Админ панель: управление заказами

Исходный код проекта

В этой статье мы продолжим построение приложения GameStore за счет предоставления администратору сайта способа обработки заказов и управления каталогом товаров.

Мы собираемся создать мастер-страницу и таблицу стилей CSS, которые являются специфичными для административного раздела приложения GameStore. В этой статье мы настроим общие строительные блоки и подготовим их для веб-форм, которые будут добавлены позже.

Расширение конфигурации маршрутизации

В приложении должны поддерживаться два новых URL, ссылающиеся на две веб-формы, которые будут добавлены позже для обеспечения обработки заказов и управления каталогом товаров. В примере ниже показана расширенная конфигурация маршрутизации в файле \App_Start\RouteConfig.cs:

```
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");

            routes.MapPageRoute("cart", "cart", "~/Pages/CartView.aspx");
            routes.MapPageRoute("checkout", "checkout", "~/Pages/Checkout.aspx

            // Новые маршруты для административных страниц
            routes.MapPageRoute("admin_orders", "admin/orders", "~/Pages/Admin
            routes.MapPageRoute("admin_games", "admin/games", "~/Pages/Admin/G
        }
    }
}
```

Согласно этим добавлениям, URL вида /admin/orders приведет к обработке файла веб-формы \Pages\Admin\Orders.aspx, а URL вида /admin/products — к обработке файла веб-формы \Pages\Admin\Products.aspx. Указанные файлы веб-форм пока еще не существуют, но будут добавлены далее.

Добавление мастер-страницы для администрирования

Мы будем использовать специальную мастер-страницу, предназначенную только для административных страниц приложения — это поможет избежать дублирования разметки во множестве веб-форм и обеспечит более четкое разделение административной и пользовательской частей приложения.

Щелкните правой кнопкой мыши на папке \Pages\Admin в окне Solution Explorer (Проводник решения) и выберите в контекстном меню пункт Add New Item. Выберите вариант Master Page (Мастер-страница) в списке шаблонов элементов, установите имя в Admin.Master и щелкните на кнопке Add (Добавить), чтобы создать новый файл. Приведите содержимое этого файла в соответствие со следующим кодом:

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

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <link rel="stylesheet" href="~/Content/Admin.css" />
</head>
<body>
    <form id="form1" runat="server">
        <h1>GameStore: admin-панель</h1>
        <div class="adminContent">
            <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">

            </asp:ContentPlaceHolder>
        </div>
    </form>
    <div id="nav">
        <a href="<%= OrdersUrl %>">Управление заказами</a>
        <a href="<%= GamesUrl %>">Управление каталогом игр</a>
    </div>
</body>
</html>
```

Никаких новых приемов в этой мастер странице не применяется. Здесь определен элемент управления ContentPlaceHolder, который позволяет помещать контент из веб-форм, использующих эту мастер-страницу и ряд дополнительных элементов, которые обеспечат согласованную структуру для административных страниц. Для ссылки на административные страницы предусмотрена пара элементов <a>, так что пользователь может переключаться между ними; атрибут href устанавливается с помощью фрагментов кода, которые читают свойства OrdersUrl и GamesUrl класса отделенного кода:

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

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

        }

        public string OrdersUrl
        {
            get
            {
                return generateURL("admin_orders");
            }
        }

        public string GamesUrl
        {
            get
            {
                return generateURL("admin_games");
            }
        }

        private string generateURL(string routeName)
        {
            return RouteTable.Routes.GetVirtualPath(null, routeName, null).Vir
        }
    }
}
```

У нас не было необходимости работать с классом отделенного кода для мастер-страницы, применяемой к веб-формам в предыдущих страницах, но в примере можно заметить, что класс отделенного кода мастер-страницы похож на такой класс для веб-формы, хотя базовым классом является `System.Web.UI.MasterPage`. Подробные сведения о классе `MasterPage` и возможностям, которые он предлагает, описаны в статье "Усовершенствованные мастер-страницы", а пока что нас интересуют только свойства `OrdersUrl` и `GamesUrl`, возвращающие URL, которые сгенерированы на основе конфигурации

маршрутизации.

Добавление таблицы стилей CSS

В мастер-страницу Admin.Master был добавлен элемент link для таблицы стилей CSS, который создается в настоящем разделе. Щелкните правой кнопкой мыши на папке Content в окне Solution Explorer и выберите в контекстном меню пункт Add New Item. Укажите шаблон элемента Style Sheet (Таблица стилей), установите имя в AdminStyles.css и щелкните на кнопке Add, чтобы создать новый файл. Приведите содержимое таблицы стилей в соответствие со следующим кодом:

```
body {
    font-family: "Arial";
}

h1.title {
    background-color: black;
    color: white;
    width: 100%;
    text-align: center;
}

div#nav {
    text-align: center;
}

div#nav a {
    font: bold 1.1em "Arial Narrow","Franklin Gothic Medium",Arial;
    display: inline-block;
    color: black; padding: 6px; border: solid medium black;
    text-decoration: none; width: 200px; text-align: center;
}

div#nav a:hover {
    background: #888;
    color: white;
}
```

Таблица стилей содержит стили, необходимые для элементов мастер-страницы, а также стили для веб-форм, которые будут добавлены позже.

Добавление веб-формы

Просмотреть мастер-страницу саму по себе невозможно, поэтому мы добавим

веб-форму, чтобы увидеть достигнутые результаты. В предшествующих статьях мы создавали стандартный файл веб-формы и затем редактировали его контент, чтобы он использовал мастер-страницу. Тем не менее, в Visual Studio доступен более удобный способ создания веб-форм в случае, если мастер-страница уже существует. В качестве демонстрации мы создадим веб-форму \Pages\Admin \Orders.aspx, которая будет применять мастер-страницу, построенную в предыдущем разделе.

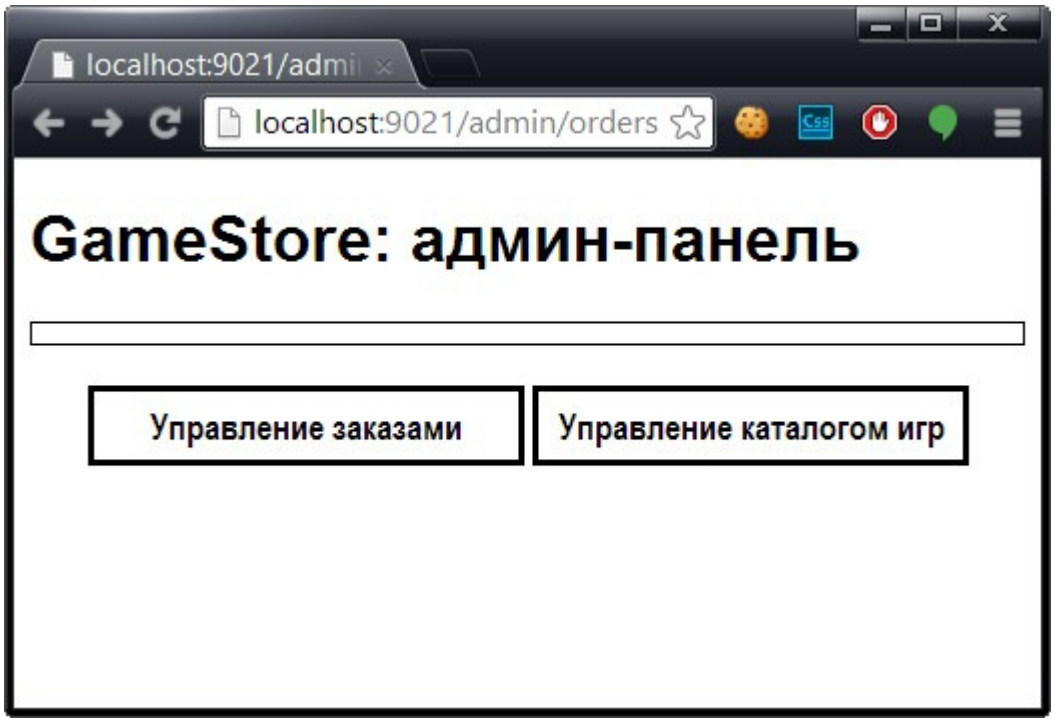
Щелкните правой кнопкой мыши на папке \Pages\Admin в окне Solution Explorer и выберите в контекстном меню пункт Add New Item. Укажите шаблон элемента Web Form Using Master Page (Веб-форма, использующая мастер-страницу), установите имя в Orders.aspx и щелкните на кнопке Add. Среда Visual Studio отобразит диалоговое окно Select a Master Page (Выбор мастер-страницы), в котором можно выбрать мастер-страницу для применения. Перейдите в папку Pages\Admin и укажите файл Admin.Master.

Мастер-страницы не обязательно должны находиться в тех же папках, что и файлы веб-форм. Мы предпочитаем группировать их вместе, но знаем многих разработчиков приложений ASP.NET Framework, которые помещают свои мастер-страницы в папку, отдельную от папок с файлами веб-форм.

Щелкните на кнопке OK, чтобы создать новую веб-форму. В примере ниже приведено содержимое нового файла, сгенерированное Visual Studio:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Orders.aspx.cs"
     Inherits="GameStore.Pages.Admin.Orders"
     MasterPageFile="~/Pages/Admin/Admin.Master" %>

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

Среда Visual Studio добавила директиву Page, сконфигурированную для использования нужной мастер-страницы, и элемент управления Content, который можно применять для вставки контента в полную разметку, отправляемую браузеру. Этого вполне достаточно, чтобы иметь возможность просмотреть эффект от использования мастер-страницы. Понадобится только запустить приложение и перейти в браузере на URL вида /admin/orders, как показано на рисунке ниже:



Очевидно, что есть еще немало работ, которые придется сделать для получения какого-то полезного контента, однако уже можно видеть общие элементы, которые будут применяться к административной части приложения. В последующих разделах мы построим приложение для предоставления функций администрирования.

Добавление средств управления заказами

Ранее была добавлена поддержка для получения заказов от пользователя и помещения их в базу данных через классы хранилища. В этой статье мы собирается завершить процесс и создать административную страницу, которая позволит просматривать заказы и помечать их как переданные в службу доставки. Мы будем использовать те же самые приемы, которые задействовали в предыдущих статьях, для создания стандартной HTML-разметки, отображающей заказы из базы данных.

Очистка и заполнение базы данных

Перед началом добавления кода для отображения и управления заказами необходимо очистить базу данных. Был момент, когда для простоты мы позволяли пользователям отправлять заказы без какой-либо проверки достоверности, и сейчас мы собираемся очистить базу от таких данных, заменив их примерами заказов, с которыми можно будет работать в этом разделе.

Откройте окно Server Explorer в Visual Studio и разверните его контент, пока не будет виден элемент GameStore (наша база данных). Щелкните на нем правой кнопкой мыши и выберите в контекстном меню пункт New Query (Новый запрос). В текстовой области открывшегося окна введите операторы SQL, приведенные в примере ниже:

```
DELETE FROM OrderLines
DELETE FROM Orders
```

Щелкните правой кнопкой мыши внутри текстовой области и выберите в контекстном меню пункт Execute (Выполнить), чтобы выполнить эти SQL-операторы. После этого вы можете заполнить произвольной информацией таблицы Orders и OrdersLine используя интерфейс нашего интернет-магазина.

Добавление контента веб-формы

Файл веб-формы \Pages\Admin\Orders.aspx уже создан, так что можно было бы протестировать мастер-страницу, но необходимо еще добавить контент, который позволит администратору управлять заказами. В примере ниже показаны дополнения, внесенные в файл Orders.aspx:

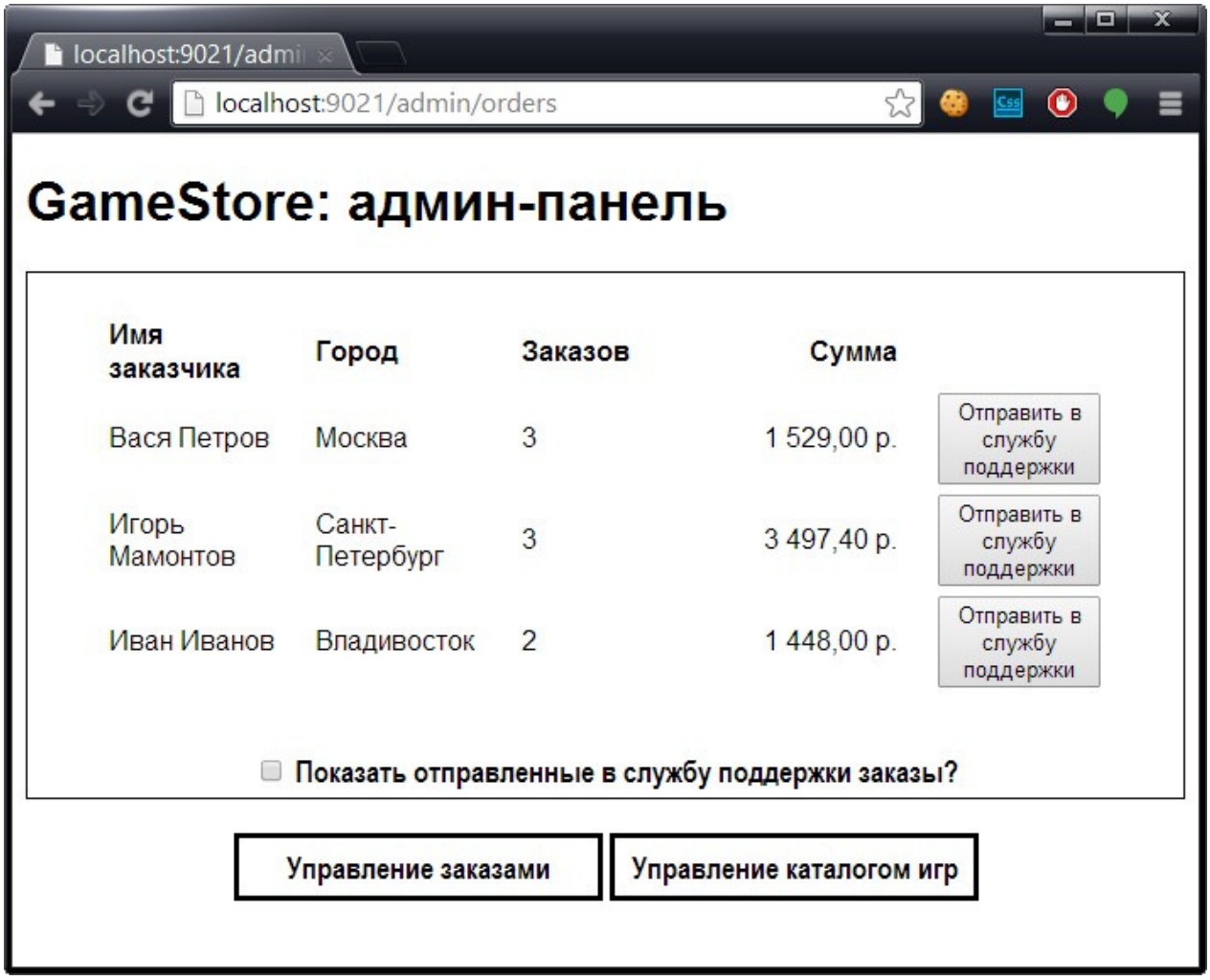
```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Orders.aspx.cs"
     Inherits="GameStore.Pages.Admin.Orders"
     MasterPageFile="~/Pages/Admin/Admin.Master" EnableViewState="false" %>

<asp:Content ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
    <div class="outerContainer">
        <table id="ordersTable">
            <tr>
                <th>Имя заказчика</th>
                <th>Город</th>
                <th>Заказов</th>
                <th>Сумма</th>
                <th></th>
            </tr>
            <asp:Repeater ID="Repeater1" runat="server" SelectMethod="GetOrder
                ItemType="GameStore.Models.Order">
                <ItemTemplate>
                    <tr>
                        <td><%#: Item.Name %></td>
                        <td><%#: Item.City %></td>
                        <td><%# Item.OrderLines.Sum(ol => ol.Quantity) %></td>
                        <td><%# Total(Item.OrderLines).ToString("C") %> </td>
                        <td>
                            <asp:PlaceHolder ID="PlaceHolder1" Visible="<%# !I
                                <button type="submit" name="dispatch"
                                    value="<%# Item.OrderId %>">
                                    Отправить в службу поддержки</button>
                            </asp:PlaceHolder>
                        </td>
                    </tr>
                </ItemTemplate>
            </asp:Repeater>
        </table>
    </div>

    <div id="ordersCheck">
        <asp:CheckBox ID="showDispatched" runat="server" Checked="false" AutoP
        Показать отправленные в службу поддержки заказы?
    </div>
</asp:Content>
```

Центральной частью этой веб-формы является элемент управления Repeater.

Мы применяем поддержку привязки данных в Web Forms, выраженную через атрибуты ItemType и SelectMethod, для получения набора объектов данных Order внутри класса отделенного кода и затем повторяем раздел разметки для каждого объекта. Генерируемым разделом разметки будет строка в HTML-элементе table с ячейками, которые отображают детали заказа.

Обратите внимание, что с помощью директивы Page отключено средство состояния представления для всей веб-формы.

Пояснить контент этой веб-формы проще всего, показав финальный результат, а затем постепенно возвращаться назад. На рисунке ниже показана веб-форма Order.aspx, отображающая примеры заказов, которые были добавлены в базу данных. (Вы не получите результат, представленный на рисунке, до тех пор, пока не определите класс отделенного кода, который вскоре будет описан.)



Выражения привязки данных

Чтобы продемонстрировать возможность помещения значений из объектов данных в HTML-разметку, мы применяли множество различных выражений привязки данных, для свойств Name и City использовались закодированные фрагменты кода привязки данных наподобие следующих:

```
<td><%#: Item.Name %></td>
<td><%#: Item.City %></td>
```

Обратите внимание на наличие двоеточия (:) после символа # в дескрипторе фрагмента кода. Закодированные фрагменты кода привязки данных работают таким же образом, как обычные фрагменты подобного рода — свойство Item ссылается на текущий объект данных, обрабатываемый элементом управления Repeater. Показанные фрагменты вставляют значения свойств Name и City внутрь элементов td для формирования части строки таблицы, причем они гарантируют, что значения данных безопасны для отображения в браузере. Это является рекомендуемым приемом при отображении любых данных, которые поступают из неизвестного или ненадежного источника, включая пользователей.

Мы применяем обычные фрагменты кода привязки данных для данных, которым доверяем — в приложении GameStore мы решили доверять данным, поступающим из таблицы Games. (Однако в реальных проектах следует проявлять намного большую осторожность в отношении доверия к данным, поэтому мы рекомендуем широко пользоваться закодированными фрагментами кода.)

Для обработки значений данных можно также применять LINQ, например:

```
<td><%# Item.OrderLines.Sum(ol => ol.Quantity) %></td>
```

В этом фрагменте используется расширяющий метод LINQ по имени Sum() для вычисления суммы значений свойств Quantity по всем объектам OrderLine, ассоциированным с объектом Order, который обрабатывается элементом управления Repeater, что дает в результате общее количество заказанных товаров.

Внутри фрагментов кода привязки данных можно также вызывать методы, как показано ниже:

```
<td><%# Total(Item.OrderLines).ToString("C") %> </td>
```

Здесь вызывается метод Total(), которому передается в качестве аргумента свойство Item (метод Total() будет скоро определен в классе отделенного кода). В отношении результата вызывается метод ToString(), чтобы сформатировать значение как денежную сумму. Как видите, фрагмент кода привязки данных предоставляет свойство Item, при этом доступна высокая гибкость в плане того, что с ним можно делать.

Привязка данных и заполнители

Мы хотим предоставить пользователю возможность выбора между просмотром всех заказов или только тех, которые не были отправлены. Необходимо лишь отобразить кнопку "Отправить в службу поддержки" для неотправленных заказов.

Мы добились этого за счет применения привязки данных посредством элемента

управления PlaceHolder. Элемент управления PlaceHolder — это оболочка вокруг раздела контента, который будет вставлен в результирующую HTML-разметку, если свойство Visible равно true. Значение свойства Visible устанавливается с использованием фрагмента кода привязки данных:

```
<td>
    <asp:PlaceHolder ID="PlaceHolder1" Visible="<%# !Item.Dispatched %>" runat
        <button type="submit" name="dispatch"
            value="<%# Item.OrderId %>">
            Отправить в службу поддержки</button>
    </asp:PlaceHolder>
</td>
```

Элемент управления PlaceHolder содержит кнопку "Отправить в службу поддержки", щелчок на которой приводит к отправке формы серверу. Мы применяли еще один фрагмент кода привязки данных для установки атрибута value в элементе button, так что можно определить, какой заказ был отправлен по назначению. Посредством этих фрагментов кода можно получить представление о важности данных и привязки данных в проектах Web Forms.

Элемент управления CheckBox

Последней частью веб-формы, к которой мы хотим привлечь внимание, является элемент управления CheckBox:

```
<div id="ordersCheck">
        <asp:CheckBox ID="showDispatched" runat="server" Checked="false" AutoP
        Показать отправленные в службу поддержки заказы?
</div>
```

Элемент управления CheckBox представляет собой удобный способ генерации элемента input с атрибутом type, установленным в checkbox. Мы могли бы просто добавить элемент input непосредственно, как это делалось с другими HTML-элементами, но элемент управления CheckBox обладает парой полезных средств.

Первое средство конфигурируется с помощью атрибута AutoPostBack. В случае если этот атрибут имеет значение true, элемент управления CheckBox добавляет к HTML-разметке, посылаемой браузеру, простой код JavaScript, который автоматически отправляет форму, когда пользователь отмечает или снимает отметку с флажка. Эта задача легко решается с помощью JavaScript-библиотек, подобных jQuery, но мы хотим отразить дополнительную пользу, обеспечиваемую элементами управления от Microsoft. Второе полезное средство элемента управления CheckBox относится к привязке данных и позволяет фильтровать отображаемые объекты данных в классе отделенного кода, который описан в

следующем разделе.

Создание класса отделенного кода

Класс отделенного кода, определенный в файле `\Pages\Admin\Orders.aspx.cs` довольно прост. В нем применяются в основном средства, которые были представлены в предыдущих статьях. Код этого класса отделенного кода показан в примере ниже:

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

namespace GameStore.Pages.Admin
{
    public partial class Orders : System.Web.UI.Page
    {
        private Repository repository = new Repository();

        protected void Page_Load(object sender, EventArgs e)
        {
            if (IsPostBack)
            {
                int dispatchID;
                if (int.TryParse(Request.Form["dispatch"], out dispatchID))
                {
                    Order myOrder = repository.Orders.Where(o => o.OrderId ==
                    if (myOrder != null)
                    {
                        myOrder.Dispatched = true;
                        repository.SaveOrder(myOrder);
                    }
                }
            }
        }

        public IEnumerable<Order> GetOrders([Control] bool showDispatched)
        {
            if (showDispatched)
            {
                return repository.Orders;
            }
            else
            {
                return repository.Orders.Where(o => !o.Dispatched);
            }
        }
```

В методе Page_Load() с помощью свойства IsPostBack обнаруживаются запросы POST, а через коллекцию Request.Form выясняется, на какой кнопке был совершен щелчок и, следовательно, какой заказ был отправлен по назначению. Затем из запроса извлекается значение, из хранилища получается соответствующий объект данных Order и выполняется обновление.

Метод Total() вычисляет общую стоимость заказа. Этот метод вызывается в одном из фрагментов кода привязки данных внутри веб-формы; он выполняет простое вычисление и возвращает результат.

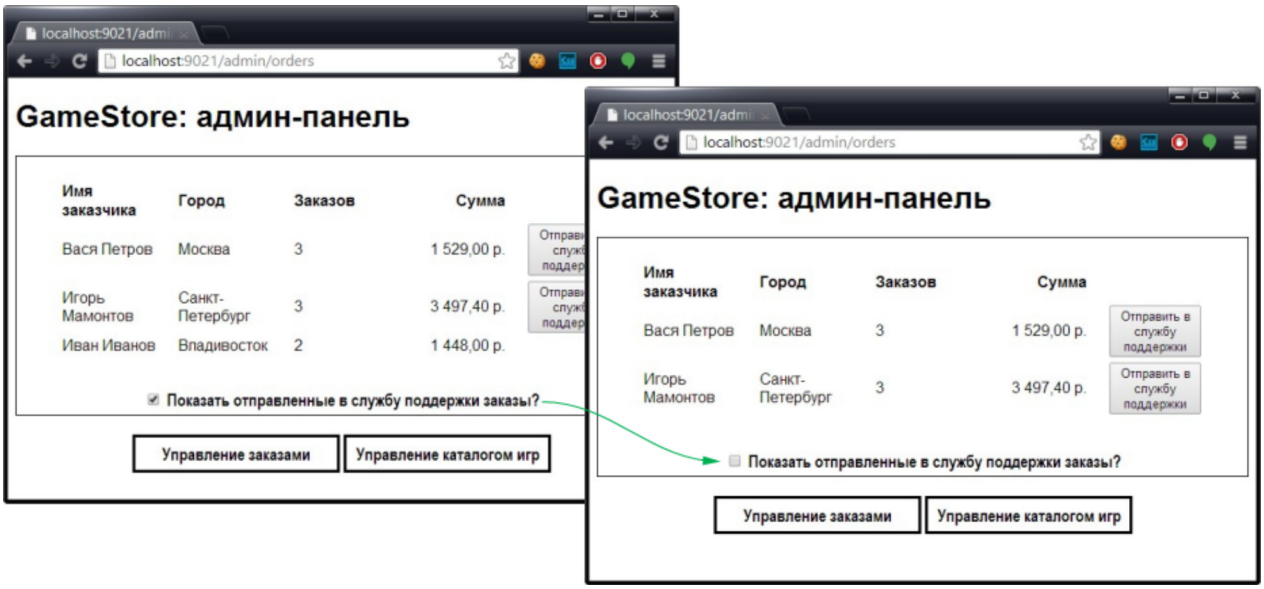
Мы хотим привлечь ваше внимание к методу GetOrders(), который используется в качестве значения атрибута SelectMethod при конфигурировании элемента управления Repeater внутри веб-формы. Взгляните на сигнатуру этого метода:

```
public IEnumerable<Order> GetOrders([Control] bool showDispatched)
```

В сигнатуре определен параметр типа bool, который применяется для определения какие объекты Order должны возвращаться. Элементу управления Repeater не известно, какие данные нам нужны — тогда откуда поступает значение для этого параметра? Ответ заключается в еще одном полезном средстве привязки данных — атрибуте Control.

Атрибут Control сообщает ASP.NET Framework о необходимости извлечь значение из элемента управления, определенного в веб-форме. В этом случае значение параметра берется из showDispatched, который представляет собой элемент управления CheckBox, добавленный в файле Orders.aspx. Никакого дополнительного кода для получения значения из элемента управления CheckBox и передачи его в качестве параметра не понадобится - все происходит автоматически, когда применяется атрибут Control. Чтобы увидеть как это работает, запустите приложение и перейдите на URL вида /admin/orders. Обеспечьте отправку по назначению одного из заказов (щелкнув на соответствующей кнопке "Отправить в службу поддержки") и затем отметьте и снимите отметку с флажка "Показать отправленные в службу поддержки заказы?".

Когда применяется этот флажок, форма отправляется автоматически, а данные, посланные серверу, содержат новую установку флажка. Новое значение флажка используется как аргумент метода GetOrders(), который действует в качестве фильтра для данных, передаваемых элементу управления Repeater и впоследствии возвращается пользователю. В конечном итоге получается простой и элегантный способ предоставления пользователям контроля над данными, которые они просматривают, с минимальным написанием кода. Результат можно видеть на следующем рисунке:



Привязка данных - невероятно полезное средство.

Apper SIM - EDI-plattform

Strukturerad kommunikation oavsett storlek på partner.