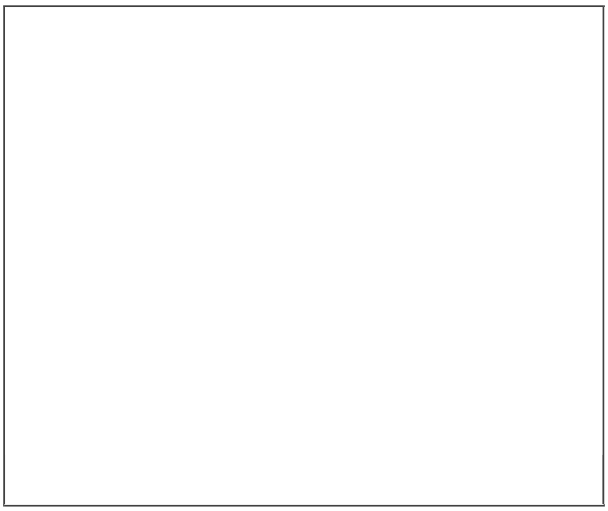
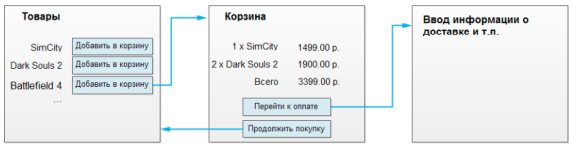
Корзина покупок



Исходный код проекта

Разработка приложения движется успешно, но мы не сможем продавать товары до тех пор, пока не будет реализована корзина для покупок. В этом разделе мы создадим корзину для покупок, как показано на рисунке ниже. Это должно выглядеть знакомо всем, кто приобретал хоть что-нибудь в онлайновых магазинах.



Кнопка добавления в корзину будет отображаться рядом с каждым товаром в каталоге. Щелчок на ней будет приводить к выводу сводки по товарам, которые уже были выбраны пользователем, включая их общую стоимость. В этой точке пользователь может с помощью кнопки продолжения покупки вернуться в каталог товаров, а с помощью кнопки перехода к оплате — сформировать заказ и завершить сеанс покупки.

Определение класса для представления корзины

Чтобы представить корзину для покупок и ее содержимое, мы добавили в папку `Models` проекта `GameStore` новый файл класса по имени `Cart.cs`. Содержимое этого файла показано в примере ниже:

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

namespace GameStore.Models
{
    public class Cart
    {
        private List<CartLine> lineCollection = new List<CartLine>();

        public void AddItem(Game game, int quantity)
        {
            CartLine line = lineCollection
                .Where(p => p.Game.GameId == game.GameId)
                .FirstOrDefault();

            if (line == null)
            {
                lineCollection.Add(new CartLine
                {
                    Game = game,
                    Quantity = quantity
                });
            }
            else
            {
                line.Quantity += quantity;
            }
        }

        public void RemoveLine(Game game)
        {
            lineCollection.RemoveAll(l => l.Game.GameId == game.GameId);
        }

        public decimal ComputeTotalValue()
        {
            return lineCollection.Sum(e => e.Game.Price * e.Quantity);

        }
        public void Clear()
        {
            lineCollection.Clear();
```

Класс `Cart` использует класс `CartLine`, определенный в том же самом файле, для представления выбранного пользователем товара и приобретаемого количества единиц этого товара. Мы определили методы для добавления элемента в корзину, удаления из корзины ранее добавленного элемента, вычисления общей стоимости элементов в корзине и сброса корзины за счет удаления всех помещенных в нее элементов. Мы также предоставили свойство, которое обеспечивает доступ к содержимому корзины, используя `IEnumerble<CartLine>`. Все это легко реализуется на `C#` с небольшой долей `LINQ`.

Добавление кнопок для помещения товаров в корзину

В веб-форму `\Pages\Listing.aspx` необходимо добавить кнопки, которые позволят пользователю помещать товары в корзину. В примере ниже показано, как это делается:

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

<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>
                            <button name='add' type='submit' value='{3}'>
                                Добавить в корзину
                            </button>
                        </div>",
                    game.Name, game.Description, game.Price, game.GameId));
            }
        %>
    </div>
    <div class="pager">
        <%
            for (int i = 1; i <= MaxPage; i++)
            {
                string category = (string)Page.RouteData.Values["category"]
                    ?? Request.QueryString["category"];

                string path = RouteTable.Routes.GetVirtualPath(null, null,
                    new RouteValueDictionary() { {"category", category}, { "pa
                Response.Write(
                    String.Format("<a href='{0}' {1}>{2}</a>",
                        path, i == CurrentPage ? "class='selected'" : "", i));
            }
        %>
    </div>
</asp:Content>
```

Здесь видно, что внутри фрагмента кода предусмотрен оператор, который добавляет элемент button к каждому товару. Щелчок на одной из таких кнопок приведет к отправке HTML-формы, которая была определена внутри мастер-

страницы. (Для приложений ASP.NET Framework принято соглашение определять элементы form в мастер-страницах, а не в отдельных веб-формах.)

Использование серверных элементов управления и привязки данных

В более широком плане также можно увидеть, что фрагмент кода теперь стало трудно читать, а чем труднее читать код, тем выше вероятность, что он содержит ошибку. Проблема связана с отсутствием удобного способа выражения фрагментов HTML разметки с помощью операторов C# — это верно как для фрагмента кода, так и для класса отделенного кода.

К счастью, существует альтернативный подход. В примере ниже приведено содержимое файла Listing.aspx, которое упорядочено за счет устранения фрагмента кода и добавления вместо него пары важных средств Web Forms: серверных элементов управления и привязки данных.

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

<asp:Content ContentPlaceHolderID="bodyContent" runat="server">
    <div id="content">
        <asp:Repeater ItemType="GameStore.Models.Game"
            SelectMethod="GetGames" runat="server">
            <ItemTemplate>
                <div class="item">
                    <h3><%# Item.Name %></h3>
                    <%# Item.Description %>
                    <h4><%# Item.Price.ToString("c") %></h4>
                    <button name="add" type="submit" value="<%# Item.GameId %>
                        Добавить в корзину
                    </button>
                </div>
            </ItemTemplate>
        </asp:Repeater>
    </div>
    <div class="pager">
        <%
            for (int i = 1; i <= MaxPage; i++)
            {
                string category = (string)Page.RouteData.Values["category"]
                    ?? Request.QueryString["category"];

                string path = RouteTable.Routes.GetVirtualPath(null, null,
                    new RouteValueDictionary() { {"category", category}, { "pa
                Response.Write(
                    String.Format("<a href='{0}' {1}>{2}</a>",
                        path, i == CurrentPage ? "class='selected'" : "", i));
            }
        %>
    </div>
</asp:Content>
```

Серверный элемент управления — это порция многократно используемой функциональности, похожая на пользовательский элемент управления, но созданная с применением другого процесса — более подробно об этом рассказывается в статье "Специальные элементы управления". В состав ASP.NET Framework включено множество готовых серверных элементов управления, которые решают распространенные задачи; элемент управления **Repeater** используется для

генерации одного и того же набора элементов для каждого объекта данных в наборе.

Определить, что работа производится с серверным элементом управления от Microsoft, можно по наличию префикса asp — например, asp:Repeater.

Элементу управления Repeater с помощью атрибута ItemType сообщается тип объекта данных, с которым выполняется работа, а посредством атрибута SelectMethod указывается способ получения объектов данных:

```
...
   <asp:Repeater ItemType="GameStore.Models.Game" SelectMethod="GetGames" runa
   ...
```

В этом примере мы указали класс Game и поручили элементу управления Repeater извлечение объектов данных с помощью метода GetGames() класса отделенного кода. Атрибуты ItemType и SelectMethod являются частью нового средства привязки данных ASP.NET 4.5, которое называется ***строго типизированными элементами управления данными***. В ранних версиях ASP.NET процесс создания элементов управления, предназначенных для представления данных приложения, был довольно утомительным. В последней версии он существенно упрощен и усовершенствован.

Элемент управления Repeater будет генерировать контент, содержащийся в его дочернем элементе ItemTemplate, для каждого объекта данных, который получается из метода GetGames(). Для включения значений из объектов данных используется специальный вид фрагмента кода:

```
...
    <h3><%# Item.Name %></h3>
...
```

Символ # сообщает ASP.NET Framework о необходимости вставки значения данных. Переменная Item применяется для ссылки на текущий объект данных, с которым оперирует элемент управления Repeater — в этом фрагменте требуется свойство Name, поэтому просто указывается Item.Name. Здесь не приходится иметь дело с форматированием и комбинированием строк, т.к. работа выполняется с HTML-элементами, а не операторами C#, что в конечном итоге дает веб-форму, которая проще в восприятии и сопровождении.

В результате использования элемента управления Repeater создается шаблон, в который можно вставлять значения данных. Платформа ASP.NET Framework прекрасно справляется с обработкой данных, а серверные элементы управления содержат множество полезной функциональности.

Превращение метода выдачи данных в открытый метод

Большинство методов в классе отделенного кода помечены как `protected` — это обеспечивает возможность обращения к ним из веб-формы, но не из других мест. Тем не менее, методы, используемые в атрибуте `SelectMethod` серверных элементов управления, должны быть открытыми, т.е. `public`. Следовательно, в файл отделенного кода `\Pages\Listing.aspx.cs` понадобится внести небольшое изменение, как показано в примере ниже:

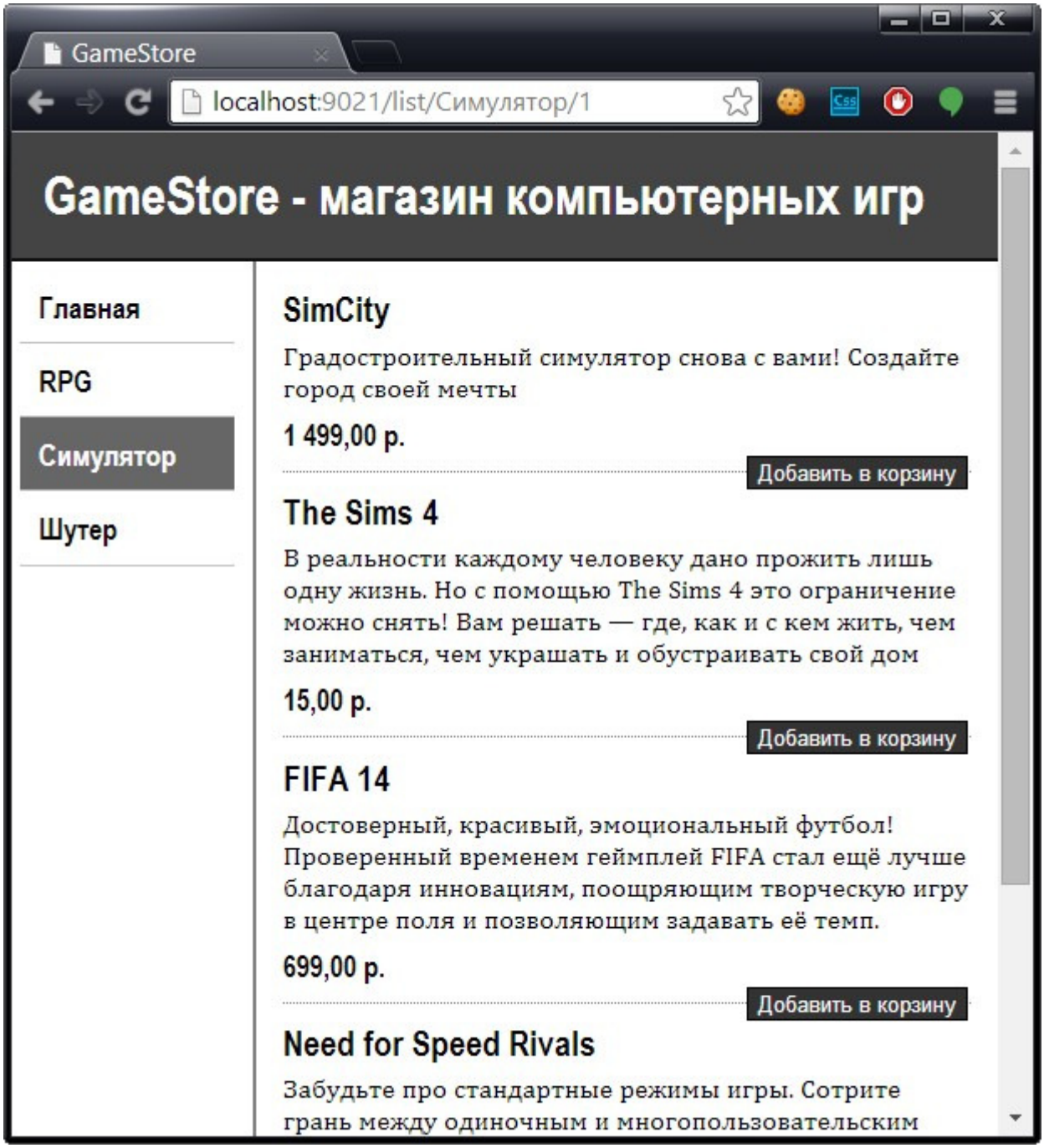
```
using System;
using System.Collections.Generic;
using GameStore.Models;
using GameStore.Models.Repository;
using System.Linq;
using GameStore.Pages.Helpers;
using System.Web.Routing;

namespace GameStore.Pages
{
    public partial class Listing : System.Web.UI.Page
    {
        // ...

        public IEnumerable<Game> GetGames()
        {
            // ...
        }

        // ...
     }
}
```

Добавление стиля CSS для кнопок добавления в корзину

Для управления внешним видом кнопок добавления в корзину мы добавили в файл `\Content\Styles.css` стиль, представленный в примере ниже:

```
div.item button {
    color:White;
    background-color: #333;
    border: 1px solid black;
    float: right;
}
```

Запустив приложение, можно увидеть результат всех показанных выше добавлений. Каждый товар отображается с кнопкой, которая выполняет отправку формы, определенной внутри мастер-страницы, как продемонстрировано на рисунке:



Создание вспомогательного класса для

сеанса

Платформа ASP.NET Framework включает средство, предназначенное для поддержки состояния сеанса, которое позволяет ассоциировать данные с сеансом. Сам сеанс охватывает множество запросов, которые потенциально могут относиться к разным веб-формам в приложении. Это означает возможность сохранения объектов данных во время множества взаимодействий с приложением для каждого пользователя, что идеально подходит для класса Cart.

Каждый пользователь должен иметь собственную корзину, которая должна сохраняться между запросами. Данные, ассоциированные с сеансом, удаляются по окончании этого сеанса (обычно из-за того, что пользователь не делал запросы в течение определенного времени), а это значит, что нам не придется управлять хранением или жизненным циклом объектов Cart.

Каждая инфраструктура для разработки веб-приложений имеет какой-нибудь вид механизма поддержки состояния — это связующий механизм, который позволяет строить приложения поверх запросов, использующих лишенный состояния протокол HTTP. В платформе ASP.NET Framework реализован другой механизм поддержки состояния, который называется состоянием представления.

Доступ к данным сеанса внутри веб-формы или в классе отделенного кода производится через свойство Session, которое возвращает объект System.Web.SessionState.**HttpSessionState**. Чтобы добавить объект к состоянию сеанса, мы устанавливаем значение для определенного ключа в объекте Session, примерно так:

```
...
   Session["Cart"] = cart;
...
```

Чтобы извлечь ранее добавленный объект, мы просто читаем значение для того же самого ключа:

```
...
   Cart cart = (Cart)Session["Cart"];
...
```

Механизм состояния сеанса исключительно удобен, но с его использованием связано несколько распространенных проблем, особенно в крупных проектах, в которых задействовано множество программистов.

Проблема первого вида возникает, когда приходится иметь дело с ключами типа string и значениями данных типа object. В разных частях приложения может применяться один и тот же ключ для сохранения отличающихся типов данных, что приводит к проблемам во время приведения результата типа object. С другой стороны, для одних и тех же данных могут использоваться отличающиеся ключи;

это означает, что данные помещаются в состояние сеанса, но не извлекаются из него корректно. Проблему такого вида решают обобщенные типы.

Еще одна проблема связана с повторением кода, когда одна и та же функциональность реализована много раз в рамках приложения. Повторения кода нужно избегать, поскольку оно существенно затрудняет тестирование и сопровождение приложения.

По умолчанию объекты состояния сеанса хранятся в памяти сервера ASP.NET, но для них можно настроить различные виды хранилищ, включая базу данных SQL.

Для демонстрации проблем обоих типов ниже приведен пример кода, который извлекает объект Cart из состояния сеанса, при необходимости создавая его:

```
Cart cart = (Cart)Session["Cart"];
if(cart == null)
{
        cart = new Cart();
    Session["Cart"] = cart;
}
```

В рамках приложения этот код будет повторяться в любой веб-форме, которой необходимо пользоваться объектом Cart. Чтобы такой код работал, понадобится обеспечить применение везде ключа Cart для получения или установки значений данных сеанса и ассоциировать объекты Cart только с ключом Cart. В случае изменения способа создания или управления объектами Cart придется найти все случаи повторения этого фрагмента кода и соответствующим образом модифицировать их.

Чтобы избежать этих проблем, мы создадим класс, который содержит статические методы для работы с данными сеанса, добавив новый файл класса по имени SessionHelper.cs в папку \Pages\Helpers. Содержимое этого файла приведено в примере ниже:

```
using System;
using System.Web.SessionState;
using GameStore.Models;

namespace GameStore.Pages.Helpers
{
    public enum SessionKey
    {
        CART,
        RETURN_URL
    }

    public static class SessionHelper
    {

        public static void Set(HttpSessionState session, SessionKey key, objec
        {
            session[Enum.GetName(typeof(SessionKey), key)] = value;
        }

        public static T Get<T>(HttpSessionState session, SessionKey key)
        {
            object dataValue = session[Enum.GetName(typeof(SessionKey), key)];
            if (dataValue != null && dataValue is T)
            {
                return (T)dataValue;
            }
            else
            {
                return default(T);
            }
        }

        public static Cart GetCart(HttpSessionState session)
        {
            Cart myCart = Get<Cart>(session, SessionKey.CART);
            if (myCart == null)
            {
                myCart = new Cart();
                Set(session, SessionKey.CART, myCart);
            }
            return myCart;
```

В файле было определено перечисление (enum) с именем SessionKey, которое содержит значения для типов данных, сохраняемых в сеансе. В перечислении определены значения CART (будет использоваться для объектов Cart) и RETURN_URL (будет применяться для URL, на который пользователи возвращаются в результате щелчка на кнопке "Продолжить покупку", гарантируя сохранение значений категории и разбиения на страницы).

Класс SessionHelper содержит метод Set(), предназначенный для помещения нового объекта данных в состояние сеанса с использованием значения SessionKey. Метод Get<T>() принимает значение SessionKey и возвращает соответствующий объект данных. Метод Get() имеет параметр обобщенного типа, который применяется для обеспечения того, что ожидаемый тип данных совпадает с типом сохраненных данных. На основе методов Get<T>() и Set() построен метод GetCart(), который решает проблемы дублирования кода и управляет объектом Cart для пользователя в единственном месте.

Вспомогательный класс для сеанса, подобный показанному выше, не предотвращает прямое использование разработчиками свойства Session внутри веб-формы или класса отделенного кода, но по нашему опыту это обычно не приводит к желаемому результату. Мы конфигурируем свою систему управления версиями так, чтобы она отклоняла файлы, в которых свойство Session используется напрямую.

Обработка отправки формы

Для элемента form, определенного внутри мастер-страницы, будет выполняться обратная отправка в адрес текущей страницы, которой в этом случае является \Pages\Listing.aspx. Мы должны добавить в файл Listing.aspx.cs код для обработки отправки формы и помещения выбранного товара в корзину пользователя.

В примере ниже показан результирующий код, в котором применяется класс SessionHelper, определенный в предыдущем разделе:

```
using System;
using System.Collections.Generic;
using GameStore.Models;
using GameStore.Models.Repository;
using System.Linq;
using GameStore.Pages.Helpers;
using System.Web.Routing;

namespace GameStore.Pages
{
    public partial class Listing : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (IsPostBack)
            {
                int selectedGameId;
                if (int.TryParse(Request.Form["add"], out selectedGameId))
                {
                    Game selectedGame = repository.Games
                        .Where(g => g.GameId == selectedGameId).FirstOrDefault

                    if (selectedGame != null)
                    {
                        SessionHelper.GetCart(Session).AddItem(selectedGame, 1
                        SessionHelper.Set(Session, SessionKey.RETURN_URL,
                            Request.RawUrl);

                        Response.Redirect(RouteTable.Routes
                            .GetVirtualPath(null, "cart", null).VirtualPath);
                    }
                }
            }
        }

        // ...для краткости другие методы и свойства не показаны...
    }
}
```

Мы находим значение идентификатора требуемого товара в получаемых данных формы и затем извлекаем соответствующий объект `Game` из хранилища. С помощью класса `SessionHelper` мы получаем объект `Cart`, ассоциированный с сеансом

пользователя, и добавляем к нему выбранный товар.

Реагирование на отправку формы завершается перенаправлением пользователя на другой URL с применением метода Response.Redirect(). При этом URL, на который перенаправляется браузер, генерируется на основе конфигурации маршрутизации — при генерации этого URL по-прежнему передается много значений null, но данная версия GetVirtualPath() создает URL из маршрута по имени cart. В примере ниже показано добавление к файлу \App_Start \RouteConfig.cs, которое делает это возможным:

```
using System;
using System.Web.Routing;

namespace GameStore
{
    public class RouteConfig
    {
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.MapPageRoute(null, "list/{category}/{page}",
                                      "~/Pages/Listing.aspx");
            routes.MapPageRoute(null, "list/{page}", "~/Pages/Listing.aspx");
            routes.MapPageRoute(null, "", "~/Pages/Listing.aspx");
            routes.MapPageRoute(null, "list", "~/Pages/Listing.aspx");

            // Обратите внимание что это именованный маршрут
            routes.MapPageRoute("cart", "cart", "~/Pages/CartView.aspx");
        }
    }
}
```

В первом аргументе метода MapPageRoute() указывается имя маршрута. Мы обычно не назначаем имена главному набору маршрутов в своих приложениях (поэтому для всех них передаются значения null), но наличие имени может быть полезно для перенаправления пользователя из одной части приложения в другую. Новый маршрут добавляет поддержку URL вида /cart, который обрабатывается с помощью веб-формы \Pages\CartView.aspx. Вскоре мы создадим этот файл и будем его применять для отображения содержимого корзины.

Использование системы маршрутизации для того, чтобы сгенерировать URL вида /cart, может показаться несколько странным. В конце концов, почему бы просто не передать /cart методу Response.Redirect()? Мы поступаем так потому, что это позволит изменять URL, отображаемый на страницу CartView.aspx, модифицируя только конфигурацию маршрутизации, а не веб-формы и классы отделенного кода, полагающиеся на данный URL. Придется изменять только один класс конфигурации

маршрутизации, а не потенциальные десятки веб-форм и классов отделенного кода.

Отображение содержимого корзины

Как было показано в предыдущем разделе, веб-форма `Listing.aspx` после добавления товара в корзину выполняет перенаправление браузера пользователя на маршрут `/cart`. Теперь необходимо создать веб-форму, которая будет применяться для поддержки этого `URL`. Мы добавили в папку `Pages` проекта `GameStore` новый файл веб-формы по имени `CartView.aspx`. В примере ниже приведено содержимое файла отделенного кода `CartView.aspx.cs`:

```
using System;
using System.Collections.Generic;
using GameStore.Models;
using GameStore.Pages.Helpers;

namespace GameStore.Pages
{
    public partial class CartView : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }

        public IEnumerable<CartLine> GetCartLines()
        {
            return SessionHelper.GetCart(Session).Lines;
        }

        public decimal CartTotal
        {
            get
            {
                return SessionHelper.GetCart(Session).ComputeTotalValue();
            }
        }

        public string ReturnUrl
        {
            get
            {
                return SessionHelper.Get<string>(Session, SessionKey.RETURN_UR
            }
        }
    }
}
```

В этом классе определен метод и пара свойств, которые понадобятся для отображения содержимого корзины. В этом примере хорошо видно, как используется класс `SessionHelper` — методам `GetCart()` и `Get<T>()` передается значение свойства `Session` для получения требуемых объектов данных сеанса, при этом не приходится беспокоиться о приведении типов или создании объекта `Cart`, если он не существует.

В примере ниже показан код веб-формы \Pages\CartView.aspx, в котором применяется метод и свойства класса отделенного кода для отображения содержимого корзины пользователю:

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

<asp:Content ID="Content1" ContentPlaceHolderID="bodyContent" runat="server">
    <div id="content">
        <h2>Ваша корзина</h2>
        <table id="cartTable">
            <thead>
                <tr>
                    <th>Кол-во игр</th>
                    <th>Название</th>
                    <th>Цена</th>
                    <th>Общая стоимость</th>
                </tr>
            </thead>
            <tbody>
                <asp:Repeater ID="Repeater1" ItemType="GameStore.Models.CartLi
                    SelectMethod="GetCartLines" runat="server">
                    <ItemTemplate>
                        <tr>
                            <td><%# Item.Quantity %></td>
                            <td><%# Item.Game.Name %></td>
                            <td><%# Item.Game.Price.ToString("c")%></td>
                            <td><%# ((Item.Quantity *
                                Item.Game.Price).ToString("c"))%></td>
                        </tr>
                    </ItemTemplate>
                </asp:Repeater>
            </tbody>
            <tfoot>
                <tr>
                    <td colspan="3">Итого:</td>
                    <td><%= CartTotal.ToString("c") %></td>
                </tr>
            </tfoot>
        </table>
        <p class="actionButtons">
            <a href="<%= ReturnUrl %>">Продолжить покупки</a>
        </p>
    </div>
</asp:Content>
```

Эта веб-форма построена на основе приемов, которые уже были представлены. Мы используем атрибут MasterPageFile дескриптора Page для указания, что должна применяться страница Store.Master. Это обеспечивает внешний вид, согласованный со страницей Listing.aspx. Отображаемая разметка помещена в элемент управления Content.

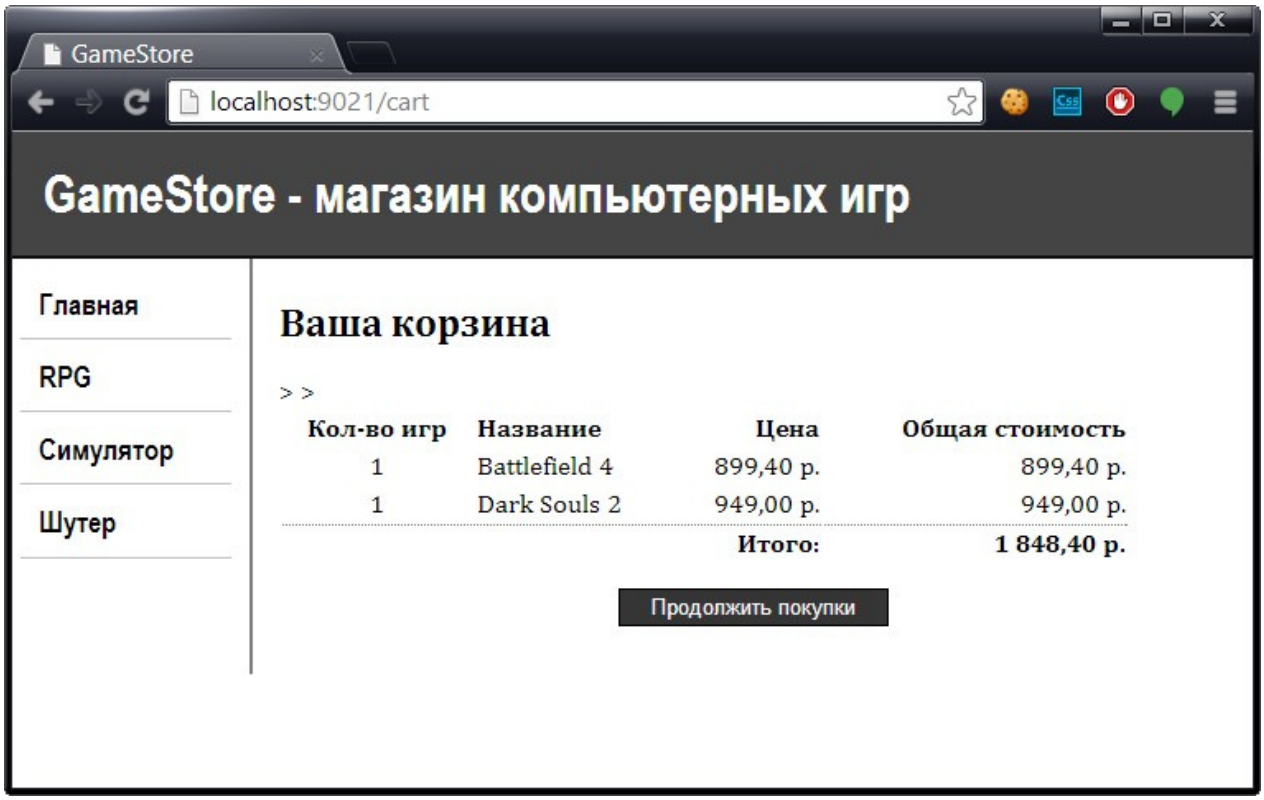
Элемент управления Repeater служит для отображения деталей отдельных объектов CartLine, которые получаются с применением средства привязки данных в методе GetCartLines() класса отделенного кода. Другие значения данных извлекаются с помощью фрагментов кода, отображающих значения ранее определенных свойств класса отделенного кода. Это хорошая демонстрация использования ряда основных приемов Web Forms при построении приложения.

Финальный шаг в отображении содержимого корзины связан со стилизацией HTML-элементов. В примере ниже показаны добавления в файл \Content \Styles.css. (Эти стили применяются к файлу CartView.aspx, потому что ранее мы добавили в мастер-страницу Store.Master элемент link для данной таблицы стилей CSS.):

```
h2 { margin-top: 0.3em }
#cartTable { width: 90%;}
#cartTable tfoot td { border-top: 1px dotted gray; font-weight: bold; }
#cartTable thead th { text-align: right;}
#cartTable thead th:first-child { text-align: center;}
#cartTable thead th:nth-child(2) { text-align: left;}
#cartTable tbody td { text-align: right;}
#cartTable tbody td:first-child { text-align: center;}
#cartTable tbody td:nth-child(2) { text-align: left;}
#cartTable tfoot tr td { text-align: right;}
p.actionButtons { text-align: center;}
.actionButtons a, button.actionButtons {
    font: .8em Arial; color: White; margin: .5em;
    text-decoration: none; padding: .15em 1.5em .2em 1.5em;
    background-color: #353535; border: 1px solid black;
}
```

Тестирование корзины

Запустите приложение и просмотрите списки товаров. Щелкните на кнопке "Добавить в корзину" для заинтересовавшего товара. Вы увидите представление корзины, подобное показанному на рисунке ниже:



Щелкните на кнопке "Продолжить покупки", чтобы вернуться на страницу товаров, на которой до этого был совершен щелчок на кнопке "Добавить в корзину". Товары внутри корзины отображаются в порядке их добавления клиентом и содержат промежуточные суммы и итоговую стоимость.

Удаление из корзины нежелательных элементов

Мы должны предоставить пользователю средства для удаления элементов из корзины. В примере ниже показано, что к веб-форме `CartView.aspx` добавлены кнопки "Удалить" для каждого элемента в корзине:

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

<asp:Content ID="Content1" ContentPlaceHolderID="bodyContent" runat="server">
    <div id="content">
        <h2>Ваша корзина</h2>
        <table id="cartTable">
            <thead>
                <tr>
                    <th>Кол-во игр</th>
                    <th>Название</th>
                    <th>Цена</th>
                    <th>Общая стоимость</th>
                </tr>
            </thead>
            <tbody>
                <asp:Repeater ID="Repeater1" ItemType="GameStore.Models.CartLi
                    SelectMethod="GetCartLines" runat="server">
                    <ItemTemplate>
                        <tr>
                            <td><%# Item.Quantity %></td>
                            <td><%# Item.Game.Name %></td>
                            <td><%# Item.Game.Price.ToString("c")%></td>
                            <td><%# ((Item.Quantity *
                                Item.Game.Price).ToString("c"))%></td>
                            <td>
                                <button type="submit" class="actionButtons" na
                                    value="<%#Item.Game.GameId %>">
                                    Удалить</button>
                            </td>
                        </tr>
                    </ItemTemplate>
                </asp:Repeater>
            </tbody>
            <tfoot>
                <tr>
                    <td colspan="3">Итого:</td>
                    <td><%= CartTotal.ToString("c") %></td>
                </tr>
            </tfoot>
        </table>
        <p class="actionButtons">
```

Мы добавили элемент `button` с типом `submit` в раздел `ItemTemplate` элемента управления `Repeater`, используя привязку данных для установки атрибута `value` в идентификатор товара. Элемент управления `Repeater` создаст для каждого товара в корзине элемент `button`. Щелчок на одной из этих кнопок приведет к отправке серверу формы, определенной внутри мастер-страницы.

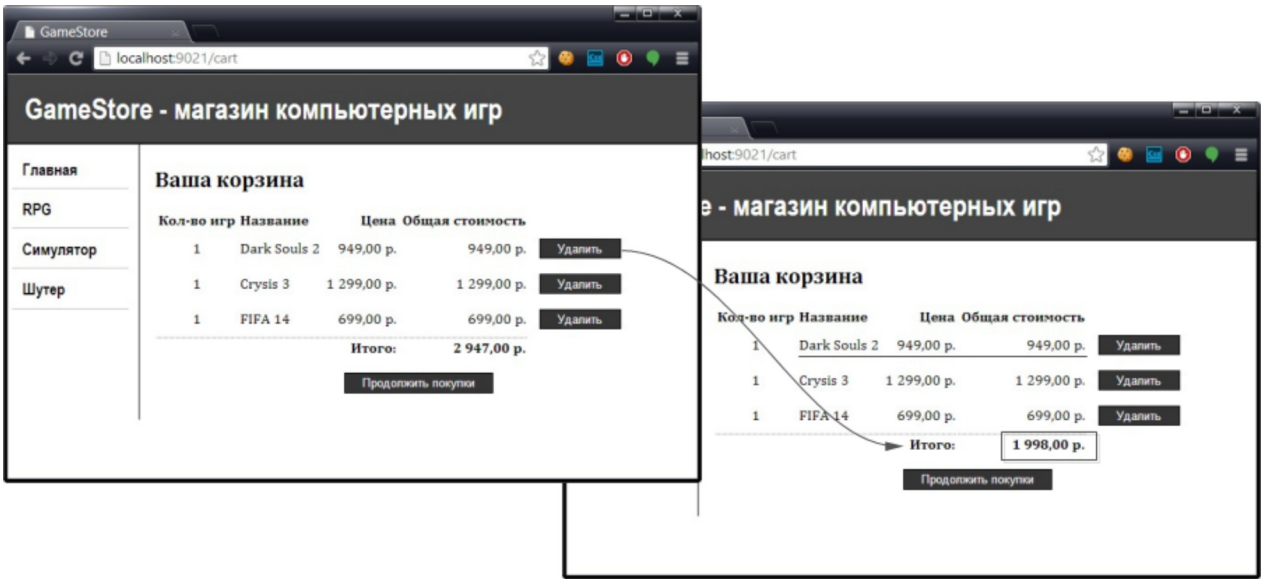
В примере ниже приведен обновленный файл отделенного кода `CartView.aspx.cs` с добавленной обработкой `HTTP`-запроса `POST`, который поступает после щелчка на одной из кнопок `Remove`. С помощью `Request.Form` из формы извлекается значение `remove` — это даст идентификатор товара, который пользователь желает удалить. Данный идентификатор применяется для получения объекта `Game` из хранилища, а затем получения объекта `Cart` и вызова метода `RemoveLine()`:

```
using System;
using System.Collections.Generic;
using System.Linq;
using GameStore.Models;
using GameStore.Models.Repository;
using GameStore.Pages.Helpers;
using System.Web.Routing;

namespace GameStore.Pages
{
    public partial class CartView : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (IsPostBack)
            {
                Repository repository = new Repository();
                int gameId;
                if (int.TryParse(Request.Form["remove"], out gameId))
                {
                    Game gameToRemove = repository.Games
                        .Where(g => g.GameId == gameId).FirstOrDefault();
                    if (gameToRemove != null)
                    {
                        SessionHelper.GetCart(Session).RemoveLine(gameToRemove
                    }
                }
            }
        }

        // ...для краткости другие методы и свойства не показаны...
    }
}
```

Как и можно было ожидать, здесь используется поддержка Web Forms обработки запросов POST для доступа к объектам Cart и Game. Тем не менее, в данный момент кнопки "Удалить" не работают так, как должны.

Состояние представления

Чтобы понять, что имеется в виду, запустите приложение, добавьте в корзину два или три товара и щелкните на одной из кнопок "Удалить". Вы увидите, что общая стоимость обновилась корректно, но удаленный элемент так и остался в корзине:



Мы столкнулись с одним из самых неправильно понимаемых и некорректно используемых средств ASP.NET Framework — состоянием представления. Базовая идея заключается в том, что состояние веб-приложения содержится в скрытом элементе input и отправляется браузеру как часть ответа. Данные состояния представления служат для обеспечения постоянства между множеством запросов, что во многом похоже на состояние сеанса, но хранятся на стороне клиента и отправляются в виде части данных формы, посылаемых серверу.

Воспользовавшись инструментами <F12> браузера для просмотра отображаемой в браузере HTML-разметки, можно увидеть элемент состояния представления:

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="/wEPDwUKLTg3Mz
```

Эти данные были добавлены к HTML-разметке, отправляемой браузеру, так что элемент управления Repeater может кэшировать данные корзины, которые отображаются, а не запрашивать их из актуального объекта Cart в состоянии сеанса. Мы знаем, что объект Cart был корректно модифицирован, но это значение не было кэшировано в состоянии представления, т.к. изменения происходили за рамками элемента управления Repeater.

При наличии большого количества элементов управления, использующих состояние представления, объем данных, которые добавляются к каждому запросу, может быть значительным. Это одно из главных направлений критики инфраструктуры Web Forms. Состояние представления дублирует данные, которые уже содержатся в HTML-элементах, отправляемых браузеру, и требует более широкой полосы пропускания для доставки контента пользователю. Это не является особой проблемой для интранет-приложений, но в случае Интернет-приложений необходимо уделять внимание генерируемому ими трафику. Данные состояния представления могут быстро превратиться в проблему. Это вовсе не говорит о том, что средство состояния представления бесполезно, а означает лишь то, что оно должно применяться осторожно и умеренно — такие характеристики не обеспечиваются по умолчанию.

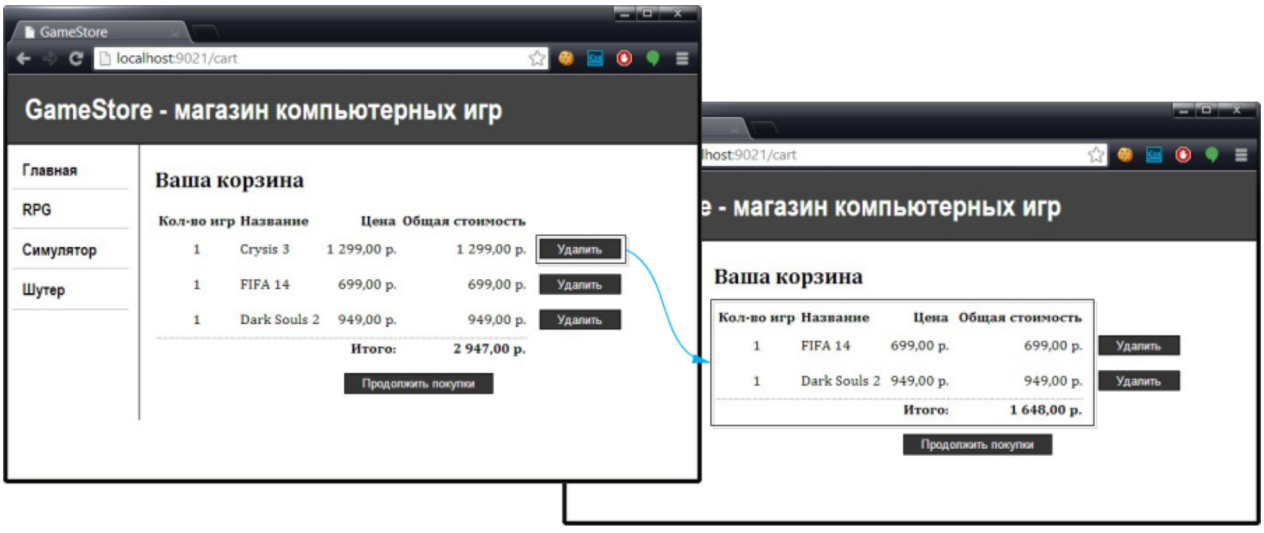
Отключение состояния представления

Чтобы элемент управления Repeater прекратил пользоваться состоянием представления, нужно установить атрибут EnableViewState в false, как показано в примере ниже. Это изменение заставляет элемент управления Repeater загружать актуальные данные из объекта Cart, а не применять кэшированную копию, которая скрыта в HTML-разметке, отправленной браузеру.

Вас может заинтересовать, почему кто-то в свое время посчитал состояние представления удачным решением. Мы можем только предположить, что в Microsoft действительно стремились воспроизвести среду разработки приложений Visual Basic для веб-приложений, и это распространилось на пользовательские интерфейсы, поддерживающие состояние. Лучше всего рассматривать состояние представления как пережиток прежних времен, который иногда может оказываться полезным в современных веб-приложениях.

```
<asp:Repeater ID="Repeater1" ItemType="GameStore.Models.CartLine"
      SelectMethod="GetCartLines" runat="server" EnableViewState="false">
      <ItemTemplate>
          ...
      </ItemTemplate>
</asp:Repeater>
```

Примененный подобным образом атрибут EnableViewState влияет только на одиночный элемент управления. После применения атрибута EnableViewStart можно повторно протестировать кнопки "Удалить", поведение будет ожидаемым, а содержимое корзины и общая стоимость корректно обновятся:



Добавление итоговой информации по корзине

Мы имеем функционирующую корзину, но еще должны определить способ ее встраивания в пользовательский интерфейс. Пользователи могут определить, что находится в их корзинах, только за счет просмотра веб-формы с представлением корзины. Однако попасть на эту веб-форму можно только путем добавления

нового элемента в корзину.

Для решения этой проблемы мы создадим виджет (т.е. графический элемент) с итоговой информацией по содержимому корзины; щелчок на виджете должен приводить к отображению содержимого корзины. Это будет реализовано почти так же, как при добавлении виджета для навигации — в виде пользовательского элемента управления, добавляемого на мастер-страницу. Но мы хотим подчеркнуть, что разработка приложений Web Forms не сводится только к фрагментам кода и состоянию представления, поэтому продемонстрируем возможность работы напрямую с HTML-элементами, отправляемыми браузеру.

Мы добавили в папку Controls проекта GameStore новый элемент Web User Control (Пользовательский веб-элемент управления) по имени CartSummary.ascx с разметкой, приведенной в примере ниже:

```
<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="CartSummary.ascx.
   Inherits="GameStore.Controls.CartSummary" %>

<div id="cartSummary">
    <span class="caption">
        <b>В корзине:</b>
        <span id="csQuantity" runat="server"></span> товаров,
        <span id="csTotal" runat="server"></span>
    </span>
    <a id="csLink" runat="server">Корзина</a>
</div>
```

Разметка максимально упрощена — есть два элемента , посредством которых можно отображать общее количество элементов в корзине и итоговую стоимость. Кроме того, имеется также элемент <a>, который будет настроен так, что щелчок на нем переместит пользователя на веб-форму CartView.aspx.

Мы собираемся конфигурировать элементы с применением класса отделенного кода. Для этого к каждому элементу понадобится применить атрибут runat со значением server. Когда среда ASP.NET Framework обрабатывает веб-форму или пользовательский элемент управления, она создает переменные для всех найденных HTML-элементов, которые имеют атрибут runat. В качестве имени переменной используется значение атрибута id, т.е. показанная в примере разметка приведет к созданию переменных csQuantity и csTotal, которые представляют элементы , и переменной csLink, представляющей элемент <a>.

Работа с этими переменными иллюстрируется в примере ниже, в котором приведено содержимое файла отделенного кода CartSummary.ascx.cs:

```
using System;
using System.Linq;
using GameStore.Models;
using System.Web.Routing;
using GameStore.Pages.Helpers;

namespace GameStore.Controls
{
    public partial class CartSummary : System.Web.UI.UserControl
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            Cart myCart = SessionHelper.GetCart(Session);
            csQuantity.InnerText = myCart.Lines.Sum(x => x.Quantity).ToString(
            csTotal.InnerText = myCart.ComputeTotalValue().ToString("c");
            csLink.HRef = RouteTable.Routes.GetVirtualPath(null, "cart",
                null).VirtualPath;
        }
    }
}
```

В коде видно, что каждый элемент конфигурируется через соответствующую ему переменную, созданную ASP.NET Framework. Такие переменные возвращают объекты из пространства имен System.Web.UI.HtmlControls; простые элементы, такие как span, представлены экземплярами класса HtmlGenericControl, тогда как для представления более сложных элементов предусмотрены собственные классы. Например, элемент <a> представлен классом HtmlAnchor, в котором определены свойства, применяемые для настройки уникальных характеристик элемента <a>. В этом примере свойство HRef используется для установки значения атрибута href, позволяя конфигурировать ссылку так, чтобы она пользовалась схемой маршрутизации URL для отображения пользователю содержимого корзины.

Применение объектов для представления элементов в разметке похоже на использование API-интерфейса DOM (Document Object Model — модель документных объектов) для навигации по контенту в браузере с помощью кода JavaScript, что должно выглядеть знакомо, если вам приходилось писать сценарии клиентской стороны.

Определение стилей CSS

Как и со всеми дополнениями к приложению, мы должны определить стили CSS для элементов, отображающих итоговую информацию по корзине. В примере ниже показаны стили, добавленные в файл \Content\Styles.css:

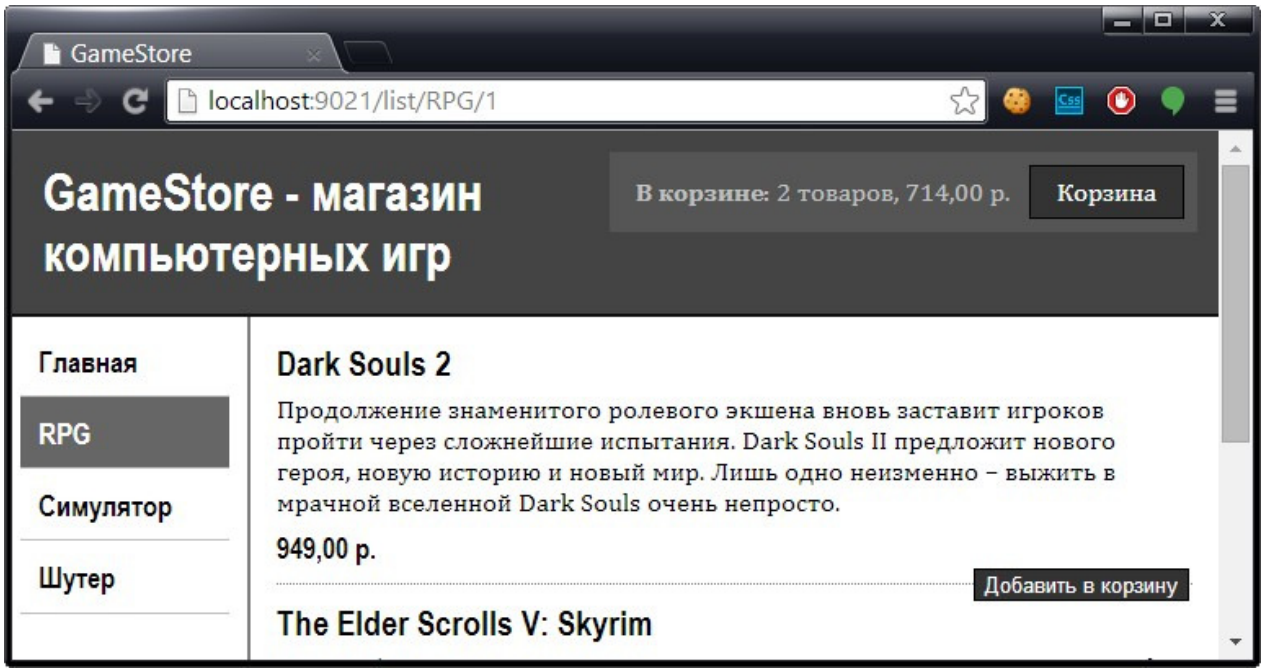
```
div#cartSummary {
    float:right;
    margin: .8em;
    color: Silver;
    background-color: #555;
    padding: .5em .5em .5em 1em;
}
div#cartSummary a {
    text-decoration: none;
    padding: .4em 1em .4em 1em;
    line-height:2.1em;
    margin-left: .5em;
    background-color: #333;
    color:White;
    border: 1px solid black;
}
```

Применение элемента управления, отображающего итоговую информацию по корзине

Для применения элемента управления `CartSummary` мы с помощью директивы `Register` сообщаем `ASP.NET Framework` о пользовательском элементе управления и затем добавляем элемент, указывающий место отображения этого элемента управления. Необходимо, чтобы элемент управления `CartSummary` отображался в приложении повсеместно, поэтому мы изменяем файл `\Pages\Store.Master`, как показано в примере ниже:

```
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Store.master.cs" I
<%@ Register TagPrefix="GS" TagName="CategoryLinks" Src="~/Controls/CategoryLi
<%@ Register TagPrefix="GS" TagName="CartSummary" Src="~/Controls/CartSummary.

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>GameStore</title>
    <link rel="stylesheet" href="/Content/Styles.css" />
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <div id="header">
                <GS:CartSummary runat="server" />
                <div class="title">GameStore - магазин компьютерных игр</div
            </div>
            <div id="categories">
                <GS:CategoryLinks runat="server" />
            </div>
            <div>
                <asp:ContentPlaceHolder ID="bodyContent" runat="server" />
            </div>
        </div>
    </form>
</body>
</html>
```

Теперь пользователь приложения видит итоговую информацию по своей корзине и может в любой момент просмотреть ее содержимое:



Объединение объявлений пользовательских элементов управления

Нам не нравится использование директив `Register` в файлах веб-форм и мастер-страниц. В сложном приложении может существовать большое количество пользовательских элементов управления, что в конечном итоге приводит к дублированию одной и той же информации `Register` по всему приложению. Намного более удачный подход предусматривает объявление пользовательских элементов управления в файле `Web.config`. Это означает, что они будут доступны везде в приложении без необходимости в указании директив `Register`.

В примере ниже демонстрируется объявление двух пользовательских элементов управления в разделе `<system.web>` файла `Web.config`:

```
<system.web>
    <pages controlRenderingCompatibilityVersion="4.0">
      <controls>
        <add tagPrefix="GS" tagName="CategoryLinks" src="~/Controls/CategoryLi
        <add tagPrefix="GS" tagName="CartSummary" src="~/Controls/CartSummary.
      </controls>
    </pages>
</system.web>
```

Мы применяем для элементов управления тот же самый префикс дескриптора и те же имена. Определение элементов управления в файле `Web.config` позволяет использовать их где угодно в приложении, не задавая отдельные директивы `Register`. В следующем примере видно, что упомянутые директивы удалены из файла `Store.Master`:

```
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Store.master.cs"
   Inherits="GameStore.Pages.Store" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>GameStore</title>
    <link rel="stylesheet" href="/Content/Styles.css" />
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <div id="header">
                <GS:CartSummary runat="server" />
                <div class="title">GameStore - магазин компьютерных игр</div
            </div>
            <div id="categories">
                <GS:CategoryLinks runat="server" />
            </div>
            <div>
                <asp:ContentPlaceHolder ID="bodyContent" runat="server" />
            </div>
        </div>
    </form>
</body>
</html>
```

В результате получается мастер-страница, более ориентированная на контент, и единственное место объявления элементов управления, куда должны вноситься изменения, если в этом возникнет необходимость.

Объявлять пользовательские элементы управления, находящиеся в той же самой папке, где расположена веб-форма или мастер-страница, в которой они используются, в файле `Web.config` не разрешено. Именно по этой причине применяется отдельная папка `Controls`.

Alexandr Erohin ✯ alexerohinzzz@gmail.com