Работа с Razor

Работа с Razor Razor – это движок представления, который Microsoft представил в MVC 3 и который был немного переделан в MVC 4 (хотя изменения являются относительно незначительными). Движок представления обрабатывает ASP.NET контент и ищет инструкции, как правило, для вставки динамического контента в выходные данные, отправленные браузеру. Microsoft поддерживает два вида движков: движок ASPX работает с тегами <% и %>, которые являлись основой развития ASP.NET в течение многих лет. А движок Razor, который работает с отдельными областями контента, обозначается символом @. По большому счету, если вы знакомы с синтаксисом <% %>, Razor не доставит слишком много проблем, хотя и есть несколько новых правил. В этом разделе мы дадим вам краткий обзор синтаксиса Razor, так что вы сможете распознать новые элементы, когда вы их встретите. Мы не собираемся углубленно изучать Razor в этой главе, скорее это ускоренный курс синтаксиса. Более подробно мы изучим Razor далее в этой книге.

Define and access the model type Use the @model and @Model expressions	Определение и доступ к типу модели Используйтемodel и выражениямodel
Reduce duplication in views Use a layout	Избежание дублирования видом Используйте макет
Specify a default layout Use the view start view	Укажите расположение по умолчанию Используйте Просмотр начала
Pass data values to the view from the controller Pass a view model object or the view bag	Передача значений данных на вид из контроллера Передать объект модели представления или представления мешок
Generate different content based on data values Use Razor conditional statements	Генерируют различное содержимое на основе значений данных Используйте Razor условных операторов
Enumerate an array or a collection Use a @foreach expression	Перечислять массив или коллекцию Используйте выражениеforeach
Add a namespace to a view Use a @using expression	Добавить пространство имен для зрения Используйтеusing выражение

Подготовка проекта для примера

Для демонстрации возможностей и синтаксиса Razor мы создали новый проект Visual Studio, используя шаблон ASP.NET MVC 4 Web Application и выбрав вариант Empty. Определение модели

Разработка модели

Мы будем использовать очень простую доменную модель и воспроизведем тот же доменный класс Product, который мы использовали в первой части книги. Добавьте файл класса Product.cs в папку Models и убедитесь, что содержание соответствует тому, что показано в следующем листинге.

```
. . .
namespace WebApplication1.Models
{   public class Product
```

```
    {   public int ProductID { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public decimal Price { get; set; }
        public string Category { set; get; }
    }
}
```

Определение контроллера

Чтобы добавить контроллер в проект, щелкните правой кнопкой мыши в вашем проекте по папке Controllers и выберите Add, а затем Controller, из всплывающего меню. Назовите его HomeController и выберите Empty MVC Controller в опции Template. Измените содержимое файла, чтобы оно соответствовало следующему листингу.

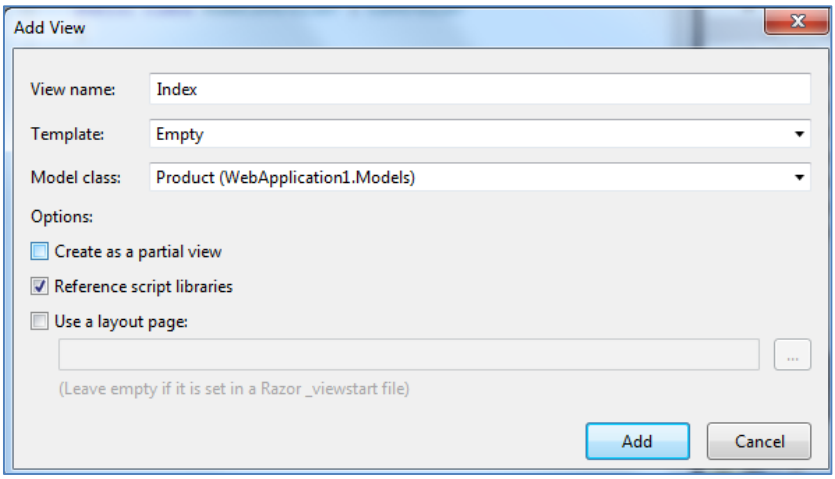
```
...
using WebApplication1.Models;
namespace WebApplication1.Controllers
{
    public class HomeController : Controller
    {
        Product myProduct = new Product
        {   ProductID = 1,
            Name = "Kayak",
            Description = "A boat for one person",
            Category = "Watersports",
            Price = 273
    };
        public ActionResult Index()
        {  return View(myProduct);
        }
}
}
```

Мы определили метод действия Index, в котором мы создаем и заполняем свойства объекта Product. Мы передаем Product методу View, поэтому он используется в качестве модели, когда воспроизводится представление. Мы не указываем имя файла представления, когда вызываем метод View, поэтому для метода действия будет использоваться представление по умолчанию.

Создание представления

Чтобы создать представление, щелкните правой кнопкой мыши по методу Index класса HomeController и выберите Add View. Проверьте опцию, что создается строго типизированное представление и выберите класс из раскрывающегося списка Product, как показано на рисунке 5-2.

Примечание. Если вы не видите в раскрывающемся списке класс Product, скомпилируйте ваш проект и попробуйте создать представление еще раз. Visual Studio не распознает классы модели, пока они не скомпилированы.



Пример добавление представления.

Убедитесь, что отключена опция для использования макета и мастер страницы (use a layout or master page), как показано на рисунке. Нажмите кнопку Add, чтобы создать представление, которое появится в папке Views/Home и будет называться Index.cshtml. Файл представления будет открыт для редактирования, и вы увидите, что это тот же базовый файл представления, который мы создавали в предыдущей главе, как показано в следующем листинге.

```
@model WebApplication1.Models.Product
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
    </div>
</body>
</html>
```

При изучении Razor важно понимать, что представления существуют, чтобы выразить одну или несколько частей модели пользователю: а это обозначает генерирование HTML, который отображает данные, полученные от одного или нескольких объектов. Если вы запомните, что мы всегда пытаемся выстроить HTML страницу, которая может быть отправлена клиенту, то все, что делает Razor, будет иметь для вас смысл.

Работа с объектом модели

Давайте начнем с первой строки в представлении:

```
@model WebApplication1.Models.Product
```

Операторы Razor начинаются с **символа @**. В данном случае выражение **@model** объявляет тип объекта модели, который мы передадим в представление из метода действия. Это позволяет нам обратиться к методам, полям и свойствам объекта модели представления через @Model, как показано в листинге ниже. Здесь представлено простое дополнение к методу Index.

```
@model WebApplication1.Models.Product

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
        @Model.Name
    </div>
</body>
</html>
```

Примечание. Обратите внимание, что мы объявили тип объекта модели представления при помощи @model, а получили доступ к свойству Name при помощи @Model (заглавная M).

Если вы запустите проект, вы увидите результат, показанный на рисунке 1. Вам не нужно указывать конкретный URL, потому что по умолчанию в MVC проекте запрос для корневого URL (/) направляется к методу действия Index в контроллере Home (но есть возможность изменить).

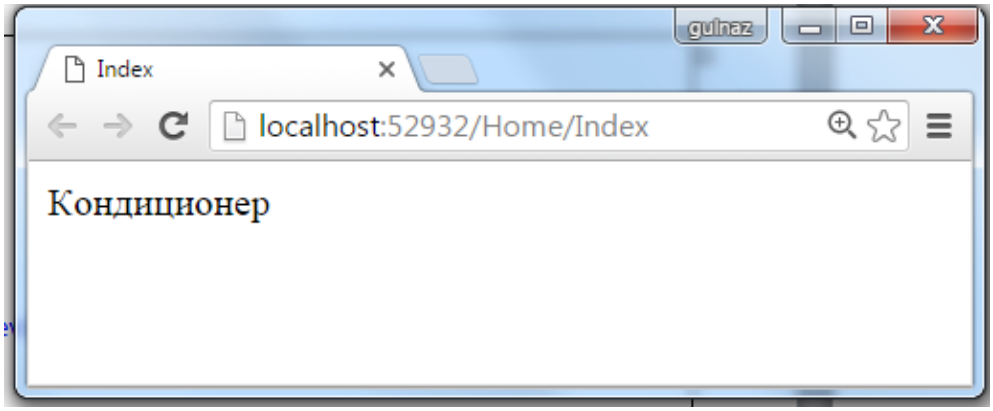
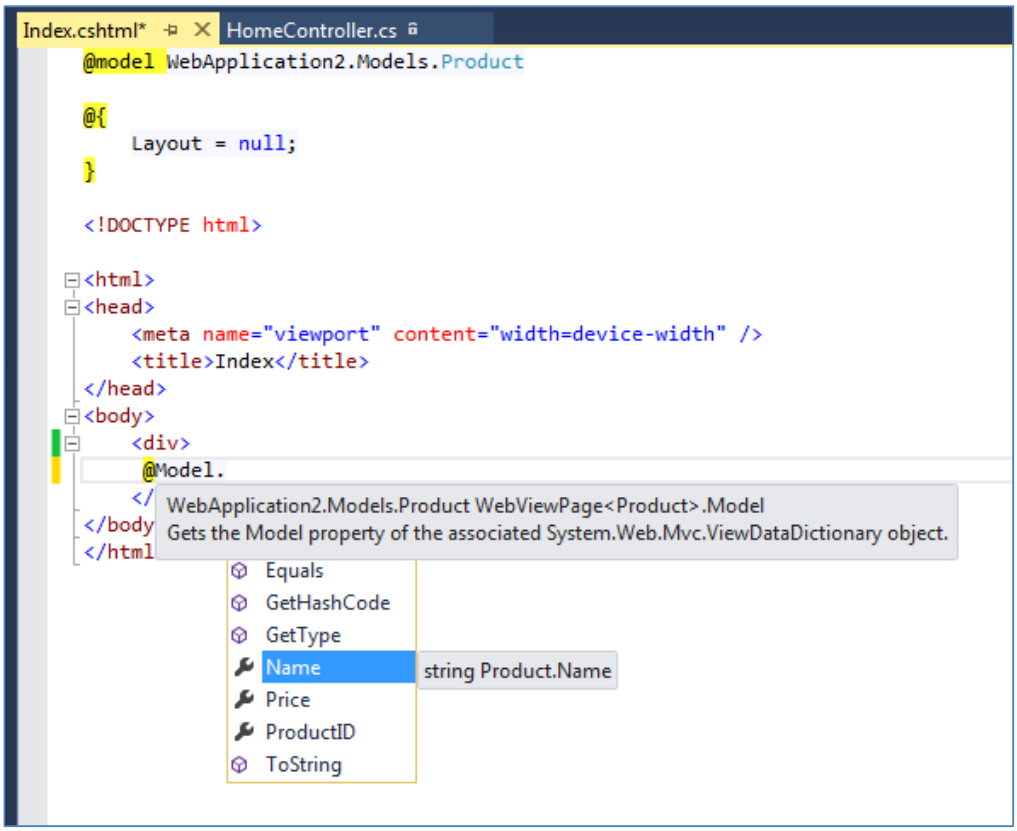


Рисунок - Результат прочтения значения свойства в представлении

При помощи выражения @model мы говорим MVC, с каким видом объекта мы будем работать, а Visual Studio использует это в несколькими способами. Во-первых, когда вы пишете код представления, Visual Studio покажет вам несколько вариантов, касательных имен членов объекта,если вы наберете @Model и поставите точку, как показано на следующем рисунке.



Работа с макетами

Другое Razor выражение в файле представления Index.cshtml следующее:

```
@{
    Layout = null;
}
```

Это пример блока кода Razor, который позволяет нам включать в представление C# выражения. Блок кода открывается @{ и закрывается }, а выражения, которые он содержит, обрабатываются тогда, когда отрабатывает представление.

Конкретно этот блок кода устанавливает значение свойства Layout на null. В MVC приложении представления Razor компилируются в C# классы, и используемый базовый класс определяет свойство Layout. Но результат установки свойства Layout на null заключается в том, что мы говорим MVC фреймоврку, что наше представление является автономным, и оно будет показывать все содержимое, которое мы должны вернуть клиенту.

Автономные представления хороши для простых приложений-примеров, но реальный проект может иметь множество представлений, и макеты являются эффективными шаблонами, которые содержат разметку, используемую для создания логичности и постоянства в веб приложении. Это может заключаться в том, что в приложение будут включены необходимые JavaScript библиотеки, или в создании общего гармоничного вида всего приложения.

Создание макета

Для создания макета щелкните правой кнопкой мыши по папке Views в Solution Explorer, нажмите на Add New Item в меню Add и выберите шаблон MVC 5 Layout Page (Razor):

Примечание. Файлы в папке Views, чьи имена начинаются с символа подчеркивания (_) не возвращаются пользователю. Это позволяет нам использовать имена файлов, чтобы различия представления, которые мы хотим показать, и файлы, которые их поддерживают. Имена макетов, которые являются поддерживающими файлами, начинаются с подчеркивания.

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
</head>
<body>
    <div>
        @RenderBody()
    </div>
</body>
</html>
```

Макеты являются специализированной формой представления, и вы видите, что мы выделили в листинге выражения с @. Вызов метода *@RenderBody вставляет содержимое представления*, указанного методом действия в разметку макета.

Другое Razor выражение в макете ищет свойство Title во *ViewBag для того, чтобы установить содержание элемента title*.

Любые элементы в макете будут применяться к любому представлению, которое использует макет, и именно поэтому макеты по существу являются шаблонами. В следующем листинге мы добавили немного простой разметки, чтобы показать, как это работает.

Добавление элементов в макет

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
</head>
<body>
    <h1>Информация о бытовой технике</h1>
    <div style="padding: 20px; border: solid medium black; font-size: 20pt">
        @RenderBody()
    </div>
    <h2>Visit <a href="http://apress.com">ССылка</a></h2>
</body>
</html>
```

Мы добавили пару элементов заголовков и применили некоторые CSS стили к элементу div, который содержит выражение @RenderBody, просто чтобы было понятно, какой контент приходит из макета, а какой из представления.

Применение макета

Чтобы применить макет к представлению, нам просто нужно установить значение свойства Layout. Мы также можем удалить элементы, которые обеспечивают структуру полной HTML страницы, потому что это будет выпадать из макета. Вы можете увидеть, как мы применили макет, в следующем листинге, который показывает радикально упрощенный файл Index.cshtml.

Мы также установили значение для свойства ViewBag.Title, которое будет использоваться в качестве контента элемента title в HTML документе, отправленном обратно пользователю: это не является обязательным, но это хорошая практика. Если значение для свойства не установлено, MVC фреймворк вернет пустой элемент title.

Файл Index.cshtml:

```
@model WebApplication2.Models.Product
@{
    ViewBag.Title = "Климатическая техника";
    Layout = "~/Views/_BasicLayout.cshtml";
}
Товар: @Model.Name
```

Преобразование довольно интересно, даже для такого простого представления. То, с чем мы остались, ориентировано на показе данных из объекта модели представления пользователю, а структура HTML документа исчезла.

Использование макетов имеет ряд преимуществ. Это позволяет упростить наши представления (что и показано в листинге), это позволяет нам создать общий HTML, который мы можем применить к нескольким представлениям, и, естественно, это сильно упрощает поддержку, потому что мы можем в одном месте изменить общий HTML и быть уверенными, что это изменение будет применяться везде, где используется макет. Чтобы увидеть результат использования макета, запустите приложение из примера. Результат представлен в следующем рисунке.

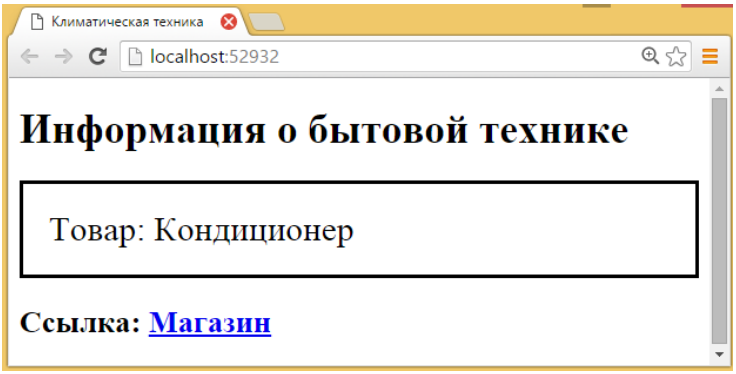


Рисунок - Результат применения простого макета к представлению

Использование файла _ViewStart.

Но есть еще кое-что, с чем надо разобраться: то есть, нам нужно указывать файл макета для каждого представления. Это обозначает, что если нам понадобится переименовать файл макета, нам нужно будет найти каждое представление, которое к нему относится, и внести изменения. Это будет процесс, полный ошибок, который идет вразрез с общей темой простоты обслуживания, являющейся фактически лозунгом MVC фреймворка.

Эту проблему можно решить при помощи файла _ViewStart.cshtml. При показе представления MVC фреймворк будет искать файл _ViewStart.cshtml. Содержимое этого файла будет рассматриваться так, как если бы оно содержалось в самом файле представления, и мы можем использовать эту функцию, чтобы автоматически устанавливать значение свойства Layout.

Чтобы создать файл _ViewStart.cshtml, добавьте новый файл макета в папку Views, повторив те действия, которые мы показали вам ранее. Назовите файл _ViewStart.cshtml и измените его так, чтобы он соответствовал следующему листингу.

```
@{
    Layout = "~/Views/_BasicLayout.cshtml";
}
```

Наш файл _ViewStart.cshtml содержит значение для свойства *Layout*, поэтому, можно удалить соответствующее выражение в файле Index.cshtml, как показано в листинге ниже.

Index.cshtml.

```
@model WebApplication2.Models.Product
@{
    ViewBag.Title = "Климатическая техника";
}
Товар: @Model.Name
```

Нам совсем не нужно указывать, что мы хотим использовать файл *_ViewStart.cshtml*. MVC фреймворк автоматически найдет файл и будет использовать его содержимое. Значения, определенные в файле представления, имеют преимущество, которое позволяет легко изменить файл _ViewStart.cshtml.

Примечание. Важно понимать разницу между тем, опускается ли свойство Layout в файле представления или устанавливается на null. Если ваше представление является автономным, и вы не хотите использовать макет, установите значение свойства Layout на null. Если же вы опустите свойство Layout, то MVC фреймворк посчитает, что вы хотите использовать макет и что он должен воспользоваться значением, которое найдет в файле _ViewStart.cshtml.

Демонстрация макетов с общим доступом

Чтобы быстро и просто показать, как предоставить общий доступ для макетов, мы добавили новый метод действия NameAndPrice в контроллер Home. Вы можете посмотреть определение этого метода в листинге 5-10, который демонстрирует изменения, внесенные в файл /Controllers/HomeController.cs.

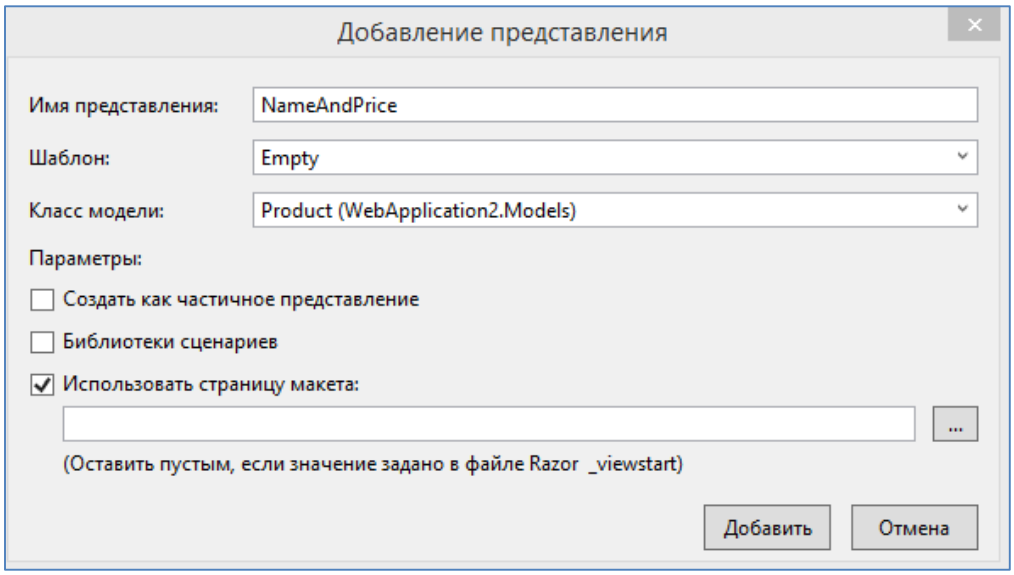
```
...
using WebApplication2.Models;

namespace WebApplication2.Controllers
{
    public class HomeController : Controller
    {
        Product myProduct = new Product
        {   ProductID = 1,
            Name = "Кондиционер",
            Description = "Кондиционер LG G09LH White",
            Category = "Бытовая техника",
            Price = 126990
        };

        public ActionResult Index()
        {
            return View(myProduct);
        }

        public ActionResult NameAndPrice()
        {
            return View(myProduct);
        }
    }
}
```

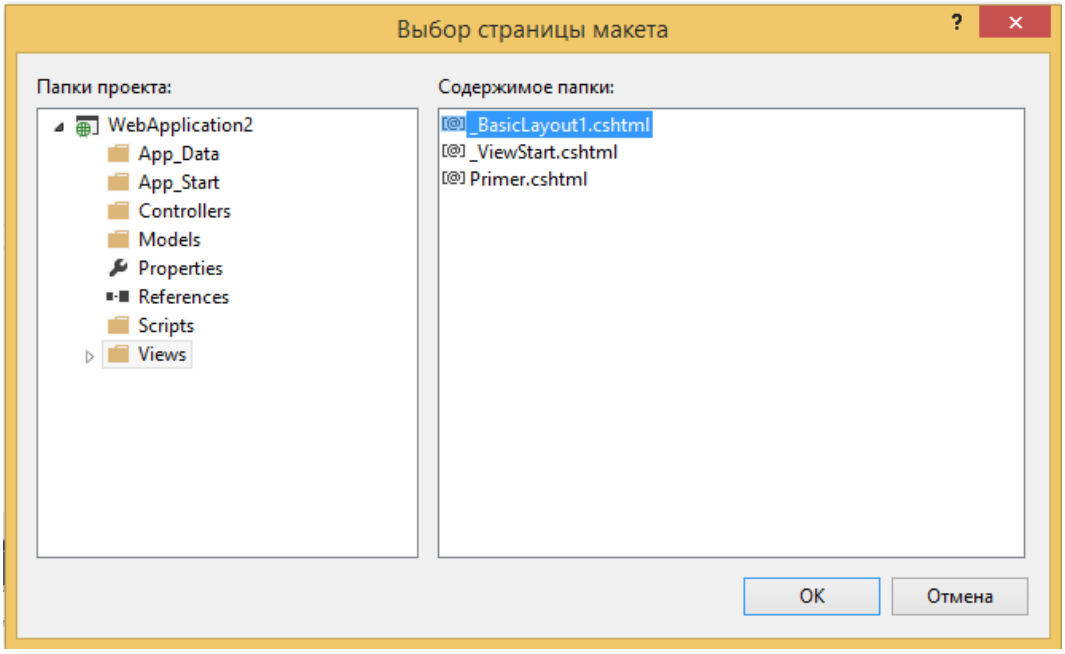
Щелкните правой кнопкой мыши в редакторе по методу NameAndPrice и выберите из всплывающего меню пункт добавления представления , название представления - NameAndPrice, значение шаблона – Empty, класс модели - Product.



Создание представления с использованием макета

Обратите внимание на текст под опцией Use a layout (испозовать страницу макета). Необходимо оставить текстовое поле пустым, если вы указали представление, которое вы хотите использовать в файле _ViewStart.cshtml. Иначе будет создано представление без оператора C#, и устанавливается как значение для свойства Layout.

Мы собираемся явно указать представление, поэтому щелкните по кнопке с многоточием (...), которая находится справа от текстового поля. Visual Studio покажет вам диалоговое окно, которое позволяет выбрать файл макета, как представлено на рисунке.

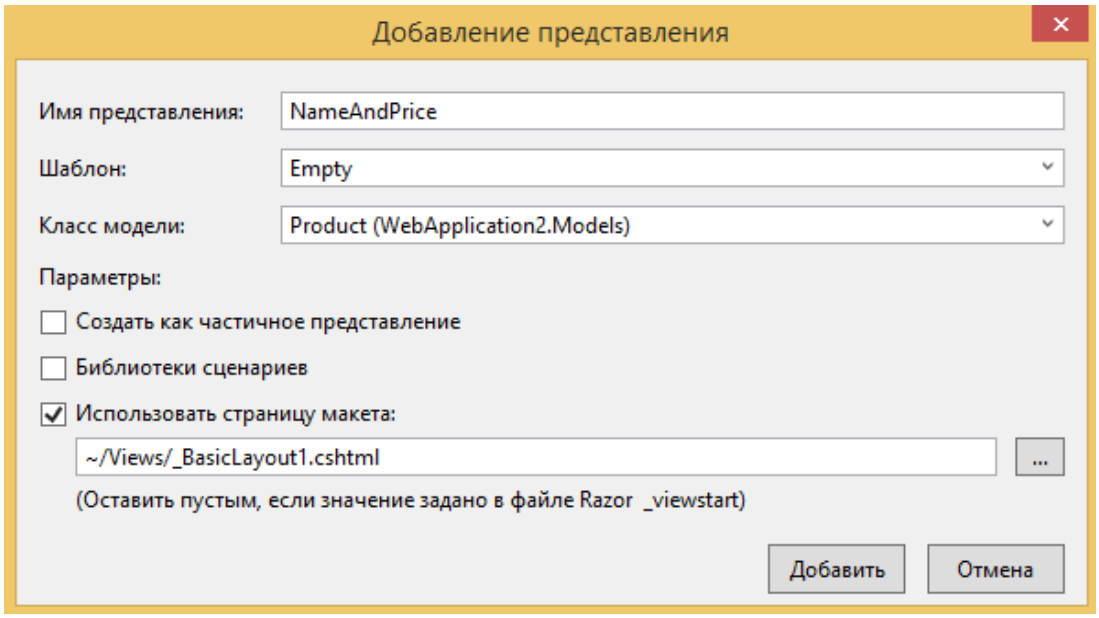


Выбор файла макета

Соглашение для MVC проектов заключается в размещение файла макета в папке Views, и поэтому диалоговое окно представляет для выбора

содержимое именно этой папки. Но это всего лишь соглашение, и поэтому левая панель диалогового окна позволяет перемещаться по проекту, если вдруг вы решили не следовать соглашению.

Мы определили только один файл макета, поэтому выберите _BasicLayout.cshtml и нажмите кнопку ОК, чтобы вернуться в диалоговое окно добавления представления (Add View). Вы увидите, что имя файла макета было помещено в текстовое поле, как показано на рисунке.



Определение файла макета при создании представления

Нажмите кнопку добавления (Add) для создания файла /Views/Home/NameAndPrice.cshtml. Содержимое этого файла представлено в следующем листинге.

NameAndPrice.cshtml

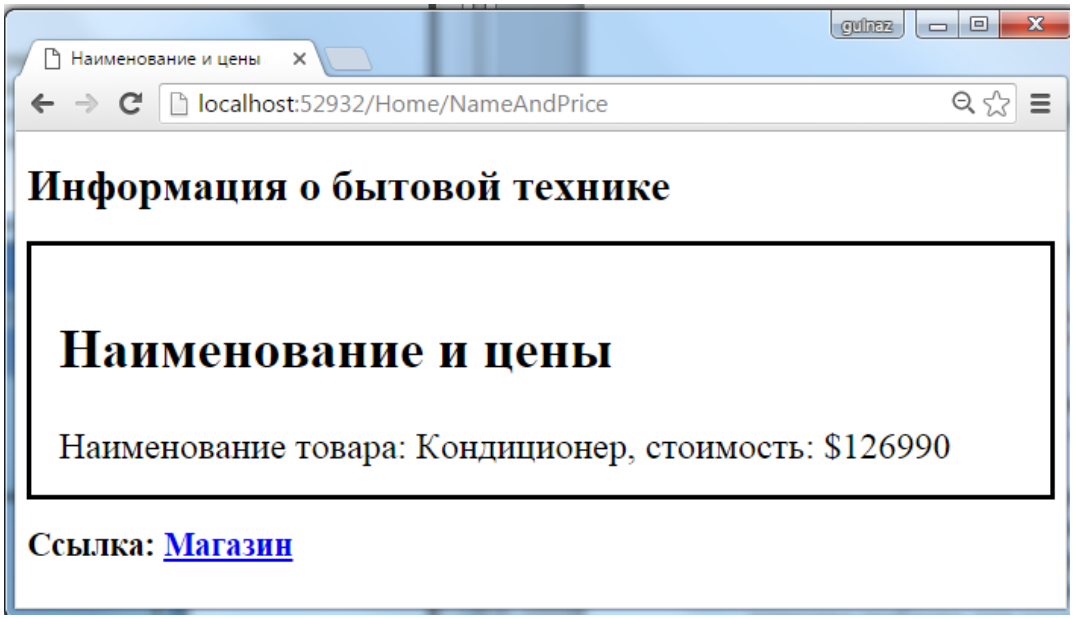
```
@model WebApplication2.Models.Product
@{
    ViewBag.Title = "NameAndPrice";
    Layout = "~/Views/_BasicLayout1.cshtml";
}
<h2>NameAndPrice</h2>
```

Visual Studio использует немного другой конетент по умолчанию для файлов представления, если вы указываете макет, но вы видите, что результат содержит те же Razor выражения, которые мы использовали, когда сами применяли макет к представлению. Чтобы завершить этот пример, мы добавили в следующий листинг простое дополнение к файлу NameAndPrice.cshtml, где отображено значение данных объекта модели представления.

Листинг дополнения в макет NameAndPrice

```
@model WebApplication2.Models.Product
@{
    ViewBag.Title = "Наименование и цены";
    Layout = "~/Views/_BasicLayout1.cshtml";
}
<h2> Наименование и цены</h2>
Наименование товара: @Model.Name,   стоимость:  $@Model.Price
```

Если вы запустите приложение и перейдете к /Home/NameAndPrice, вы увидите результаты, показанные на рисунке 222. Как вы и ожидали, общие элементы и стили, определенные в макете, были применены к представлению. Это показывает, как макет может быть использован в качестве шаблона для создания общего внешнего вида приложения (хотя он довольно простой и не совсем привлекательный).



Содержание файла макета, применяемое к представлению NameAndPrice

Примечание. У нас был бы тот же результат, если бы мы оставили текстовое поле в диалоговом окне Add View пустым и работали с файлом _ViewStart.cshtml. Мы указали файл явно только потому, что мы хотели показать вам функцию Visual Studio, которая помогает выбрать необходимый файл.

Использование выражений Razor

Теперь, когда мы показали вам основы представлений и макетов, мы собираемся обратиться к различным видам выражений, которые поддерживает Razor, и рассказать вам, как их использовать для создания контента представления.

В хорошем MVC приложении существует четкое разделение между ролями, которые выполняют метод действия и представление.

Таблица 5-: Роли, выполняемые методом действия и представлением

Чтобы получить все лучшее от MVC фреймворка, вам нужно уважать разделение различных частей приложения и соблюдать его.

Razor не используется для выполнения операций с бизнес логикой или для манипулирования объектами доменной модели.

Равным образом, вы не должны форматировать данные, которые ваш метод действия передает представлению. Вместо этого, позвольте представлению вычислить данные, которые оно должно отобразить.

Мы определили метод действия NameAndPrice, который отображает значения свойств Name и Price объекта Product. И хотя мы знали, какие именно свойства нам нужно отобразить, мы передали весь объект Product модели представления, вот так:

```
public ActionResult NameAndPrice() {
return View(myProduct);
}
```

Мы могли бы создать строку, которую хотим отобразить, в методе действия и передать ее в качестве объекта модели представления в представление. Это сработало бы, но такой подход подрывает преимущество MVC паттерна и уменьшает возможность реагировать на изменения в будущем. Как мы уже говорили, мы вернемся к этой теме снова, но вы должны знать, что MVC фреймворк не навязывает надлежащее использование MVC паттерна, и вы должны постоянно помнить, к какому результату могут привести созданный вами код и дизайнерские решения.