

Разработка web-приложений с использованием Microsoft® Visual Studio

В этом руководстве рассматривается разработка web-приложений с помощью технологии ASP.NET.

Содержание

Лабораторная работа 1. Создание основы веб-сайта приглашения на семинар	2
Упражнение 1. Создание основы веб-приложения ASP.NET	2
Упражнение 2. Создание форм веб-приложения	5
Упражнение 3. Работа с файлом Global.cs.....	7
Лабораторная работа 2. Обработка событий формы.....	8
Упражнение 1. Обработка события загрузки формы	8
Упражнение 2. Создание HTML-файлов	9
Лабораторная работа 3. Применение операторов C# в коде страницы	10
Упражнение 1. Создание итогового представления	10
Упражнение 2. Вызов метода из отделенного кода.....	12
Лабораторная работа 4. Использование Master Page при построении интернет-приложений.....	13
Упражнение 1. Добавление и применение главной страницы	13
Упражнение 2. Преобразование веб-форм в страницы содержимого	16
Упражнение 2. Добавление веб-формы как страницы содержимого	17
Лабораторная работа 5. Выполнение проверки достоверности	18
Упражнение 1. Добавление проверяющих элементов управления.....	18
Лабораторная работа 6. Применение технологии ASP.NET AJAX.....	20
Упражнение 1. Анимирование элемента управления UpdatePanel	20
Лабораторная работа 7. Управление состоянием в веб-приложениях.....	21
Упражнение 1. Отображение значения счетчика посещений	21
Лабораторная работа 8. Работа с базой данных.....	22
Упражнение 1. Использование Code-First	23
Упражнение 2. Отображение данных	30

Лабораторная работа 1. Создание основы веб-сайта приглашения на семинар

Постановка задачи

Требуется создать веб-сайт, который дал бы возможность приглашать на семинар, а приглашенным отправлять ответы на приглашение (repondez s'il vous plait - RSVP).

Должны быть следующие основные элементы:

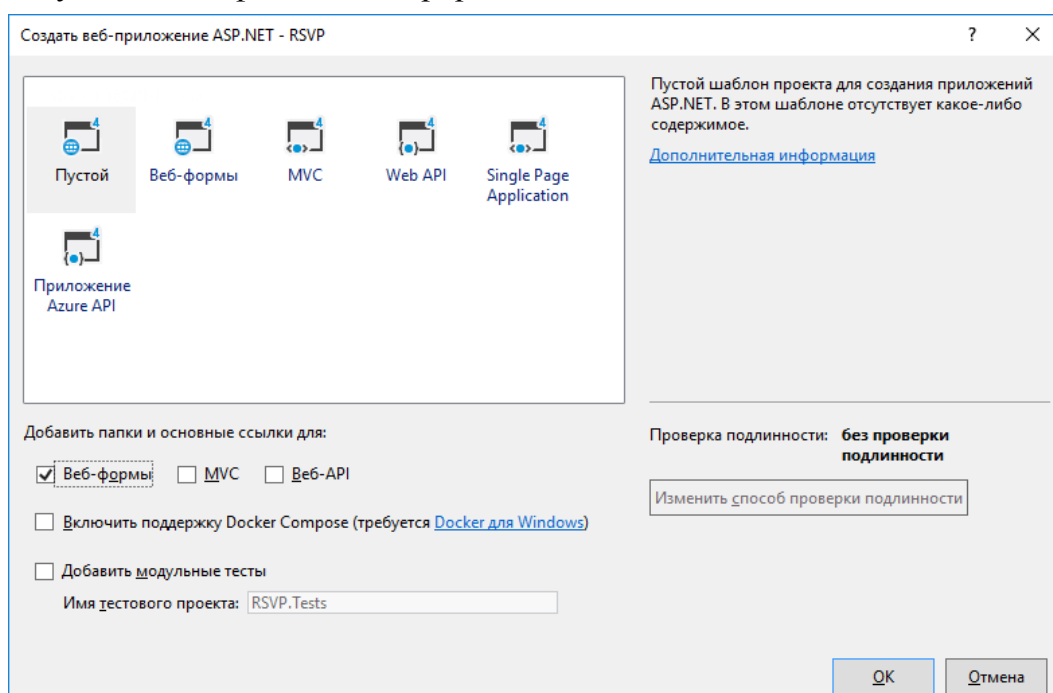
- страница, которая отображает информацию о семинаре и форму RSVP;
- проверка достоверности для формы RSVP, которая будет отображать страницу подтверждения;
- страница, на которой выводится список ответов от приглашенных.

Упражнение 1. Создание основы веб-приложения ASP.NET

Создание веб-проекта

В этом упражнении вы создадите простой веб-проект ASP.NET.

1. Запустите Visual Studio.
2. Создайте веб-проект, для этого выберите пункт **File (Файл) – New (Создать) – Project (Проект)**.
3. Откроется диалоговое окно **Создание проекта**. В списке шаблонов выберите **Web** и тип приложения – **Веб-приложение ASP.NET** с именем RSVP.
4. Откроется окно **Создать веб-приложение ASP.NET**. Выберите **Empty (Пустой)** шаблон, в разделе *Добавить папки и основные ссылки для:* установите флажок *Веб-формы* и нажмите OK:



5. Изучите структуру проекта по шаблону *ASP.NET Empty* – простейший из всех шаблонов сайтов. Он создает сайт, содержащий лишь файлы *Global.asax* (класс приложения – позволяет записывать обработчики событий, реагирующие на глобальные события) и *Web.config* (конфигурационная информация для приложения ASP.NET).

Создание модели данных

В веб-проекте будет использоваться модель данных.

1. Добавьте в папку **Models** проекта класс. Для этого в контекстном меню папки выберите пункт **Add → Class** (Добавить → Класс).
2. Укажите в качестве имени класса *GuestResponse.cs* и щелкните на кнопке **Add (Добавить)**.

Среда Visual Studio создаст новый файл класса C# и откроет его для редактирования.

3. Объявите в классе **GuestResponse** автоматически реализуемые свойства:

```
public class GuestResponse
{
    public int GuestResponseId { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    public string Phone { get; set; }
    public bool? WillAttend { get; set; }
    public DateTime Rdata { get; set; }
}
```

Первое свойство определяет уникальное значение, в дальнейшем оно будет использовано в качестве первичного ключа. Суффикс *Id* в имени требуется для генерации в базе данных первичного ключа. По умолчанию при генерации базы данных в качестве первичных ключей будут рассматриваться свойства с именами *Id* или *[Имя_класса]Id*.

Свойство *WillAttend* определено с типом *bool*, допускающим *null*. Это означает, что свойство может принимать значения *true*, *false* или *null*. Для представления ответов от приглашенных будут применяться экземпляры класса *GuestResponse*.

4. Добавьте в класс два конструктора, первый по умолчанию, второй с параметрами, принимающий имя, электронный адрес, телефон и информацию о согласии, дата должна вычисляться в теле конструктора как текущее значение:

```
public GuestResponse() { }

public GuestResponse(string name, string email, string phone,
bool? willattend)
{
```

```

        Name = name;
        Email = email;
        Phone = phone;
        WillAttend = willattend;
        Rdata = DateTime.Now;
    }

```

Создание хранилища данных

Созданные объекты `GuestResponse` должны записываться в хранилище.

Позднее в качестве хранилища будет выступать база данных. Пока будет реализовано хранение объектов в памяти. Преимущество такого подхода связано с простотой реализации, однако следует учитывать, что данные будут теряться при каждом останове или перезапуске приложения, поэтому лучше сохранять коллекцию в файл с помощью механизма сериализации.

1. Чтобы определить хранилище, добавьте в папку **Models** новый файл класса по имени `ResponseRepository.cs` и поместите в него код, показанный ниже:

```

using System.Collections.Generic;

public class ResponseRepository
{
    private static ResponseRepository repository = new
ResponseRepository();

    private List<GuestResponse> responses = new
List<GuestResponse>();

    public static ResponseRepository GetRepository()
    {
        return repository;
    }

    public IEnumerable<GuestResponse> GetAllResponses()
    {
        return responses;
    }

    public void AddResponse(GuestResponse response)
    {
        responses.Add(response);
    }
}

```

Хранилище обычно располагает методами для создания, чтения, обновления и удаления объектов данных (вместе называемых методами CRUD (creating, reading, updating, deleting – создание, чтение, обновление, удаление), но в этом приложении реализуется только возможность чтения всех объектов данных и добавления новых объектов данных.

Упражнение 2. Создание форм веб-приложения

Создание основной формы

Следующий шаг заключается в создании стартовой страницы, которая содержит информацию о семинаре.

1. Добавьте в проект сайта новую веб-форму с именем Start.aspx.
2. В раздел `<div>` внесите необходимые изменения для отображения информации на форме:

```
<div>
  <div class="rek">
    <h1>Приглашение на семинар</h1>
    <p>Вы приглашены на наш семинар</p>
    <p>Подтвердите свое согласие, пройдя регистрацию</p>
  </div>
  <div class="info">
    Семинар состоится 1 января 2021 года в 7.30
  </div>
</div>
```

3. Постройте решение.
4. Задайте форму Start.aspx в качестве начальной страницы, выбрав соответствующую команду контекстного меню файла веб-формы в обозревателе решений.
5. Запустите проект, выбрав пункт **Start Debugging** в меню **Debug** и проверьте отображение стартовой страницы.

Создание формы регистрации

Следующий шаг заключается в создании страницы регистрации, которая содержит информацию о клиенте, позволяющую приглашенным подготовить ответ.

1. Добавьте в проект сайта новую веб-форму с именем Reg.aspx.
2. Внесите необходимые изменения для отображения информации на форме

```
<div>
  <h1>Приглашаем на семинар</h1>
<p></p>
</div>
```

3. Добавьте серверные элементы, предназначенные для сбора информации от пользователей и кнопку для ее отправки:

```
<div>
  <label>Ваше имя:</label><asp:TextBox ID="name"
runat="server"></asp:TextBox>
</div>
<div>
  <label>Ваш email:</label><asp:TextBox ID="email"
runat="server"></asp:TextBox>
</div>
<div>
  <label>Ваш телефон:</label><asp:TextBox ID="phone"
runat="server"></asp:TextBox>
</div>
```

```

    <div>
      <label>Вы будете делать доклад?</label>
      <asp:CheckBox ID="CheckBoxYN" runat="server" />
    </div>
    <div>
      <button type="submit">Отправить ответ на приглашение
RSVP</button>
    </div>

```

4. Откройте страницу в браузере и посмотрите, как выглядят указанные элементы.

Стилизация созданных форм

Элементы веб-формы стилизуются с применением каскадных таблиц стилей (Cascading Style Sheets - CSS).

1. Добавьте в приложение таблицу стилей с помощью команды контекстного меню **Add → StyleSheet** (**Добавить → Таблица стилей**). В появившемся диалоговом окне в качестве имени укажите Styles и щелкните на кнопке ОК.

Среда Visual Studio добавит к проекту новый файл Styles.css.

2. Укажите в этом файле следующий код:

```

h1 {
    margin: 0px;
    padding: 2px 0;
    font-size: 30px;
    font-weight: bold;
}
#form1 label {
    width: 120px;
    display: inline-block;
}

#form1 input {
    margin: 2px;
    margin-left: 4px;
    width: 150px;
}

#form1 select {
    margin: 2px 0;
    width: 154px;
}

button[type=submit] {
    margin-top: 15px;
    padding: 5px;
}

```

3. Добавьте элемент `link` (таблица стилей CSS ассоциируется с веб-формой с помощью этого элемента) в раздел `head` файлов `Reg.aspx` и `Start.aspx`:

```
...
<head runat="server">
    <title></title>
    <link rel="stylesheet" href="Styles.css" />
</head>
...
```

Обратите внимание, что для ссылки на файл, содержащий базовые стили CSS, используется стандартный HTML-элемент.

4. Постройте приложение. Запустите приложение и посмотрите влияние стилей на отображение форм в браузере (для открытия формы регистрации укажите в адресной строке браузера `http://localhost:ваш_port/Reg.aspx`).

Упражнение 3. Работа с файлом Global.cs

В этом упражнении вы используете файл `Global.asax`, который позволяет записывать обработчики событий, реагирующие на глобальные события. Пользователи не могут запрашивать файл `Global.asax` напрямую. Код в файле `Global.asax` пишется точно так же, как и для веб-формы. Отличие заключается в том, что файл `Global.asax` не содержит дескрипторов HTML или ASP.NET. Вместо этого он содержит методы со специфическими предопределенными именами.

Файл `global.asax` является необязательным, но каждому веб-приложению разрешено иметь не более одного файла `global.asax`, который должен находиться в корневом каталоге приложения, а не в каком-то из его подкаталогов.

1. Откройте файл `Global.asax`.
2. В раздел скрипта `<script runat="server">` добавьте обработчик события `EndRequest` класса приложения **`HttpApplication`**. Это событие происходит при каждом запросе страницы:

```
protected void Application_OnEndRequest()
{
    Response.Write("<hr />Эта страница была загружена " +
        DateTime.Now.ToString());
}
```

В этом коде используется объект `Response`, который является встроенным свойством класса **`HttpApplication`**.

Поскольку обработчик событий реагирует на событие `HttpApplication.EndRequest`, метод `Application_OnEndRequest()` будет выполняться при каждом запросе страницы, сразу же после завершения выполнения всего имеющегося в ней кода обработки событий.

3. Запустите снова приложение. Проверьте, что метод `Application_OnEndRequest()` выводит в нижней части страницы колонтитул с датой и временем ее создания.

Лабораторная работа 2. Обработка событий формы

В этой работе вы реализуете реакцию на отправку данных на семинар как реакцию на кнопку "Отправить ответ приглашение RSVP".

Упражнение 1. Обработка события загрузки формы

1. Откройте файл Reg.aspx.cs в режиме кода. Изучите его содержимое.

Базовым для класса отделенного кода является класс `System.Web.UI.Page`, который содержит несколько методов и свойств для ответа на веб-запросы. По умолчанию добавлен пустой метод `Page_Load()`, вызываемый ASP.NET при поступлении запросов для Reg.aspx и предоставляющий возможность отреагировать на такие запросы. В данном случае метод `Page_Load()` будет вызван один раз во время первоначальной загрузки страницы и еще раз, когда пользователь отправит форму.

2. В обработчике события загрузки формы сначала с помощью проверки свойства `IsPostBack` определите, относится ли запрос, на который производится ответ, к форме, отправленной обратно серверу, и если это так, то создайте новый экземпляр объекта `GuestResponse` модели данных с полученными от элементов формы данными:

```
protected void Page_Load(object sender, EventArgs e)
{
    if (IsPostBack)
    {
        GuestResponse rsvp = new GuestResponse(name.Text,
        email.Text, phone.Text, CheckBoxYN.Checked);
    }
}
```

3. Затем объект `GuestResponse` поместите в хранилище (далее весь код размещается в теле оператора выбора):

```
ResponseRepository.GetRepository().AddResponse(rsvp);
```

4. Пользователю необходимо предоставить какой-либо отклик после того, как он отправил форму, и это делается с помощью метода `Response.Redirect()`, который выполняет перенаправление пользовательского браузера. Реализуйте проверку свойства `WillAttend`: если оно равно `true`, пользователь будет делать на семинаре доклад, поэтому он перенаправляется на файл `seeyouthere.html`. В противном случае перенаправление происходит на файл `sorryyoucantcome.html`:

```
if (rsvp.WillAttend.HasValue && rsvp.WillAttend.Value)
{
    Response.Redirect("seeyouthere.html");
}
else
{
}
```



```

        Response.Redirect("sorryyoucantcome.html");
    }

```

5. В итоге обработчик события Page_Load() для реагирования на запросы должен выглядеть следующим образом:

```

protected void Page_Load(object sender, EventArgs e)
{
    if (IsPostBack)
    {
        GuestResponse rsvp = new GuestResponse(name.Text,
        email.Text, phone.Text, CheckBoxYN.Checked);

        ResponseRepository.GetRepository().AddResponse(rsvp);

        if (rsvp.WillAttend.HasValue && rsvp.WillAttend.Value)
        {
            Response.Redirect("seeyouthere.html");
        }
        else
        {
            Response.Redirect("sorryyoucantcome.html");
        }
    }
}

```

6. Постройте приложение.

Упражнение 2. Создание HTML-файлов

В этой части вы добавите в проект сайта обычные статические HTML-файлы, играющие роль ответов на отправку ответа на приглашение.

1. Для создания первого файла ответа выберите в контекстном меню проекта пункт **Add → New Item** (Добавить → Создать элемент).
2. В открывшемся диалоговом окне **Add New Item** укажите **шаблон HTML Page** (HTML-страница) и задайте странице имя seeyouthere.html, щелкните на кнопке **Add (Добавить)** для создания HTML-файла.
3. Добавьте в тег заголовка нового файла:

```
<title>Приглашение на семинар</title>
```

4. В теге <body> укажите текст реакции на действия клиента:

```

<body>
    <h1>Увидимся на семинаре</h1>
    <p>Презентация для доклада должна быть загружена до начала работы секции
</p>
    <p>Начало регистрации в 9.00.</p>
</body>

```

5. Создайте второй файл ответа с отказом от участия с докладом - sorryyoucantcome.html и укажите в нем соответствующий текст:

```
<title> Приглашение на семинар </title>
```

...

```
<body>
  <h1>Жаль, что вы не сможете выступить с докладом</h1>
  <p>Начало регистрации в 9.00.</p>
</body>
```

Тестирование формы регистрации

1. Откройте форму Reg.aspx в режиме просмотра в браузере.
2. Заполните поля формы и выберите установите флажок. После отправки формы вы увидите ответ, который должен отображаться в случае выбора участия с докладом.
3. Используя кнопки браузера вернитесь в стартовую форму, снимите флажок и снова отправьте ответ, должна отобразиться страница в случае отказа от участия с докладом

Лабораторная работа 3. Применение операторов C# в коде страницы

Упражнение 1. Создание итогового представления

В этом разделе вы добавите поддержку для отображения итоговых сведений по полученным ответам, чтобы можно было видеть, кто планирует выступить на семинаре с докладом.

1. Добавьте в проект сайта новую web форму – Summary.aspx.
2. Приведите содержимое файла в соответствие с примером:

```
<!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>Участники семинара</title>
  <link rel="stylesheet" href="Styles.css" />
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <h2>Приглашения</h2>

      <h3>Выступающие с докладом: </h3>
      <table>
        <thead>
          <tr>
            <th>Имя</th>
            <th>Email</th>
            <th>Телефон</th>
          </tr>
        </thead>
        <tbody>

        </tbody>
      </table>

    </div>
  </form>
```

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

3. Изучите содержимое файла.
4. Внутри блока кода `<tbody>` `</tbody>` добавьте код C#, который обеспечит заполнение таблицы с помощью запроса LINQ:

```

        <tbody>
            <% var yesData =
                RSVP.Models.ResponseRepository.GetRepository().GetAllResponses()
                    .Where(r => r.WillAttend.Value);
                foreach (var rsvp in yesData) {
                    string htmlString =
                        String.Format("<tr><td>{0}</td><td>{1}</td><td>{2}</td><td>{3}</td>",
                            rsvp.Name, rsvp.Email, rsvp.Phone, rsvp.Rdata);
                    Response.Write(htmlString);
                } %>
        </tbody>
```

Имя пространства имен класса

5. Изучите данный фрагмент.

В нем дополнительно применяются дескрипторы `<%` и `%>` для добавления динамического контента к генерируемому HTML-коду, когда браузер запрашивает данный файл. Дескрипторы `<%` и `%>` формально называются ограничителями сценариев серверной стороны.

Внутри блока кода используются обычные операторы C# для генерации набора HTML-элементов, представляющих собой строки в элементе `table`, в этих строках перечислены люди, принявшие решение выступить на семинаре с докладом.

Для получения всех объектов данных из хранилища вызывается метод `ResponseRepository.GetRepository().GetAllResponses()`, а для выборки всех подтверждающих участие ответов применяется LINQ-выражение `Where()`.

Затем в цикле `foreach` выполняется запрос и генерируются HTML-строки для каждого объекта данных.

Метод `String.Format()` позволяет компоновать HTML-строки, содержащие значения свойств из каждого объекта `GuestResponse`, который необходимо отобразить.

Для добавления HTML-кода в вывод, отправляемый браузеру, используется метод `Response.Write()`.

Форматирование динамического HTML-кода

В файл `Summary.aspx` был включен элемент `link`, который импортирует файл `Styles.css` с содержащимися внутри него стилями.

1. Добавьте в файл `Styles.css` стиль для применения внутри `Summary.aspx`.

```
...
table, td, th {
    border: thin solid black;
    border-collapse: collapse;
    padding: 5px;
    background-color: lemonchiffon;
```

```

text-align: left;
margin: 10px 0;
}

```

Тестирование динамического кода

2. Чтобы протестировать файл Summary.aspx, откройте форму Reg.aspx в режиме просмотра в браузере и воспользуйтесь страницей регистрации для добавления данных в хранилище.
3. После нескольких отправок формы перейдите на URL вида `"/Summary.aspx"`; появится таблица с приглашенными на семинар.

Упражнение 2. Вызов метода из отделенного кода

В этом упражнении вы примените другой подход, который предусматривает определение методов в файле отделенного кода, а затем использование фрагментов кода для вызова этих методов и вставки результатов в HTML-код, отправляемый браузеру.

1. В файле отделенного кода Summary.aspx.cs определите новый метод `GetNoShowHtml()`. Этот метод генерирует таблицу строк с теми, кто не будет выступать с докладом:

```

using System.Text;

public partial class Summary : System.Web.UI.Page
{
    protected string GetNoShowHtml()
    {
        StringBuilder html = new StringBuilder();
        var noData = ResponseRepository.GetRepository()
            .GetAllResponses().Where(r => !r.WillAttend.Value);
        foreach (var rsvp in noData)
        {
            html.Append(String.Format("<tr><td>{0}</td><td>{1}</td><td>{2}</td>",
                                     rsvp.Name, rsvp.Email, rsvp.Phone));
        }
        return html.ToString();
    }
}

```

2. В режиме разметки формы Summary.aspx добавьте разметку для формирования таблицы и вызовите новый метод внутри фрагмента кода `<tbody> </tbody>`:

```

...
</table>
<h3>Участники без доклада: </h3>
<table>
  <thead>
    <tr>
      <th>Имя</th>

```

```

        <th>Email</th>
        <th>Телефон</th>
    </tr>
</thead>
<tbody>
    <%= GetNoShowHtml()%>
</tbody>
</table>

</body></html>

```

В этом примере применяется фрагмент кода с открывающим дескриптором `<%=`. Это сообщает ASP.NET о необходимости вставки результата выполнения метода в вывод, отправляемый браузеру, что представляет собой более аккуратный и читабельный подход, чем включение кода непосредственно в страницу. Генерируется такой же HTML-код, что и с помощью предыдущего фрагмента кода, но только в данном случае получается таблица строк для людей, которые не выступают с докладом.

3. Протестируйте работу форм. После нескольких отправок формы перейдите на URL вида `"/Summary.aspx"` и проверьте таблицу с людьми, выступающими и не выступающими на семинаре с докладом.

Лабораторная работа 4. Использование Master Page при построении интернет-приложений

В этой работе вы обеспечите единообразный пользовательский интерфейс для всех клиентов. Для того чтобы реализовать такую возможность вам необходимо добавить главную (мастер) страницу (Master Page) и преобразовать существующие веб-формы в страницы содержимого.

Мастер-страница содержит фиксированные элементы, одинаковые для всех страниц, и заполнитель содержимого для остальной части страницы. Наиболее типичными фиксированными элементами являются верхний и нижний колонтитулы, панель навигации, панель меню и т.д. Страница содержимого получает от мастер-страницы фиксированные элементы и предоставляет дополнительное содержимое

Упражнение 1. Добавление и применение главной страницы

В этом упражнении вы добавите в существующий веб-сайт главную страницу, разработаете ее структуру (верхний, нижний колонтитулы и панель навигации – общие для всех страниц элементы) и определите элемент управления **ContentPlaceHolder**.

1. Добавьте в существующий веб-сайт **RSVP** главную страницу (**Master Page**) со следующими параметрами:
 - a. **Name** – Site.master

- b. Установите флажки: *Place code in separate file* (Разместить код в отдельном файле) – включен, *Select master page* (Выбрать главную страницу) – снят.

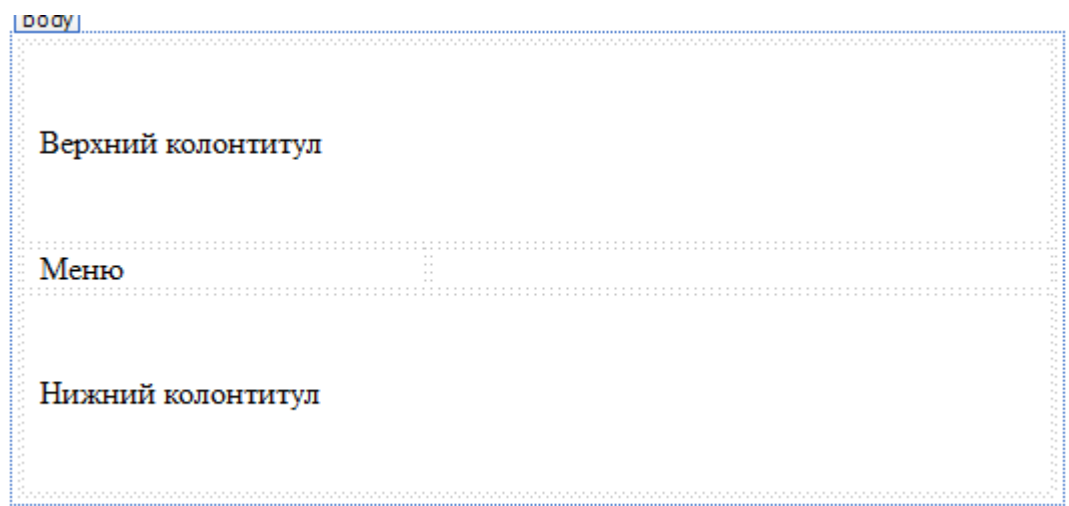
2. Изучите содержимое главной страницы.

Как видно из ее исходного кода, она подобна обычной веб-странице ASP.NET, отличие заключается в том, что главная страница начинается с директивы **Master** и содержит элемент управления **ContentPlaceHolder**, который предназначен для определения области, куда страница содержимого может вставлять содержимое. При создании новой мастер-страницы элемент **ContentPlaceHolder** создается по умолчанию, хотя его можно удалить или добавить еще один или несколько.

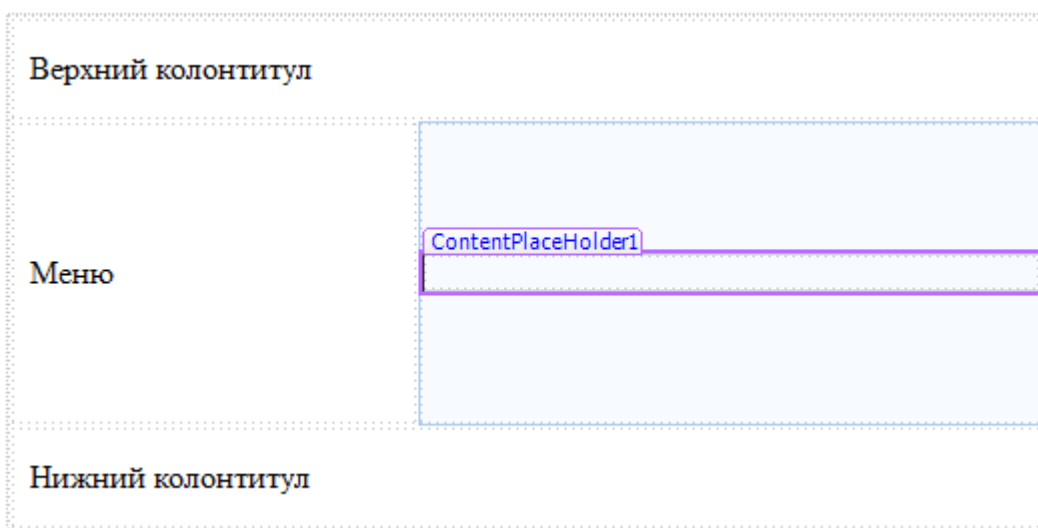
3. Удалите автоматически добавленные элементы **ContentPlaceHolder** со свойствами **id** равными **head** и **ContentPlaceHolder1**. Требуемые элементы будут потом добавлены.
4. Перейдите в режим конструктора. Добавьте на страницу таблицу, содержащую три строки и два столбца – меню **Таблица** → **Вставить таблицу**. В окне **Вставка таблицы** укажите 3 строки и 2 столбца и нажмите ОК.
5. С помощью команды **Изменить** меню **Таблица** объедините ячейки в первой и последней строках, так чтобы в первой и последней строках таблица содержала по одной ячейке, а во второй – две.
6. Перейдите в режим разметки и внесите код для настройки размера ячеек и содержания:

```
<table class="auto-style1">
  <tr>
    <td colspan="2" style="height:100px">&nbsp;
      Верхний колонтитул
    </td>
  </tr>
  <tr>
    <td style="width:200px">&nbsp;
      Меню
    </td>
    <td>&nbsp;</td>
  </tr>
  <tr>
    <td colspan="2" style="height:100px">&nbsp;
      Нижний колонтитул
    </td>
  </tr>
</table>
```

7. В режиме конструктора посмотрите, что в результате была добавлена обычная таблица, для которой были установлены значения ширины и высоты некоторых ячеек:



8. В режиме конструктора добавьте элемент управления **ContentPlaceholder** во вторую ячейку второй строки таблицы, а также измените высоту строк, содержащих верхний и нижний колонтитулы (по 50px) и строку меню (150px). В результате главная страница в режиме разработки будет выглядеть следующим образом:



9. Вместо строки "верхний колонтитул" укажите строку "RSVP", помещенную внутри тэга `<h2>`:
- ```
<h2>RSVP</h2>
```
10. Установите содержимое элемента **title** равным **RSVP**:
- ```
<title>RSVP</title>
```
11. Перенесите сгенерированные параметры стиля главной страницы в существующий файл стилей Styles.css и добавьте после закрывающего тега элемента **title** ссылку на файл стилей:
- ```
<link rel="stylesheet" type="text/css" href="Styles.css" />
```
12. Сохраните главную страницу.

В результате выполнения данного упражнения вы создали главную страницу с именем Site.master, а также определили в главной странице элемент управления **ContentPlaceholder**.

## Упражнение 2. Преобразование веб-форм в страницы содержимого

В этом упражнении вы преобразуете веб-форму по умолчанию в страницу содержимого и добавите в главную страницу возможности навигации.

### Преобразование веб-формы по умолчанию в страницу содержимого

1. Откройте веб-форму Start.aspx. Добавьте в директиву **Page** свойство **MasterPageFile** и установите его значение равным ~/Site.master:

```
...MasterPageFile="~/Site.master" %>
```

2. В веб-форме Start.aspx удалите элементы HTML верхнего уровня, за исключением **div** элемента:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
</form>
</body>
</html>
```

3. После директивы **Page** добавьте серверный элемент управления **Content**:

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

4. Перенесите весь код, ограниченный элементом **div** внутри элемента управления **Content**.
5. Сохраните изменения, внесенные в главную страницу Site.master. Сохраните и закройте веб-форму Start.aspx.
6. Постройте и запустите приложение. Должна открыться страница с разделением на области.
7. Аналогичным образом преобразуйте страницы Reg.aspx и Summary.aspx в страницы содержимого.

### Добавление в главную страницу возможности навигации

1. Откройте главную страницу в конструкторе и добавьте в левую область второй строки таблицы меню, добавив элемент управления с именем **MainMenu**, завернутый в **div**-элемент со значением атрибута класса **menu**, ориентацией вертикально и цветом команд красный:

```
<div class="menu">
<asp:Menu ID="NavigationMenu" runat="server" CssClass="menu"
EnableViewState="false" Orientation="Vertical" ForeColor="Red">
```



&lt;/asp:Menu&gt;

2. Добавьте в меню коллекцию элементов меню с указанием путей навигации к существующим страницам:

```
<div class="menu">
 <asp:Menu ID="NavigationMenu" runat="server" CssClass="menu"
EnableViewState="false" Orientation="Vertical" ForeColor="Red">
 <Items>
 <asp:MenuItem NavigateUrl="~/Start.aspx"
Text="Главная"/>
 <asp:MenuItem NavigateUrl="~/Reg.aspx"
Text="Регистрация"/>
 <asp:MenuItem NavigateUrl="~/Summary.aspx"
Text="Отчет о регистрации"/>
 </Items>
 </asp:Menu>
```

3. Добавьте в нижний колонтитул главной страницы ссылки на страницы в виде маркированного списка, устанавливаемый тегом `<ul>`:

```
<td colspan="3" class="auto-style4">
 <%--Нижний колонтитул--%>
 <div id="temo_footer">
 <ul class="footer_menu">
 Главная
 Регистрация
 Отчет о регистрации
 Подробности

 </div>
</td>
```

4. Добавьте в файл стилей Styles.css параметры для форматирования списка ссылок:

```
.footer_menu li {
margin: 0px;
padding: 0 20px;
display: inline;
border-right: 1px solid #000000;
}
```

5. Постройте и запустите приложение. Должна открыться стартовая страница с меню. Протестируйте ее работу.

### Упражнение 2. Добавление веб-формы как страницы содержимого

В этом упражнении вы добавите веб-форму как страницу содержимого.

## Добавление веб-формы как страницы содержимого

1. Добавьте в проект сайта новую веб-форму About.aspx и при добавлении укажите главную страницу – Site.master.
2. Изучите содержимое новой страницы, обратите внимание на свойство **MasterPageFile**, равное значению ~/Site.master:

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true"
CodeFile="About.aspx.cs" Inherits="About" %>
```

```
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
</asp:Content>
```

3. Вставьте внутрь элемента Content справочную информацию, например:

```
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
 <h1>Семинар</h1>
 <h2>Приглашаем на семинар</h2>
 <p>Напоминаем Вам, что если Вы планируете выступить с докладом, то презентация для
доклада должна быть загружена до начала работы секции </p>
 <p>Начало регистрации 1.01.2017 в 9.00.</p>
</asp:Content>
```

4. Сохраните изменения, внесенные в страницу.

5. Добавьте в файл главной страницы в коллекцию меню новый элемент с указанием пути к новой странице:

```
<asp:MenuItem NavigateUrl="~/About.aspx" Text="Подробнее"/>
```

6. Постройте и запустите приложение, проверьте работу внесенных изменений.

## Лабораторная работа 5. Выполнение проверки достоверности

В этой работе вы реализуете проверку вводимых пользователями данных на предмет наличия значений во всех полях формы, корректность специфических данных, таких как электронный адрес и телефон.

### Упражнение 1. Добавление проверяющих элементов управления

В этом упражнении вы добавите в пользовательский элемент управления проверяющие элементы управления.

1. Откройте форму Reg.aspx в режиме конструктора.
2. Перенесите с панели инструментов проверяющий элемент управления **RequiredFieldValidator** и разместите его справа от элемента управления – текстового поля name, с помощью окна свойств настройте его свойства:
  - a. укажите свойству ControlToValidate значение идентификатора проверяемого элемента (name),
  - b. свойству ErrorMessage укажите текст "Заполните поле имени", отображаемый в элементе **ValidationSummary** (он будет добавлен позже) в случае ошибки,
  - c. цвет шрифта ошибки – красный,
  - d. текст ошибки – свойство Text укажите фразу "Не оставляйте поле пустым".
3. Проверьте изменения в режиме разметки после внесенных изменений:

```
<div>
 <label>Ваше имя:</label><asp:TextBox ID="name" runat="server"></asp:TextBox>
 <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
 ControlToValidate="name"
 ErrorMessage="Заполните поле имени">
```

```

 ForeColor="Red">Не оставляйте поле пустым</asp:RequiredFieldValidator>
 </div>

```

4. Аналогичным образом добавьте и настройте элементы **RequiredFieldValidator** для полей ввода электронного адреса и телефона:

```

<div>
 <label>Ваш email:</label><asp:TextBox ID="email"
runat="server"></asp:TextBox>
 <asp:RequiredFieldValidator ID="RequiredFieldValidator4" runat="server"
ControlToValidate="email" ErrorMessage="Заполните поле адреса" ForeColor="Red">Не
оставляйте поле пустым</asp:RequiredFieldValidator>
</div>

<div>
 <label>Ваш телефон:</label><asp:TextBox ID="phone"
runat="server"></asp:TextBox>
 <asp:RequiredFieldValidator ID="RequiredFieldValidator3" runat="server"
ControlToValidate="phone" ErrorMessage="Заполните поле ввода телефона" ForeColor="Red">Не
оставляйте поле пустым</asp:RequiredFieldValidator>
</div>

```

5. После элемента **RequiredFieldValidator** добавьте для проверки поля ввода электронного адреса элемент **RegularExpressionValidator**, который проверяет правильность формата адреса согласно шаблона:

```

<asp:RegularExpressionValidator runat="server" ControlToValidate="email"
ValidationExpression="\w+([-+.']\w+)*@\w+([-.\w+)*\.\w+([-.\w+)*"
ErrorMessage="E-mail is not in a valid format" Display="Dynamic"
ForeColor="Red">Адрес введен неверно!!!</asp:RegularExpressionValidator>

```

6. Для отображения сообщений в одном месте, когда возникают проблемы с проверкой достоверности данных формы добавьте элемент управления **ValidationSummary** (этот элемент отображает значение свойства **ErrorMessage** элементов проверки) перед элементом ввода имени:

```

<asp:ValidationSummary ID="validationSummary" runat="server"
ShowModelStateErrors="true" />

```

7. Для обеспечения корректной работы проверяющих элементов необходимо отключить свойство **UnobtrusiveValidationMode**. Для этого в директиве **Page** этому свойству укажите значение **None**:

```

<%@ Page Language="C#" . . . UnobtrusiveValidationMode="None" %>

```

В случае необходимости эту функциональность можно обеспечить для всего сайта, добавив требуемое значение в файл конфигурации **web.config**:

```

<configuration>
 <appSettings>
 <add key="ValidationSettings:UnobtrusiveValidationMode"
value="None" />
 </appSettings>
</configuration>

```

8. Для проверки страницы добавьте в файл **Reg.aspx.cs** в обработчик события **Page\_Load** вызов метода **Validate()**, осуществляющий проверку данных при выполнении возврата данных веб-формы на сервер

(postback) и затем проверку на правильность страницы – в случае ошибок следует запретить инициализацию свойств модели:

```
if (IsPostBack)
{
 Page.Validate();
 if (!Page.IsValid)
 return;
 . . .
}
```

9. Постройте и запустите приложение. Протестируйте работу элементов проверки вводимых данных.

## Лабораторная работа 6. Применение технологии ASP.NET AJAX

В этой лабораторной работе вы реализуете частичное обновление страниц с помощью технологии ASP.NET AJAX.

Для обновления части интерфейса, представляемый элементом `updatePanel` будет использоваться элемент ASP.NET AJAX `Timer`, который можно сконфигурировать на запуск обратных отправок через регулярные интервалы времени. Если элемент управления содержится внутри элемента `UpdatePanel`, то этот `UpdatePanel` будет обновляться при каждом срабатывании элемента `Timer`.

### *Упражнение 1. Анимирование элемента управления UpdatePanel*

В этом упражнении вы добавите на стартовую страницу отчет обратного времени до наступления семинара.

1. Откройте стартовую страницу в режиме конструктора.
2. Перенесите со вкладки AJAX-расширения в панели элементов элемент управления **ScriptManager** и разместите его после заголовка «Приглашение на семинар».
3. Добавьте на форму после информации о дате семинара элемент управления **UpdatePanel**.
4. Откройте страницу в режиме разметки и изучите код добавления новых элементов.
5. Добавьте в элемент **ContentTemplate** элемента `asp:UpdatePanel` элемент **Timer**. и укажите свойству `Interval` значение 1000:

```
<asp:Timer runat="server" Interval="1000"></asp:Timer>
```

6. Добавьте в элемент **ContentTemplate** элемента `asp:UpdatePanel` код вычисляющий разность дней, минут и секунд между назначенной датой семинара и текущей датой и отобразите результат в виде строки:

```
<ContentTemplate>
 <%
```

```

DateTime dataseminar = new DateTime(2018,1,1,7,30,0);
DateTime dnow = DateTime.Now;
int rd = (dataseminar - dnow).Days;
int rm = (dataseminar - dnow).Minutes;
int rsec = (dataseminar - dnow).Seconds;
%>

```

```

<h3>До семинара осталось <%=rd.ToString()%> дн., <%=rm.ToString()%> мин. и
<%=rsec.ToString()%> с.</h3>

```

7. Постройте и запустите приложение. Протестируйте работу счетчика времени.

## Лабораторная работа 7. Управление состоянием в веб-приложениях

### Упражнение 1. Отображение значения счетчика посещений

В этом упражнении вы добавите счетчик посещений и реализуете его обновление в каждой новой сессии.

#### Инициализация переменной приложения

1. Добавьте в проект сайта глобальный класс приложения – **Global.asax**. Файл Global.asax содержит обработчики событий, которые реагируют на глобальные события приложения
2. В метод **Application\_Start()** добавьте переменную с именем **Visitors** и значением **0**.

```
Application["Visitors"] = 0;
```

3. Сохраните файл Global.asax.

#### Добавление счетчика посещений

1. Откройте файл главной страницы в представлении разметки.
2. Добавьте **div**-элемент с атрибутом **class** равному значению **footer** после закрывающего тега списка навигационных ссылок и вставьте в него управления типа **Literal** с именем **visitorLiteral**:

```

<div class="footer">
 <asp:Literal ID="VisitorLiteral" runat="server" />
</div>

```

3. Сохраните и закройте главную форму.
4. В файл стилей Site.css добавьте в конец файла следующий стиль:

```

.footer
{
 font-size: 18px;
 text-align: left;
}

```

5. Сохраните и закройте файл Site.css.
6. В обработчике события **Page\_Load** главной формы **Site.master** создайте локальную переменную типа **long** с именем **numVisitors**, предназначенную для хранения числа посещений. Инициализируйте переменную значением **0**.

```
long numVisitors = 0;
```

7. Далее в том же обработчике проверьте, существует ли переменная **Visitors** в объекте приложения (**Application**). В случае если переменная существует, выполните присваивание ее значения, тип которого приведен к типу **long**, локальной переменной **numVisitors**.

```
if (Application["Visitors"] != null)
{
 numVisitors =
 long.Parse(Application["Visitors"].ToString());
}
```

8. Затем выполните присваивание текста “**Число посещений:**” и значения переменной **numVisitors** свойству **Text** элемента управления **visitorLiteral**.

```
VisitorLiteral.Text = "Число посещений: " + numVisitors.ToString();
```

9. Сохраните Site.master.aspx.cs.

### Обновление счетчика посещений в новой сессии

1. Для того, чтобы при каждом создании пользователем новой сессии увеличивалось значение переменной **Visitors** на единицу добавьте в метод **Session\_Start** файла Global.asax следующий код:

```
void Session_Start(object sender, EventArgs e)
{
 // Код, выполняемый при запуске нового сеанса
 // Increment Visitors counter
 Application["Visitors"] =
long.Parse(Application["Visitors"].ToString()) + 1;
}
```

2. Сохраните и закройте файл Global.asax.

### Тестирование счетчика посещений

1. Постройте и запустите приложение.

Обратите внимание на то, что счетчик посещений установлен в значение **1**.

2. Закройте браузер и снова запустите приложение.

Обратите внимание на то, что счетчик посещений установлен в значение **2**.

## Лабораторная работа 8. Работа с базой данных

В этой работе вы будете применять для работы с базой данных объектно-ориентированную технологию доступа к данным, являющееся *object-relational mapping* (ORM) решением для .NET Framework от Microsoft – **ADO.NET Entity Framework (EF)**.

Платформа ADO.NET Entity Framework позволяет разработчикам создавать приложения для доступа к данным, работающие с концептуальной моделью приложения, а не напрямую с реляционной схемой хранения. Ее целью является уменьшение объема кода и усилий по обслуживанию приложений, ориентированных на обработку данных.

В первом упражнении рассматривается подход **Code-First**, при использовании которого разработчик создает классы модели данных, которые будут храниться в базе данных, а затем **Entity Framework** по этой модели генерирует эту базу данных и ее таблицы. определяется модель в коде, а затем, на ее основе создается (или модифицируется) база данных.

Во втором упражнении вы используете подход **Database first**, при котором средства Visual Studio на основе созданной ранее базы данных формируют ее модель – Entity Data Model (EDM). Платформа Entity Framework поддерживает модель EDM для определения данных на концептуальном уровне. При использовании конструктора ADO.NET EDM Designer концептуальная модель, модель хранения и сведения о сопоставлениях содержатся в файле EDMX.

### ***Упражнение 1. Использование Code-First***

В этом упражнении сначала определяется модель в коде, а затем, на ее основе создается база данных. В итоге вы создадите две таблицы, описывающие данные участника семинара и его доклады на этом семинаре. Отношение между этими таблицами будет “один ко многим” (one-to-many).

#### **Изменение существующей модели**

В приложении, разработанном в предыдущих работах простая модель, состоящая из одного класса. Для наглядной работы с **Entity Framework** вы ее усложните, теперь модель будет включать два класса: участника семинара и его доклады на этом семинаре.

1. Откройте файл GuestResponse.cs и добавьте новый класс **Report**, моделирующий сущность «доклад», включающий следующие свойства:
  - a. Идентификатор (ReportId),
  - b. Название доклада (NameReport),
  - c. Аннотация (Annotation),
  - d. Ссылка на автора доклада (GuestResponse)

```
public class Report
{
 public int ReportId { get; set; }
 public string NameReport { get; set; }
 public string Annotation { get; set; }

 public GuestResponse GuestRes { get; set; }

 public Report() { }
 public Report(string title, string annot)
 {
 NameReport = title;
 Annotation = annot;
 }
}
```

2. В существующий класс **GuestResponse** добавьте ссылку на коллекцию докладов в виде виртуального свойства обобщенного типа `List<T>` и в конструкторе добавьте код ее создания:

```
public class GuestResponse
{
 public int GuestResponseId { get; set; }
 public string Name { get; set; }

 public string Email { get; set; }

 public string Phone { get; set; }
 public bool? WillAttend { get; set; }
 public DateTime Rdata {get; set;}

 public virtual List<Report> Reports { get; set; }

 public GuestResponse(string name, string email, string phone,
bool? willattend)
 {
 Name = name;
 Email = email;
 Phone = phone;
 WillAttend = willattend;
 Rdata = DateTime.Now;

 Reports = new List<Report>();
 }

 public GuestResponse() { }
}
```

3. Постройте и запустите стартовую страницу. Ничего не должно измениться.

### Изменение интерфейса

Будем предполагать, что по требованиям организаторов семинара каждый участник может выступить не более чем с двумя докладами. Перед тем как реализовывать работу с данными теперь требуется внести изменения в форму регистрации.

1. Откройте файл `Reg.aspx` в режиме разметки и добавьте после закрывающего тега `</div>` чекбокса два текстовых поля с надписями для ввода аннотации и темы докладов:

```
<div>
 Введите название доклада:
 <asp:TextBox ID="TextBoxTitle" runat="server"
Width="345px"></asp:TextBox>
</div>

<div>
 Введите аннотацию доклада:
 <asp:TextBox ID="TextBoxTextAnnot" runat="server"
Width="345px"></asp:TextBox>
</div>
```



```

<div>
Введите название доклада:
<asp:TextBox ID="TextBoxTitle2" runat="server"
Width="345px"></asp:TextBox>
</div>

<div>
Введите аннотацию доклада:
<asp:TextBox ID="TextBoxTextAnnot2" runat="server"
Width="345px"></asp:TextBox>
</div>

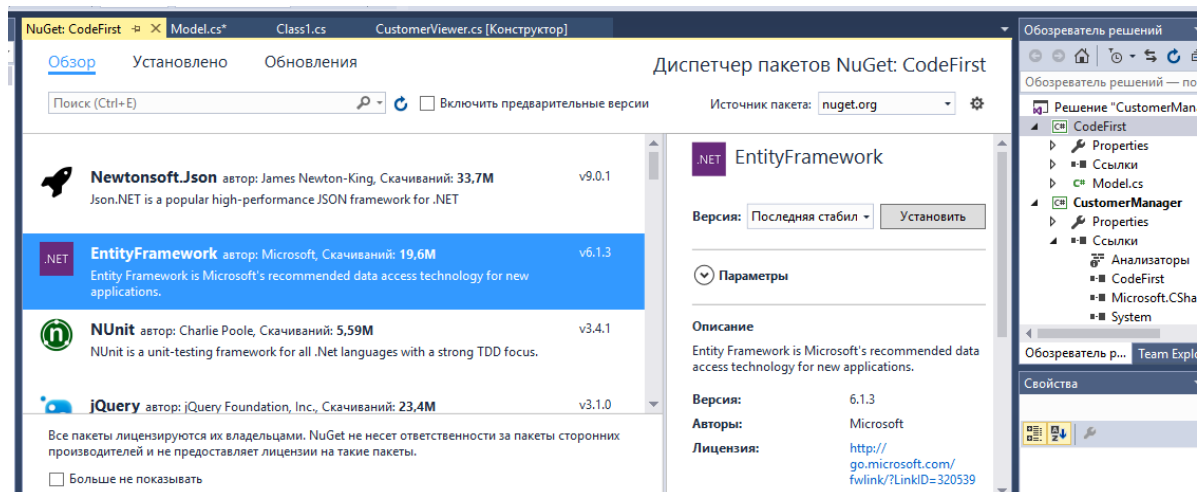
```

## Установка Entity Framework 6 в проект

Перед тем как использовать Entity Framework (EF) в проекте при подходе Code-First необходимо добавить ссылки на библиотеки EF.

Выполните следующие действия:

1. Для добавления поддержки EF с помощью **NuGet** (начиная с версии 4, библиотека EF входит в менеджер пакетов NuGet), выберите в окне **Solution Explorer** данный проект, щелкните по нему правой кнопкой мыши и выполните команду из контекстного меню **Manage Nuget Packages**.
2. В появившемся диалоговом окне на вкладке **Обзор** выберите последнюю версию **Entity Framework** и нажмите кнопку **Install** (Установить):



3. После этого появится окно с описанием лицензии на использование Entity Framework. Согласитесь с условиями, после чего NuGet установит в проект Entity Framework.
4. Добавьте ссылку на ключевое пространство имен в EF - `System.Data.Entity` в файле `GuestResponse.cs`:  

```
using System.Data.Entity;
```

## Класс контекста данных

Созданные классы **GuestResponse** и **Report** описывают структуру бизнес-модели, которая используется в приложении. Для того чтобы эти классы служили также для управления данными в базе данных, нужно использовать класс **контекста**.

EF имеет два базовых класса контекста: **ObjectContext** – этот класс является более общим классом контекста данных, и используется, начиная с самых ранних версий EF и **DbContext** – этот класс контекста данных появился в Entity Framework 4.1, и он обеспечивает поддержку подхода Code-First (ObjectContext также обеспечивает работу подхода Code-First, но он труднее в использовании). Далее будет использоваться DbContext.

1. Для создания класса контекста добавьте новый класс **SampleContext** в папку **App\_Code**.
2. В добавленном классе имя будущей базы данных укажите через вызов конструктора базового класса и укажите соответствующие таблицы (по традиции во множественном числе) базы данных с помощью свойств с типом DbSet:

```
using System.Data.Entity;

public class SampleContext : DbContext
{
 public SampleContext()
 : base("SeminarBD")
 {
 }
 public DbSet<GuestResponse> GuestResponses { get; set; }
 public DbSet<Report> Reports { get; set; }
}
```

Этот класс контекста представляет полный слой данных, который можно использовать в приложениях. Благодаря **DbContext**, можно запросить, изменить, удалить или вставить значения в базу данных. Обратите внимание на использование конструктора в этом классе с вызовом конструктора базового класса **DbContext** и передачей ему строкового параметра.

В этом параметре указывается либо имя базы данных, либо строка подключения к базе данных. В данном случае указывается явно имя базы данных, т.к. по умолчанию, при генерации базы данных EF использует имя приложения и контекста данных, которое использовать неудобно.

3. Постройте приложение.

### Установка подключения к базе данных

Для установки подключения обычно используется файл конфигурации приложения, в проектах для web приложений это файл Web.config.

1. Откройте файл конфигурации **Web.config** и после закрывающего тега `</configSections>` добавьте секцию `connectionStrings`, в которой с помощью элемента укажите подключение:

```

<connectionStrings>
 <add name="SeminarBD" connectionString="data
source=(localdb)\MSSQLLocalDB;Initial
Catalog=RSVP.SeminarBD.mdf;Integrated Security=True;"
providerName="System.Data.SqlClient" />
</connectionStrings>

```

В конструкторе класса контекста **SampleContext** была передана в качестве названия подключения строка "SeminarBD", поэтому данное название и указывается в атрибуте `name="SeminarBD "`.

## Работа с данными

На текущий момент есть классы данных и класс контекста, необходимые для работы с базой данных, которая еще не существует. Простое создание этих классов не приводит к автоматическому созданию (изменению) базы данных при запуске приложения.

Чтобы создать базу данных, нужно добавить код работы с данными с помощью EF, написать код вставки данных в таблицы.

1. Откройте файл кода Reg.aspx.cs.
2. В обработчике события загрузки формы после создания объекта участника семинара `rsvp` в случае включения флажка создайте первый доклад и добавьте его в коллекцию докладов участника:

```

if (CheckBoxYN.Checked)
{
 Report report1 = new Report(TextBoxTitle.Text,
 TextBoxTextAnnot.Text);
 rsvp.Reports.Add(report1);
}

```

3. Далее в случае заполнения полей для второго доклада создайте второй доклад и также добавьте его в коллекцию:

```

if (TextBoxTitle2.Text != "" || TextBoxTextAnnot2.Text != "")
{
 Report report2 = new Report(TextBoxTitle2.Text,
 TextBoxTextAnnot2.Text);
 rsvp.Reports.Add(report2);
}

```

4. Далее создайте экземпляр класса контекста базы данных, после этого добавьте методом `Add()` созданного участника семинара в базу данных и методом `SaveChanges()` сохраните изменения в базе данных:

```

try
{
 SampleContext context = new SampleContext();
 context.GuestResponses.Add(rsvp);
 context.SaveChanges();
}
catch (Exception ex)
{
 Response.Redirect("Ошибка " + ex.Message);
}

```

Данный код создает объект контекста **SampleContext** и использует его, чтобы добавить новые данные в таблицу **GuestResponses**. Вызов метода `SaveChanges()` сохраняет изменения в базе данных, а при первой вставке данных вызов `SaveChanges()` создаст базу данных.

5. Постройте и запустите приложение.
6. Заполните поля регистрации и введите данные о двух докладах. Нажмите кнопку для отправки данных.

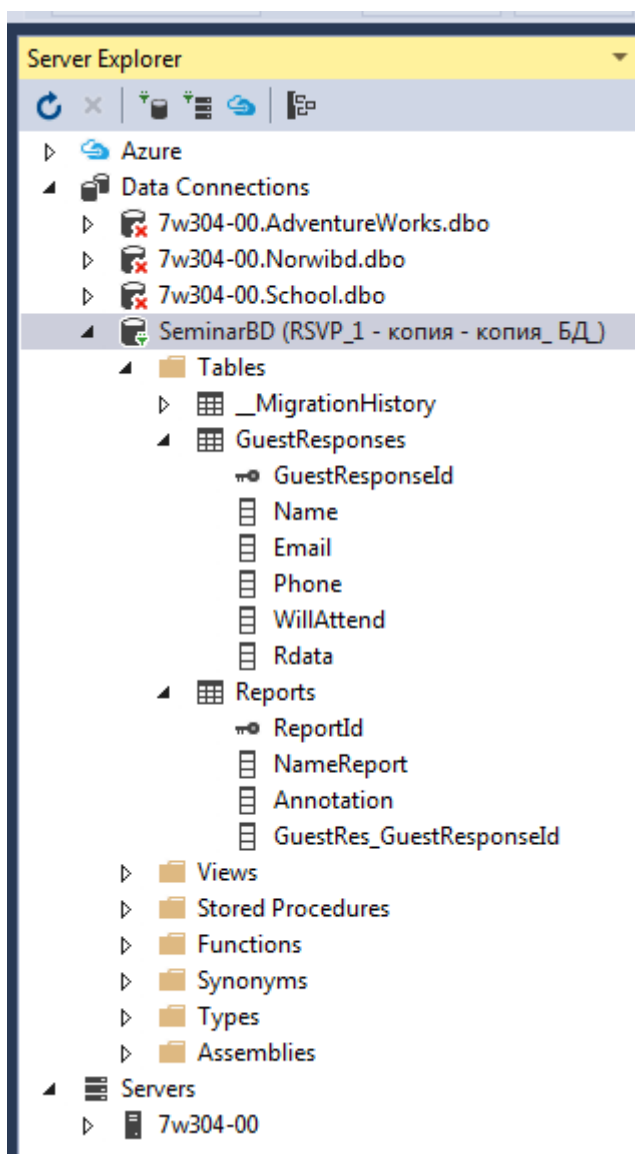
Обратите внимание, что при первом добавлении данных возникает заметная задержка, потому что в данном моменте и создается база данных.

7. Введите данные о втором участнике. Обратите внимание, что данные добавились в этом случае быстро, т.к. они вставляются в уже существующую базу данных.

### Соглашения о конфигурации сгенерированной базы данных

База данных создана. Далее вы просмотрите ее структуру.

1. В окне обозревателя серверов **Server Explorer** проверьте подключение:



2. Проверьте, что в базе данных были созданы две таблицы, имеющие соответствующие столбцы.

Согласно соглашениям EF по именованию таблиц, их название генерируется в множественном числе, основываясь на правилах английского языка. Названия столбцов соответствуют названиям свойств сущностных классов.

3. В контекстном меню таблицы **GuestResponses** выберите пункт *Open Table Defenition* (Открыть определение таблицы) и посмотрите, что типы данных .NET преобразованы в типы данных T-SQL: Int32 в INT, String в NVARCHAR(max), bool в bit:

Name	Data Type	Allow Nulls	Default
GuestResponseId	int	<input type="checkbox"/>	
Name	nvarchar(MAX)	<input checked="" type="checkbox"/>	
Email	nvarchar(MAX)	<input checked="" type="checkbox"/>	
Phone	nvarchar(MAX)	<input checked="" type="checkbox"/>	
WillAttend	bit	<input checked="" type="checkbox"/>	
Rdata	datetime	<input type="checkbox"/>	

```
CREATE TABLE [dbo].[GuestResponses] (
 [GuestResponseId] INT IDENTITY (1, 1) NOT NULL,
 [Name] NVARCHAR (MAX) NULL,
 [Email] NVARCHAR (MAX) NULL,
 [Phone] NVARCHAR (MAX) NULL,
 [WillAttend] BIT NULL,
 [Rdata] DATETIME NOT NULL,
 CONSTRAINT [PK_dbo.GuestResponses] PRIMARY KEY CLUSTERED ([GuestResponseId] ASC)
```

4. Изучите структуру таблиц, обратите внимание на некоторые соглашения о проецировании сущностных классов, которые используются в Code-First.

Например, свойства `GuestResponseId` и `ReportId` в сущностном классе EF преобразовал в первичные ключи в базе данных (EF автоматически ищет подстроку “Id” в именах свойств модели с помощью механизма рефлексии). Эти поля используют автоинкремент с помощью инструкции `IDENTITY (1,1)` и не могут иметь значение NULL.

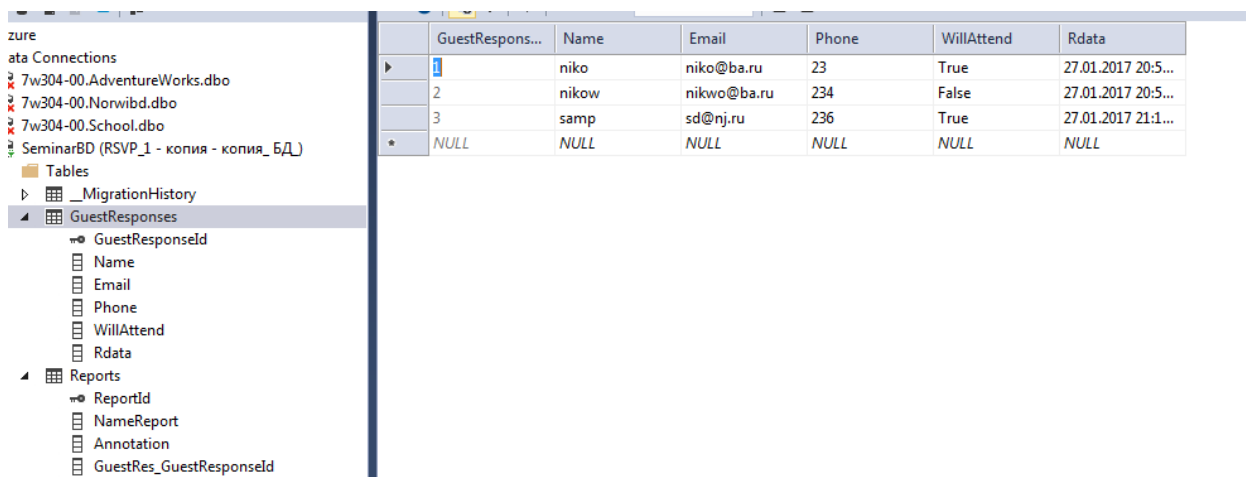
5. Откройте структуру таблицы **Reports** и проверьте, что EF создал внешний ключ `GuestRes_GuestResponseId`, ссылающийся на таблицу **GuestResponses**.

Это было достигнуто за счет того, что мы указали виртуальное свойство в классе **GuestResponse** ссылающееся на класс **Report** и добавили обратное свойство в классе **Report**, ссылающееся на **GuestResponse**. Благодаря тому, что тип виртуального свойства унаследован от `IEnumerable<T>`, EF догадался, что нужно реализовать отношение “один ко многим” (one-to-many) между этими таблицами. При подходе к именованию внешнего ключа EF использовал следующее соглашение:

[Имя навигационного свойства]\_[Имя первичного ключа родительской таблицы]

Обратите внимание также на то, что EF сгенерировал еще одну таблицу с названием **\_\_MigrationHistory**. Эта таблица хранит различные версии изменения структуры базы данных. В частности, в поле Model эта таблица хранит метаданные модели, представленные в виде двоичного объекта BLOB. Если позже вы измените модель в своем коде, EF вставит в эту таблицу новую запись, с метаданными новой модели.

6. С помощью команды контекстного меню *Показать таблицу данных* таблицы **GuestResponses** просмотрите ее содержимое – вы увидите те данные, которые и были вами добавлены, например:



The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'Tables' folder is expanded, showing the 'GuestResponses' table. On the right, the table's data is displayed in a grid. The table has columns: GuestResponses..., Name, Email, Phone, WillAttend, and Rdata. The data rows are as follows:

GuestResponses...	Name	Email	Phone	WillAttend	Rdata
1	niko	niko@ba.ru	23	True	27.01.2017 20:5...
2	nikow	nikwo@ba.ru	234	False	27.01.2017 20:5...
3	samp	sd@nj.ru	236	True	27.01.2017 21:1...
*	NULL	NULL	NULL	NULL	NULL

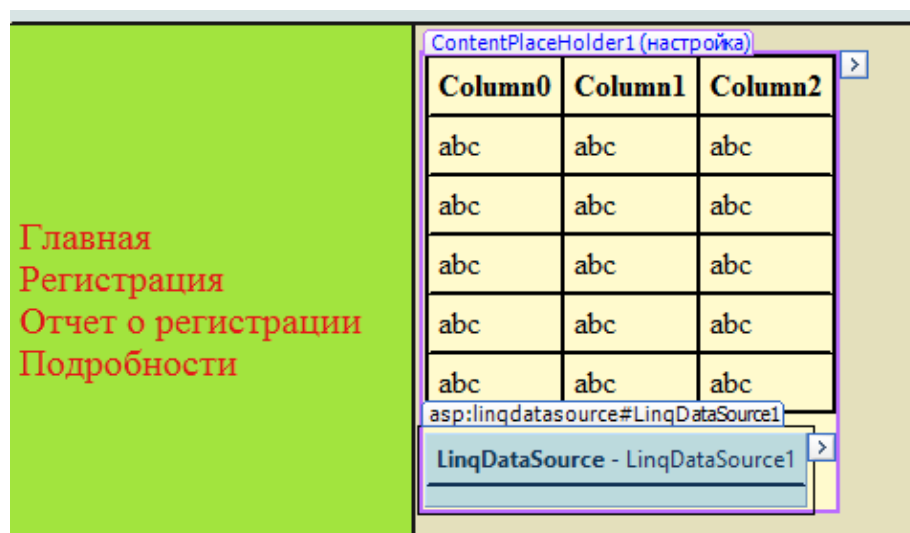
## Упражнение 2. Отображение данных

В этом упражнении вы реализуете вывод данных из базы в подходящий для этого элемент управления: **GridView**.

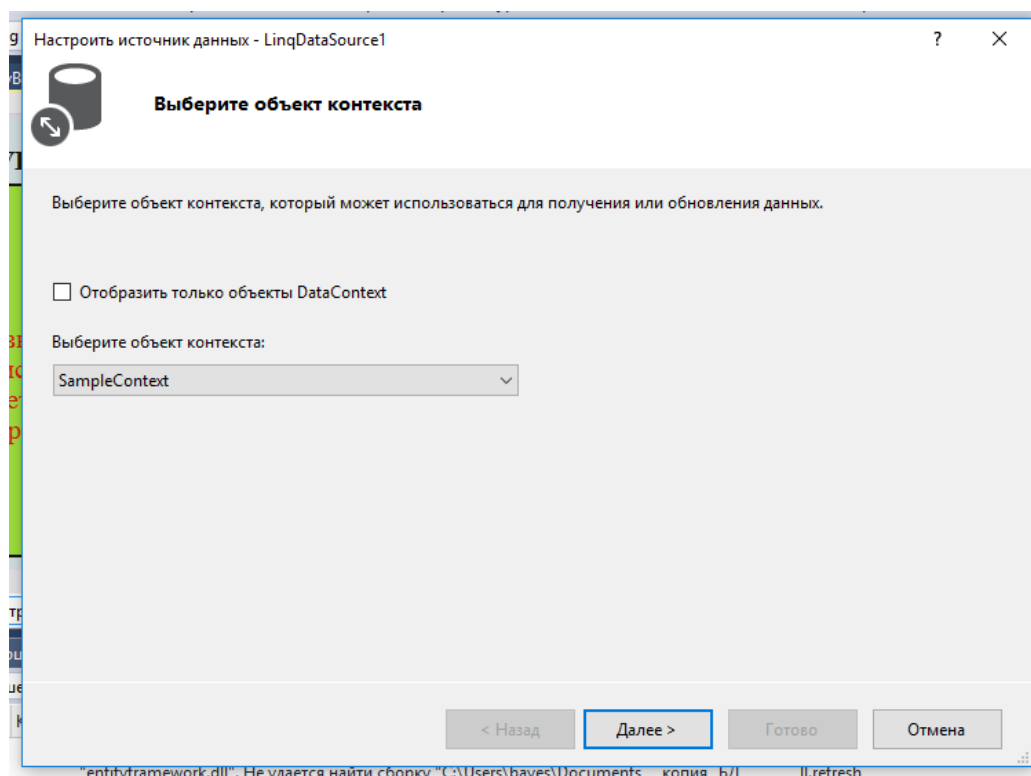
1. Добавьте в проект сайта новую форму SummaryBD.aspx как страницу содержимого.
2. В элемент Content добавьте заголовок второго уровня:

<h2>Список участников</h2>

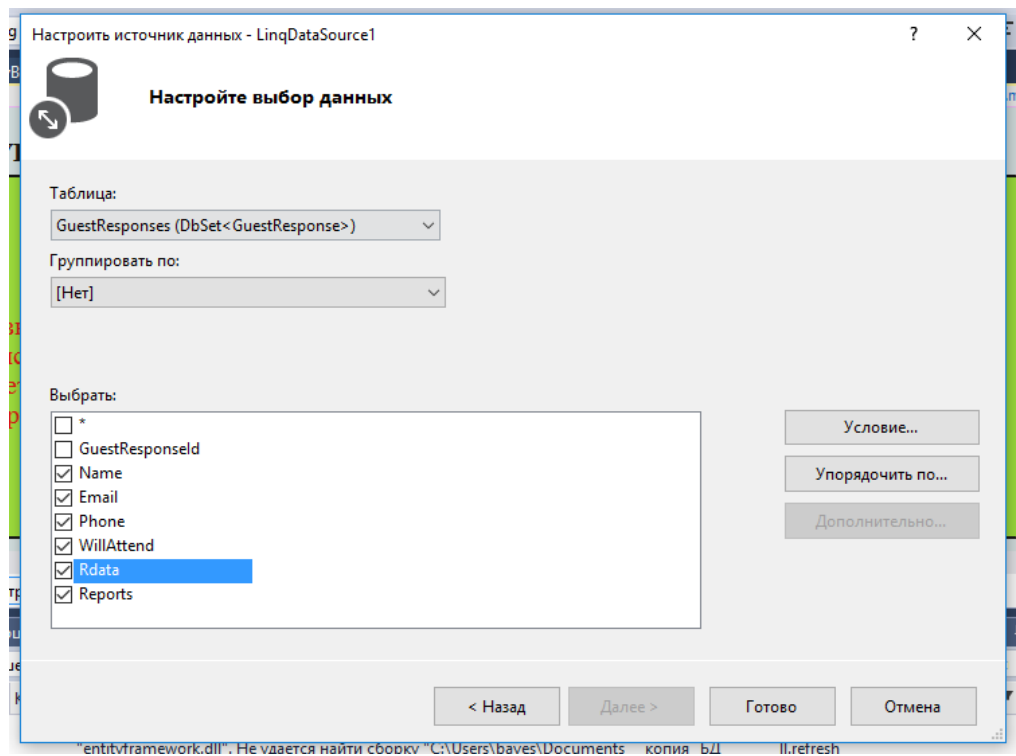
3. После заголовка перенесите на форму из панели инструментов компоненты **GridView** и **LinqDataSource**



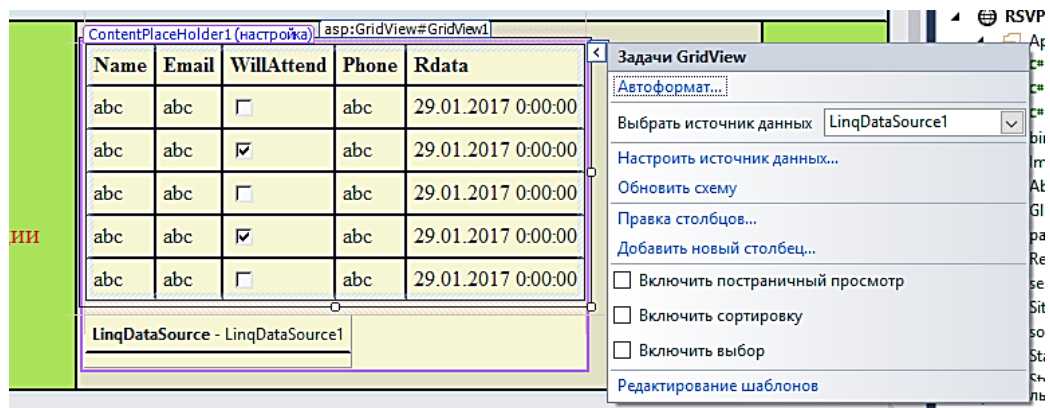
4. Настройте источник данных — компонент **LinqDataSource**, для этого кликните смарт-тег элемента и выберите соответствующую команду.
5. В первом окне настройки источника данных выберите объект контекста — **SampleContext** и нажмите кнопку **Далее**:



6. Во втором окне выберите нужные для отображение поля как на рисунке и нажмите **Готово**:

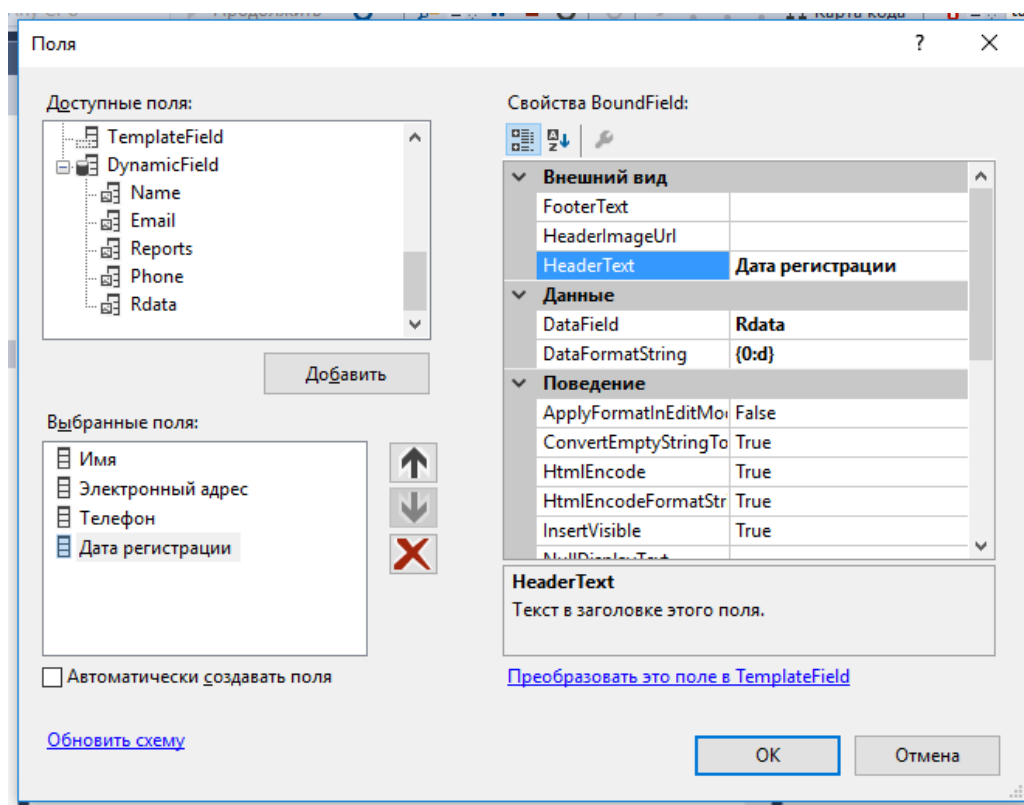


7. Настройте источник данных для элемента **GridView**, для этого выделите его, кликните на смарт-тег и в списке выбора источника данных укажите LinqDataSource1:



8. В меню смарт-тега выберите команду **Правка столбцов** и в открывшемся окне с помощью окна свойств укажите имена столбцов, а для поля даты еще и формат отображения (при необходимости можно настроить и очередность отображения заголовков):





9. Добавьте в меню стартовой страницы ссылку на новую страницу:

```
<asp:MenuItem NavigateUrl="~/SummaryBD.aspx" Text="Список участников"/>
```

и добавьте ссылку в колонтитул:

```
Список участников
```

10. Постройте и запустите приложение. Перейдите на страницу списка участников. Компонент должен отобразить данные примерно, как на рисунке:

RSVP				
<a href="#">Главная</a> <a href="#">Регистрация</a> <a href="#">Отчет о регистрации</a> <a href="#">Список участников</a> <a href="#">Подробности</a>	<b>Список участников</b>			
	<u>Имя</u>	<u>Электронный адрес</u>	<u>Телефон</u>	<u>Дата регистрации</u>
	nilo	niko@bayes.ru	567	29.01.2017
	bas	ba@mk.ru	908	29.01.2017
	poil	poli@mh.ru	45	29.01.2017
	niko	niko@bayes.ru	45	29.01.2017
<a href="#">Главная</a>   <a href="#">Регистрация</a>   <a href="#">Отчет о регистрации</a>   <a href="#">Список участников</a>   <a href="#">Подробности</a>				
Copyright © 2017 Число посещений: 1				

