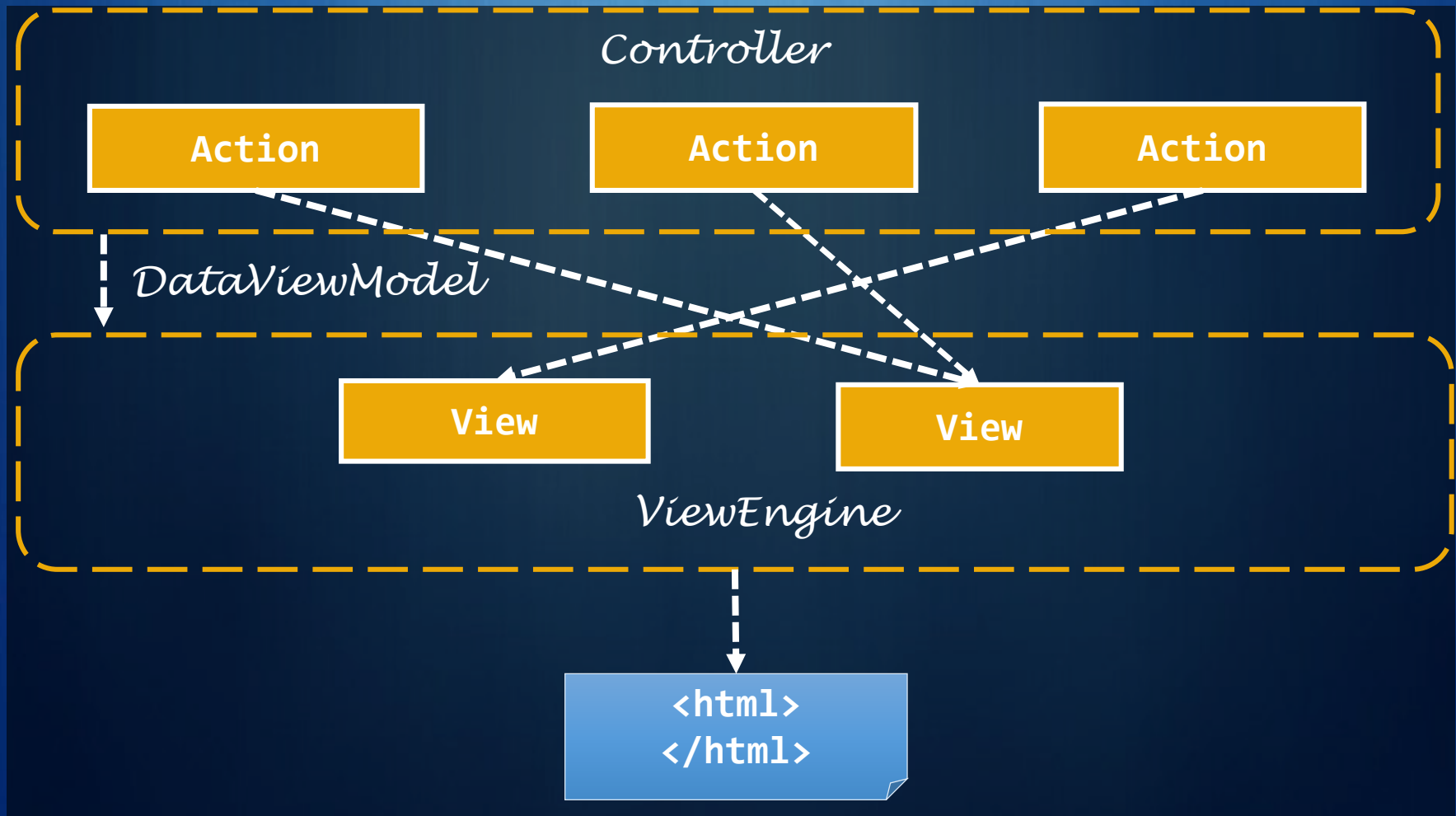


ASP MVC

v^*

Представление

Введение в представление



Введение в представление

```
protected internal ViewResult View();  
protected internal ViewResult View(IView view);  
protected internal ViewResult View(object model);  
protected internal ViewResult View(string viewName);  
protected internal virtual ViewResult View(IView view, object model);  
protected internal ViewResult View(string viewName, object model);  
protected internal ViewResult View(string viewName, string masterName);  
protected internal virtual ViewResult View(string viewName, string masterName, object model);  
  
protected internal PartialViewResult PartialView();  
protected internal PartialViewResult PartialView(object model);  
protected internal PartialViewResult PartialView(string viewName);  
protected internal virtual PartialViewResult PartialView(string viewName, object model);  
  
protected internal JsonResult JsonResult(object data);  
protected internal JsonResult JsonResult(object data, JsonRequestBehavior behavior);  
protected internal JsonResult JsonResult(object data, string contentType);  
protected internal virtual JsonResult JsonResult(object data, string contentType, Encoding contentEncoding);  
protected internal JsonResult JsonResult(object data, string contentType, JsonRequestBehavior behavior);  
protected internal virtual JsonResult JsonResult(object data, string contentType, Encoding contentEncoding, Js
```

Введение в представление

Работа приложения MVC управляется главным образом контроллерами, но непосредственно пользователю приложение доступно в виде представления (**View**), которое и формирует внешний вид приложения.

Целью представления является прием переданной в него модели и использование данной модели для отображения содержимого приложения. В связи с тем, что контроллер и соответствующие службы уже исполнили всю бизнес-логику приложения и упаковали результаты в объект модели представления, представлению остается всего лишь узнать, как принять эту модель и преобразовать ее в HTML.

В ASP.NET MVC представления представляют файлы с расширением **cshtml/vbhtml/aspx/ascx**, которые содержат код с интерфейсом пользователя, как правило, на языке html.

Введение в представление

Компонент	Выполняет	Не выполняет
Action Method	Передаёт объект модели представления в представление	Не передаёт отформатированные данные в представление
View	Использует объект модели представления для показа контента пользователю	Не меняет ни один из аспектов объекта модели представления

Передача данных в представление

- TempData : TempDataDictionary
- ViewBag : Dynamic
- ViewData : ViewDataDictionary
- Строго типизированные представления

Передача данных в представление

First request

http://localhost/Home/Index

```
public ActionResult Index()  
{  
    TempData["Test"] = "Test";  
}
```

Second request

http://localhost/Home/About

```
public ActionResult About()  
{  
    var test = TempData["Test"];  
}
```

Third request

http://localhost/Home/Contact

```
public ActionResult Index()  
{  
    var test = TempData["Test"];  
}
```



Example

1_FromControllerToView

02_TempData

2015 © EPAM Systems

9

Передача данных в представление

First request

http://localhost/Home/Index

```
public ActionResult Index()  
{  
    TempData["Test"] = "Test";  
}
```

Second request

http://localhost/Home/About

```
public ActionResult About()  
{  
    var test =  
        TempData.Peek("Test");  
}
```

Third request

http://localhost/Home/Contact

```
public ActionResult Index()  
{  
    var test = TempData["Test"];  
}
```



Передача данных в представление



- **ViewBag** служит для передачи данных от контроллера в представление — как правило это временные данные, которые не включены в модель
- **ViewBag** это динамическое свойство, которое использует в себе новые динамические функции C# 4.0
- Для **ViewBag** можно назначить любое количество свойств и значений
- Цикл жизни **ViewBag** определяется только текущим HTTP-запросом, значения **ViewData** будут нулевыми, если происходит редирект
- **ViewBag** на самом деле является оберткой **ViewData**



Передача данных в представление

```
public ActionResult Action2()
{
    // Использование объекта ViewBag (это динамический тип данных).
    ViewBag.Date = DateTime.Now;
    return View();
}
```

@*Чтение данных из объекта ViewBag*@

<p>День недели: @ViewBag.Date.Year</p>

(dynamic expression)

This operation will be resolved at runtime.



Передача данных в представление



- **ViewData** служит для передачи данных от контроллера в представление, а не наоборот
- **ViewData** является словарем производным от **ViewDataDictionary**
- Цикл жизни **ViewData** определяется только текущим HTTP-запросом, значения **ViewData** будут потеряны, если происходит редирект
- Над значениями **ViewData** перед использованием должны быть выполнены приведение типа
- Поскольку **ViewBag** внутренне вставляет данные в словарь **ViewData**, ключи **ViewData** и свойства **ViewBag** **НЕ должны** совпадать



Передача данных в представление

```
public ActionResult Action3()
{
    // Запись данных во ViewData для представления.
    ViewData["Date"] = DateTime.Now;
    return View();
}
```

@*Чтение данных из ViewData*@

День недели: @(((DateTime)ViewData["Date"]).Year)

- ⊗ ToLocalTime
- ⊗ ToLongDateString
- ⊗ ToLongTimeString
- ⊗ ToOADate
- ⊗ ToShortDateString
- ⊗ ToShortTimeString
- ⊗ ToString
- ⊗ ToUniversalTime
- 🔧 Year



Example

1_FromControllerToView 02_TempData

2015 © EPAM Systems

14

Строго типизированные представления

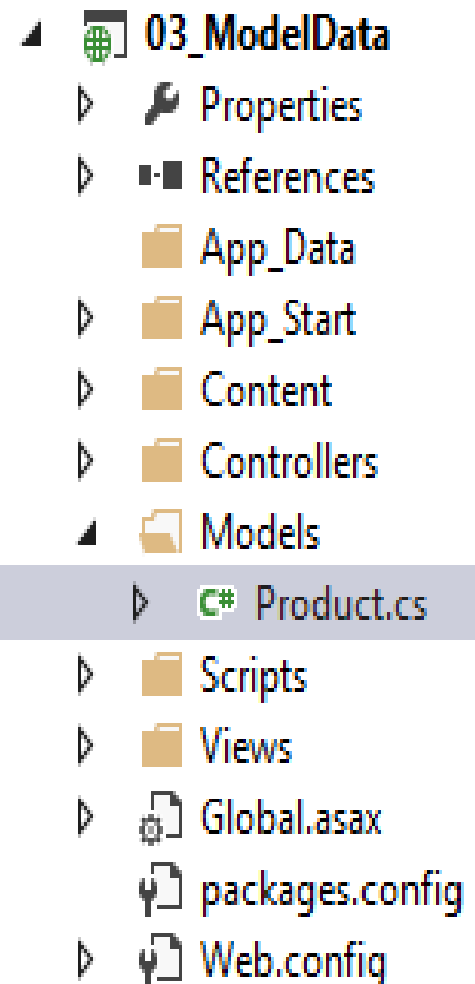
При использовании представлений на базе движка Razor, представления могут по умолчанию наследоваться от двух типов: `System.Web.Mvc.WebViewPage` или `System.Web.Mvc.WebViewPage<T>`

```
public class WebViewPage<TModel> : WebViewPage
{
    public new AjaxHelper<TModel> Ajax { get; set; }
    public new HtmlHelper<TModel> Html { get; set; }
    public new TModel Model { get; }
    public new ViewDataDictionary<TModel> ViewData { get; set; }
}
```



Строго типизированные представления

```
namespace _03_ModelData.Models
{
    9 references
    public class Product
    {
        5 references
        public int ProductId { get; set; }
        5 references
        public string Name { get; set; }
        5 references
        public string Description { get; set; }
        5 references
        public decimal Price { get; set; }
        5 references
        public string Category { get; set; }
    }
}
```



Строго типизированные представления

```
namespace _03_ModelData.Controllers
```

```
{
```

```
    References
```

```
    public class ProductController : Controller
```

```
    {
```

```
        References
```

```
        public ActionResult Index()
```

```
        {
```

```
            List<Product> products = new List<Product>();
```

```
            products.Add(new Product()
```

```
            {
```

```
                ProductId = 1,
```

```
                Name = "Шариковая ручка",
```

```
                Description = "Синяя шариковая ручка с колпачком и прозрачным корпусом.",
```

```
                Price = 3m,
```

```
                Category = "Канцтовары"
```

```
            });
```

```
            products.Add(new Product()
```

```
            {
```

```
                ProductId = 2,
```

```
                Name = "Мобильный телефон",
```

```
                Description = "Мобильный телефон с фотокамерой.",
```

```
                Price = 250m,
```

```
                Category = "Техника"
```

```
            });
```

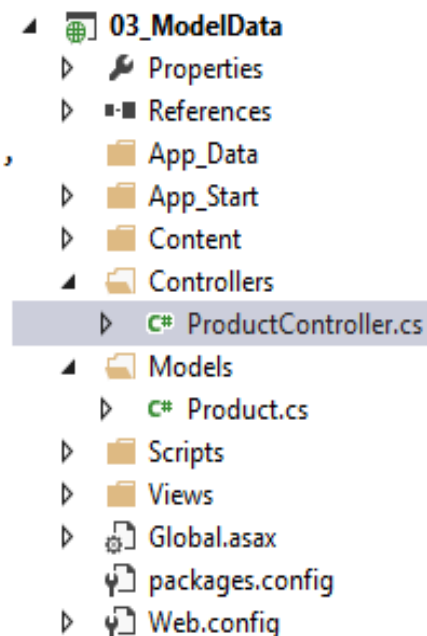
```
            // Возвращаем представление из директории Views/Products/Index.cshtml
```

```
            // Параметр передающийся в метод View() является моделью,
```

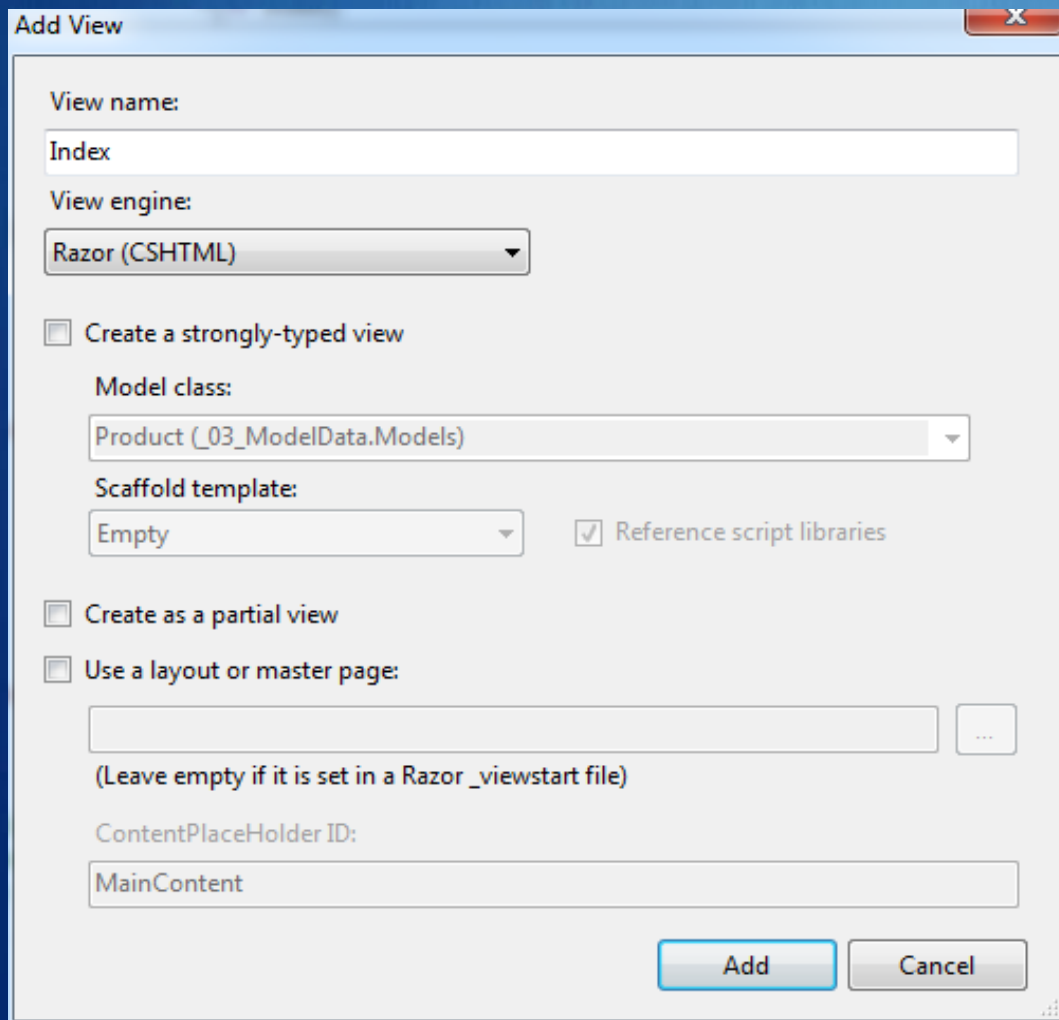
```
            //которая будет доступна только на чтение в представлении Index
```

```
            return View(products);
```

```
        }
```



Строго типизированные представления



Add View

View name:
Index

View engine:
Razor (CSHTML)

☐ Create a strongly-typed view

Model class:
Product (_03_ModelData.Models)

Scaffold template:
Empty

☒ Reference script libraries

☐ Create as a partial view

☐ Use a layout or master page:

...

(Leave empty if it is set in a Razor _viewstart file)

ContentPlaceHolder ID:
MainContent

Add Cancel

Строго типизированные представления

Опции

Описание

Create a strongly-typed view	Создание строго типизированного представления. После ее выбора нужно будет указать модель в выпадающем списке, который содержит список всех моделей. Но чтобы все модели приложения можно было использовать, перед добавлением представления нужно скомпилировать проект.
Reference Script Libraries	Подключения набора файлов JavaScript. По умолчанию файл _Layout.cshtml содержит ссылку на главную библиотеку jQuery, однако не имеет ссылки на библиотеки jQuery Validation и Unobtrusive jQuery Validation.
Create as a Partial View	выбор этой опции указывает, что созданное представление будет неполным. В итоге представление будет вполне обычным, однако в его шапке не будет таких тегов, как <html> и <head>
Use a layout or Master Page	Подключение Layout представления. С возможностью добавления не стандартного шаблона.

Строго типизированные представления

Add View

View name:
Index

View engine:
Razor (CSHTML)

☒ Create a strongly-typed view

Model class:
RouteConfig (_03_ModelData)

Scaffold template:
Empty
Create
Delete
Details
Edit
Empty
List

☐ Reference script libraries

(Leave empty if it is set in a Razor _viewstart file)

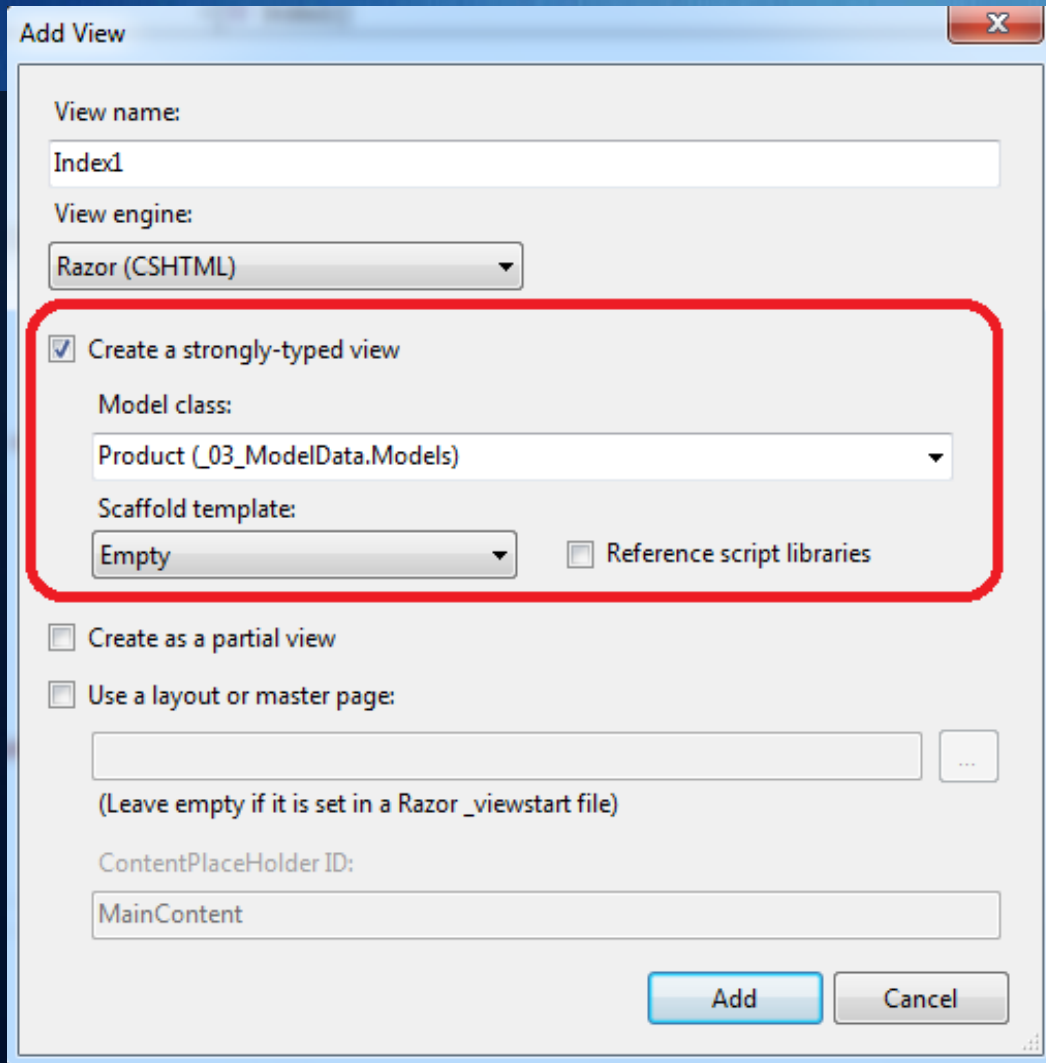
ContentPlaceholder ID:
MainContent

Add Cancel

Строго типизированные представления

Опции	Описание
Scaffold template	шаблон формирования нового представления
Empty	Создается пустое представление. В представлении только определен тип модели с помощью директивы @model
Create или Edit	Создается представление с формой для создания или редактирования новых объектов модели. Генерируется метка и поле редактирования для каждого свойства модели
Delete	Создается представление с формой для удаления существующих объектов модели. Отображаются метка и текущее значение каждого свойства модели
Details	Создается представление, которое отображает метку и значение каждого свойства модели.

Строго типизированные представления



Add View

View name:
Index1

View engine:
Razor (CSHTML)

☒ Create a strongly-typed view

Model class:
Product (_03_ModelData.Models)

Scaffold template:
Empty

☐ Reference script libraries

☐ Create as a partial view

☐ Use a layout or master page:

(Leave empty if it is set in a Razor _viewstart file)

ContentPlaceHolder ID:
MainContent

Add Cancel

Строго типизированные представления

```
@using _03_ModelData.Models
@model IEnumerable<Product>

@* @model _03_ModelData.Models.Product *@

{
    Layout = null;
}

<!DOCTYPE html>

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

    </div>
</body>
</html>
```

03_ModelData

- Properties
- References
- App_Data
- App_Start
- Content
- Controllers
 - ProductController.cs
- Models
 - Product.cs
- Scripts
- Views
 - Product
 - Index.cshtml
 - Shared
 - _ViewStart.cshtml
 - Web.config
 - Global.asax

Синтаксис Razor

Razor – это движок представления, который 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. **Движок Razor**, который работает с отдельными областями контента, обозначается символом `@`.

`@` – предотвращение Razor от интерпретации строки как C# выражения

`@:` – оператор Razor (по умолчанию кодируется для предотвращения XSS-атак)

`@{ ... }` – блок кода Razor

`@* ... *@` – комментарий



Example

Синтаксис Razor

Основные синтаксические правила **Razor** для C#

- **Кодовые блоки** Razor заключаются в **@ {...}**
- **Встроенные выражения** (переменные и функции) начинаются с **@**
- **Утверждения** заканчиваются точкой с запятой
- **Переменные** объявляются с ключевым словом **var**
- **Строки** заключаются в кавычки
- **Код C#** чувствителен к регистру
- Файлы C# имеют расширение **.cshtml**

http://www.w3schools.com/aspnet/razor_intro.asp



Example

Строго типизированные представления

```
@using _03_ModelData.Models
@model IEnumerable<Product>

@* @model _03_ModelData.Models.Product *@

@{
    Layout = null;
}

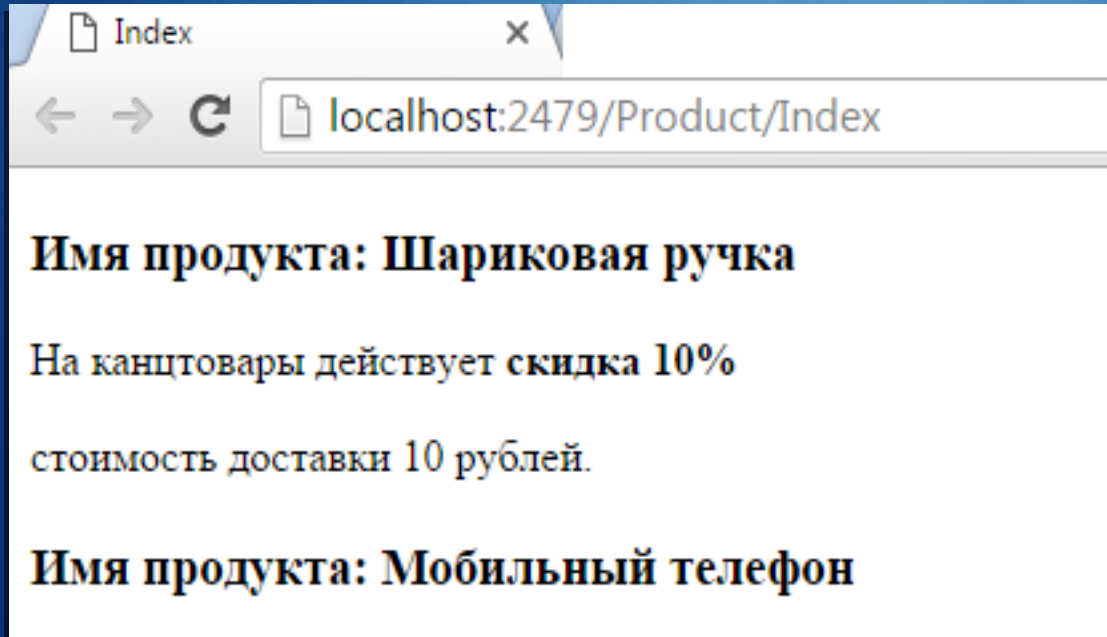
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
        @foreach (var item in Model)
        {
            <h3>Имя продукта: @item.Name</h3>

            if (item.Category == "Канцтовары")
            {
                <p>На канцтовары действует <b>скидка 10%</b></p>
            }

            if (item.Price < 5)
            {
                @:стоимость доставки 10 рублей.
            }

            <br />
        }
    </div>
</body>
</html>
```

Строго типизированные представления



Строго типизированные представления

Add View

View name:
Index1

View engine:
Razor (CSHTML)

☒ Create a strongly-typed view

Model class:
Product (_03_ModelData.Models)

Scaffold template:
Create

☒ Reference script libraries

☐ Create as a partial view

☐ Use a layout or master page:

(Leave empty if it is set in a Razor _viewstart file)

ContentPlaceHolder ID:
MainContent

Add Cancel

Строго типизированные представления

```
Index1.cshtml  Product.cs  ProductController.cs  Index.cshtml  A
@model _03_ModelData.Models.Product

@{
    ViewBag.Title = "Index1";
}

<h2>Index1</h2>

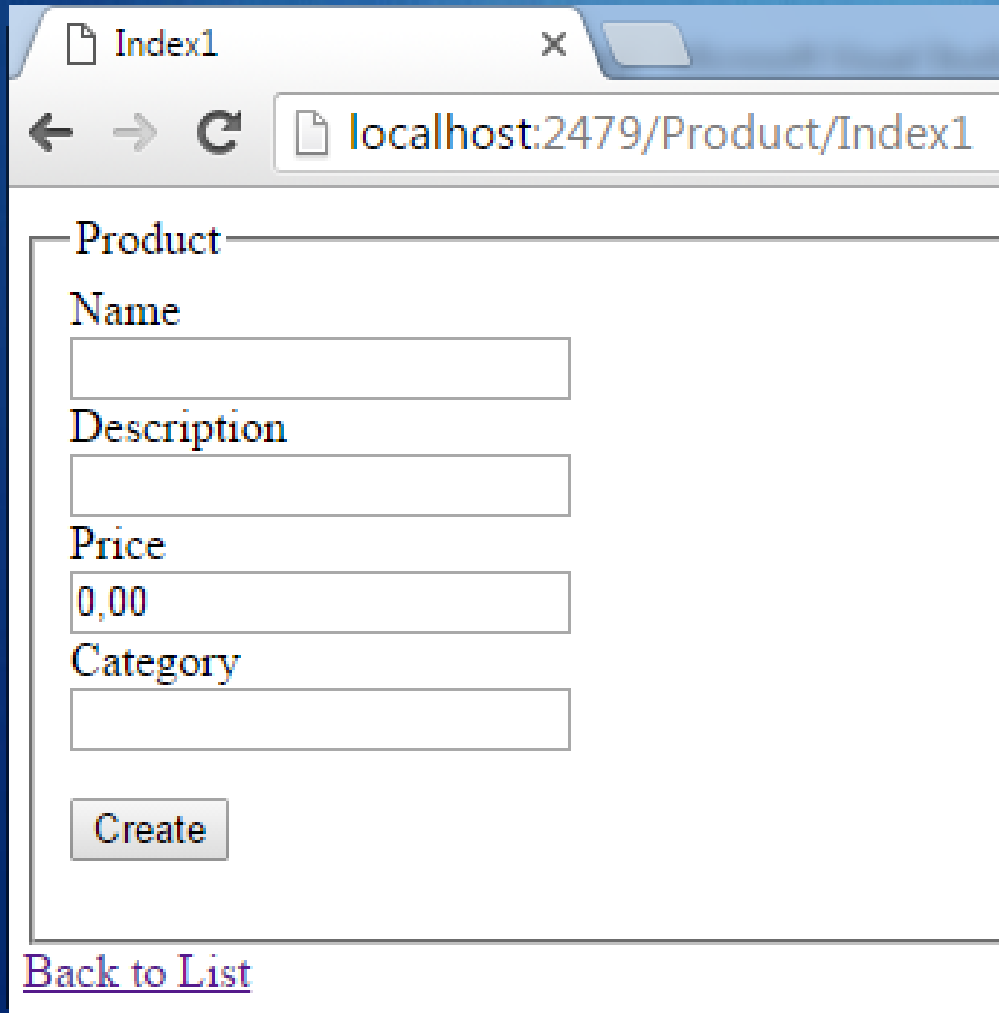
@using (Html.BeginForm()) {
    @Html.AntiForgeryToken()
    @Html.ValidationSummary(true)

    <fieldset>
        <legend>Product</legend>

        <div class="editor-label">
            @Html.LabelFor(model => model.Name)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Name)
            @Html.ValidationMessageFor(model => model.Name)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.Description)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Description)
            @Html.ValidationMessageFor(model => model.Description)
        </div>
    </fieldset>
}
```

Строго типизированные представления



Index1

localhost:2479/Product/Index1

Product

Name

Description

Price

Category

Create

[Back to List](#)

Строго типизированные представления

```
namespace System.Web.Mvc
```

```
{
```

```
    public abstract class WebViewPage<TModel> : WebViewPage
```

```
    {
```

```
        protected WebViewPage();
```

```
        public AjaxHelper<TModel> Ajax { get; set; }
```

```
        public HtmlHelper<TModel> Html { get; set; }
```

```
        public TModel Model { get; }
```

```
        public ViewDataDictionary<TModel> ViewData { get; set; }
```

```
        public override void InitHelpers();
```

```
        protected override void SetViewData(ViewDataDictionary
```

```
    }
```

```
}
```

```
namespace System.Web.Mvc
```

```
{
```

```
    public abstract class WebViewPage : WebPageBase, IViewDataCont
```

```
    {
```

```
        protected WebViewPage();
```

```
        public AjaxHelper<object> Ajax { get; set; }
```

```
        public override HttpContextBase Context { get; set; }
```

```
        public HtmlHelper<object> Html { get; set; }
```

```
        public object Model { get; }
```

```
        public TempDataDictionary TempData { get; }
```

```
        public UrlHelper Url { get; set; }
```

```
        public dynamic ViewBag { get; }
```

```
        public ViewContext ViewContext { get; set; }
```

```
        public ViewDataDictionary ViewData { get; set; }
```

```
    }
```

```
}
```

Строго типизированные представления

@Html.

- GetType
- GetUnobtrusiveValidationAttributes
- Hidden
- HiddenFor<>
- HttpMethodOverride
- Id
- IdFor<>
- IdForModel
- Label

Web.config X

```
<pages>
  <namespaces>
    <add namespace="System.Web.Helpers" />
    <add namespace="System.Web.Mvc" />
    <add namespace="System.Web.Mvc.Ajax" />
    <add namespace="System.Web.Mvc.Html" />
    <add namespace="System.Web.Optimization" />
    <add namespace="System.Web.Routing" />
    <add namespace="System.Web.WebPages" />
  </namespaces>
</pages>
```

Html-хелперы. Встроенные хелперы

Фреймворк MVC предоставляет большой набор встроенных html-хелперов, которые позволяют генерировать ту или иную разметку, главным образом, для работы с формами.

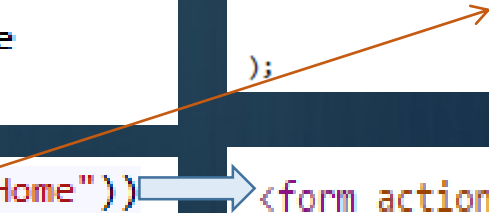
Наиболее полезные (и наиболее часто используемые) вспомогательные методы - это **Html.BeginForm** и **Html.EndForm**. Они создают теги формы HTML и генерируют для нее допустимый атрибут **action**, основываясь на механизме маршрутизации приложения

В MVC определен широкий набор хелперов ввода практически для каждого html-элемента

Строго типизированные представления



```
public static MvcForm BeginForm(  
    this HtmlHelper htmlHelper,  
    string actionName,  
    string controllerName  
)
```

```
routes.MapRoute(  
    name: "Default",  
    url: "{controller}/{action}/{id}",  
    defaults: new { controller = "Home",  
                    action = "Index",  
                    id = UrlParameter.Optional }  
);
```



```
@using (Html.BeginForm("Index", "Home"))  
{  
    <div>  
        <p>Содержание формы</p>  
    </div>  
  
    <input type="submit" />  
}
```

```
<form action="/" method="post">  
    <div>  
        <p>Содержание формы</p>  
    </div>  
    <input type="submit" />  
</form>
```



Html-хелперы. Встроенные хелперы

Элемент HTML	Пример
Html.CheckBox	<pre>Html.CheckBox("checkbox", false)</pre> <p>Вывод: <input id="checkbox" name="checkbox" type="checkbox" value="true" /> <input name="checkbox" type="hidden" value="false" /></p>
Html.Hidden	<pre>Html.Hidden("hidden", "val")</pre> <p>Вывод: <input id="hidden" name="hidden" type="hidden" value="val" /></p>
Html.RadioButton	<pre>Html.RadioButton("radiobutton", "val", true)</pre> <p>Вывод: <input checked="checked" id="radiobutton" name="radiobutton" type="radio" value="val" /></p>

Html-хелперы. Встроенные хелперы

Элемент HTML	Пример
Html.Password	<pre>Html.Password("cpassword", "val")</pre> <p>Вывод: <input id="cpassword" name="cpassword" type="password" value="val" /></p>
Html.TextArea	<pre>Html.TextArea("ctextarea", "val", 5, 20, null)</pre> <p>Вывод: <textarea cols="20" id="ctextarea" name="ctextarea" rows="5"> val</textarea></p>
Html.TextBox	<pre>Html.TextBox("ctextbox", "val")</pre> <p>Вывод: <input id="ctextbox" name="ctextbox" type="text" value="val" /></p>

Html-хелперы. Встроенные хелперы

Элемент HTML	Пример
Html.DropDownList	<pre>Html.DropDownList("countires", new SelectList(new string[] {"Russia","USA", "Canada","France"}), "Countries")</pre> <p>Вывод: <select id="countires" name="countires"><option value="">Countries</option> <option>Russia</option> <option>USA</option> <option>Canada</option> <option>France</option> </select></p>
Html.Label	<pre>Html.Label("Name")</pre> <p>Вывод: <label for="Name">Name</label></p>

HTML-хелперы. Строго типизированные хелперы

Кроме базовых хелперов в ASP.NET MVC имеются их двойники - строго типизированные хелперы. Этот вид хелперов принимает в качестве параметра лямбда-выражение, в котором указывается то свойство модели, к которому должен быть привязан данный хелпер. Важно учитывать, что строго типизированные хелперы могут использоваться только в строго типизированных представлениях, а тип модели, которая передается в хелпер, должен быть тем же, что указан для всего представления с помощью директивы `@model`.

HTML-хелперы. Строго типизированные хелперы

Базовые HTML-хелперы

```
Html.CheckBox("checkbox", false)
```

```
Html.Hidden("hidden", "val")
```

```
Html.RadioButton("radio",  
    "val", true)
```

```
Html.Password("password", "val")
```

```
Html.TextArea("text", "val",  
    5, 20, null)
```

```
Html.TextBox("text", "val")
```

Строго типизированные HTML-хелперы

```
Html.CheckBoxFor(x => x.IsApproved)
```

```
Html.HiddenFor(x => x.SomeProperty)
```

```
Html.RadioButtonFor(  
    x => x.IsApproved, "val")
```

```
Html.PasswordFor(x => x.Password)
```

```
Html.TextAreaFor(x => x.Bio,  
    5, 20, new{})
```

```
Html.TextBoxFor(x => x.Name)
```

HTML-хелперы. Шаблонные хелперы

Шаблонные вспомогательные методы позволяют только указать свойство, которое необходимо визуализировать, не уточняя при этом, какой элемент HTML для него требуется - MVC Framework выясняет это самостоятельно. Это более гибкий подход к отображению данных пользователю, хотя он и требует немного больше внимания и осторожности.

HTML-хелперы. Шаблонные хелперы

- **Display** создает элемент разметки, который доступен только для чтения, для указанного свойства модели: `Html.Display("Name")`
- **DisplayFor** создает строго типизированный аналог хелпера `Display`: `Html.DisplayFor(e => e.Name)`
- **Editor** создает элемент разметки, который доступен для редактирования, для указанного свойства модели: `Html.Editor("Name")`
- **EditorFor** строго типизированный аналог хелпера `Editor`: `Html.EditorFor(e => e.Name)`
- **DisplayText** создает выражение для указанного свойства модели в виде простой строки: `Html.DisplayText("Name")`
- **DisplayTextFor** строго типизированный аналог хелпера `DisplayText`: `Html.DisplayTextFor(e => e.Name)`

HTML-хелперы. Шаблонные хелперы

Существует несколько шаблонов, которые позволяют сгенерировать разом все поля для определенной модели:

- **DisplayForModeld** – создает поля для чтения для всех свойств модели:
`Html.DisplayForModel()`
- **EditorForModel** – создает поля для редактирования для всех свойств модели:
`Html.EditorForModel()`

HTML-хелперы. Строго типизированные хелперы

```
[HttpGet]
[References]
public ActionResult Create()
{
    GuestBookEntity guestBook = new GuestBookEntity();
    guestBook.Name = "Ivanov";
    guestBook.Message = "Hello!!!";

    return View(guestBook);
}
```

@*

При использовании перегрузки `@Html.TextBoxFor` (лямбда выражение) создается input с типом text и атрибутом name и id указанным в параметре. Важная особенность данного метода - поиск данных для атрибута value элемента при вызове.

При следующем вызове `@Html.EditorFor(model => model.Name)` поиск будет происходить в такой последовательности:

- 1) ViewBag.Name
- 2) ViewData["Name"]
- 3) @Model.Name

*@

```
@using (Html.BeginForm()) {
    <div>
        @Html.LabelFor(model => model.Name)
    </div>
    <div>
        @Html.EditorFor(model => model.Name)
    </div>
}
```

HTML-хелперы. Строго типизированные хелперы

The diagram illustrates the mapping from C# code to HTML output for ASP.NET MVC helpers. It consists of two columns: the left column contains C# code, and the right column contains the resulting HTML. Arrows indicate the mapping between specific C# calls and their HTML equivalents.

C# Code (Left):

```
@using (Html.BeginForm()) {  
    <div>  
        @Html.LabelFor(model => model.Name)  
    </div>  
    <div>  
        @Html.EditorFor(model => model.Name)  
    </div>  
  
    <div >  
        @Html.LabelFor(model => model.Message)  
    </div>  
    <div >  
        @Html.TextAreaFor(model => model.Message, 5, 20, new{})  
    </div>  
    <p>  
        <input type="submit" value="Create" />  
    </p>  
}  
<div>  
    @Html.ActionLink("Back to List", "Index")  
</div>
```

HTML Output (Right):

```
<form action="/GuestBook/Create" method="post">  
    <div>  
        <label for="Name">Name</label>  
    </div>  
    <div>  
        <input id="Name" name="Name" type="text" value="Ivanov" />  
    </div>  
    <div >  
        <label for="Message">Message</label>  
    </div>  
    <div >  
        <textarea cols="20" id="Message" name="Message" rows="5">  
            Hello!!!</textarea>  
        </div>  
    <p>  
        <input type="submit" value="Create" />  
    </p>  
</form>  
  
<div>  
    <a href="/GuestBook">Back to List</a>  
</div>
```

Mapping Arrows:

- From `@using (Html.BeginForm()) {` to `<form action="/GuestBook/Create" method="post">`
- From `@Html.LabelFor(model => model.Name)` to `<label for="Name">Name</label>`
- From `@Html.EditorFor(model => model.Name)` to `<input id="Name" name="Name" type="text" value="Ivanov" />`
- From `@Html.TextAreaFor(model => model.Message, 5, 20, new{})` to `<textarea cols="20" id="Message" name="Message" rows="5">Hello!!!</textarea>`
- From `@Html.ActionLink("Back to List", "Index")` to `Back to List`

Html-хелперы

Представления используют html-разметку для визуализации содержимого. Однако фреймворк ASP.NET MVC обладает также таким мощным инструментом как **HTML-хелперы**, позволяющие генерировать html-код.

- Внутренние хелперы
- Внешние хелперы
- Встроенные (built-in)-хелперы

Html-хелперы. Внутренние хелперы

Внутренние хелперы похожи на обычные определения методов на языке C#, начинающихся с тега `@helper`

```
@helper CreateList(IEnumerable<string> list)
{
    <ul>
        @foreach (var b in list)
        {
            <li>@b</li>
        }
    </ul>
}
```

Хелперы наиболее полезны, если приходится многочисленно создавать сложную, но однотипную html-разметку



Html-хелперы. Внутренние хелперы

References

```
public ActionResult Index()
{
    ViewBag.Days = new string[] { "Monday", "Tuesday", "Wednesday" };
    ViewBag.Fruits = new string[] { "Mango", "Banana", "Apple" };

    return View();
}
```

@*

Встраиваемый вспомогательный метод HTML.
Метод может использоваться только в этом представлении.

*@

```
@helper CreateLit(string[] items)
{
    <ul>
        @foreach (var item in items)
        {
            <li>@item</li>
        }
    </ul>
}
```

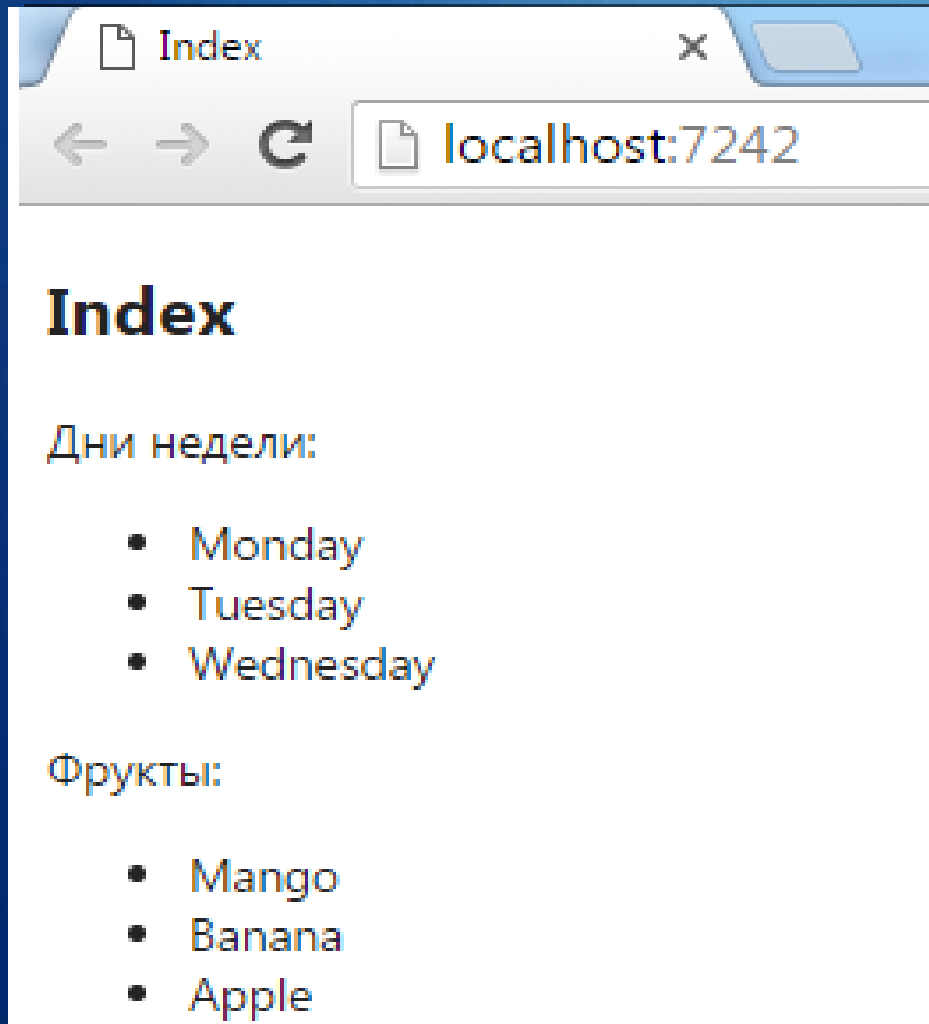
Дни недели:

```
@CreateLit(ViewBag.Days)
```

Фрукты:

```
@CreateLit(ViewBag.Fruits)
```

Html-хелперы. Внутренние хелперы



Example

Html-хелперы. Внешние хелперы

В качестве альтернативы можно использовать **внешние вспомогательные методы HTML**, которые выражаются как методы расширения C#

```
public static MvcHtmlString CreateList(this HtmlHelper html,
    IEnumerable<string> items)
{
    var ul = new TagBuilder("ul");
    foreach (string item in items)
    {
        var li = new TagBuilder("li");
        li.SetInnerText(item);
        ul.InnerHtml += li.ToString();
    }
    return new MvcHtmlString(ul.ToString());
}
```



Html-хелперы. Внешние хелперы

```
namespace _02_ExternalHelperMethods.Infrastructure
```

```
{
```

```
    References
```

```
    public static class CustomHelperMethods
```

```
    {
```

```
        // Для того чтобы использовать данный вспомогательный метод,
```

```
        //необходимо импортировать пространство имен _02_ExternalHelperMethods.Infrastructure.
```

```
        // В данном проекте, пространство имен импортировано с помощью файла Views/web.config
```

```
        References
```

```
        public static MvcHtmlString UnorderedList(this HtmlHelper html, IEnumerable<string> items)
```

```
        {
```

```
            // TagBuilder - класс для создания HTML элементов.
```

```
            TagBuilder tag = new TagBuilder("ul");
```

```
            foreach (var item in items)
```

```
            {
```

```
                TagBuilder liTag = new TagBuilder("li");
```

```
                liTag.SetInnerText(item);
```

```
                tag.InnerHtml += liTag.ToString();
```

```
            }
```

```
            // MvcHtmlString - представляет HTML кодированную строку, которая не должна кодироваться повторно.
```

```
            // Необходимо самостоятельно позаботится о кодировании тех данных, с которыми связан риск XSS атаки
```

```
            return new MvcHtmlString(tag.ToString());
```

```
        }
```

```
    }
```

```
}
```

02_ExternalHelperMethods

Properties

References

App_Data

App_Start

Content

Controllers

Infrastructure

CustomHelperMethods.cs

Models

Scripts

Views

Global.asax

packages.config

Web.config



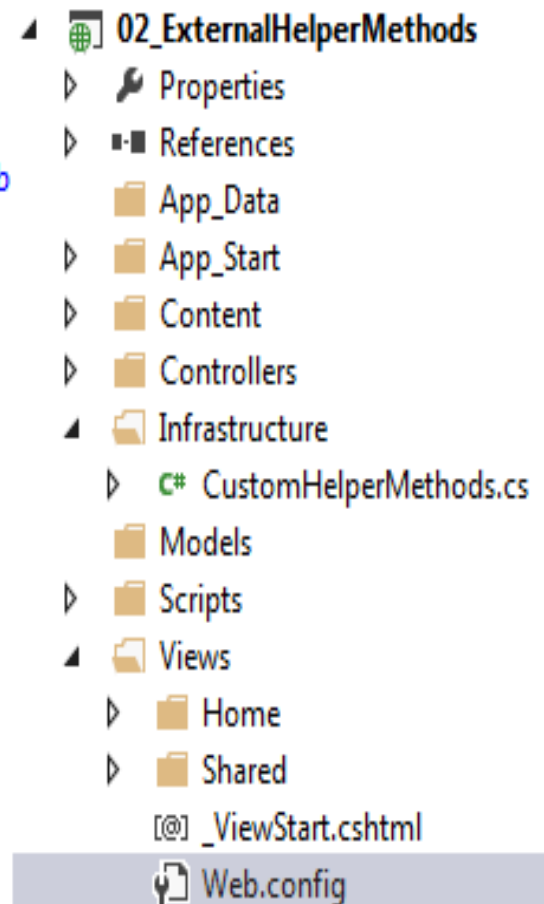
Example

2. HtmlHelper 02_ExternalHelperMethods

2019 © EPAM Systems

Html-хелперы. Внешние хелперы

```
<system.web.webPages.razor>
  <host factoryType="System.Web.Mvc.MvcWebRazorHostFactory, System.Web
  <pages pageBaseType="System.Web.Mvc.WebViewPage">
    <namespaces>
      <add namespace="System.Web.Mvc" />
      <add namespace="System.Web.Mvc.Ajax" />
      <add namespace="System.Web.Mvc.Html" />
      <add namespace="System.Web.Optimization"/>
      <add namespace="System.Web.Routing" />
      <add namespace="_02_ExternalHelperMethods.Infrastructure" />
    </namespaces>
  </pages>
</system.web.webPages.razor>
```



Example

2. HtmlHelper 02_ExternalHelperMethods

2019 © EPAM Systems

Html-хелперы. Внешние хелперы

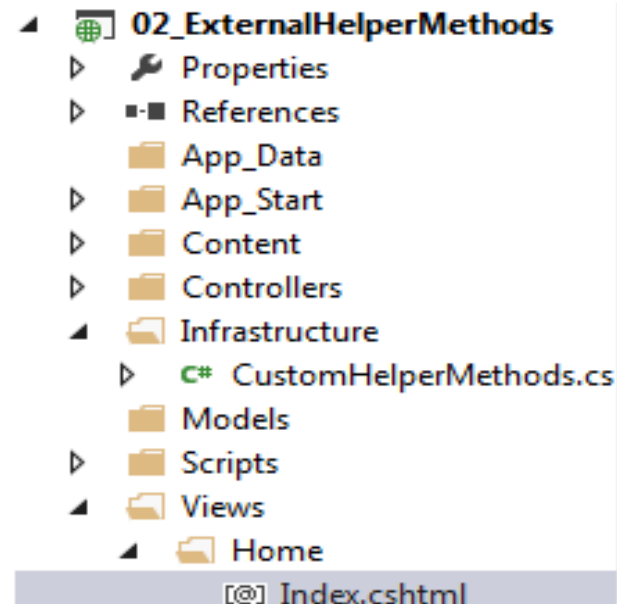
Index.cshtml* ↵ ✕

```
@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>

Дни недели:
@Html.UnorderedList(ViewBag.Days as string[])

Фрукты:
@Html.UnorderedList(ViewBag.Fruits as string[])
```



Index

Дни недели:

- Monday
- Tuesday
- Wednesday

Фрукты:

- Mango
- Banana
- Apple



Example

Html-хелперы. Класс TagBuilder

Член	Описание
InnerHTML	Свойство, которое позволяет записать содержимое элемента как строку HTML. Значение, присвоенное этому свойству, не будет закодировано, что означает, что с его помощью можно создавать вложенные элементы HTML.
SetInnerText	Устанавливает текстовое содержимое элемента HTML. Параметр string кодируется для безопасности отображения.
AddCssClass	Добавляет класс CSS к элементу HTML.
MergeAttribute	Добавляет атрибут к элементу HTML. Первый параметр - это имя атрибут, второй - его значение. Параметр bool определяет, нужно ли заменить существующий атрибут с таким же названием.

Html-хелперы. Управление кодировкой строк

В MVC Framework для защиты от вредоносного ввода применяется автоматическое кодирование, которое позволяет гарантировать безопасность добавления данных на страницу

Браузеру запрещается интерпретировать значения данных как действительную (допустимую) разметку, так как это является основой для распространенной формы атак, при которой злоумышленники пытаются изменить поведение приложения, пытаясь добавлять свой собственный код HTML или разметку JavaScript. Razor автоматически кодирует значения данных, когда они используются в представлении

Поскольку вспомогательные методы должны генерировать HTML, то они пользуются высоким уровнем доверия со стороны движка представления - и в силу этого требуют более пристального внимания.

