("Author", "*") %>
        </p>
        <p>
        <label
for="PublicationDate">PublicationDate: </label>
        <%= Html.TextBox("PublicationDate") %>
        <%=
        Html.ValidationMessage("PublicationDate",
        "*" ) %>
        </p>
        <p>
        <input type="submit" value="Create" />
        </p>
        </fieldset>
        <% } %>
        <div>
        <%=Html.ActionLink("Back to List", "Index") %>
        </div>
</asp:Content>

```

ВАЖНО! В Create и Edit представлениях нужно обязательно удалять следующие строки (если они есть), т.к. если произойдет попытка вставки в поле с типом identity (автоинкремент), то произойдет ошибка (Листинг 10):

Листинг 10 – Удаление лишних полей из представления Views\Home\Create.cshtml

```

<p>
    <label for="ID">ID: </label>
    <%= Html.TextBox("ID") %>
    <%= Html.ValidationMessage("ID", "*") %>
</p>

```

Теперь нужно запустить проект. В результате получим следующую страницу, через которую можно добавлять записи в таблицу GuestBook (рис. 16):

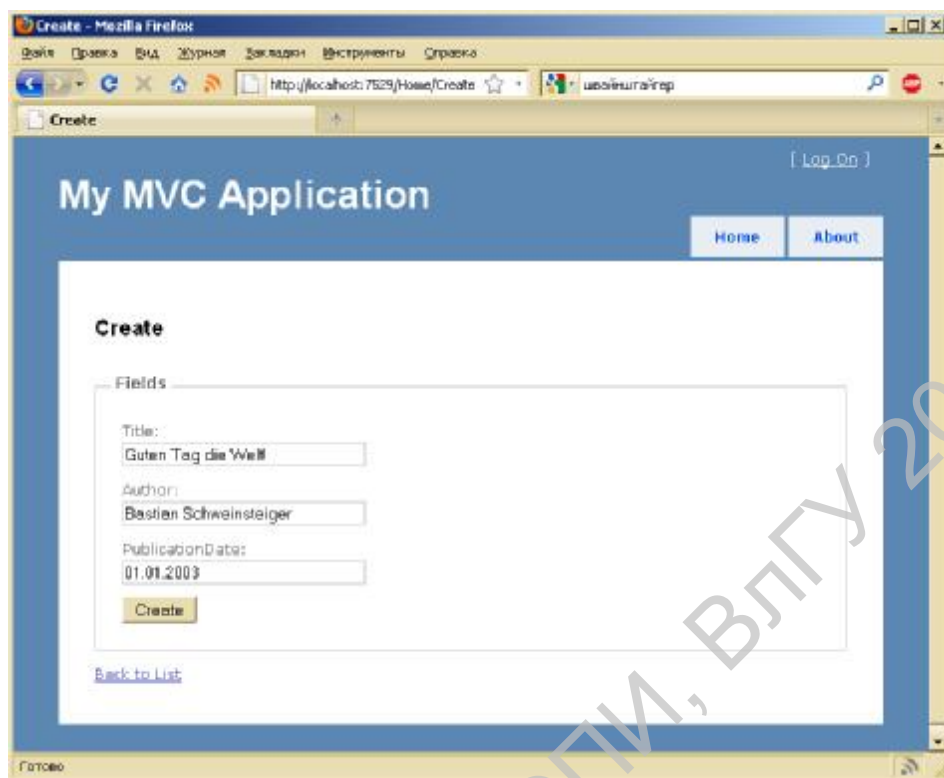


Рисунок 16. Create-представление

Задание для самостоятельного выполнения

В программе, соответствующей вашему индивидуальному заданию, добавить следующие возможности:

1. страница детального просмотра элемента (Details);
2. страница редактирования элемента (Edit);
3. страница удаления элемента (Delete).

Подводя итоги

В созданной программе мы установили проект на основе шаблона MVC, разобрали технологию доступа к данным DAO и создали небольшое приложение, позволяющее работать с записи в БД через веб-интерфейс. В данную лабораторную работу не вошли такие темы как валидация данных, работа с навигацией (построение меню), шаблонами сайта и работа с авторизацией на основе Membership.

Вопросы для самопроверки

1. Что такое MVC? Какова его архитектура?
2. Как работает шаблон MVC?
3. В чем ценность шаблона MVC?
4. Что такое DAO? Каково его предназначение?

5. Что такое суррогатный ключ? Из чего он строится? Какого его предназначение?

Вершинин В.В. Каф. ИСПИ, ВлГУ 2017

Лабораторная работа №6

РЕАЛИЗАЦИЯ АУТЕНТИФИКАЦИИ В MVC ASP.NET

Цель работы

Познакомиться с реализацией аутентификации MVC на платформе ASP.NET.

Общие сведения

В платформе ASP.NET MVC существует несколько видов аутентификаций:

- **Windows Authentication** (Аутентификация Windows) – одним из примеров являются пользователи, добавленные в дерево AD. Все проверки делаются непосредственно AD, с помощью IIS, через специальный провайдер. Данная аутентификация часто применяется для корпоративных приложений;
- **Password Authentication** (аутентификация через Password) – централизованная служба аутентификации, предлагаемая Microsoft;
- **Form Authentication** (Аутентификация с помощью форм) – данный вид аутентификации подходит для приложений, доступных через Интернет. Она работает через клиентскую переадресацию, на указанную html страницу, с формой авторизации. На форме клиент вводит свои учетные данные и отправляет на сервер, где они обрабатываются специфичной для данного приложения логикой. Именно такой вид аутентификации мы и будем писать.

В аутентификации через форму на стороне клиента хранится зашифрованный cookie-набор. Cookie передается в запросах к серверу, показывая, что пользователь авторизован. Для создания такого зашифрованного набора, в стандартной аутентификации через форму, служит метод `Encrypt` в классе `System.Web.Security.FormAuthentication`. Для декодирования используется метод `Decrypt`. Чем хорош класс `FormAuthentication`? Тем, что его методы кодирования и декодирования шифруют и подписывают данные с помощью машинных ключей сервера. Без данных ключей информацию из cookie файла невозможно прочесть или изменить, а нам не нужно изобретать велосипед.

Ход работы

Файлы для аутентификации.

Для того, чтобы реализовать аутентификацию, нужно добавить новую модель, контроллер и естественно представления для этого. Чтобы немного облегчить себе задачу можно использовать уже готовые файлы из проекта, который можно создать в Visual Studio. Для этого сначала создадим новый проект ASP.NET MVC 4 Web Application (рис. 17).

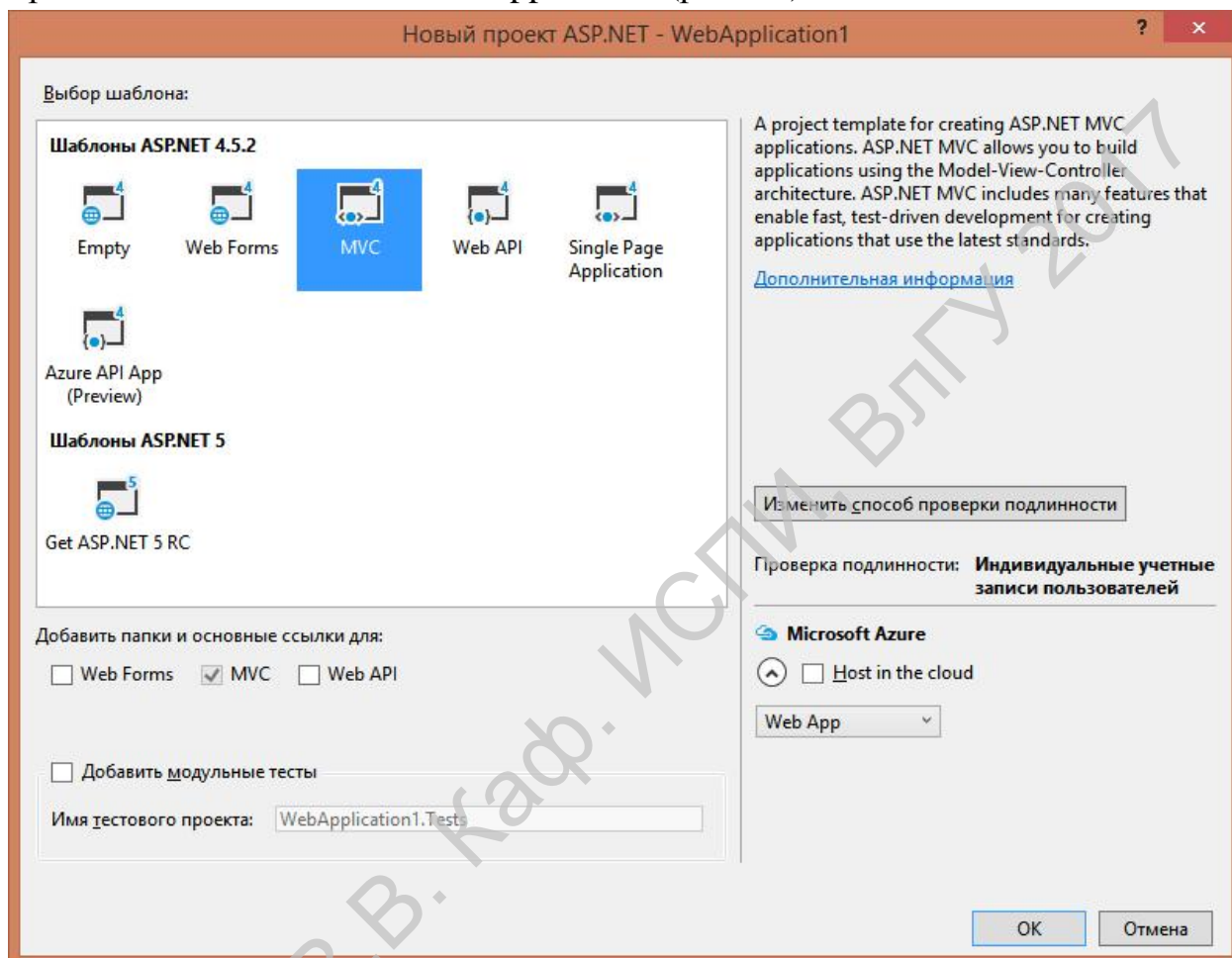


Рисунок 17. Создание шаблонного проекта MVC.

Справа вы видите кнопку «Изменить способ проверки подлинности». Нажмите на нее и выберите «Учетные записи отдельных пользователей» и нажмите ОК (рис. 18).

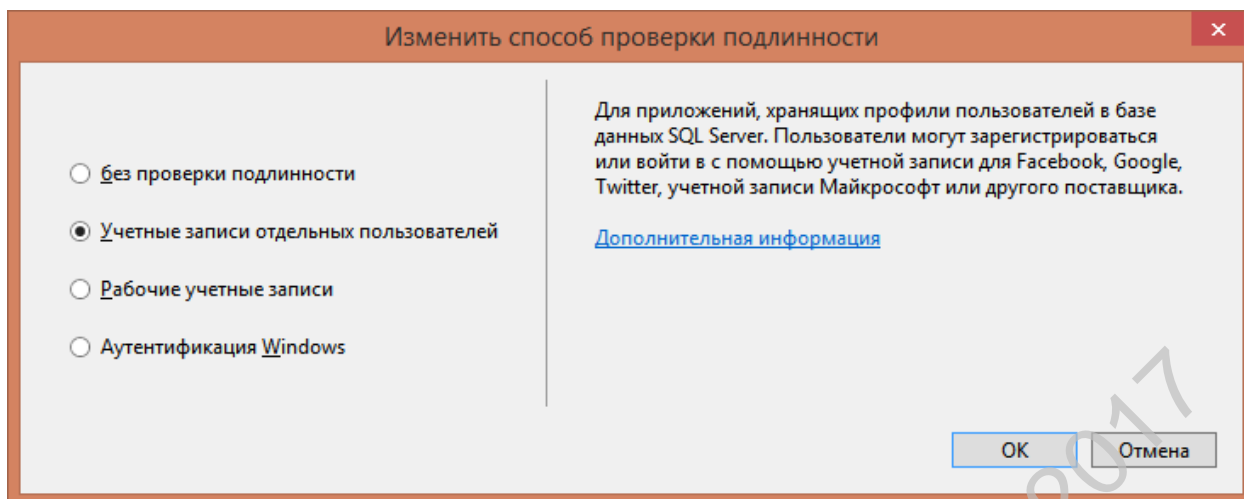


Рисунок 18. Изменение типа аутентификации.

Создание пользователя и роли.

Теперь создадим пользователя и роль. Запустим наше веб-приложение в браузере и перейдем на страницу регистрации (рис. 19). Здесь мы можем создавать пользователей. Создайте как минимум двух пользователей.

Рисунок 19. Создание пользователя.

Как только вы создадите пользователя, в папке проекта App_Data появится база данных (БД конфигурации), которая содержит роли и данные о пользователях. Если папка кажется пустой, нажмите кнопку Проект → Показать все файлы. Открыв таблицуAspNetUser, вы увидите данные о зарегистрированных пользователях (рис. 20).

	Id	Email	EmailConfirmed	PasswordHash	SecurityStamp	PhoneNumber	PhoneNumber...	TwoFactorEna...
▶	c84-1317c6213c84	admin@mail.ru	False	AlgItXI/m0oQ6...	9b3da8b7-1473...	NULL	False	False
	e83568e4-3db6...	ghost@yandex.ru	False	ABLcdha7axOa...	61fd285-0510-...	NULL	False	False
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Рисунок 20. Фрагмент таблицы пользователей.

Создадим новую роль. Для этого перейдем к таблице ролей – AspNetRoles. Создадим две роли – администратор и пользователь (рис. 21).

	Id	Name
	1	Admin
	2	Visitor
▶*	NULL	NULL

Рисунок 21. Фрагмент таблицы ролей.

Теперь нам нужно определить роли для пользователей. Перейдем в таблицу AspNetUserRoles и пропишем роли пользователям. Для тех, кто знаком со структурой баз данных, нетрудно будет догадаться, что в первый столбец следует внести ID существующего пользователя, а во второй – ID существующей роли. В случае ошибки Visual Studio непременно сообщит вам об этом.

	UserId	RoleId
	ccd1fa27-4528-4d51-bc84-1317c6213c84	1
	e83568e4-3db6-4361-80f2-ff1b49693bd9	2
▶*	NULL	NULL

Рисунок 22. Фрагмент назначения роли пользователю.

Проверка доступа

Для того, чтобы разрешить доступ к некоторым функциям сайта только определенным ролям, перед каждым методом в соответствующем контроллере следует прописать:

```
[Authorize(Roles = "Role1, Role2 ...")]
```

В кавычках пишутся роли, которым открыт доступ к этому методу. Вы также можете указать непосредственно пользователей, которые могут пользоваться данным методом:

```
[Authorize(Users = "ghost@yandex.ru")]
```

Задание для самостоятельного выполнения

В созданном в предыдущей лабораторной работе проекте MVC, реализовать аутентификацию:

1. Создать как минимум две роли
2. Создать пользователя для каждой роли
3. Организовать ролям и пользователям доступ к определённым действиям на сайте.

Вопросы для самопроверки

1. Что такое аутентификация?
2. Что такое авторизация?
3. В чём отличия аутентификации от авторизации?
4. Значение аутентификации в приложениях MVC?
5. Значение авторизации в приложениях MVC?

Содержание отчёта

1. Цель работы
2. Добавление пользователей
3. Добавление ролей и назначение их
4. Скриншоты, иллюстрирующие привилегии каждой роли
5. Выводы к работе

Лабораторная работа №7

ЯЗЫК ИНТЕГРИРОВАННЫХ ЗАПРОСОВ LINQ TO ENTITIES

Цель работы

Познакомиться с технологией использования языка интегрированных запросов LINQ to Entities.

Общие сведения

Язык интегрированных запросов LINQ

Language Integrated Query (LINQ) — язык интегрированных запросов, добавленный компанией Microsoft в платформу .NET, напоминающий SQL. Данный язык является унифицированным единым языком для работы с различными источниками данных. Это могут быть базы данных, XML-хранилища, файловые системы, массивы и т.д. Особенностью данного языка, в отличие от SQL, является работа не с таблицами базы данных, а объектами программы.

LINQ выпущен вместе с Visual Studio 2008 в конце ноября 2007 года.

Польза данного языка в том, что он не привязан к какому-либо конкретному источнику данных и реализует работу благодаря внутреннему языку запросов. Это позволяет, поняв язык один раз, использовать его в работе с любыми видами структурированных данных.

Ниже приведен пример выборки данных из массива для консольного приложения:

Листинг 1 – Hello LINQ

```
using System;
using System.Linq;

namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            string[] greetings = { "Hello world!", "hello LINQ", "hello
Entity Framework" };
            var items =
                from s in greetings
                where s.EndsWith("LINQ")
                select s;
            foreach (var item in items)
                Console.WriteLine(item);
            Console.ReadLine();
        }
    }
}
```

Результатом выполнения данной программы будет строка «**Hello LINQ**».

ADO.NET Entity Framework

ADO.NET Entity Framework (EF) — объектно-ориентированная технология доступа к данным, является ORM-решением (Object Relational Mapping) для .NET Framework от Microsoft. Предоставляет возможность взаимодействия с объектами посредством LINQ в виде LINQ to Entities.

Релиз данной технологии состоялся в августе 2008 года в составе .NET Framework 3.5 Service Pack 1 и Visual Studio 2008 Service Pack 1.

LINQ to Entities

LINQ to Entities - это интерфейс LINQ, используемый для обращения к базе данных через язык интегрированных запросов. Он отделяет сущностную объектную модель данных от физической базы данных, вводя логическое отображение между ними.

Польза LINQ to Entities в том, что он дает возможность ослабить связь между сущностной объектной моделью данных и физической моделью базы данных, например, если сущностные классы строятся из нескольких таблиц или если необходима большая гибкость в моделировании сущностных объектов, то используется LINQ to Entities.

Литература

Джозеф С. Раттц-мл. LINQ: язык интегрированных запросов в C# 2008 для профессионалов = Pro LINQ: Language Integrated Query in C# 2008. — М.: «Вильямс», 2008. — С. 560. — ISBN 978-5-8459-1427-9

При необходимости строить сложные запросы рекомендую обратиться к части II данной книги: LINQ to Objects – здесь можно найти ответы на любые вопросы, касаемые нетривиальных запросов.

Ход работы

В предыдущей работе рассматривались вопросы построения модели данных и взаимодействия с данными в самом простом и понятном виде. В ходе данной работы будет построена сущностная модель данных и взаимодействие с ней.

Воспользуемся приложением, разработанным в ходе первой лабораторной работы. Отмечу, что данное решение выбрано не случайно. Я отказался от создания нового проекта (ведь мы строим проект на ASP.NET

MVC Framework), в силу того, что MVC нам позволяет очень гибко работать со структурой проекта. Одним из его достоинств является модификация частей системы в отдельности, не модифицируя другие. Иными словами мы в первом разделе работы заменим существующую модель данных на сущности, а также класс, реализующий работу с ними, при этом не затронув ни контроллер, ни представления.

Удаление старых файлов в проекте

В старом проекте необходимо удалить следующие файлы:

- 1) \DAO\DAO.cs
- 2) \Models\Contacts.cs

Разработка групп контактов

На данный момент в нашем приложении есть только таблица контактов. Но необходима функция их классификации по группам. Дальнейшая деятельность будет направлена на создание второй таблицы, связке её с первой таблицей и разработке программной части для работы с группами контактов.

Для этого введем в БД новую таблицу **Groups** следующей структуры:

Таблица 1 – Структура таблицы Groups

Column Name	Data Type	Allow Nulls
Id	int	false
Name	nvarchar(50)	false
Description	nvarchar(255)	false

Определим колонку **Id** первичным ключем таблицы (индексом), которая так же будет автоинкрементом (Identity Increment):

- а) Задание первичного ключа. Щелкаем правой кнопкой мыши на колонке Id и выбираем “**Set Primary Key**”. После этого появится иконка с изображением ключа.
- б) Задание автоинкремента. Необходимо выделить колонку Id и в окне Column Properties установить значение свойства (**Is Identity**) как **Yes**.

Следующий шаг – добавим 2-3 записи в созданную таблицу. Данный шаг строго обязателен для выполнения. Для этого в Server Explorer щелкаем правой кнопкой мыши на таблице **Groups**, выбираем **Show Table Data** и вносим любые 2-3 записи.

Установка связей между таблицами

Для обеспечения целостности данных, свяжем таблицы.

Для этого в таблицу **Contacts** введем поле **GroupId**, типа **int**, которое будет содержать в себе внешний ключ на первичный ключ таблицы **Contacts**.

Для этого щелкаем правой кнопкой мыши на поле **FirstName** и выбираем **Insert Column** (рисунок 1):

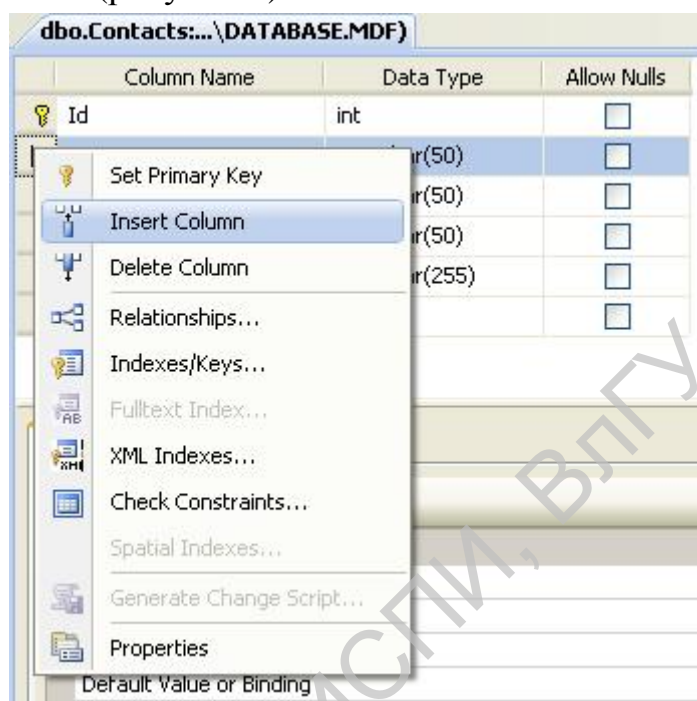


Рисунок 23. Вставка поля **GroupId** в таблицу **Contacts**

Таким образом, структура таблицы **Contacts** будет выглядеть следующим образом:

Таблица 2 – Структура таблицы **Contacts после модификации**

Column Name	Data Type	Allow Nulls
Id	int	false
GroupId	int	false
FirstName	nvarchar(50)	false
SecondName	nvarchar(50)	false
Phone	nvarchar(50)	false
Email	nvarchar(255)	false

Подсказка: чтобы сделать поле **Allow Nulls=false** необходимо сначала сделать его типа **true**, заполнить в таблице это поле данными (действительными **Id** из таблицы **Groups**) и только потом модифицировать его в тип **false**.

Теперь непосредственно установка связей между таблицами. Щелкаем правой кнопкой мыши на поле **GroupId** таблицы **Contacts** и выбираем пункт **Relationships...**:

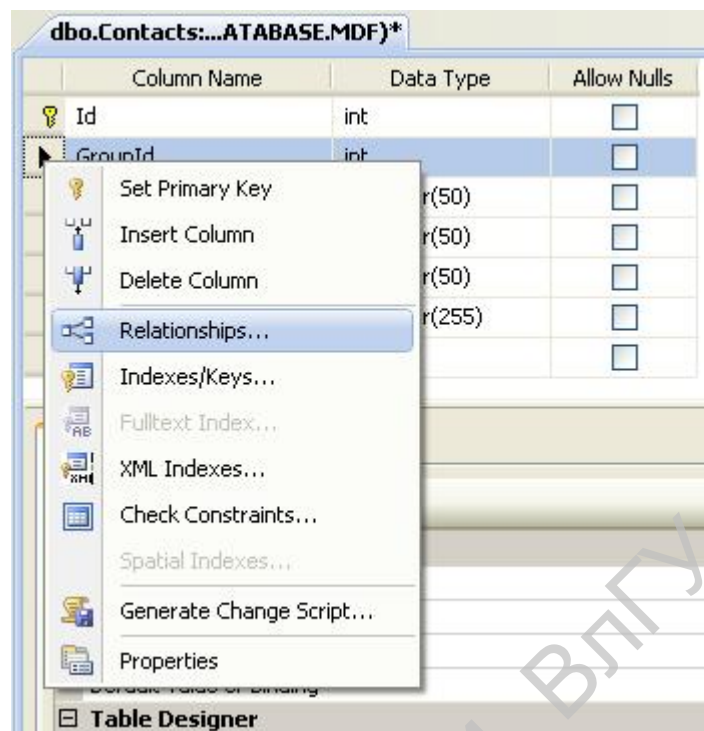


Рисунок 24. Установка связей

Далее добавим связь, нажав кнопку **Add**. На данный момент мы внесли связь, но не привязали её ни к чему. Поэтому нажимаем на пункт **Tables And Columns Specific** и на кнопку возле этого пункта (рисунок 3).

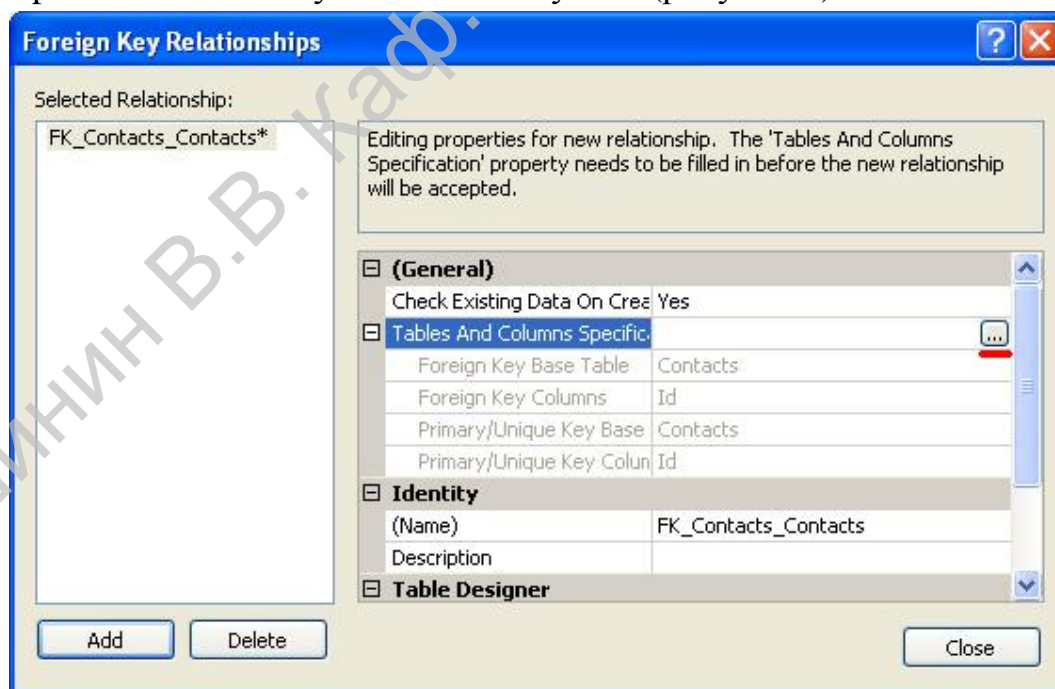


Рисунок 25. Диалог создания связей между таблицами

Установим связь, как это показано на рисунке 4:

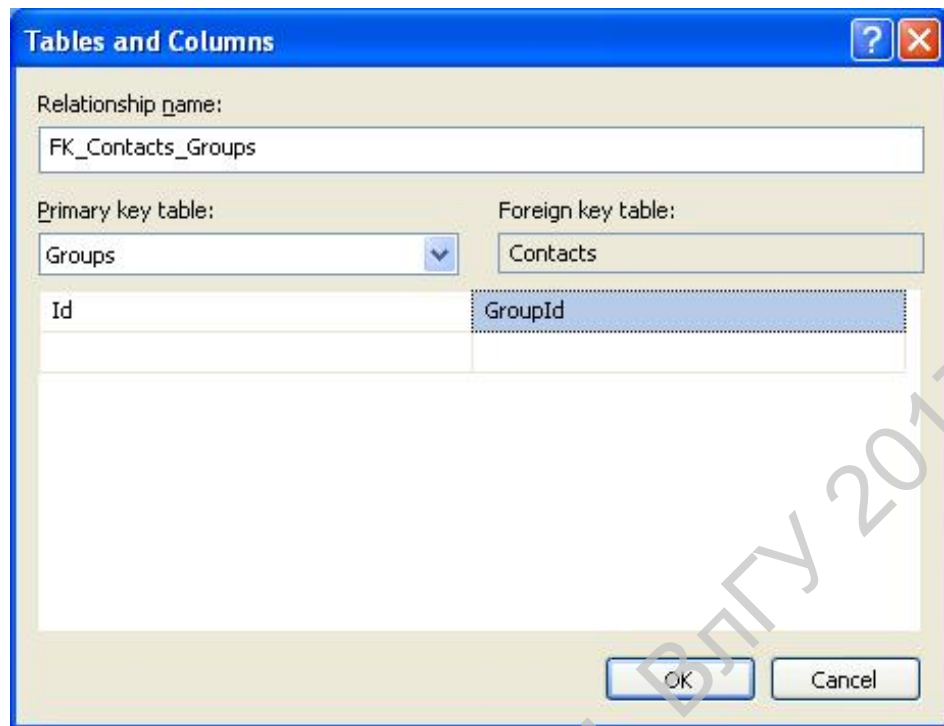


Рисунок 26. Установка связи между таблицами **Groups** и **Contacts**

В разделе Table Designer в правилах **INSERT and UPDATE Specification** укажите свойство **Cascade** для **Delete Rule**.

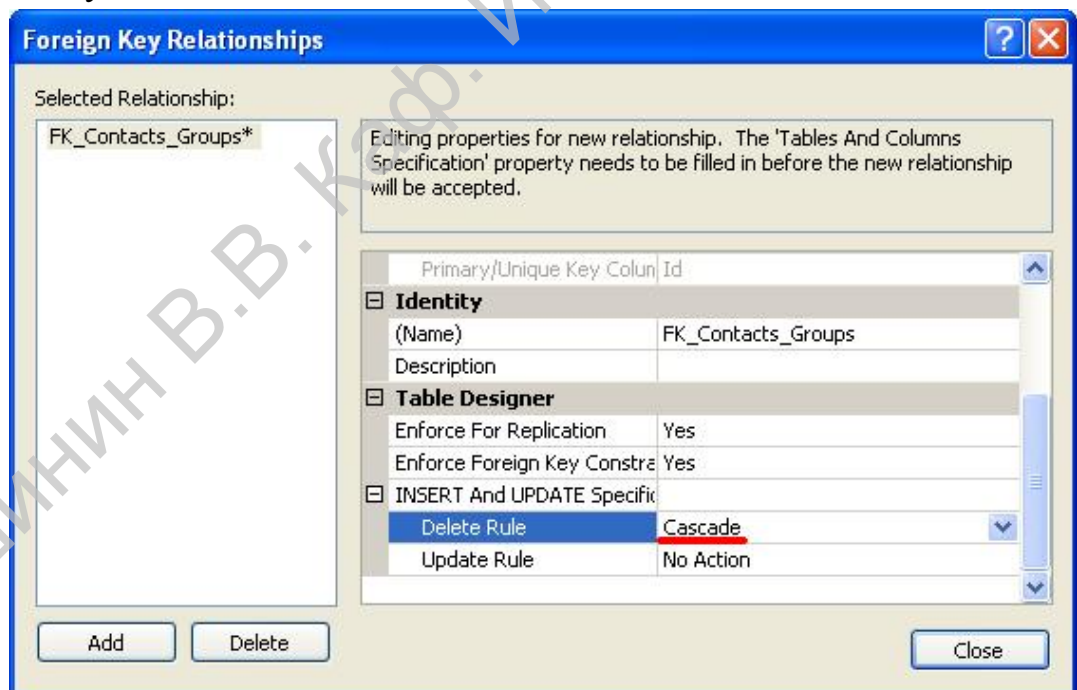


Рисунок 27. Установка правил удаления записей. Сохраняем изменения в таблицах.

Теперь нужно убедиться, что SQL Server принял структуру базы данных. Для этого щелкаем правой кнопкой мыши на пункте **Database Diagrams** и выбираем **Add New Diagram** (рисунок 6):

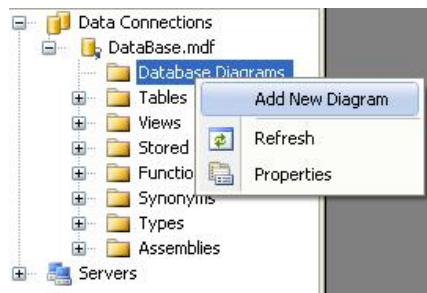


Рисунок 28. Построение диаграммы базы данных.

Указываем таблицы для построения диаграммы (рисунок 7):

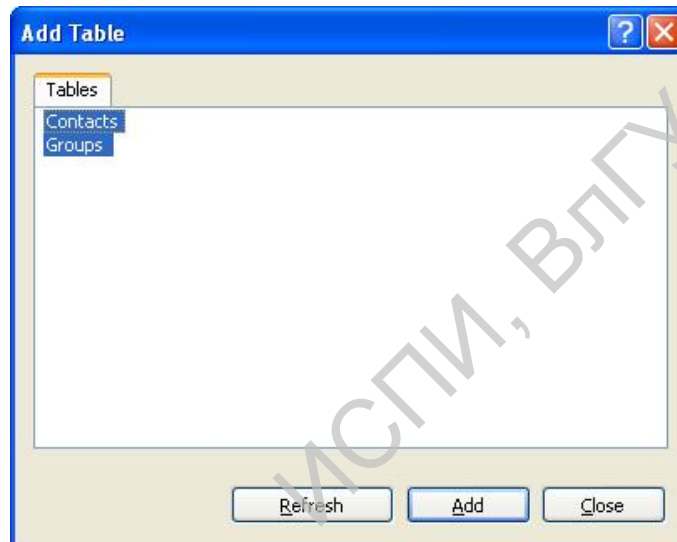


Рисунок 29. Выбор таблиц для внесения в диаграмму

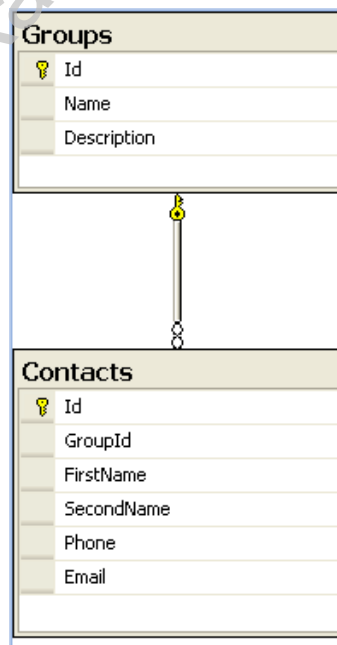


Рисунок 30. Результат построения диаграммы

Если диаграмма вашей базы данных совпадает с диаграммой, приведенной на рисунке N, то можно переходить к следующему шару работы.

Создание сущностной модели данных

Создание сущностной модели данных в ADO.NET Entity Framework на сегодняшний день возможно двумя путями:

- 1) генерация сущностей по базе данных. При этом структура существующих таблиц переходит в структуру классов сущностей.
- 2) самостоятельная постройка структуры классов сущностей и ручная установка соответствия между таблицами и классами сущностей.

Первый подход используется при построении простой модели данных, повторяющей структуру базы данных. Второй подход позволяет создать собственную модель данных, структура которой будет отличаться от структуры БД, например для построения класса сущностей из нескольких таблиц. Генерация структуры базы данных по структуре объектов на данный момент не поддерживается.

Для создания сущностной объектной модели данных щелкаем правой кнопкой мыши на папке **Models** и выбираем команду **Add/New Item...**

В окне выбираем **Visual C#/Data/ADO.NET Entity Data Model** (напоминаю: для этого должна быть установлена последняя версия .NET Framework и Visual Studio (или Web Developer))

Назовем **Model.edmx** и нажимаем **Add**:

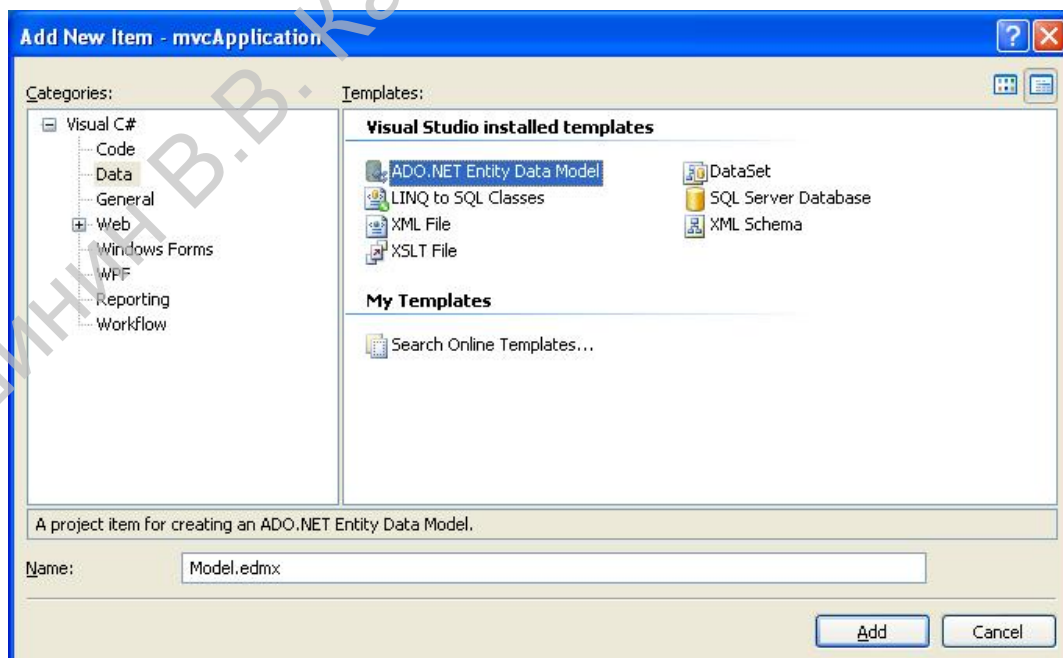


Рисунок 31. Создание Entity Data Model

В следующем диалоге программа предложит выбрать способ генерации сущностей (рисунок 32):

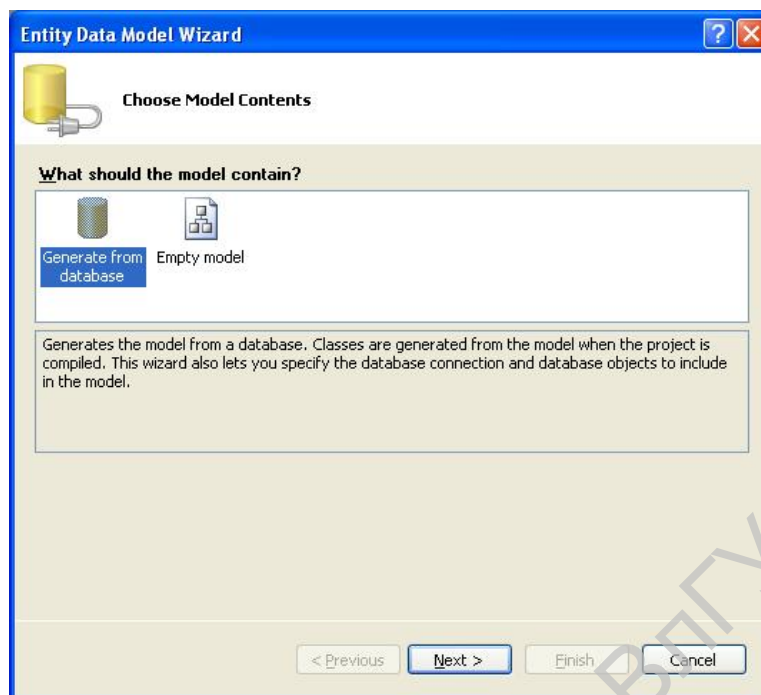


Рисунок 32. Диалог выбора способа генерации сущностей

Выберем в качестве источника данных БД **DataBase.mdf** и сохраним название класса сущностей как **DataBaseEntities** (рисунок 33):

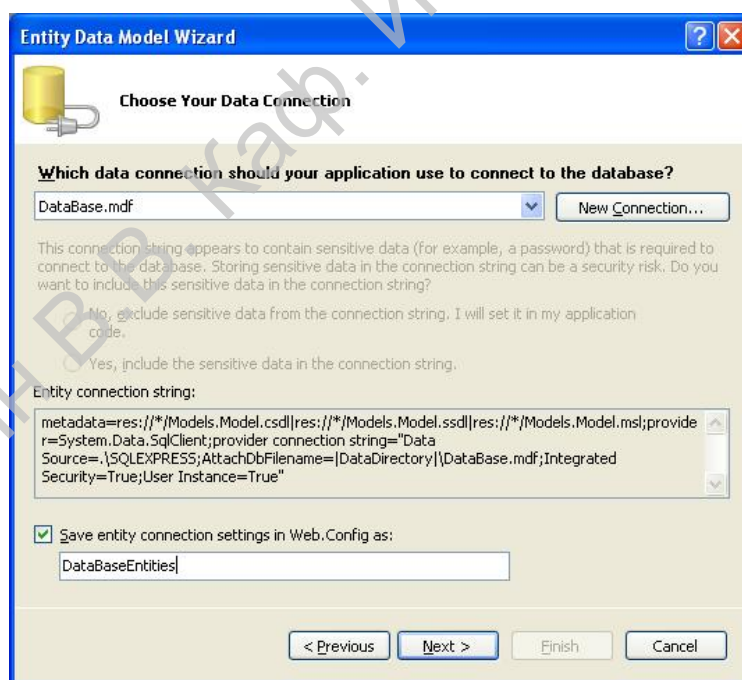


Рисунок 33. Выбор подключения к БД

Выберем таблицы для импорта в класс сущностей. Раскроем список Tables, и выберем только созданные таблицы, т.к. база может содержать еще и служебные таблицы (рисунок 34):

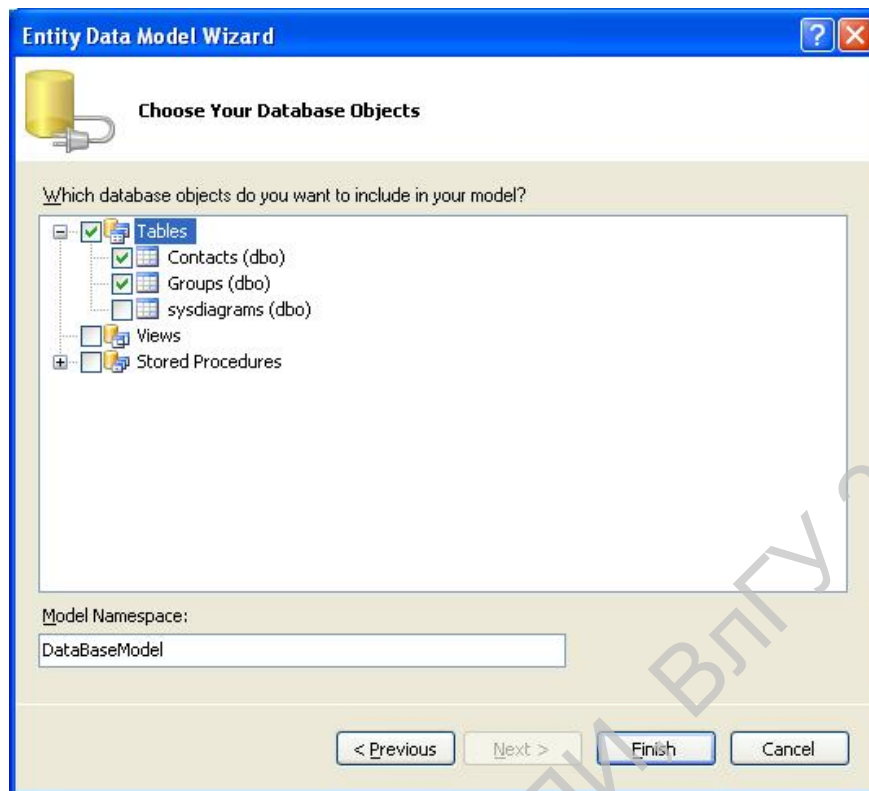


Рисунок 34. Выбор таблиц для импорта в класс сущностей

Результат объектно-реляционного преобразования: структура таблицы Contacts перешла в структуру класс сущностей Contacts (рис. 35):

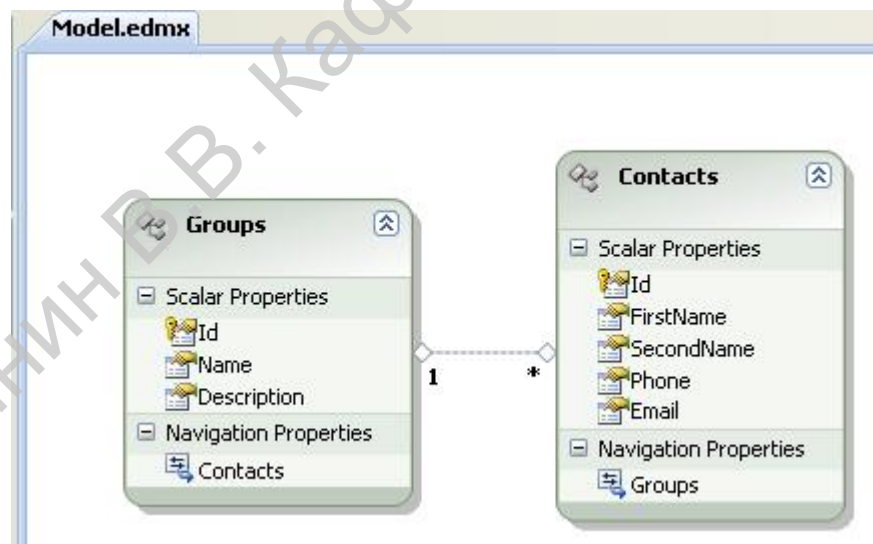


Рисунок 35. Результат генерации классов сущностей

Сохраним модель.

Модификация объекта доступа к данным и контроллера

Для успешной работы с сущностями, модифицируем файл DAO\ContactsDAO.cs следующим образом (Листинг 2):

Листинг 2 – DAO\ContactsDAO.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using MvcApplication.Models;

namespace MvcApplication.DAO
{
    public class ContactsDAO
    {
        // создаем экземпляр класса сущностей
        private DataBaseEntities _entities = new DataBaseEntities();

        public IEnumerable<Contacts> getAllContacts()
        {
            // простая выборка через класс сущностей
            return (from c in _entities.Contacts
                    select c);
        }

        public Groups GetContactGroup(int? id)
        {
            if (id != null) //возвращает запись по её Id
                return (from c in _entities.Groups
                        where c.Id == id
                        select c).FirstOrDefault();
            else // возвращает первую запись в таблице
                return (from c in _entities.Groups
                        select c).FirstOrDefault();
        }

        public Contacts getContact(int id)
        {
            return (from c in _entities.Contacts.Include("Groups")
                    where c.Id == id
                    select c).FirstOrDefault();
        }

        public bool addContact(int GroupId, Contacts contact)
        {
            try
            {
                // Associate Contacts with Groups
                contact.Groups = GetContactGroup(GroupId);

                // добавление записи в таблицу Contacts
                _entities.AddToContacts(contact);
                _entities.SaveChanges();
            }
            catch
            {
                return false;
            }
            return true;
        }

        public bool updateContact(int GroupId, Contacts contact)
        {

```

```

        Contacts originalContact = getContact(contact.Id);

        originalContact.Groups = GetContactGroup(Group.Id);

        try
        {
            // редактирование записи в таблице

_entities.ApplyPropertyChanges(originalContact.EntityKey.EntitySetName,
contact);

        _entities.SaveChanges();
    }
    catch
    {
        return false;
    }
    return true;
}

public bool deleteContact(Contacts contact)
{
    Contacts originalContact = getContact(contact.Id);

    try
    {
        // удаляем запись из таблицы
        _entities.DeleteObject(originalContact);
        _entities.SaveChanges();
    }
    catch
    {
        return false;
    }
    return true;
}
}
}
}

```

Добавим класс GroupsDAO (Листинг 3):

Листинг 3 - \DAO\GroupsDAO.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Mvc.Ajax;
using MvcApplication.Models;
using MvcApplication.DAO;

namespace MvcApplication.DAO
{
    public class GroupsDAO
    {
        private DataBaseEntities _entities = new DataBaseEntities();

        public IEnumerable<Groups> getAllGroups()
        {
            return (from c in _entities.Groups
                    select c);
        }
    }
}

```


Модифицируем HomeController следующим образом:

Листинг 4 - \Controllers\ HomeController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Mvc.Ajax;
using MvcApplication.Models;
using MvcApplication.DAO;

namespace MvcApplication.Controllers
{
    public class HomeController : Controller
    {
        ContactsDAO contactsDAO = new ContactsDAO();
        GroupsDAO groupsDAO = new GroupsDAO();

        //
        // GET: /Home/
        public ActionResult Index(int? id)
        {
            ViewData["groups"] = groupsDAO.getAllGroups();
            return View(contactsDAO.getAllContacts());
        }

        //
        // GET: /Home/Details/5
        public ActionResult Details(int id)
        {
            return View(contactsDAO.getContact(id));
        }

        protected bool ViewDataSelectList(int GroupId)
        {
            var groups = groupsDAO.getAllGroups();
            ViewData["GroupId"] = new SelectList(groups, "Id", "Name",
            GroupId);

            return groups.Count() > 0;
        }

        //
        // GET: /Home/Create
        public ActionResult Create()
        {
            if (!ViewDataSelectList(-1))
                return RedirectToAction("Index");
            return View("Create");
        }

        //
        // POST: /Home/Create
        [AcceptVerbs(HttpVerbs.Post)]
        public ActionResult Create(int GroupId, [Bind(Exclude = "Id")]
        Contacts contact)
        {
            if (contactsDAO.addContact(GroupId, contact))
                return RedirectToAction("Index");
            else
            {
                ViewDataSelectList(GroupId);
            }
        }
    }
}
```

```

        return View("Create");
    }
}

//
// GET: /Home/Edit/5
public ActionResult Edit(int id)
{
    Contacts contact = contactsDAO.getContact(id);
    if (!ViewDataSelectList(contact.Groups.Id))
        return RedirectToAction("Index");
    return View(contactsDAO.getContact(id));
}

//
// POST: /Home/Edit/5
[AcceptVerbs(HttpVerbs.Post)]
public ActionResult Edit(int GroupId, int id, Contacts contact)
{
    if (contactsDAO.updateContact(GroupId, contact))
        return RedirectToAction("Index");
    else
    {
        ViewDataSelectList(-1);
        return View("Edit", contactsDAO.getContact(id));
    }
}

//
// GET: /Home/Delete
public ActionResult Delete(int id)
{
    return View("Delete", contactsDAO.getContact(id));
}

//
// POST: /Home/Delete
[AcceptVerbs(HttpVerbs.Post)]
public ActionResult Delete(int id, Contacts contact)
{
    if (contactsDAO.deleteContact(contact))
        return RedirectToAction("Index");
    else
        return View("Delete", contactsDAO.getContact(id));
}
}
}

```

ViewData – специальный объект в ASP.NET MVC Framework, который призван передавать данные от контроллера в представление. В нашем случае в данном объекте будут передавать данные о группах. Данные о контактах, как и прежде, будут передаваться непосредственно по структуре модели.

Сборка проекта

В Visual Studio выполняем команду из меню **Build / Build Solution**. Данное действие обязательно, т.к. далее пойдет генерация представлений, и они должны «знать» структуру модели.

Вывод групп контактов на главной странице

Теперь добавим в представление **Index.aspx** следующий код, сразу после заголовка `<h2>Index</h2>`, который будет выводить группы контактов:

Листинг 5 – Вывод групп контактов на странице **Index.aspx**

```
<ul id="groups">
    <% foreach (var item in
(IEnumerable<mvcApplication.Models.Groups>) ViewData[ "groups" ])
    {
        <li>
            <div><%= Html.ActionLink(item.Name, "Index", new { id =
item.Id }) %></div>
        </li>
    }
</ul>
```

Добавление выпадающего списка на страницу создания и редактирования

Мы создали группы контактов. Логично выводить их не только на главной странице, но и на страницах создания и редактирования контакта.

Добавим следующий код в файлы **Create.aspx** и **Edit.aspx** (листинг 6):

Листинг 6 – Вывод выпадающего списка групп контактов

```
<p>
<label for="GroupId">GroupId:</label>
<%= Html.DropDownList( "GroupId" ) %>
</p>
```

На странице **Details.aspx** вывод группы контактов добавьте самостоятельно.

Запуск проекта

Запустим проект, нажав **Debug/Start Debugging** или клавишу **F5**.

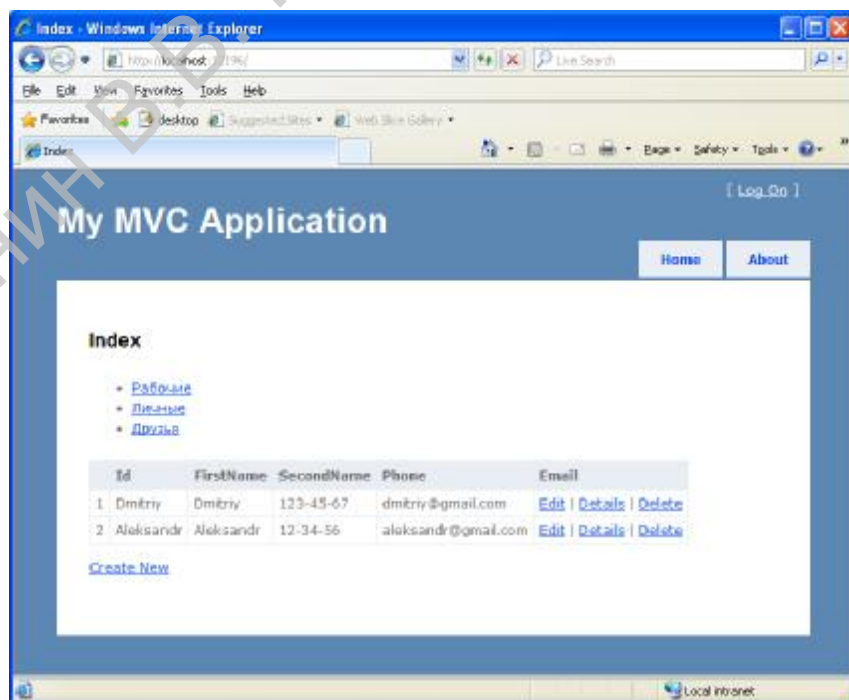


Рисунок 36. Результат выполнения программы

Задание для самостоятельного выполнения

В созданной программе добавить следующие возможности:

- 1) при выборе определенной группы должен производиться показ контактов в выбранной группе;
- 2) на странице Details.aspx добавить вывод группы, к которой принадлежит контакт;
- 3) создать раздел в веб-приложении, который позволит управлять группами контактов. Работу с данными организовать через LINQ to Entities. Никаких прямых SQL-запросов в DAO быть не должно.

Подводя итоги

В данной работе мы рассмотрели вопросы построения и работы с сущностной модели данных с помощью языка интегрированных запросов.

Вопросы для самопроверки

1. Что такое LINQ? В чем его ценность?
2. Приведите примеры источников данных, с которыми может работать LINQ.
3. Что такое ADO.NET Entity Framework?
4. Что такое LINQ to Entities? Как относятся между собой LINQ и LINQ to Entities?
5. Что такое ORM? Раскройте его предназначение.
6. Что такое класс сущностей? Чем он отличается от обычного класса?
7. Перечислите способы генерации классов сущностей.
8. Перечислите виды отношений между таблицами. Как они реализуются в плане структуры таблиц?
9. Как соотносится между собой структура таблиц базы данных и сущностей? Где она устанавливается? Как можно её модифицировать?

Лабораторная работа №3

ПОСТРОЕНИЕ ИНТЕРАКТИВНЫХ ПОЛЬЗОВАТЕЛЬСКИХ ИНТЕРФЕЙСОВ С ИСПОЛЬЗОВАНИЕМ AJAX

Цель работы

Познакомиться с технологией интерактивных пользовательских интерфейсов веб-приложений.

Общие сведения

AJAX (Asynchronous Javascript and XML — «асинхронный JavaScript и XML») — это подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером. В результате при обновлении данных веб-страница не перезагружается полностью, и веб-приложения становятся более быстрыми и удобными.

AJAX — это не самостоятельная технология, а концепция использования нескольких смежных технологий.

Основная идея AJAX состоит в построении динамических представлений с использованием специальных библиотек, написанных на JavaScript. Например, есть какое-либо приложение с формой голосования на главной странице. В случае выбора варианта ответа на форме и нажатия кнопки «Голосовать», при обычной модели взаимодействия клиент-сервера, происходит полная загрузка страницы той же страницы, но вместо формы – результат голосования. Однако если сделать форму голосования с помощью AJAX, то при нажатии кнопки погрузится только нужная область без обновления всей страницы. Это важно как для клиента (скорость работы приложения) так и для сервера (уменьшение нагрузки на веб-сервер, на сервер баз данных (меньше запросов)).

Польза AJAX.

- 1) Экономия трафика. Использование AJAX позволяет значительно сократить трафик при работе с веб-приложением благодаря тому, что часто вместо загрузки всей страницы достаточно загрузить только изменившуюся часть, иногда довольно небольшую.
- 2) Уменьшение нагрузки на сервер. AJAX позволяет несколько снизить нагрузку на сервер.

- 3) Ускорение реакции интерфейса. Поскольку нужно загрузить только изменившуюся часть, то пользователь видит результат своих действий быстрее.

3. Ход работы

Описание SiteMaps и построение навигации на его основе.

Создадим меню в веб-приложении на основе стандартного средства SiteMaps. Особенность данного средства в том, что карта сайта задается в отдельном специализированном XML-файле, из которого берется информация о разделах сайта для построения меню. Это позволяет не редактируя шаблоны сайта (Site.master) редактировать карту сайта через один файл. Синтаксисом данного файла является простое XML-описание, что не затруднит редактирование файла.

Создадим карту сайта. Для этого щелкаем правой кнопкой мыши на проекте mvсApplication, выбираем **Add/New Item...**. В диалоге выбираем **Visual C#/Web/Site Map** (рис. 37):

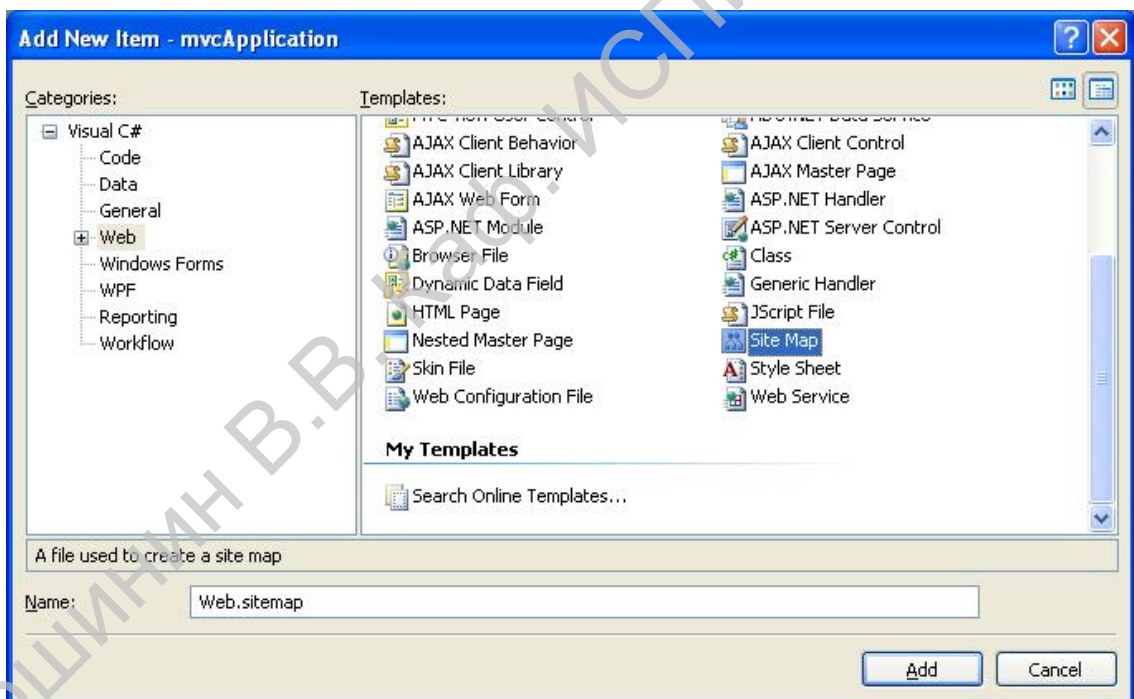


Рисунок 37. Добавление нового XML SiteMap-файла

В результате в корне проекта появится XML-файл Web.sitemap. Карта сайта содержит один корневой элемент <siteMap>, в котором находятся один или более элементов <siteMapNode>. Вы можете использовать элементы <siteMapNode> для описания связей между страницами в ASP.NET MVC application.

Каждый <siteMapNode> элемент может содержать следующие атрибуты (неполный список):

- 1) url – URL-адрес страницы. Используется при формировании URL в ссылке.
- 2) title – Title страницы. Используется при формировании названия ссылки.
- 3) description – Описание страницы. Задаёт описание страницы.
- 4) resourceKey – Ресурс, который используется при локализации siteMapNode для различных языков.
- 5) siteMapFile – Путь к другому SiteMap -файлу. Используется при разделении карты сайта на несколько частей.

Листинг 1 - Web.sitemap

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
  <siteMapNode url="" title="" description="">
    <siteMapNode url="/" title="Home" description="Contact page"
  />
    <siteMapNode url="/Groups/Index" title="Groups"
description="Groups page" />
  </siteMapNode>
</siteMap>
```

Создание Menu HTML Helper

ASP.NET MVC Framework позволяет создавать пользовательские HtmlHelper -ы. То есть например конструкция вида

```
<%= Html.ActionLink(item.Name, "Index", new { id = item.Id }) %>
```

является встроенным HtmlHelper-ом для вывода гиперссылки.

Создадим собственный Helper, который будет выводить все элементы первого порядка из Web.sitemap.

Для этого создадим в корне проекта каталог Helpers и в нем файл MenuHelper.cs (листинг 2):

Листинг 2 - \Helpers\MenuHelper.cs

```
using System.Text;
using System.Web;
using System.Web.Mvc;

namespace MvcApplication.Helpers
{
    public static class MenuHelper
    {
        public static string Menu(this HtmlHelper helper)
        {
            var sb = new StringBuilder();

            sb.Append("<ul id='menu'>");

            var topLevelNodes = SiteMap.RootNode.ChildNodes;
            foreach (SiteMapNode node in topLevelNodes)
```

```

        {
            if (SiteMap.CurrentNode == node)
                sb.AppendLine("<li class='selectedMenuItem'>");
            else
                sb.AppendLine("<li>");

            sb.AppendFormat("<a href='{0}' title='{1}'>{2}</a>",
node.Url, helper.Encode(node.Description), helper.Encode(node.Title));
            sb.AppendLine("</li>");
        }

        sb.Append("</ul>");

        return sb.ToString();
    }
}

```

На листинге 3 показано как используется метод `Helper -a Menu()` в `Site.master`

Листинг 3 – Site.master

```

<%@ Master Language="C#" Inherits="System.Web.Mvc.ViewMasterPage" %>
<%@ Import Namespace="mvcApplication.Helpers" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title><asp:ContentPlaceholder ID="TitleContent" runat="server"
/></title>
    <link href="../../../Content/Site.css" rel="stylesheet"
type="text/css" />
</head>
<body>
    <div class="page">
        <div id="header">
            <div id="title">
                <h1>My MVC Application</h1>
            </div>
            <div id="logindisplay">
                <% Html.RenderPartial("LogOnUserControl"); %>
            </div>
            <div id="menucontainer">
                <%= Html.Menu() %>
            </div>
        </div>
        <div id="main">
            <asp:ContentPlaceholder ID="MainContent" runat="server" />
            <div id="footer">
            </div>
        </div>
    </div>
</body>
</html>

```


Запустите проект, посмотрите результат.

Валидация данных web-форм.

Валидация – проверка данных на корректность, на соответствие определенным требованиям. В нашем случае необходимо проводить валидацию данных веб-форм перед их внесением в базу данных.

Замените код контроллера HomeController на код из листинга 4:

Листинг 4 – HomeController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Mvc.Ajax;
using System.Text.RegularExpressions;
using MvcApplication.Models;
using MvcApplication.DAO;

namespace MvcApplication.Controllers
{
    public class HomeController : Controller
    {
        ContactsDAO contactsDAO = new ContactsDAO();
        GroupsDAO groupsDAO = new GroupsDAO();

        protected bool ValidateContact(Contacts contactToValidate)
        {
            if (contactToValidate.FirstName.Trim().Length == 0)
                ModelState.AddModelError("FirstName", "Поле 'Фамилия' обязательно для заполнения");
            if (contactToValidate.SecondName.Trim().Length == 0)
                ModelState.AddModelError("LastName", "Поле 'Имя' обязательно для заполнения.");
            if (contactToValidate.Phone.Trim().Length == 0)
                ModelState.AddModelError("Phone", "Поле 'Телефон' обязательно для заполнения.");
            if (contactToValidate.Email.Length > 0 &&
                !Regex.IsMatch(contactToValidate.Email, @"^[\w-\.\.]+\@([\w-]+\.\.)+[\w-]{2,4}$"))
                ModelState.AddModelError("Email", "Введите корректный E-mail.");

            return ModelState.IsValid;
        }

        //
        // GET: /Home/
        public ActionResult Index(int? id)
        {
            ViewData["groups"] = groupsDAO.getAllGroups();
            return View(contactsDAO.getAllContacts(id));
        }

        //
        // GET: /Home/Details/5
        public ActionResult Details(int id)
        {
            return View(contactsDAO.getContact(id));
        }
    }
}
```

```

protected bool ViewDataSelectList(int GroupId)
{
    var groups = groupsDAO.getAllGroups();
    ViewData["GroupId"] = new SelectList(groups, "Id", "Name",
GroupId);

    return groups.Count() > 0;
}

//
// GET: /Home/Create
public ActionResult Create()
{
    if (!ViewDataSelectList(-1))
        return RedirectToAction("Index");
    return View("Create");
}

//
// POST: /Home/Create
[AcceptVerbs(HttpVerbs.Post)]
public ActionResult Create(int GroupId, [Bind(Exclude = "Id")]
Contacts contact)
{
    if (ValidateContact(contact) &&
contactsDAO.addContact(GroupId, contact))
        return RedirectToAction("Index");
    else
    {
        ViewDataSelectList(GroupId);
        return View("Create");
    }
}

//
// GET: /Home/Edit/5
public ActionResult Edit(int id)
{
    Contacts contact = contactsDAO.getContact(id);
    if (!ViewDataSelectList(contact.Groups.Id))
        return RedirectToAction("Index");
    return View(contactsDAO.getContact(id));
}

//
// POST: /Home/Edit/5
[AcceptVerbs(HttpVerbs.Post)]
public ActionResult Edit(int GroupId, int id, Contacts contact)
{
    if (ValidateContact(contact) &&
contactsDAO.updateContact(GroupId, contact))
        return RedirectToAction("Index");
    else
    {
        ViewDataSelectList(-1);
        return View("Edit", contactsDAO.getContact(id));
    }
}

//
// GET: /Home/Delete
public ActionResult Delete(int id)
{

```

```

        return View("Delete", contactsDAO.getContact(id));
    }

    //
    // POST: /Home/Delete
    [AcceptVerbs(HttpVerbs.Post)]
    public ActionResult Delete(int id, Contacts contact)
    {
        if (contactsDAO.deleteContact(contact))
            return RedirectToAction("Index");
        else
            return View("Delete", contactsDAO.getContact(id));
    }
}
}

```

Запустите проект, попробуйте ввести некорректные данные (например E-mail) при создании и редактировании контакта. Результат должен быть примерно таким, как показан на рис. 38.

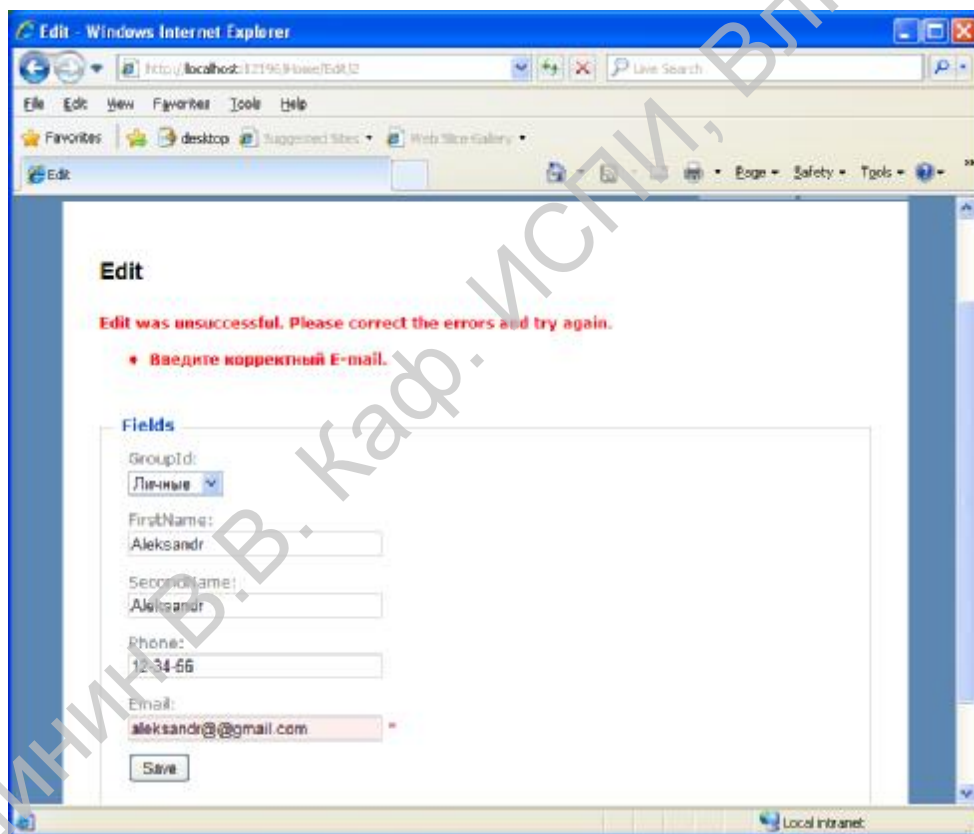


Рисунок 38. Результат валидации данных

Задание: сделать валидацию данных веб-форм для раздела Groups и проверить её работу.

Построение интерактивного приложения с использованием AJAX и добавление библиотек AJAX в Site.Master

Для использования технологии Ajax необходимо подключить JavaScript-библиотеки с данной технологией. В ASP.NET MVC Framework данные библиотеки встроены и находятся в папке scripts.

Добавим основные JavaScript-файлы внутрь тега <head> в Site.master:

Листинг 5 – Добавление AJAX-библиотек в Site.Master

```
<script src="../../Scripts/MicrosoftAjax.js"
type="text/javascript"></script>
<script src="../../Scripts/MicrosoftMvcAjax.js"
type="text/javascript"></script>
<script src="../../Scripts/jquery-1.3.2.min.js"
type="text/javascript"></script>
```

Примечание: версия библиотеки jQuery у вас может отличаться.

Рефакторинг представления Index под использование Ajax

Модифицируем страницу Index так, чтобы при выборе группы контактов, контакты из выбранной группы подгружались в активную область. Для этого щелкаем правой кнопкой мыши на папке /Views/Home и выбираем Add/View... (рис. 39).

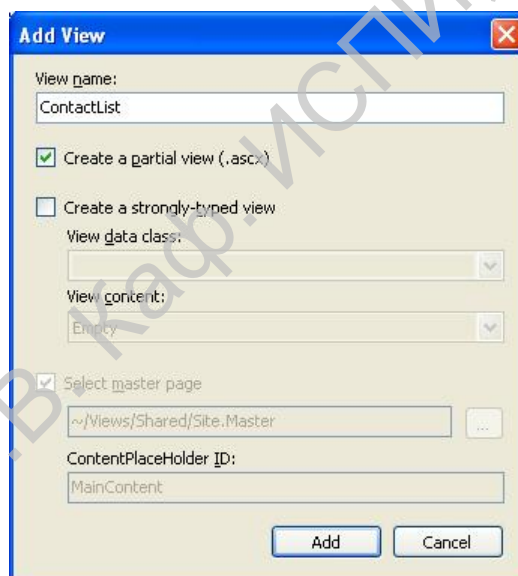


Рисунок 39. Добавление подключаемой области.

Листинг 6 - Views\Home\ContactList.ascx

```
<%@ Control Language="C#"
Inherits="System.Web.Mvc.ViewUserControl<IEnumerable<mvcApplication.Models.Co
ntacts>>" %>
<table>
<tr>
<th>Id</th>
<th>FirstName</th>
<th>SecondName</th>
<th>Phone</th>
<th>Email</th>
<th></th>
</tr>
<% foreach (var item in Model) { %>
<tr>
```

```
 <%= Html.Encode(item.Id) %></td>  <%= Html.Encode(item.FirstName) %></td>  <%= Html.Encode(item.SecondName) %></td>  <%= Html.Encode(item.Phone) %></td>  <%= Html.Encode(item.Email) %></td>  <%= Html.ActionLink("Edit", "Edit", new { id=item.Id }) %> | item.Id })%> | <%= Html.ActionLink("Details", "Details", new { id = })%> <%= Html.ActionLink("Delete", "Delete", new { id = item.Id })%> </td> </tr> <% } %> </table> | | | | | |
```

Листинг 7 – Views\Home\Index.aspx

```

<%@ Page Title="" Language="C#"
MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage<IEnumerable<MvcApplication.Models.Contacts>"
>%>

<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent"
runat="server">
    Index
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent"
runat="server">

    <script type="text/javascript">
        function beginContactList(args) {
            // Highlight selected group
            $('#leftColumn li').removeClass('selected');
            $(this).parent().addClass('selected');

            // Animate
            $('#divContactList').fadeOut('normal');
        }

        function successContactList() {
            // Animate
            $('#divContactList').fadeIn('normal');
        }

        function failureContactList() {
            alert("Could not retrieve contacts.");
        }
    </script>

    <h2>Index</h2>

    <ul id="groups">
        <% foreach (var item in
        (IEnumerable<MvcApplication.Models.Groups>)ViewData["groups"])
        { %>
            <li>
                <div><%= Ajax.ActionLink(item.Name, "Index", new { id =
                item.Id }, new AjaxOptions { UpdateTargetId = "divContactList", OnBegin =
                "beginContactList", OnSuccess = "successContactList", OnFailure =
                "failureContactList" })%></div>

```

```

        <% } %>
    </li>
</ul>

<div id="divContactList">
    <% Html.RenderPartial("ContactList", Model); %>
</div>

<p><%= Html.ActionLink("Create New", "Create") %></p>

</asp:Content>

```

Рефакторинг метода Index() HomeController

Листинг 8 - Controllers\HomeController.cs (метод Index)

```

public ActionResult Index(int? id)
{
    ViewData["groups"] = groupsDAO.getAllGroups();

    IEnumerable<Contacts> allContacts = contactsDAO.getAllContacts(id);

    if (!Request.IsAjaxRequest())
        // Normal Request
        return View(allContacts);
    else
        // Ajax Request
        return PartialView("ContactList", allContacts);
}

```

Запустите приложение. Посмотрите результат.

Задание для самостоятельного выполнения

Постройте самостоятельно для любых 2 частей приложения Ajax-подгрузку, например для страниц добавления, редактирования контактов.

Вопросы для самопроверки

- 1) Расскажите о средстве Web.sitemap.
- 2) Что такое валидация? Для чего она нужна?
- 3) Что такое Ajax? Какие библиотеки предназначены для работы в Ajax?
Опишите механизм его работы.