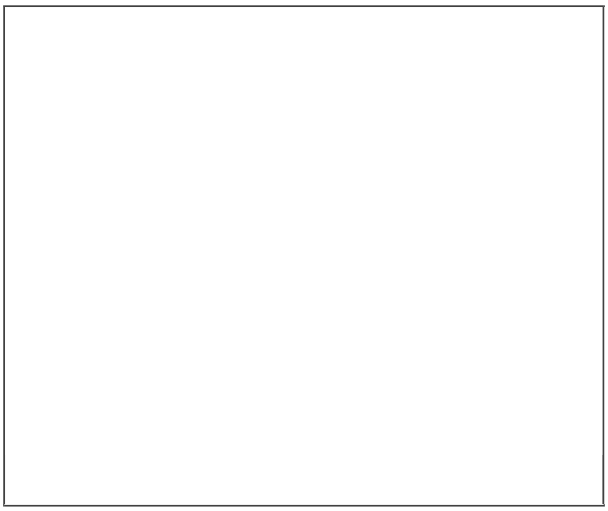
Список товаров



Исходный код проекта

Теперь, когда имеются модель данных, база данных и хранилище для интернет-магазина, можно приступать к построению функциональности, предлагаемой пользователю. Мы добавили в папку Pages новый файл веб-формы по имени Listing.aspx, содержимое которого показано в примере ниже:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Listing.aspx.cs" Inh

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>GameStore</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <%
                foreach (GameStore.Models.Game game in GetGames())
                {
                    Response.Write(String.Format(@"
                        <div class='item'>
                            <h3>{0}</h3>
                            {1}
                            <h4>{2:c}</h4>
                        </div>",
                        game.Name, game.Description, game.Price));
                }
            %>
        </div>
    </form>
</body>
</html>
```

Эта веб-форма содержит фрагмент кода, который получает набор объектов `Game`, вызывая метод `GetGames()` класса отделенного кода и генерируя для каждого объекта ряд базовых `HTML`-элементов.

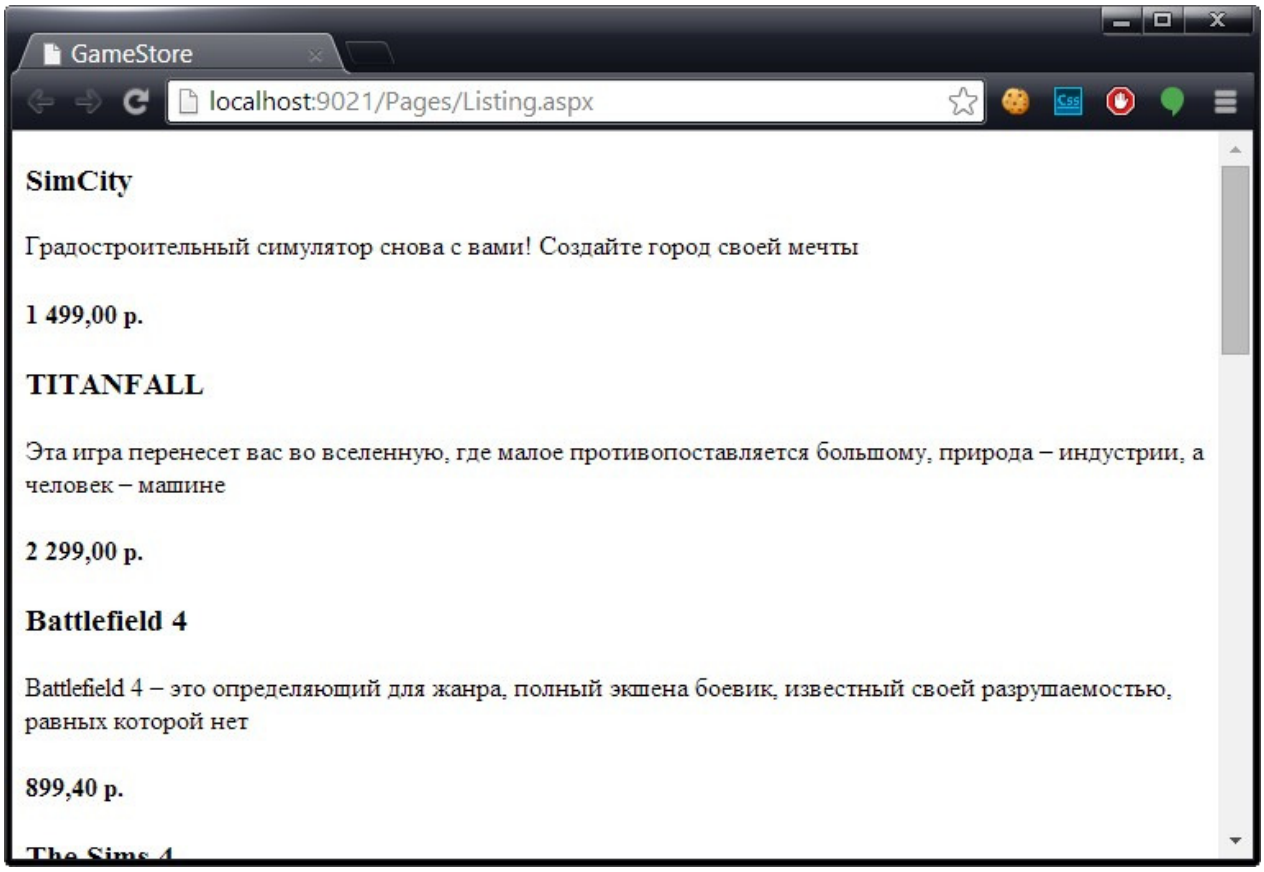
Обратите внимание, что свойство `Price` преобразуется в строку с применением метода `ToString("c")`, который представляет числовые значения в денежной форме согласно параметрам культуры, действующим на сервере `IIS` и обычно получаемым из конфигурации серверной операционной системы. Например, если сервер настроен на культуру `en-US`, то `(1952.3) ToString("c")` возвратит `$1,952.30`, а если на культуру `ru-RU`, тот же самый вызов даст в результате `1952.30 р.` Параметр культуры для сервера можно изменить, добавив в узел `<system.web>` файла `Web.config` раздел следующего вида:

```
<globalization culture="ru-RU" uiCulture="ru-RU" />
```

Код метода GetGames() можно видеть в примере ниже. В этом листинге представлено содержимое файла отделенного кода Listing.aspx.cs, созданного Visual Studio для веб-формы, к которому добавлен код использования класса Repository внутрь метода GetGames():

```
public partial class Listing : System.Web.UI.Page
{
        private Repository repository = new Repository();

        protected IEnumerable<Game> GetGames()
        {
            return repository.Games;
        }

        protected void Page_Load(object sender, EventArgs e)
        {

        }
}
```

Все, что необходимо сделать для переноса содержимого базы данных в класс отделенного кода — это создать новый экземпляр класса Repository и прочитать его свойство Games. Чтобы протестировать новую функциональность, щелкните правой кнопкой мыши на элементе \Pages\Listing.aspx в окне Solution Explorer и выберите в контекстном меню пункт Set As Start Page (Установить как стартовую страницу) Выберите пункт Start Debugging (Начать отладку) в меню Debug (Отладка). Среда Visual Studio запустит приложение, создаст новый экземпляр выбранного браузера и перейдет на URL, который отобразит страницу. Результаты представлены на рисунке ниже:



Итак, вы видели, насколько легко было создать базу данных, ассоциировать ее с классом модели данных и отобразить данные из базы для пользователя с помощью веб-формы. Хотя веб-форма выглядит не особо привлекательно, мы с весьма небольшими усилиями смогли получить базовую структуру приложения — причем эта скорость и элегантность оставляют нам больше времени на то, чтобы сосредоточиться на функциональности приложения. Мы признаем, что в настоящий момент только читаем содержимое базы данных, но по мере построения приложения `GameStore` мы будем добавлять поддержку других видов операций над данными.

Конечно, чтобы добраться до этой точки, мы вынуждены были пропустить множество деталей о том, как работает инфраструктура `Entity Framework`, и большое число доступных вариантов конфигурации. Нам нравится инфраструктура `Entity Framework` и мы рекомендуем выделить некоторое время на ее освоение.

Добавление разбиения на страницы

На рисунке выше можно заметить, что все товары из базы данных отображаются в виде длинного списка на одной странице. В этом разделе мы добавим поддержку разбиения на страницы, чтобы выводить небольшое количество товаров одновременно и позволить пользователю постранично просматривать весь каталог. Это необходимо сделать в два этапа — сначала должна быть добавлена поддержка для отображения подмножества товаров, а затем предусмотрены ссылки, которые пользователь сможет нажимать для перехода от одного подмножества товаров к другому.

Отображение страницы товаров

Отображать фиксированное количество товаров на странице можно за счет

применения запросов LINQ к коллекции объектов Game, извлекаемых из базы данных. Для этого необходимо знать, сколько товаров должно отображаться на странице, и какую страницу пользователь желает просмотреть.

В примере ниже показано, как это делается (это измененный файл \Pages \Listing.aspx.cs):

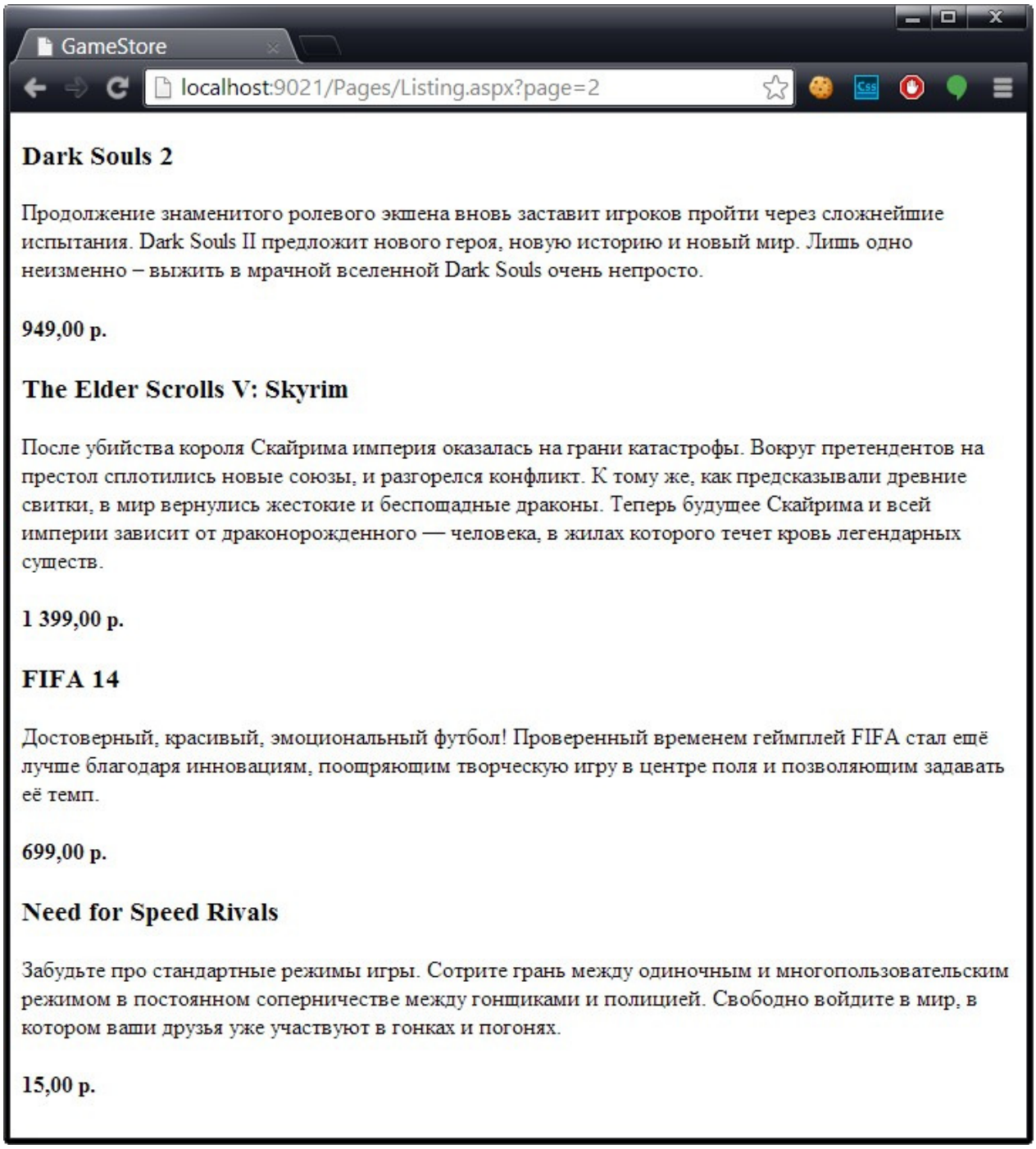
```
public partial class Listing : System.Web.UI.Page
{
        private Repository repository = new Repository();
        private int pageSize = 4;

        protected int CurrentPage
        {
            get {
                int page;
                return int.TryParse(Request.QueryString["page"], out page) ? p
            }
        }

        protected IEnumerable<Game> GetGames()
        {
            return repository.Games
                .OrderBy(g => g.GameId)
                .Skip((CurrentPage - 1) * pageSize)
                .Take(pageSize);
        }

        protected void Page_Load(object sender, EventArgs e)
        {

        }
}
```

Размер страницы задан равным четырем товарам и устанавливается через поле PageSize. Для представления текущей страницы создано свойство CurrentPage, которое использует коллекцию Request.QueryString, определенную базовым классом, для выяснения, присутствует ли значение page в запрошенном URL. Свойство Request предоставляет доступ к деталям текущего запроса. Таким образом, например, если веб-форма обрабатывается для обслуживания URL следующего вида:

```
http://localhost:9021/Pages/Listing.aspx?page=2
```

то коллекция Request.QueryString будет содержать ключ page со значением 2. Значения возвращаются из коллекции Request.QueryString в форме строк, поэтому с помощью метода int.TryParse() производится попытка преобразования строки в числовое значение. По умолчанию принято значение 1, которое указывает на отображение первой страницы товаров, когда значение page в строке запроса не задано или преобразовать его не удалось.

Значения CurrentPage и PageSize позволяют выбирать требуемые объекты Game из хранилища. Метод OrderBy() из LINQ обеспечивает обработку объектов Game в одном и том же порядке, метод Skip() дает возможность проигнорировать объекты Game, встречающиеся перед желаемой страницей, а метод Take() позволяет выбрать нужное количество объектов Game для отображения пользователю. Чтобы протестировать этот код, запустите приложение и несколько раз вручную перейдите на URL разных страниц. На рисунке ниже показан результат перехода на вторую страницу товаров:



Как видно на рисунке, на странице отображаются четыре товара. Поэкспериментировав некоторое время с этим средством, вы заметите, что можно запрашивать страницы с номерами, выходящими за пределы количества товаров в базе данных, получая в результате пустые страницы. Например, проблему может продемонстрировать следующий URL:

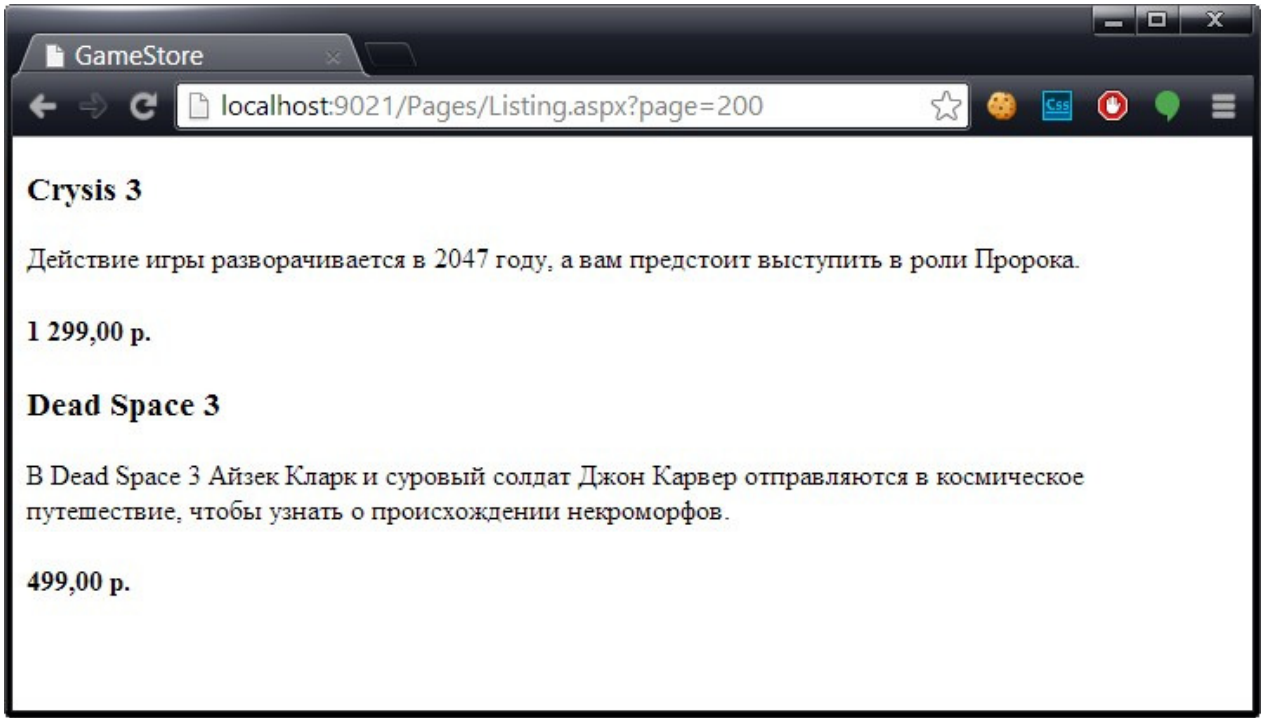
```
http://localhost:9021/Pages/Listing.aspx?page=200
```

При обработке запроса обычно полезно учитывать диапазон допустимых значений, особенно когда пользователь имеет возможность легко переходить на страницу напрямую. Мы могли бы в такой ситуации вывести пользователю сообщение об ошибке или перенаправить браузер на URL для корректной страницы.

Мы собираемся воспользоваться другим подходом, который предусматривает отображение последней страницы товаров. В примере ниже показано обновление `GetGames()` в файле `\Pages\Listing.aspx.cs` для обнаружения и обработки данной проблемы:

```
using System;
using System.Collections.Generic;
using GameStore.Models;
using GameStore.Models.Repository;
using System.Linq;

namespace GameStore.Pages
{
    public partial class Listing : System.Web.UI.Page
    {
        private Repository repository = new Repository();
        private int pageSize = 4;

        protected int CurrentPage
        {
            get {
                int page;
                page = int.TryParse(Request.QueryString["page"], out page) ? p
                return page > MaxPage ? MaxPage : page;
            }
        }

        // Новое свойство, возвращающее наибольший номер допустимой ст
        protected int MaxPage
        {
            get {
                return (int)Math.Ceiling((decimal)repository.Games.Count() / p
            }
        }

        protected IEnumerable<Game> GetGames()
        {
            return repository.Games
                .OrderBy(g => g.GameId)
                .Skip((CurrentPage - 1) * pageSize)
                .Take(pageSize);
        }

        protected void Page_Load(object sender, EventArgs e)
        {

        }
```

Свойство MaxPage возвращает наибольший номер страницы, для которой могут быть отображены товары. Мы применяем это значение в средстве get свойства CurrentPage, в результате чего запрос, к примеру, страницы 200, эквивалентен запросу последней допустимой страницы (в рассматриваемом примере это страница 3, т.к. в базе хранятся 10 элементов, а на одной странице отображаются четыре элемента). На рисунке ниже демонстрируется результат запрашивания страницы 200:



Добавление ссылок на страницы

Существует много способов предоставления ссылок на страницы для пользователя но мы собираемся следовать подходу с отображением HTML-элемента <a> для каждой доступной страницы. Щелчок на таких ссылках приводит к запросу страниц, которые они представляют.

Наш класс отделенного кода уже обеспечивает всю необходимую информацию, поэтому мы можем сгенерировать требуемые ссылки с помощью фрагмента кода в файле веб-формы \Pages\Listing.aspx, как показано в примере ниже:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Listing.aspx.cs" Inh

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>GameStore</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <%
                foreach (GameStore.Models.Game game in GetGames())
                {
                    Response.Write(String.Format(@"
                        <div class='item'>
                            <h3>{0}</h3>
                            {1}
                            <h4>{2:c}</h4>
                        </div>",
                        game.Name, game.Description, game.Price));
                }
            %>
        </div>
    </form>
    <div>
        <%
            for (int i = 1; i <= MaxPage; i++)
            {
                Response.Write(
                    String.Format("<a href='/Pages/Listing.aspx?page={0}' {1}>
                        i, i == CurrentPage ? "class='selected'" : "", i));
            }
        %>
    </div>
</body>
</html>
```

В коде используется цикл `for` для генерации элементов `<a>`, представляющих каждую страницу, для которой может быть отображен контент. Фрагмент кода генерирует примерно такие ссылки:

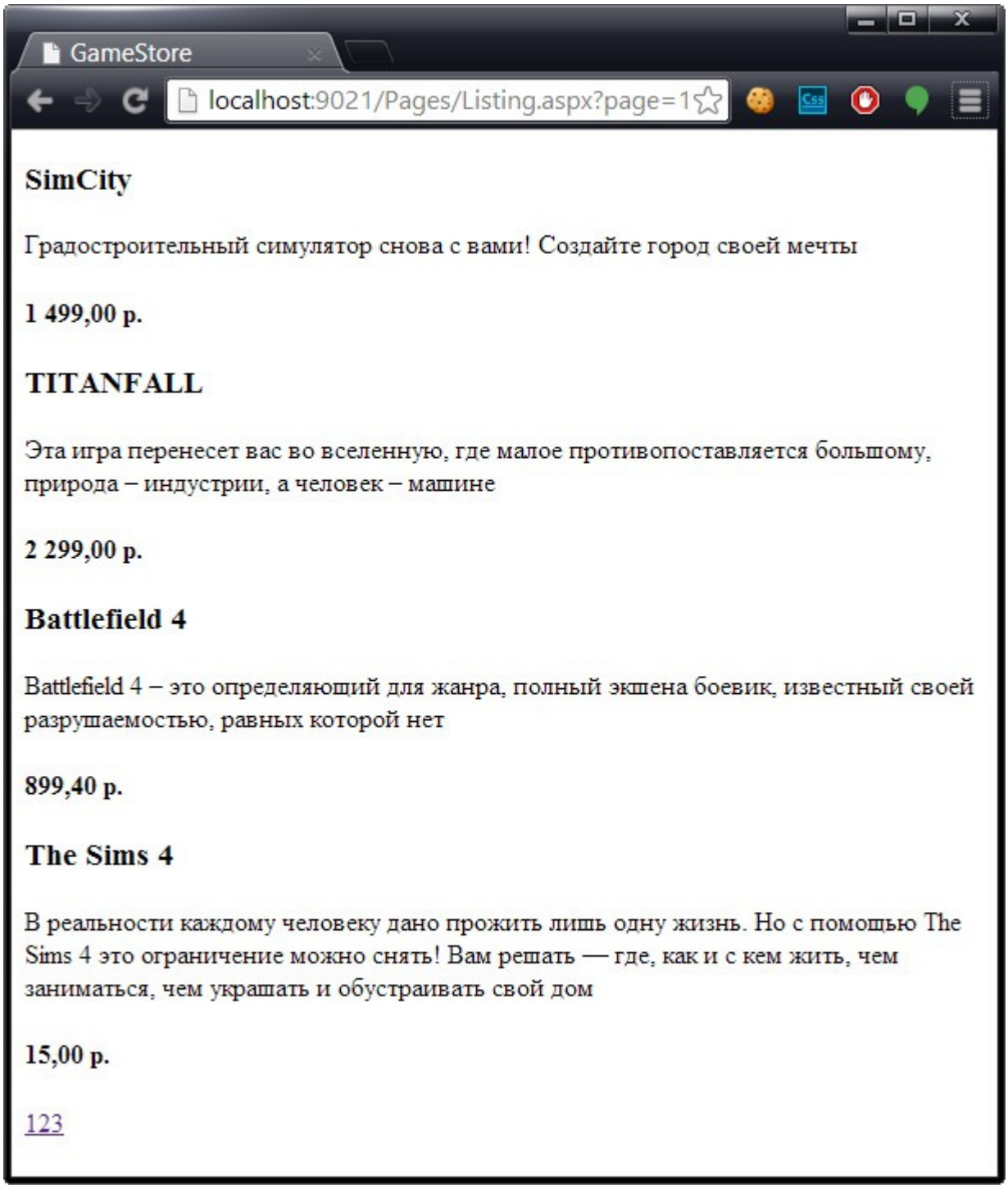
```
<a href='Pages/Listing.aspx?page=2'>2</a>
```

для каждой страницы и ссылку следующего вида для страницы, отображаемой в текущий момент:

```
<a href='Pages/Listing.aspx?page=1' class='selected'>1</a>
```

В следующем разделе мы будем использовать для стилизации элементов класс `selected`, создавая визуальную подсказку о номере отображаемой в текущий момент страницы. Генерация ссылок подобным образом требует аккуратной работы со средствами форматирования строк `C#`.

Запустив приложение, можно увидеть разбиение на страницы в действии и протестировать его. Поскольку в базе данных хранится достаточное количество товаров для их отображения на трех страницах, будут видны три страничных ссылки, что и продемонстрировано на рисунке ниже:



Стилизация веб-формы со списком товаров

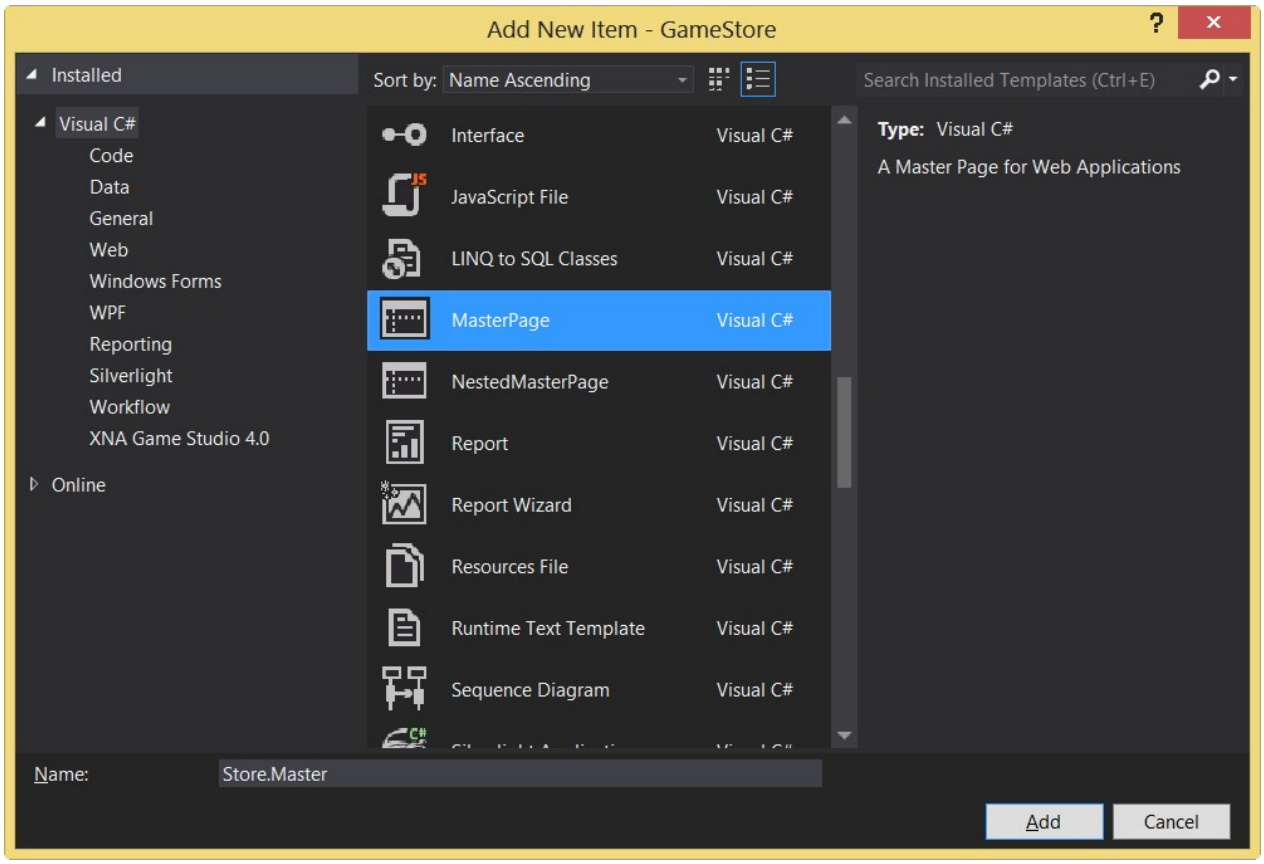
Итак, мы построили базовую структуру приложения с требуемой функциональностью — имеется база данных, список товаров для отображения пользователю и ссылки для навигации по каталогу. Тем не менее, поскольку мы были сосредоточены на функциональности, а не на внешнем виде, возникла ситуация, когда дизайн контента настолько скуден, что сводит на нет все технические достижения. В настоящем разделе мы займемся визуальным аспектом приложения. Мы собираемся реализовать классическую двухколоночную компоновку.

В этом разделе стили CSS будут добавляться без сопутствующих объяснений,

что они означают. Подробное описание CSS3 можно найти в разделе Учебник CSS3.

Создание мастер-страницы

Мы можем сформировать контент, который будет совместно использоваться множеством веб-форм, создав для этого мастер-страницу, действующую в качестве шаблона, куда вставляется контент, специфичный для конкретной страницы. Давайте добавим в приложение мастер-страницу, щелкнув правой кнопкой мыши на папке Pages в окне Solution Explorer и выбрав в контекстном меню пункт Add New Item. В открывшемся диалоговом окне Add New Item (Добавление нового элемента) отыщите шаблон Master Page (Мастер-страница), как показано на рисунке ниже, укажите в качестве имени Store.Master и щелкните на кнопке Add (Добавить):



В примере ниже представлен начальный контент, сгенерированный Visual Studio для новой мастер-страницы. Как видите, мастер-страница имеет такую же структуру, как и обычная веб-форма:

```
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Store.master.cs" I

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <asp:ContentPlaceHolder ID="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">


        </asp:ContentPlaceHolder>
    </div>
    </form>
</body>
</html>
```

Директива Master внутри фрагмента кода в начале страницы сообщает ASP.NET Framework о том, что это мастер-страница. Элементы **ContentPlaceHolder** представляют собой механизм, применяемый для вставки контента из веб-форм в шаблон. Элементы, дескрипторы которых начинаются с asp:, обозначают элемент управления Web Forms — многократно используемый пакет кода и разметки. Элемент управления ContentPlaceHolder извлекает контент из веб-форм, которые пользуются данной мастер-страницей. В стандартном контенте, показанном в примере, присутствуют элементы управления ContentPlaceHolder, которые позволяют вставлять контент в разделы head и body мастер-страницы.

Настройка мастер-страницы

От мастер-страницы будет мало толку, если ее не настроить в соответствие с приложением. Мы должны внести в мастер-страницу несколько изменений, которые показаны в примере ниже:

```
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Store.master.cs" I

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>GameStore</title>
    <link rel="stylesheet" href="/Content/Styles.css" />
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <div id="header">
            <div class="title">GameStore - магазин компьютерных игр</div>
        </div>
        <div id="categories">
            Мы добавим сюда контент позже
        </div>
        <div>
            <asp:ContentPlaceHolder ID="bodyContent" runat="server" />
        </div>
    </div>
    </form>
</body>
</html>
```

Мы добавили ряд HTML-элементов для создания заголовка и других необходимых структурных единиц. Мы удалили заполнитель из раздела `head` мастер-страницы и добавили элемент `link`, который будет импортировать стили CSS из файла таблицы стилей под названием `Styles.css`, хранящегося в папке `Content`.

В примере ниже приведено содержимое файла `Styles.css`, который был создан за счет щелчка правой кнопкой мыши на папке `Content`, выбора в контекстном меню пункта `Add New Item` и указания шаблона `Style Sheet` (Таблица стилей).

```
body {
    font-family: Cambria, Georgia, "Times New Roman";
    margin: 0;
}
div#header div.title, div.item h3, div.item h4, div.pager a {
    font: bold 1em "Arial Narrow", "Franklin Gothic Medium", Arial;
}
div#header {
    background-color: #444;
    border-bottom: 2px solid #111;
    color: white;
}
div#header div.title {
    font-size: 2em;
    padding: .6em;
}
div#content {
    border-left: 2px solid gray;
    margin-left: 9em;
    padding: 1em;
}
div#categories {
    float: left;
    width: 8em;
    padding: .3em;
}

div.item {
    border-top: 1px dotted gray;
    padding-top: .7em;
    margin-bottom: .7em;
}
div.item:first-child {
    border-top:none;
    padding-top: 0;
}
div.item h3 {
    font-size: 1.3em;
    margin: 0 0 .25em 0;
}
div.item h4 {
    font-size: 1.1em;
```

Применение мастер-страницы

Необходимо сообщить ASP.NET Framework о том, что веб-форма должна использовать мастер-страницу. Это требует внесения двух изменений – добавления атрибута MasterPageFile в директиву Page и переделки контента с целью удаления HTML-элементов, которые определены на мастер-странице. Ниже демонстрируется применение мастер-страницы к файлу \Pages\Listing.aspx:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Listing.aspx.cs" Inh
    MasterPageFile="~/Pages/Store.Master" %>

<asp:Content ContentPlaceHolderID="bodyContent" runat="server">
    <div id="content">
        <%
            foreach (GameStore.Models.Game game in GetGames())
            {
                Response.Write(String.Format(@"
                        <div class='item'>
                            <h3>{0}</h3>
                            {1}
                            <h4>{2:c}</h4>
                        </div>",
                    game.Name, game.Description, game.Price));
            }
        %>
    </div>
    <div class="pager">
        <%
            for (int i = 1; i <= MaxPage; i++)
            {
                Response.Write(
                    String.Format("<a href='/Pages/Listing.aspx?page={0}' {1}>
                        i, i == CurrentPage ? "class='selected'" : "", i));
            }
        %>
    </div>
</asp:Content>
```

Нам не нужно определять элементы html, head и body внутри веб-формы, т.к. они уже определены на мастер-странице, указанной в качестве значения атрибута MasterPageFile.

Контент, который необходимо вставить в мастер-страницу, содержится внутри

элемента управления **Content**. Он представляет собой аналог элемента управления ContentPlaceHolder на мастер-странице и вы заметите, что значения атрибута ContentPlaceHolderID элементов управления Content и ContentPlaceHolder совпадают — именно так среде ASP.NET Framework указывается место в мастер-странице, где должен быть вставлен контент веб-формы.

Тестирование мастер-страницы

Чтобы протестировать мастер-страницу, следует запустить приложение. Когда среда ASP.NET Framework получает запрос файла Listing.aspx, она обнаруживает атрибут MasterPageFile в директиве Page и применяет файл Store.Master для генерации базовой структуры HTML-ответа для браузера. Каждый раз, когда среда ASP.NET Framework находит элемент управления ContentPlaceHolder на мастер-странице, она ищет в файле веб-формы элемент управления Content с соответствующим значением атрибута ContentPlaceHolderID и добавляет контент этого элемента управления в вывод.

На рисунке ниже можно видеть, что использование мастер-страницы и стилей CSS позволяет улучшить внешний вид приложения GameStore:

Хотя нам еще осталось сгенерировать список категорий (это будет сделано позже), мы создали согласованный шаблон, который может использоваться по всему приложению. Полученный шаблон упрощает пользователям работу со страничными ссылками и позволяет видеть номер страницы товаров, отображаемой в текущий момент.

Alexandr Erohin ✯ alexerohinzzz@gmail.com © 2011 - 2016