Обработка заказов

APPER SIM - EDI-PLATTFORM

apper.com

Strukturerad kommunikation oavsett storlek på partner.

Исходный код проекта

Мы добрались до финального пользовательского средства в приложении GameStore — возможности проверки и завершения формирования заказа. В последующих разделах мы расширим проект для предоставления поддержки ввода пользователем сведений о доставке и средств обработки этих сведений.

Расширение базы данных и модели данных

Мы собираемся добавить в базу данных дополнительные таблицы для хранения сведений о пользовательском заказе, а также информации о доставке. Чтобы обновить базу данных, откройте окно Server Explorer (Проводник баз данных) в Visual Studio, щелкните правой кнопкой мыши на элементе EFDbContext (GameStore) и выберите в контекстном меню пункт New Query (Новый запрос). В текстовой области открывшегося окна введите операторы SQL, приведенные ниже:

```
CREATE TABLE Orders (
        [OrderId] INT Identity NOT NULL,
        [Name] NVARCHAR(MAX) NULL,
        [Line1] NVARCHAR(MAX) NULL,
        [Line2] NVARCHAR(MAX) NULL,
        [Line3] NVARCHAR(MAX) NULL,
        [City] NVARCHAR(MAX) NULL,
        [GiftWrap] BIT NOT NULL,
        [Dispatched] BIT NOT NULL,
        CONSTRAINT [PK_dbo.Orders] PRIMARY KEY CLUSTERED ([OrderId] ASC)
);

CREATE TABLE OrderLines (
        [OrderLineId] INT IDENTITY NOT NULL,
        [Quantity] INT NOT NULL,
        [Game_GameID] INT NULL,
        [Order_OrderId] INT NULL,
        CONSTRAINT [PK_dbo.OrderLines] PRIMARY KEY CLUSTERED ([OrderLineId] AS
        CONSTRAINT [FK_dbo.OrderLines_dbo.Games_GameID] FOREIGN KEY
                ([Game_GameID]) REFERENCES [dbo].[Games] ([GameID]),
        CONSTRAINT [FK_dbo.OrderLines_dbo.Order_OrderId] FOREIGN KEY
                ([Order_OrderId]) REFERENCES [dbo].[Orders] ([OrderId])
);
```

После ввода операторов SQL щелкните правой кнопкой мыши на текстовой области и выберите в контекстном меню пункт Execute (Выполнить). Затем щелкните на значке Refresh (Обновить) в окне Server Explorer. Вы увидите, что в базу данных были добавлены две новых таблицы, OrderLines и Orders, как показано на рисунке ниже:

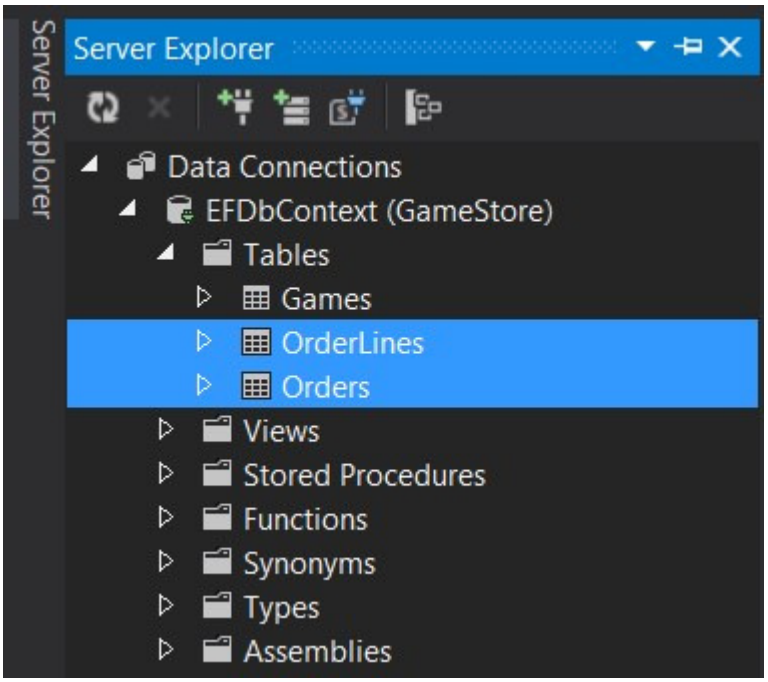


Таблица `Orders` будет предназначена для хранения информации о доставке заказа, а таблица `OrderLines` — для хранения сведений о товарах, входящих в заказ. Между таблицей `OrderLines` и таблицами `Orders` и `Games` определены отношения внешнего ключа, упрощающие работу с данными.

Добавление классов модели данных

Мы должны создать в папке `Models` классы для представления строк в таблицах `Orders` и `OrderLines`. Для этого мы создаем новый файл класса `Order.cs` с содержимым, показанным в примере ниже:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;

namespace GameStore.Models
{
    public class Order
    {
        public int OrderId { get; set; }
        public string Name { get; set; }
        public string Line1 { get; set; }
        public string Line2 { get; set; }
        public string Line3 { get; set; }
        public string City { get; set; }
        public bool GiftWrap { get; set; }
        public bool Dispatched { get; set; }
        public virtual List<OrderLine> OrderLines { get; set; }
    }

    public class OrderLine
    {
        public int OrderLineId { get; set; }
        public Order Order { get; set; }
        public Game Game { get; set; }
        public int Quantity { get; set; }
    }
}
```

Классы `Order` и `OrderLine` определены в одном и том же файле. Для выражения отношений между таблицами через свойства объектов применялись средства `Entity Framework`, поэтому для хранения ключей строк из таблиц `Games` и `Orders` в классе `OrderLine` определены свойства типа `Game` и `Order`, а не значения `int`. Инфраструктура `Entity Framework` будет автоматически использовать внешние ключи для нахождения строк в других таблицах и представления их с помощью объектов `C#`. Применение ключевого слова `virtual` к свойству `OrderLines` в классе `Order` приводит к тому, что `Entity Framework` загружает все строки `OrderLine`, которые ассоциированы с заказом, и представляет их в виде списка объектов `OrderLine`.

Расширение классов контекста и хранилища

Поддержка новых классов модели данных должна быть также добавлена в

классы контекста и хранилища. В примере ниже показаны изменения, внесенные в файл класса \Models\Repository\EFDbContext.cs:

```
using System.Data.Entity;

namespace GameStore.Models.Repository
{
    public class EFDbContext : DbContext
    {
        public DbSet<Game> Games { get; set; }
        public DbSet<Order> Orders { get; set; }
    }
}
```

Мы добавили новое свойство по имени Orders для поддержки таблицы Orders. Свойство для таблицы OrderLines добавлять не нужно, т.к. работа с ней напрямую не производится. Способ поддержки инфраструктурой Entity Framework отношений внешнего ключа означает, что объекты OrderLine будут обрабатываться автоматически через объекты Order, с которыми они ассоциированы.

Имея определенное свойство, которое Entity Framework будет применять для предоставления доступа к таблице Orders, можно обновить файл класса \Models \Repository\Repository.cs, добавив возможность чтения и записи объектов Order и OrderLine. Изменения представлены в примере ниже:

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

namespace GameStore.Models.Repository
{
    public class Repository
    {
        private EFDbContext context = new EFDbContext();

        public IEnumerable<Game> Games
        {
            get { return context.Games; }
        }

        // Чтение данных из таблицы Orders
        public IEnumerable<Order> Orders
        {
            get
            {
                return context.Orders
                    .Include(o => o.OrderLines.Select(ol => ol.Game));
            }
        }

        // Сохранить данные заказа в базе данных
```

Свойство `Orders` возвращает перечисление строк из таблицы `Orders` базы данных, в котором каждая строка представлена объектом `Order`. Методы `Include()` и `Select()` обеспечивают загрузку объектов `Game`, связанных с каждым объектом `OrderLine`, при запрашивании базы данных.

Метод `SaveOrder()` позволяет сохранять новые или модифицировать существующие объекты `Order`. Новые объекты `Order`, которые еще ни разу не сохранялись, могут быть обнаружены по значению их свойства `OrderId`, равному нулю. Объекты `Order`, созданные для представления существующих строк таблицы, будут иметь ненулевое значение `OrderId`, назначенное сервером базы данных.

В конечном итоге мы получаем довольно естественный способ работы со строками и отношениями между таблицами посредством объектов `C#`. Знание того, каким образом `Entity Framework` функционирует, необходимо при обновлении классов контекста и хранилища, но эти детали скрыты для остальной части приложения `GameStore`. Это позволяет по своему усмотрению корректировать настройку инфраструктуры `Entity Framework` (или полностью заменить ее другой системой `ORM`), изменяя всего пару классов.

Добавление ссылки на оплату и URL

Мы должны предоставить пользователям способ перехода к оплате за товары в корзине, для чего понадобится добавить к веб-форме CartView.aspx ссылку, которая будет направлять на начало процесса оплаты.

Необходимо, чтобы ссылка, по которой будут следовать пользователи, соответствовала созданной схеме URL, поэтому придется расширить существующую конфигурацию маршрутизации. В примере ниже можно видеть новый URL, определенный 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
        }
    }
}
```

Новый оператор отображает URL вида /checkout на веб-форму под названием Checkout.aspx из папки Pages — эта веб-форма пока не существует, но вскоре будет создана.

Теперь, имея маршрут, который сгенерирует желаемый URL, можно добавить к корзине ссылку Checkout (Оплата). В примере ниже представлено изменение, внесенное в файл веб-формы \Pages\CartView.aspx:

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

<asp:Content ID="Content1" ContentPlaceHolderID="bodyContent" runat="server">
    <div id="content">
        <h2>Ваша корзина</h2>
          ...
        <p class="actionButtons">
            <a href="<%= ReturnUrl %>">Продолжить покупки</a>
            <a href="<%= CheckoutUrl %>">Оформить заказ</a>
        </p>
    </div>
</asp:Content>
```

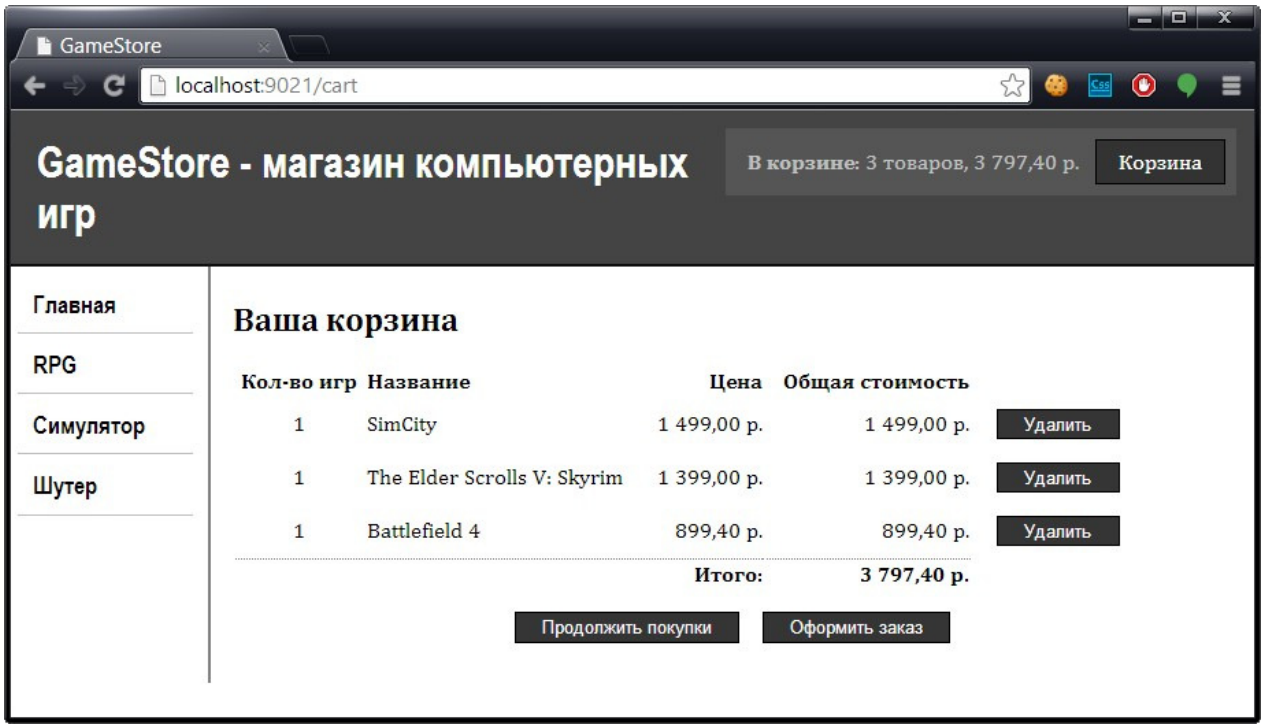
В этом примере не показано полное содержимое файла `CartView.aspx`, поскольку изменение очень мало — мы добавили элемент `<a>`, атрибут `href` которого установлен равным значению нового свойства `CheckoutUrl` класса отделенного кода. Определение свойства `CheckoutUrl` внутри файла отделенного кода приведено в примере ниже:

```
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
    {
        // ...

        public string CheckoutUrl
        {
            get
            {
                return RouteTable.Routes.GetVirtualPath(null, "checkout",
                    null).VirtualPath;
            }
        }
    }
}
```

Значение свойства генерируется из конфигурации маршрутизации с помощью того же приема, который использовался для предыдущих ссылок. Благодаря этой системе, схему URL для приложения можно изменять, просто модифицируя конфигурацию маршрутизаиции и не затрагивая веб-формы, пользовательские элементы управления и классы отделенного кода. Ссылка "Оформить заказ", стилизованная в виде кнопки, представлена на рисунке:

Apper SIM - EDI-plattform

Strukturerad kommunikation oavsett storlek på partner.



Обработка заказа

Для поддержки процесса оплаты мы создали в папке `Pages` новый файл веб-формы по имени `Checkout.aspx`. Его содержимое показано в примере ниже:

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

<asp:Content ID="Content1" ContentPlaceHolderID="bodyContent" runat="server">
    <div id="content">

        <div id="checkoutForm" class="checkout" runat="server">
            <h2>Оформить заказ</h2>
            Пожалуйста, введите свои данные, и мы отправим Ваш товар прям

        <div id="errors" data-valmsg-summary="true">
            <ul>
                <li style="display:none"></li>
            </ul>
            <asp:ValidationSummary ID="ValidationSummary1" runat="server" />
        </div>

            <h3>Заказчик</h3>
            <div>
                <label for="name">Имя:</label>
                <input id="name" name="name" runat="server" />
            </div>

            <h3>Адрес доставки</h3>
            <div>
                <label for="line1">Адрес 1:</label>
                <input id="line1" name="line1" runat="server" />
            </div>
            <div>
                <label for="line2">Адрес 2:</label>
                <input id="line2" name="line2" runat="server" />
            </div>
            <div>
                <label for="line3">Адрес 3:</label>
                <input id="line3" name="line3" runat="server" />
            </div>
            <div>
                <label for="city">Город:</label>
                <input id="city" name="city" runat="server" />
            </div>
```

Эта веб-форма состоит из двух разделов контента, каждый из которых содержится внутри элемента `div` с атрибутом `runat`, установленным в `server`, так что элементами можно манипулировать в классе отделенного кода. Один раздел содержит элементы `input`, которые собирают требуемую информацию от пользователя, а другой раздел предназначен для отображения простого сообщения, когда процесс оплаты завершен.

Во всех остальных отношениях применяется стандартная разметка для создания HTML-формы. (В качестве напоминания: элемент `form` определен внутри мастер-страницы, которая была создана ранее.) Стилизация созданного элемента осуществляется с помощью стилей, добавленных в файл `\Content\Styles.css`:

```
.checkout label {
    display: inline-block;
    width: 60px;
    text-align: right;
}
.checkout div input {
    width: 200px;
    margin: 2px;
}
#errors { color: red;}
```

В примере ниже приведено содержимое файла отделенного кода `\Pages\Checkout.aspx.cs` с обработкой запросов веб-формы `Checkout`:

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

namespace GameStore.Pages
{
    public partial class Checkout : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            checkoutForm.Visible = true;
            checkoutMessage.Visible = false;

            if (IsPostBack)
            {
                Order myOrder = new Order();
                if (TryUpdateModel(myOrder,
                   new FormValueProvider(ModelBindingExecutionContext)))
                {

                    myOrder.OrderLines = new List<OrderLine>();

                    Cart myCart = SessionHelper.GetCart(Session);

                    foreach (CartLine line in myCart.Lines)
                    {
                        myOrder.OrderLines.Add(new OrderLine
                        {
                            Order = myOrder,
                            Game = line.Game,
                            Quantity = line.Quantity
                        });
                    }

                    new Repository().SaveOrder(myOrder);
                    myCart.Clear();

                    checkoutForm.Visible = false;
                    checkoutMessage.Visible = true;
```

Запросы POST обнаруживаются с применением свойства IsPostBack. Затем с помощью привязки модели создается объект Order на основе данных, отправленных пользователем в форме. Если привязка модели работает, мы получаем объект Cart из данных Session и строим объекты OrderLine для всех экземпляров CartLine, созданных пользователем. После этого объект Order записывается в хранилище с применением метода SaveOrder(), который был определен ранее. После помещения нового объекта Order в хранилище контент объекта Cart очищается, так что пользователь может начать покупку заново.

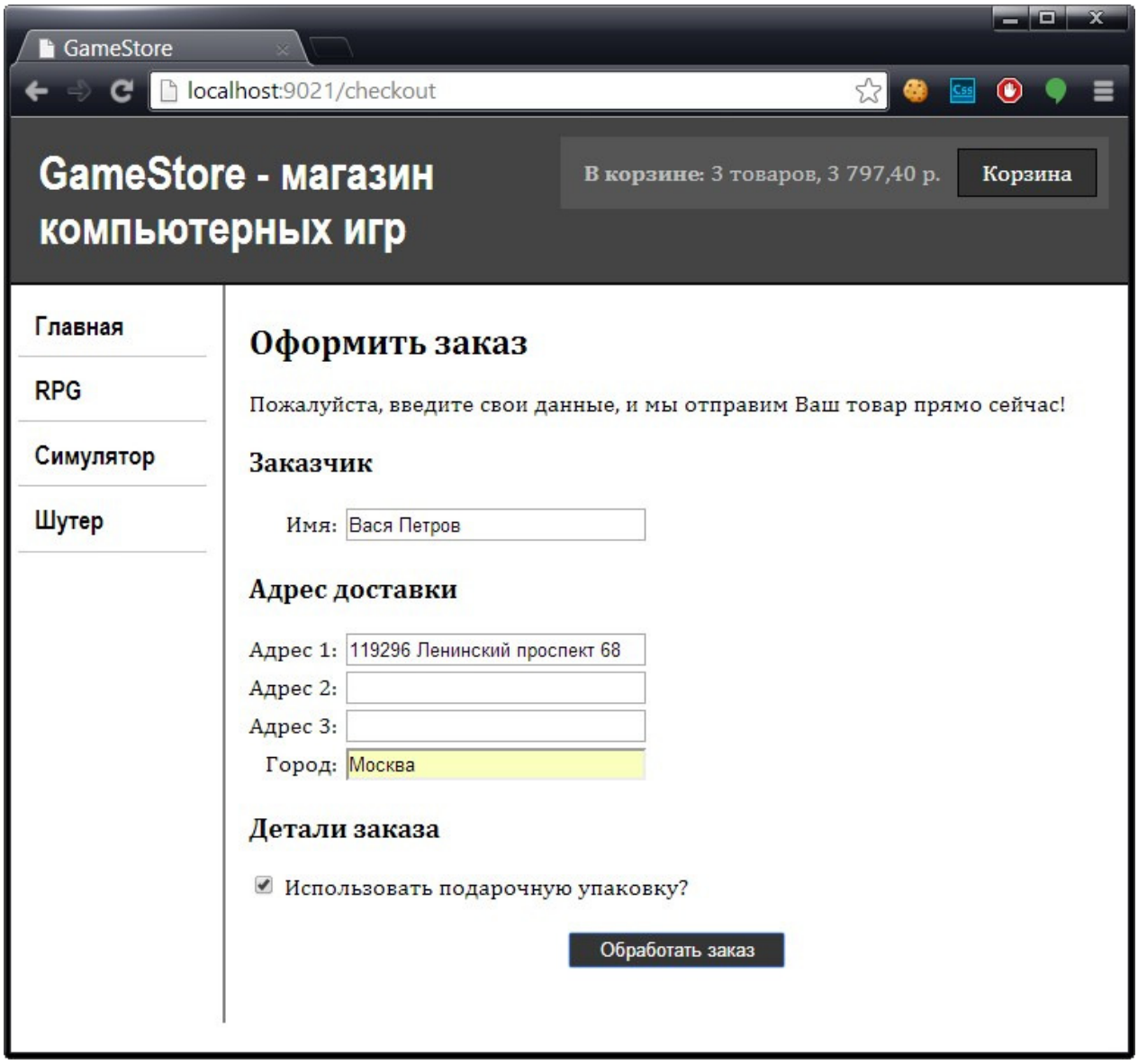
Вы наверняка заметили, что объекты CartLine и OrderLine во многом похожи. Приложив небольшие дополнительные усилия, мы могли бы использовать объект CartLine для представления элементов, ассоциированных с заказом, однако мы предпочитаем иметь отдельные объекты модели данных, даже если это означает реализацию отображения, показанного в примере. В случае применения объектов модели данных для множества ролей (вроде представления строк в корзинах и в заказах) код станет лаконичнее, но когда понадобится изменить способ представления данных из базы — кстати, на удивление распространенный тип изменений — возникнут затруднения. Мы считаем, что лучше реализовать отображение между объектами, такими как CartLine и OrderLine, чем пытаться выйти за рамки ролей, которые исполняет объект, особенно если такой объект используется повсеместно в приложении.

Обратите внимание, что для выяснения, какой контент отображается пользователю, применяется свойство Visible. Мы показываем пользователю раздел, содержащий элементы input, когда страница запрашивается первый раз, и скрываем другой раздел контента:

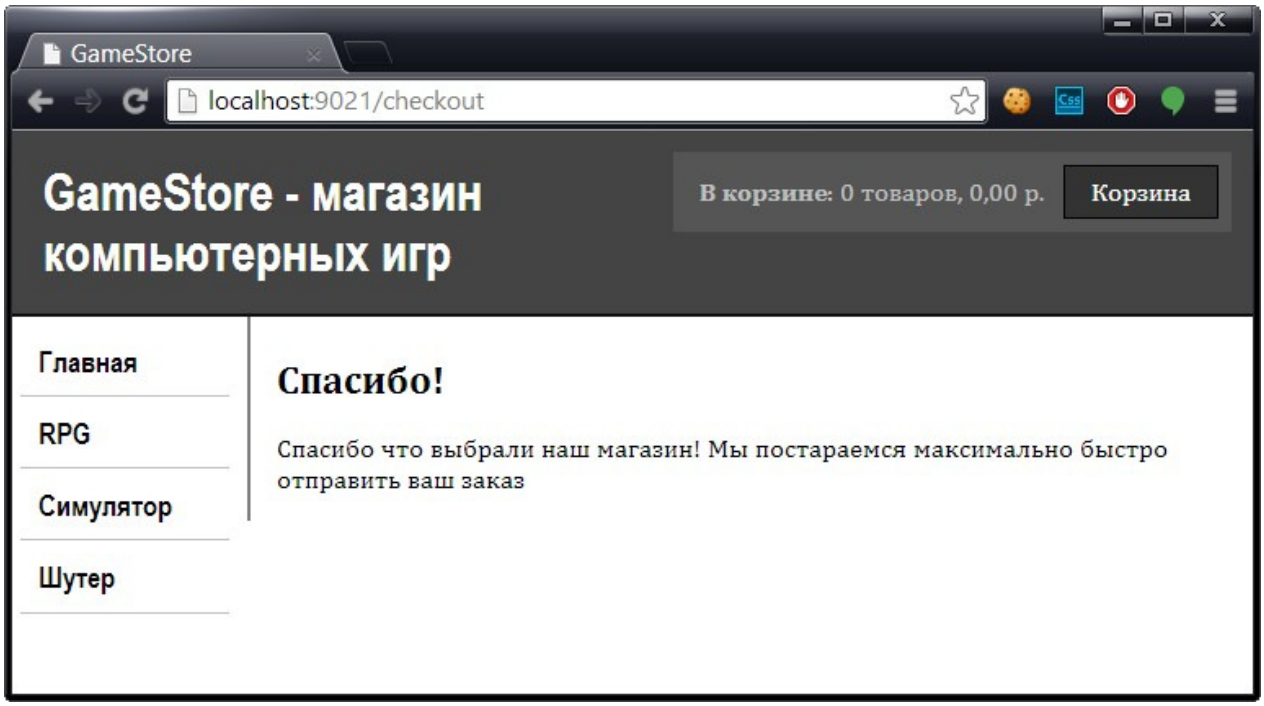
```
...
checkoutForm.Visible = true;
checkoutMessage.Visible = false;
...
```

Когда свойство Visible установлено в false, среда ASP.NET Framework не включает элемент или его контент в ответ, отправляемый браузеру. Это распространенная причина путаницы у разработчиков, которые используют код JavaScript для манипулирования DOM-моделью HTML в браузере, где признак видимости просто скрывает элементы, не удаляя их. В случае применения свойства Visible в классе отделенного кода элементы вообще не отправляются.

Чтобы протестировать новую функциональность, запустите приложение, добавьте товары в корзину и щелкните на кнопке Checkout (Оплата). Первым делом, вы увидите поля формы, представленные на рисунке ниже:



После щелчка на кнопке "Обработать заказ" форма отправляется серверу, данные записываются в базу и видимость элементов контента изменяется так, что появляется сообщение, показанное на рисунке ниже:



Вопросы оплаты заказов здесь не рассматриваются. В ASP.NET Framework отсутствуют встроенные средства для поддержки оплаты, поэтому добавление любой службы платежей свелось бы к упражнению по интеграции одного поставщика такой службы. Доступно множество разных вариантов оплаты, и все они функционируют по-разному. Чтобы сосредоточить внимание на платформе ASP.NET Framework, приложение GameStore будет позволять пользователям размещать заказы без необходимости в предоставлении деталей, связанных с оплатой.

Silverlight Doc/Docx File

Silverlight Read/Create/Edit Doc(x) Professional Silverlight Component