

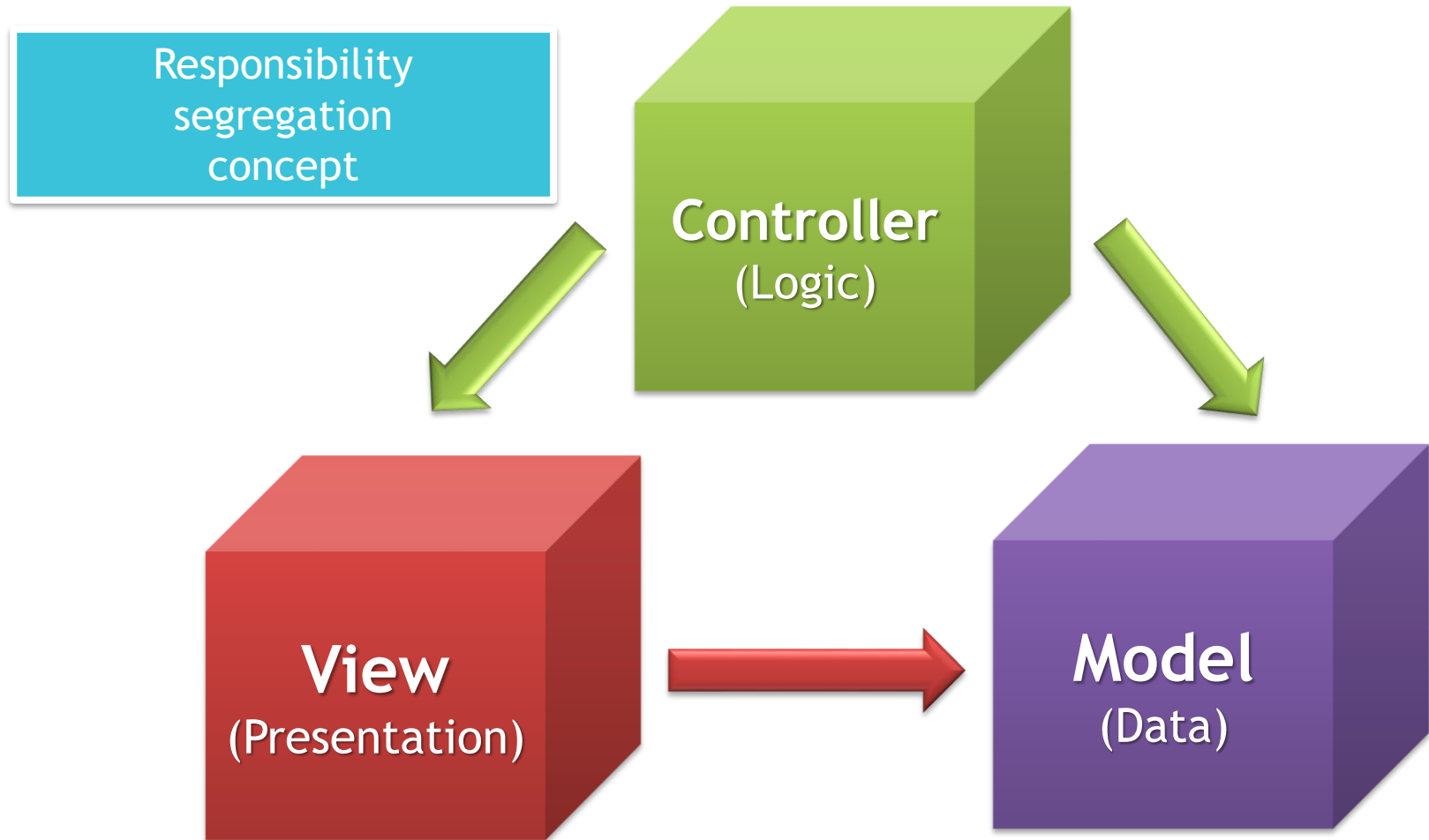


# ASP.NET MVC INTRODUCTION

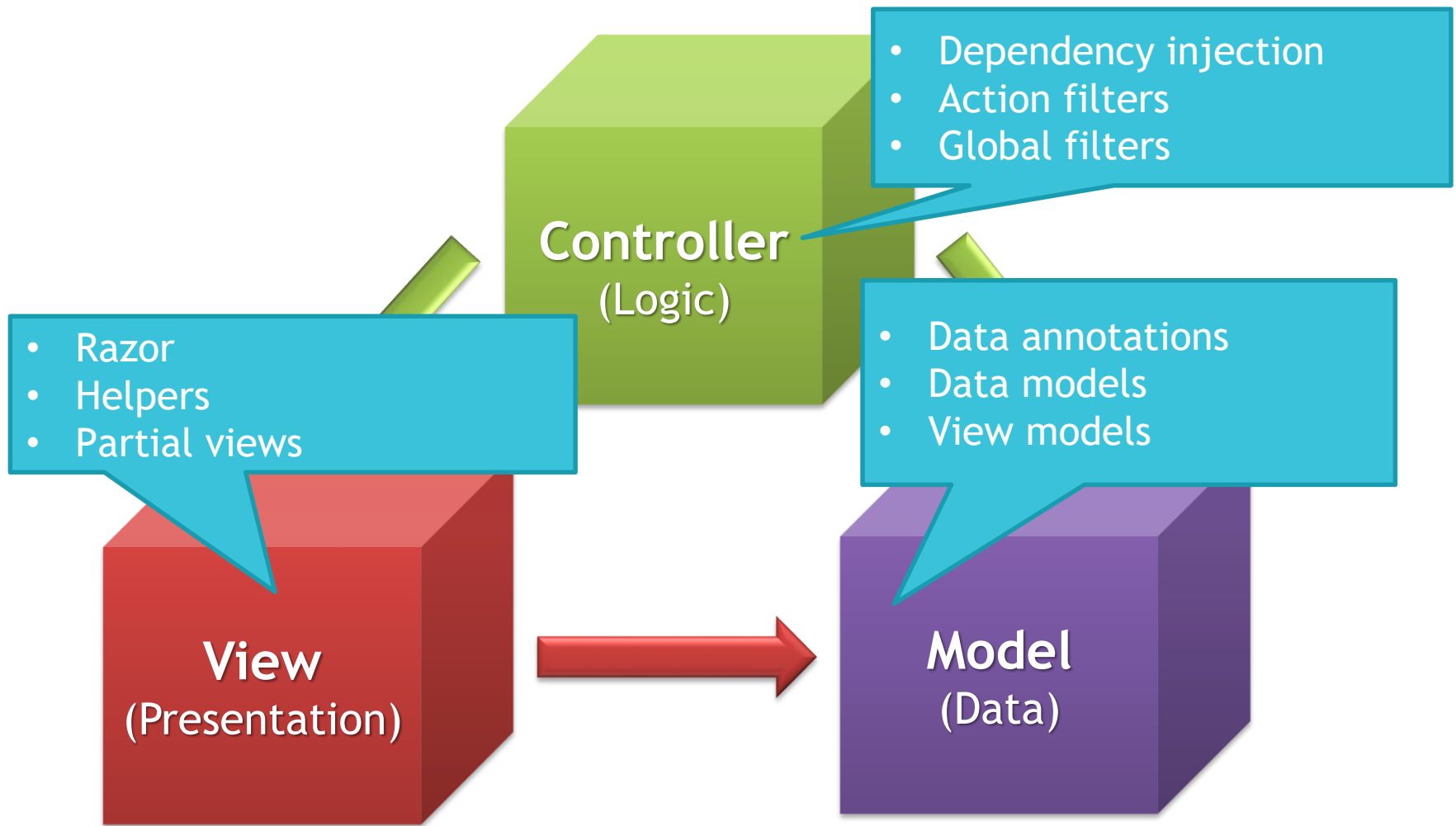
**CONTROLLER & VIEW**

September 2018

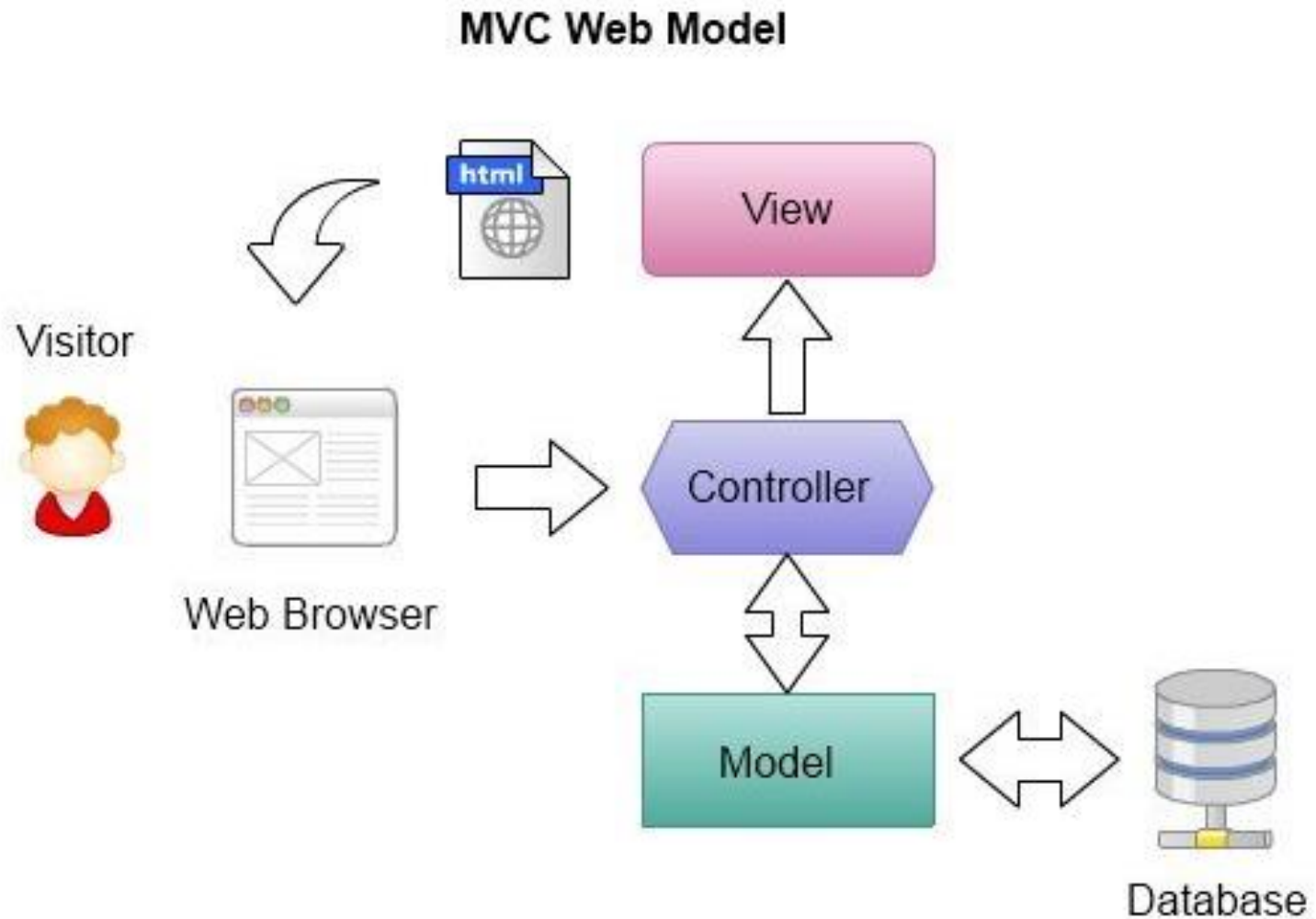
# ASP.NET MVC



# ASP.NET MVC CORE COMPONENTS



# MVC CONCEPT



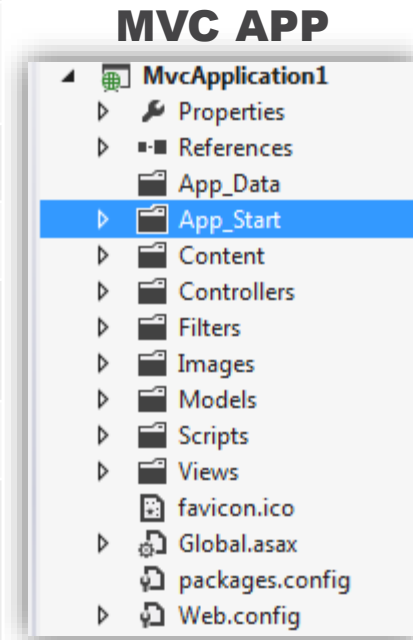
# BASIC WEB APPLICATION TEMPLATES

---

- **Empty**. Пустой шаблон проекта, содержащий базовые зависимости для создания web-приложений.
- **Web Forms**. Шаблон для создания Web Forms проектов, поддерживающих конструктор пользовательского интерфейса и, привычную для WinForms разработчиков, событийную модель.
- **MVC**. Шаблон для создания приложений, основывающихся на архитектурном шаблоне MVC.
- **Web API**. Шаблон для создания data-oriented сервисов, основывающихся на архитектурном подходе **Representational State Transfer (REST)**.

# MVC APPLICATION STRUCTURE

Каталог	Предназначение
~/Controllers	Место хранения контроллеров приложения.
~/Models	Базовое место хранения логики и поведения приложения.
~/Views	Место хранения представлений (*.cshtml)
~/Scripts	Место хранения библиотек и скриптов.
~/Content	Место хранения загружаемого контента (изображений, стилей (*.css) и т.д)
~/App_Data	Базовое место хранения данных.
~/App_Start	Статические классы с логикой инициализации приложения. Вызываются из <b>Global.asax</b> . Ранее все конфигурации находились в <b>Global.asax</b> .





A wide-angle photograph of a busy London street, likely Regent Street, during the "golden hour" of sunset. The sun is low in the sky, creating a strong, warm glow and long shadows across the pavement. On the left, a grand white stone building with classical architectural features like columns and arched windows is visible. A Union Jack flag flies from a balcony. A red traffic light is positioned at the street corner. Pedestrians, dressed in winter clothing, are walking in both directions. On the right, a Starbucks is visible under a red awning, and a building is under construction with scaffolding. A blue rectangular box with the word "CONTROLLERS" in white capital letters is superimposed over the middle of the image.

**CONTROLLERS**

# SIMPLIEST CONTROLLER

```
// ~/Controllers/HomeController.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MvcApplication.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```



# ROUTING BASICS

**Router** должен определить, какой метод у какого контроллера вызвать. Поэтому, каждый **route** обязательно должен иметь два основных параметра: **controller** и **action**.

**http://servername/{controller}/{action}/{parameter}**

Например, для **http://localhost:xxxxx/Home/Index**

Будет вызван метод **Index()** контроллера **HomeController**.

В MVC 4+ routing конфигурируется в **~/App\_Start/RouteConfig.cs**.

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute(
        "Default", // Имя маршрута
        "{controller}/{action}/{id}", // URL-адрес с параметрами
        new { controller = "Home", action = "Index", id = UrlParameter.Optional } // Параметры по умолчанию
    );
}
```

# INPUT DATA RECEPTION

```
<!-- ~/Views/Home/Index.cshtml -->
```

```
<form action="/Home/Square">
```

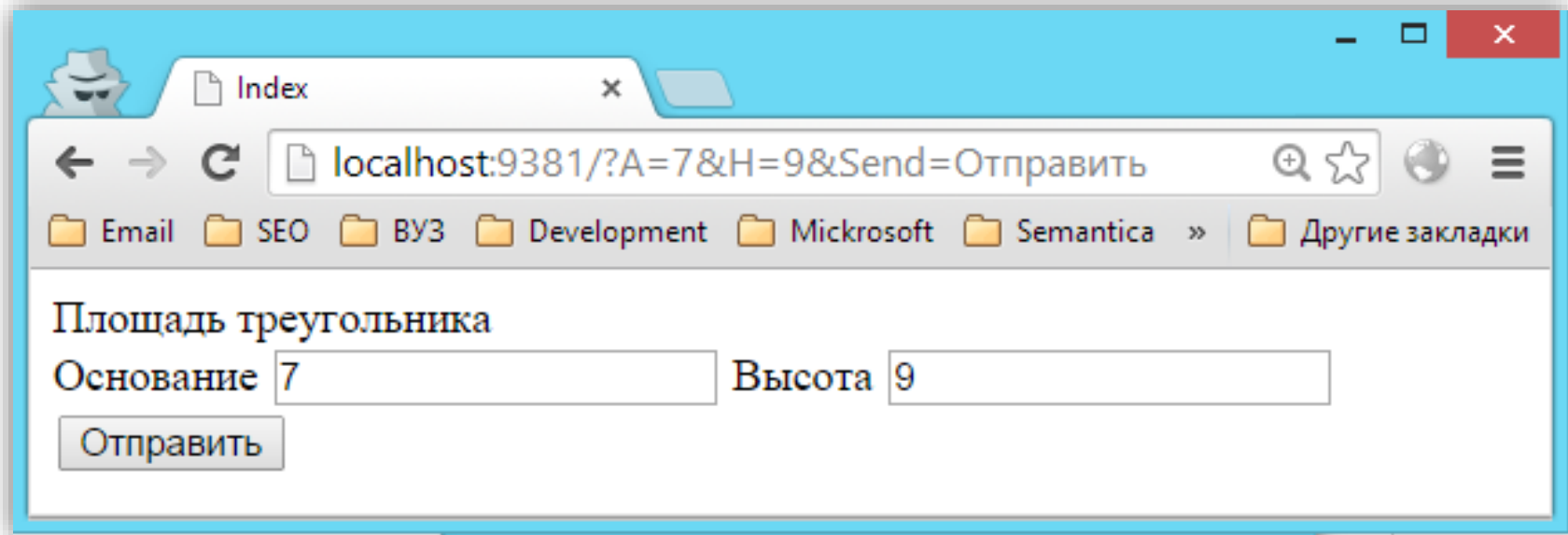
```
    <div>Площадь треугольника</div>
```

```
    <span>Основание </span> <input type="text" name="a" />
```

```
    <span>Высота </span> <input type="text" name="h" /><br />
```

```
    <input type="submit" name="Send" value="Отправить" />
```

```
</form>
```

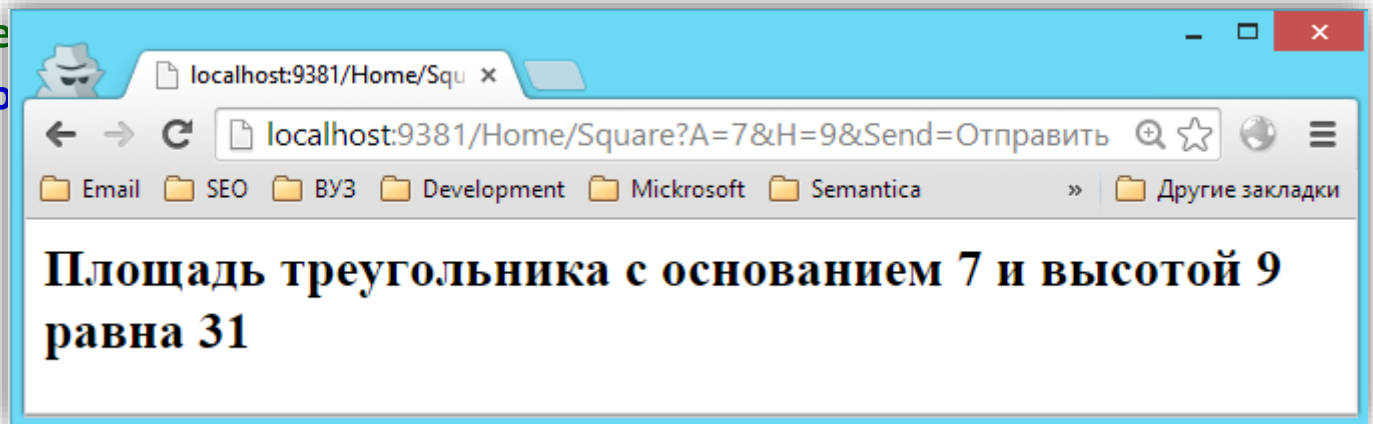


# INPUT DATA RECEPTION

```
// ~/Controllers/HomeController.cs
public string Square(int a, int h)
{
    var s = (double) a * h / 2;

    return $"<h2>Площадь треугольника с основанием {a}
        и высотой {h} равна {s}</h2>";
}
```

```
<!-- ~/Views/Home
<form action="/Home/
...
</form>
```



# ACTION RESULTS

---

Как правило, возвращаемым результатом является объект класса, производного от `ActionResult`.

```
public abstract class ActionResult
{
    public abstract void ExecuteResult(ControllerContext context);
}
```

# ACTION RESULTS

ActionResult	Method	Описание
ViewResult	View()	Генерирует <b>view</b> как веб-страницу.
PartialViewResult	PartialView()	Генерирует <b>partial view</b> , которая будет сгенерирована внутри другой <b>view</b> .
RedirectResult	Redirect()	Перенаправляет на другой <b>action</b> с указанием его URL.
RedirectToRouteResult	RedirectToAction() RedirectToRoute()	Перенаправляет на другой <b>action</b> .
ContentResult	Content()	Возвращение определенного пользователем типа контента.
JsonResult	Json()	Возвращение объекта JSON.
JavaScriptResult	JavaScript()	Возвращение скрипта, выполняемого на стороне клиента.
FileResult	File()	Возвращение бинарного потока.
EmptyResult	(NONE)	Возвращение результата, если метод возвращает null (void).
HttpStatusCodeResult	HttpSatusCodeResult()	Возвращает клиенту определенный статусный код HTTP.



# VIEW RESULT

```
public class HomeController : Controller
{
    public ViewResult SomeMethod()
    {
        return View("SomeView");
    }
}
```

В качестве view будет выбрана ~/Views/Home/SomeView.cshtml

```
public class HomeController : Controller
{
    public ViewResult SomeMethod()
    {
        return View("~/Views/Some/SomeView.cshtml");
    }
}
```

# REDIRECT RESULT

Временная переадресация (*HTTP 302*):

```
public RedirectResult SomeMethod()
{
    return Redirect("/Home/MyResult");
}
```

Постоянная переадресация (*HTTP 301*):

```
public RedirectResult SomeMethod()
{
    return RedirectPermanent("/Home/MyResult");
}
```

Перейти к определенному action определенного controller-a:

```
public RedirectToRouteResult SomeMethod()
{
    return RedirectToAction("MyResult", "Home");
}
```

# CONTENT RESULT

```
public string Square(int a, int h)
{
    var s = (double) a * h / 2;
    return $"<h2>Площадь треугольника с основанием {a}
           и высотой {h} равна {s}.</h2>";
}
```

// Из-за отсутствия ActionResult будет преобразовано в

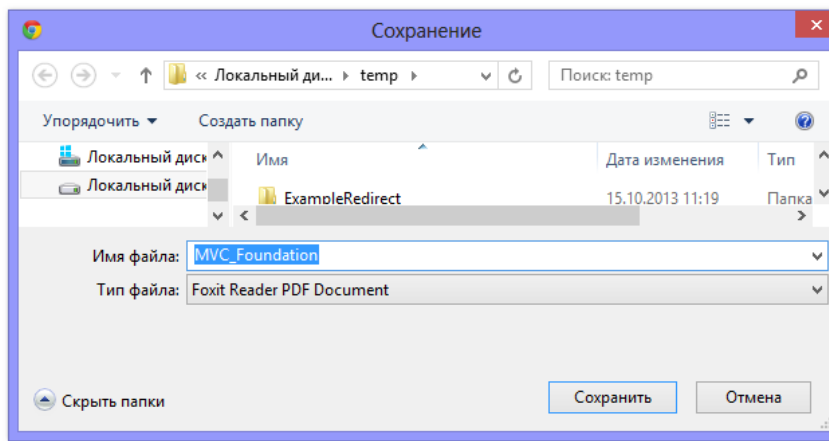
```
public ContentResult Square(int a, int h)
{
    var s = (double) a * h / 2;
    return Content($"<h2>Площадь треугольника с основанием {a}
                   и высотой {h} равна {s}.</h2>");
}
```

# FILE RESULT

```
public FileResult SendFile()  
{  
    // Путь к файлу  
    string filePath = @"D:\temp\01-mvc-1part.pdf";  
    // Тип файла - content-type  
    string fileType= "application/pdf";  
    // Имя файла  
    string  
    return  
}
```

## Index

Hello MVC [link](#)  
[Download](#)



# STATUS CODE RESULT

---

```
public ActionResult Check(int age)
{
    if (age < 21)
    {
        return new HttpStatusCodeResult(404);
    }
    return View();
}
```



# REQUEST CONTEXT

---

Объекты контекста:

**Request, Response, RoutedData, HttpContext,  
ControllerContext, Server**

Объект **HttpContext** описывает данные конкретного HTTP-запроса, который обрабатывается приложением.

А **ControllerContext** описывает данные HTTP-запроса по отношению к данному контроллеру.

# REQUEST CONTEXT

Объект `HttpContext` описывает данные конкретного HTTP-запроса, который обрабатывается приложением.

- Получение браузера пользователя:  
`HttpContext.Request.Browser`
- Иногда просто браузера недостаточно, тогда можно обратиться к агенту пользователя:  
`HttpContext.Request.UserAgent`
- Получение URL-запроса:  
`HttpContext.Request.RawUrl`
- Получение IP-адреса пользователя:  
`HttpContext.Request.UserHostAddress`
- Получение пути к странице:  
`HttpContext.Request.Path`

# REQUEST CONTEXT

---

// определяем, принадлежит ли пользователь к администраторам

```
bool isAdmin = HttpContext.User.IsInRole("admin");
```

// аутентифицирован ли пользователь

```
bool isAuthenticated = HttpContext.User.Identity.IsAuthenticated;
```

// логин авторизованного пользователя

```
string login = HttpContext.User.Identity.Name;
```

# REQUEST CONTEXT. COOKIES

Получить значение cookie:

СВОЙСТВО `HttpContext.Request.Cookies`

```
string id = HttpContext.Request.Cookies["id"].Value;
```

Установить значение cookie:

СВОЙСТВО `HttpContext.Response.Cookies`

```
var idCookie = new HttpCookie("id")
{
    Expires = DateTime.UtcNow.AddDays(-1),
    Value = null
};
Response.Cookies.Add(idCookie);
```



A wide-angle photograph of a busy London street, likely Regent Street, during the 'golden hour' of sunset. The sun is low in the sky, creating a strong, warm glow and casting long, dark shadows of the pedestrians onto the wet pavement. The street is filled with a large crowd of people walking in various directions. On the left, a grand, light-colored stone building with classical architectural features like columns and arched windows is visible. A Union Jack flag flies from a balcony on this building. A red traffic light is positioned at the street corner. A 'superdry' store is located on the ground floor of the building. A street sign above the entrance reads 'REGENT STREET W1'. Further down the street, another 'superdry' flag is visible. On the right side of the street, a Starbucks logo is seen on a building facade, and a red awning extends over the sidewalk. The overall atmosphere is one of a vibrant, bustling urban environment.

# VIEWS



# SIMPLIEST VIEW

```
@{  
    Layout = null;  
}  
<!DOCTYPE html>  
<html>  
<head>  
    <meta name="viewport" content="width=device-width" />  
    <title>Index</title>  
</head>  
<body>  
    <h2>Index</h2>  
    <div>  
        @ViewBag.Message  
    </div>  
</body>  
</html>
```

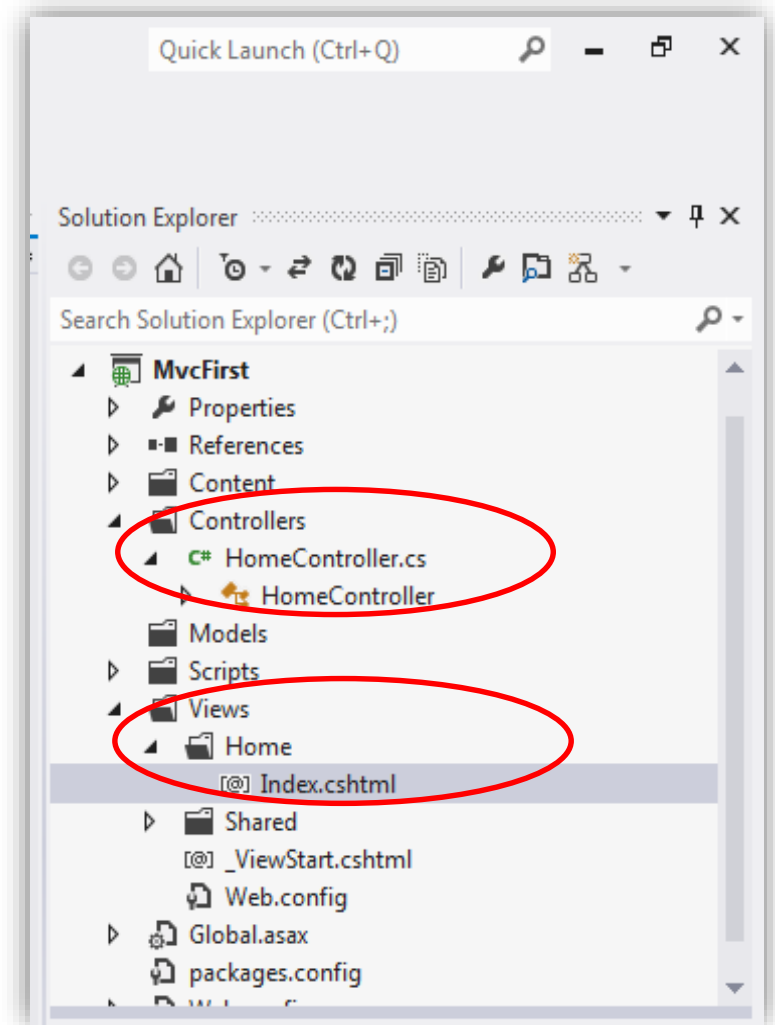
# VIEW PATHS

Все добавляемые **view**, как правило, группируются по контроллерам в соответствующие папки в каталоге `~/Views`.

Представления, которые относятся к методам **HomeController**, будут находиться в проекте в каталоге `~/Views/Home`.

Однако, мы сами можем создать в каталоге `~/Views` папку с произвольным именем, для хранения новой **view**, не связанного с определенными методами в **HomeController**.

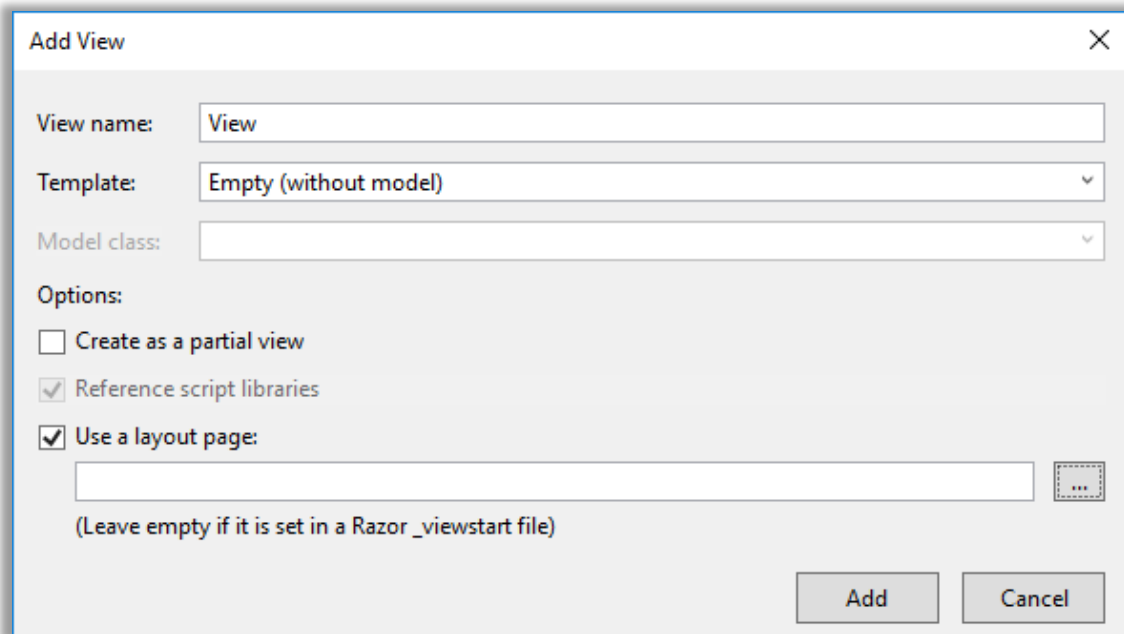
Тогда, чтобы произвести рендеринг **view** в выходной поток:



```
return View("~/Views/Some/SomeView.cshtml");
```

# NEW VIEW

- **View name** — имя **view**.
- **Template** — выбор шаблона формирования новой **view** (Empty, Create, Delete, Details, Edit, List).
- **Reference Script Libraries** — включение файлов JavaScript.



Add View

View name: View

Template: Empty (without model)

Model class:

Options:

- ☐ Create as a partial view
- ☒ Reference script libraries
- ☒ Use a layout page:

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

Add Cancel

- **Create as a Partial View** - **view** будет неполной, в ее шапке не будет таких тегов, как **<html>** и **<head>**.
- **Use a layout or Master Page** - эта опция указывает, будут ли использоваться **Layout** страницы (главные страницы) или **view** будет самодостаточной.

# CONTROLLER-VIEW DATA TRANSFER

```
// ~/Controllers/HomeController.cs
public ActionResult Index(){
    ViewBag.First = "Hello";
    ViewData["Second"] = " user";
    int number = 33;
    return View(number);
}
// ~/Views/Home/Index.cshtml
<p>
    @ViewBag.First
    @ViewData["Second"]
    @Model
</p>
```

Нельзя передавать динамические значения без приведения типа из **ViewBag** в методы расширения во **view**, так как компилятор C# должен знать тип каждого параметра во время компиляции:

```
@Html.TextBox("name",
ViewBag.Name)
```

```
@Html.TextBox("name",
ViewData["Name"])
```

# RAZOR SYNTAX

## Razor is a cut above the rest

Web Forms  
(6 markup transitions)

```
<ul>
  <% for (int i = 0; i < 10; i++) { %>
    <li><% =i %></li>
  <% } %>
</ul>
```

PHP  
(2 markup transitions  
& an echo)

```
<ul>
  <?php
    for ($i = 0; $i < 10; $i++) {
      echo("<li>$i</li>");
    }
  ?>
</ul>
```

Razor  
(2 markup transitions)

```
<ul>
  @for (int i = 0; i < 10; i++) {
    <li>@i</li>
  }
</ul>
```



# RAZOR SYNTAX

@\* Some more Razor examples \*@

<span>

Price including Sale Tax: @Model.Price \* 1.2

</span>

<span>

Price including Sale Tax: @(Model.Price \* 1.2)

</span>

@if (Model.Count > 5)

{

<ol>

@foreach (var item in Model)

{

<li>@item.Name</li>

}

</ol>

}

33 \* 1.2

39,6

# RAZOR SYNTAX

<-- Простое смешивание C# кода и HTML разметки -->

```
<div>
    @if (User.IsInRole("admin")) {
        <a href="/Edit">Edit</a>
    }
</div>
```

<!-- Включение блока кода в разметку -->

```
@{
    var name = "Ivan Ivanov";
    <span>Your name:</span> @name
}
```

<!-- Включение кода в разметку -->

```
<ul>
    @for (int i = 0; i < 10; i++) {
        <li>@i</li>
    }
</ul>
```

<!-- Комментарии в разметке -->

```
@*
    <div>
        Hello World
    </div>
*@
```

<!-- Комментарии в коде -->

```
@{
    //var name = "John Doe";
    //@name
}
```

# RAZOR SYNTAX

```
БЕЗ RAZOR <%=Math.PI %>  
RAZOR @Math.PI
```

<div>

<span>Текст внутри тега html </span><br />

если текст вне тегов html,

то можно отделять спец. тегом @\* <text> \*@

</div>

```
@for (int i = 0; i < 4; i++)  
{
```

<span>строка </span> @i <text> , </text> <br />

```
}
```

# LAYOUT SYNTAX

~/Views/Shared/\_Layout.cshtml

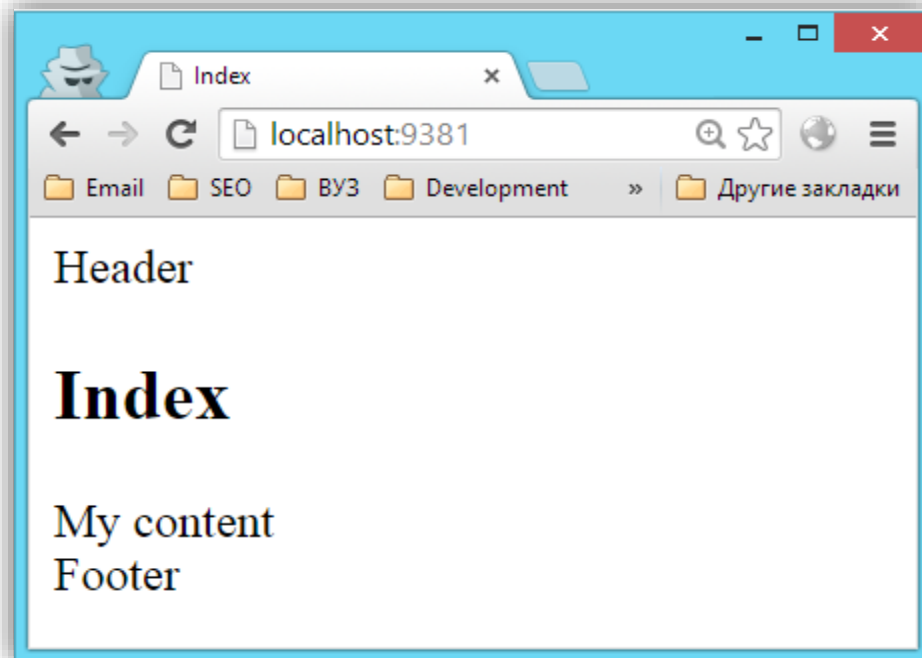
```
<html>
  <head>
    <title>Simple Layout</title>
  </head>
  <body>
    @RenderBody()
  </body>
</html>
```

MyPage.cshtml

```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
<p>
    My content goes here
</p>
```

# LAYOUT SYNTAX

```
<!-- ~/Views/Shared/_Layout.cshtml -->
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>@ViewBag.Title</title>
</head>
<body>
  Header
  <div>
    @RenderBody()
  </div>
  Footer
</body>
</html>
```



# LAYOUT SYNTAX

```
<!-- ~/Views/Home/Index.cshtml -->
```

```
@{
```

```
    ViewBag.Title = "Index";
```

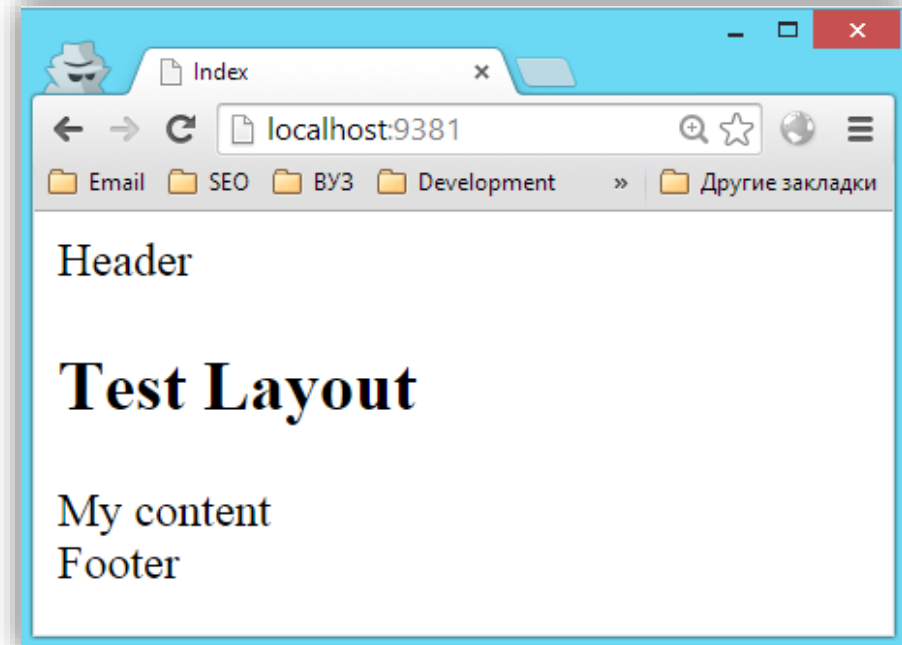
```
    Layout = "~/Views/Shared/_Layout.cshtml";
```

```
}
```

```
<h2>Test Layout</h2>
```

```
<p> @ViewBag.Content </p>
```

```
public ActionResult Index() {  
    ViewBag.Content = "My content";  
    return View();  
}
```



Если мы не используем `layout`-страницу, то мы указываем `Layout = null`;

# LAYOUT SYNTAX

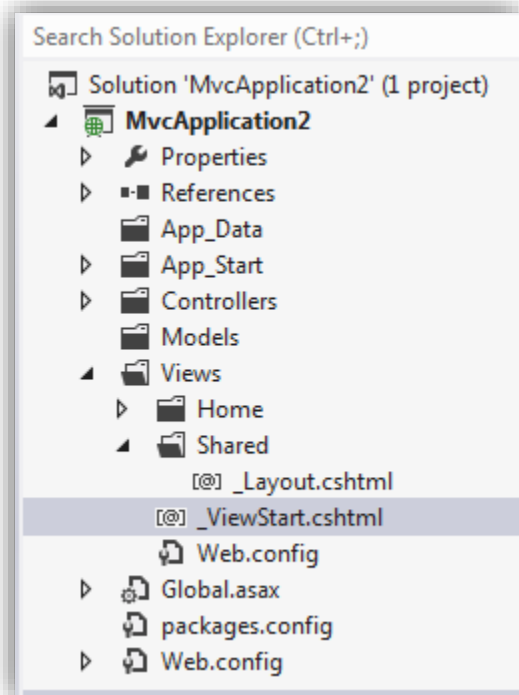
```
<!-- _ViewStart.cshtml -->
```

```
<!-- Код и разметка в этом файле применяется к каждой view,  
находящейся в этом каталоге -->
```

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

```
<!-- Index.cshtml -->
```

```
@{  
    ViewBag.Title = "MyIndex";  
}  
<h2>MVC</h2>  
<p>  
    @ViewBag.Content  
</p>
```





# LAYOUT SECTIONS

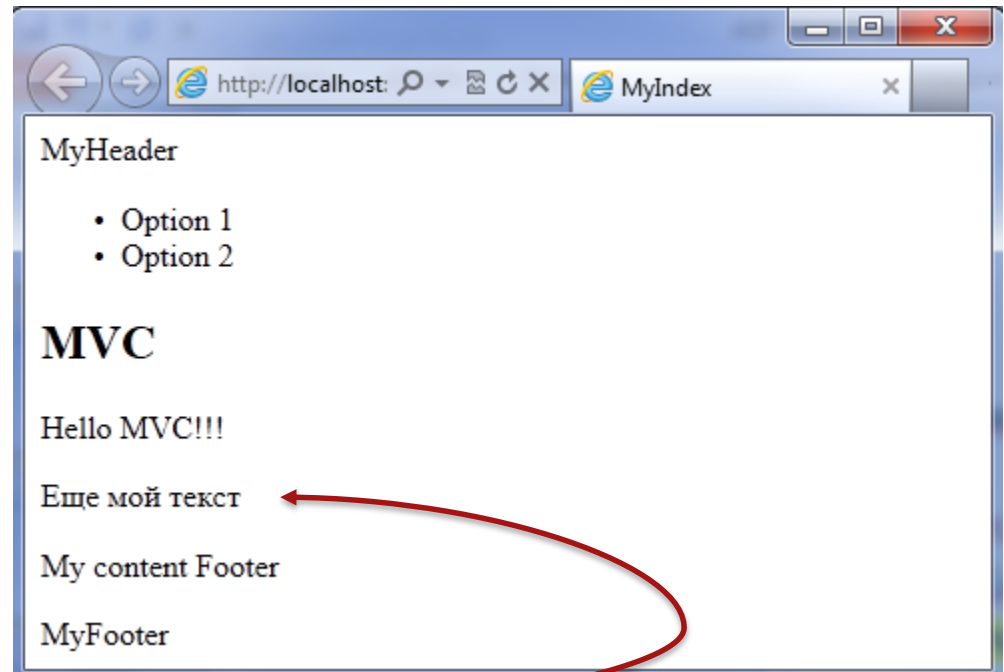
```
<!-- ~/Views/Shared/_Layout.cshtml -->
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport"
    content="width=device-width" />
  <title>@ViewBag.Title</title>
</head>
<body>
  MyHeader
  @RenderSection("menu") <!-- метка для секции menu -->
  <div>
    @RenderBody()
  </div>
  @RenderSection("news", false) <!-- метка для секции news -->
  MyFooter
</body>
</html>
```

Секции позволяют странице  
занимать несколько блоков в  
шаблоне

false - необязательная секция

# LAYOUT SECTIONS

```
<!-- Index.cshtml -->
@{
    ViewBag.Title = "MyIndex";
}
<h2>MVC</h2>
<p>    @ViewBag.Msg </p>
@section menu {
    <ul id="pageMenu">
        <li>Option 1</li>
        <li>Option 2</li>
    </ul>
}
@section news{
    <p>My content Footer</p>
}
Еще мой текст
```



# PARTIAL VIEWS

```
<!-- Partial.cshtml -->
```

<span>Это частичное представление без участия кода логики</span>

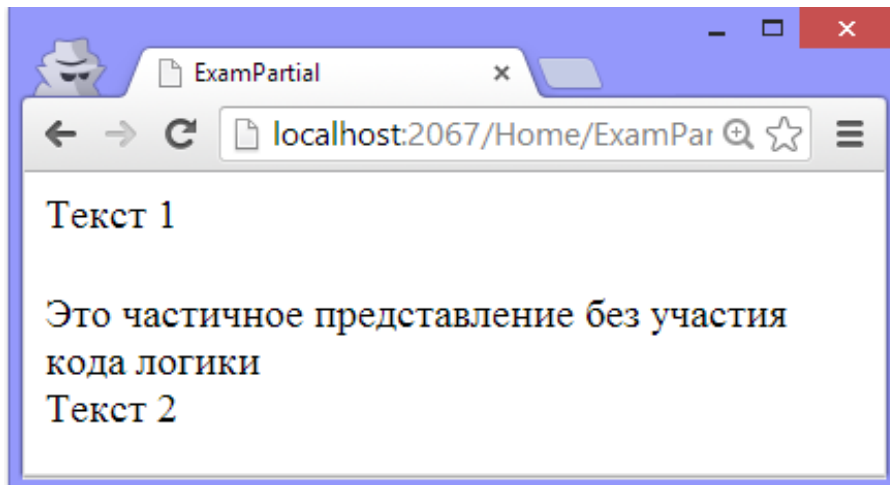
```
<!-- ExamPartial.cshtml -->
```

```
@{ Html.RenderPartial("Partial"); }
```

```
<!-- или -->
```

```
@Html.Partial("Partial")
```

<div>Текст 2</div>



Add View

View name:

Template:

Model class:

Options:

☐ Create as a partial view

☒ Reference script libraries

☒ Use a layout page:

(Leave empty if it is set in a Ra

# PARTIAL VIEWS

```
<!-- Partial.cshtml -->
```

```
<h2>@ViewBag.Msg</h2>
```

Это частичное представление без участия кода логики `<br />`

```
<!-- ExamPartial.cshtml -->
```

```
<div>Текст 1</div>
```

```
@Html.Action("Partial")
```

```
@Html.Partial("Partial")
```

```
<div>Текст 2</div>
```

```
// HomeController.cs
```

```
public PartialViewResult Partial()
```

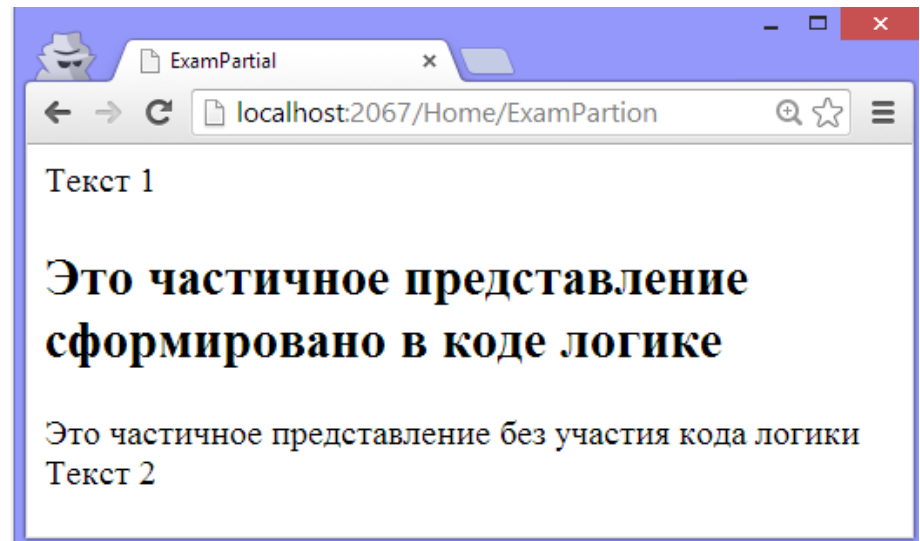
```
{
```

```
    ViewBag.Msg =
```

```
    "Это частичное представление  
сформировано в коде логики";
```

```
    return PartialView();
```

```
}
```







# HELPERS

# HTML HELPER

---

**HtmlHelper** — объект, имеющий множество различных методов, позволяющих упростить генерацию разметки.

Разделяют:

- Не типизированные **helper**-методы
- Строго типизированные **helper**-методы.

# INLINE HELPERS

Например, необходимо отобразить во view несколько однотипных списков:

```
public ActionResult Index()
{
    List<String> ordinals =
    new List<String> { "first", "second", "third" };

    ViewBag.Colors = new List<String> { "red", "green", "blue" };

    return View(ordinals);
}
```



# INLINE HELPERS

```
<!-- Index.cshtml -->
<div>
  <select name="Colors">
    @foreach (string color in ViewBag.Colors)
    {
      <option value="@color" >@color</option>
    }
  </select>
</div>
<div>
  <select name="Ordinals">
    @foreach (string ordinal in Model)
    {
      <option value="@ordinal" >@ordinal</option>
    }
  </select>
</div>
```

НЕДОСТАТОК  
ПОВТОР КОДА

# INLINE HELPERS

```
@helper CreateSelect(List<string> items, string name)
{
    <select name="@name">
        @foreach (string item in items)
        {
            <option value="@item" >@item</option>
        }
    </select>
}
<!-- использование inline helper-a -->
@CreateSelect(ViewBag.Color as List<string>, "Colors")
@CreateSelect(Model as List<string>, "Ordinals")
```

НО, если хелпер очень объемный, то он может очень сильно захламлять разметку представления.

# CUSTOM HELPERS

```
// вместо inline helper-а можно создать extension method
namespace MvcApplication1.Helpers {
    public static class SelectHelpers
    { // первый параметр - Тип, для которого создается extension method
        public static MvcHtmlString CreateSelect(this HtmlHelper html,
            List<string> items, string name)
        {
            var select = new TagBuilder("select");
            select.MergeAttribute("name", name);
            foreach (string item in items) {
                var option = new TagBuilder("option");
                option.MergeAttribute("value", item);
                option.SetInnerText(item);
                select.InnerHtml += op.ToString();
            }
            return new MvcHtmlString(select.ToString());
        }
    }
}
```

# CUSTOM HELPERS

---

Тогда, представление будет иметь следующий вид:

```
@using MvcApplication1.Helpers
```

```
@Html.CreateSelect(ViewBag.Color as List<string>, "Colors")
```

```
@Html.CreateSelect(Model as List<string>, "Ordinals")
```

# ACTION HELPERS

- **Html.ActionLink()**

```
@Html.ActionLink("Click here to view photo 1",  
"Display", new { id = 1 })
```



```
<a href="/photo/display/1">  
Click here to view photo 1  
</a>
```

- **Url.Action()**

```

```



```

```

# DISPLAY HELPERS

- **Html.DisplayNameFor()**

```
@Html.DisplayNameFor(model => model.CreatedDate)
```



Created Date

- **Html.DisplayFor()**

```
@Html.DisplayFor(model => model.CreatedDate)
```



03/12/2012

# FORM HELPERS

## Html.BeginForm()

```
@using (Html.BeginForm("Create", "Photo",  
    FormMethod.Post,  
    new { enctype = "multipart/form-data" }))  
{  
    @* Place input controls here *@  
}
```



```
<form action="/Photo/Create" method="post"  
    enctype="multipart/form-data">  
  
</form>
```



# VALIDATION HELPERS

## Html.ValidationSummary()

@Html.ValidationSummary()



```
<ul>  
  <li>Please enter your last name</li>  
  <li>Please enter a valid email address</li>  
</ul>
```

## Html.ValidationMessageFor()

@Html.ValidationMessageFor(model => model.Email)



Please enter a valid email address

# EDITOR HELPERS

## Html.LabelFor()

```
@Html.LabelFor(model => model.ContactMe)
```



```
<label for="ContactMe">  
  Contact Me  
</label>
```

## Html.EditorFor()

```
@Html.EditorFor(model => model.ContactMe)
```



```
<input type="checkbox"  
  name="Description">
```

# FORMS WITH HELPERS

Хелпер `@Html.BeginForm()` создает тег `<form>`, например:

```
@using (Html.BeginForm())
{
    @Html.ActionLink("Редактировать", "Edit")
}
<!-- преобразуется в -->
<form action="/" method="post">
    <a href="/Home/Edit">Редактировать</a>
</form>
```

Хелпер `@Html.BeginForm()` имеет множество перегрузок, позволяющих модифицировать поведение и атрибуты формы:

```
@using (Html.BeginForm("Index", "Home", FormMethod.Post)){ . . . }
```

# NON-TYPED HELPERS

Не типизированные хелперы проверяют объекты `ViewData` и `ViewBag` и получают из них значения, которое затем отображают:

```
// ~/Controllers/HomeController.cs
```

```
public ActionResult Index()
{
    ViewBag.EnterTxt = "Введите текст";
    return View();
}
```

```
<!-- ~/Views/Home/Index.cshtml -->
```

```
@Html.TextBox("EnterTxt")
```

```
<!-- будет преобразовано в -->
```

```
<input id="EnterTxt" name="EnterTxt" type="text" value="Введите текст" />
```

# NON-TYPED HELPERS

```
// ~/Controllers/HomeController.cs
```

```
public ActionResult Index()
```

```
{
```

```
    ViewData["EnterTxt"] = "Введите текст";
```

```
    return View();
```

```
}
```

```
<!-- ~/Views/Home/Index.cshtml -->
```

```
@Html.TextBox("Txt", ViewData["EnterTxt"])
```

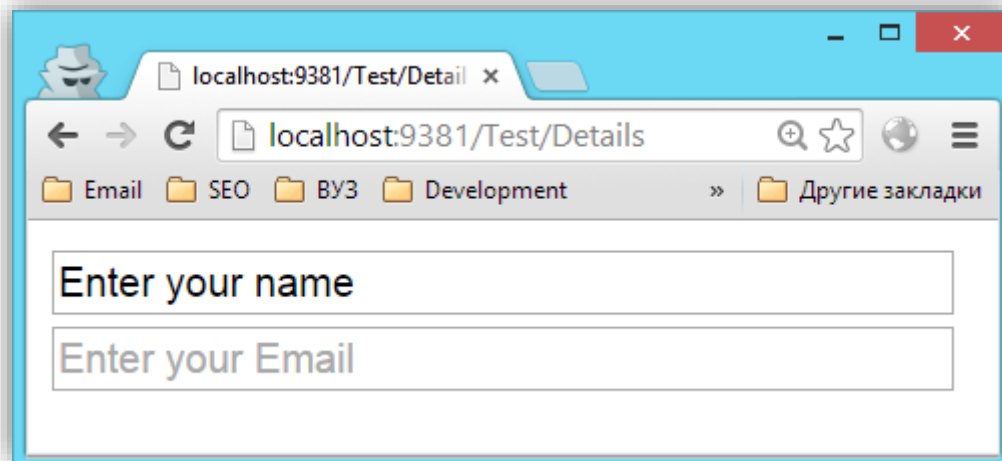
```
<!-- будет преобразовано в -->
```

```
<input id="Txt" name="Txt" type="text" value="Введите текст" />
```

# NON-TYPED HELPERS

`@Html.TextBox()` — создает элемент `<input type="text" />` (метод перегружен).

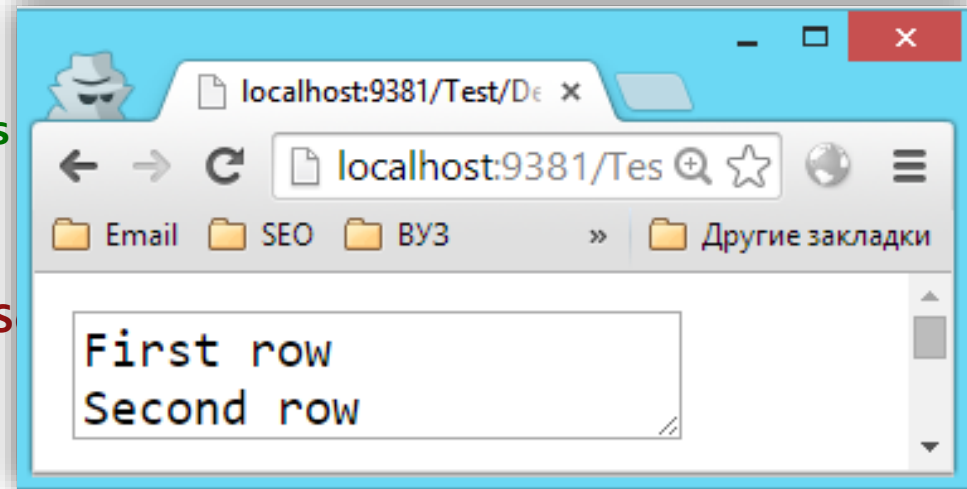
```
@using (Html.BeginForm("Details", "Test", FormMethod.Post))
{
    @Html.TextBox("UserName", "Enter your name",
        new { style = " width:300px" })
    @Html.TextBox("UserEmail", null, new {
        placeholder="Enter your Email", style = " width:300px" })
}
```



# NON-TYPED HELPERS

`@Html.TextArea()` — создает элемент `<textarea>`, который предоставляет многострочное текстовое поле:

```
<!-- ~/Controllers/HomeController.cs
public ActionResult Details()
{
    ViewData["Text"] = "First row\nS
    return View();
}
```



```
<!-- ~/Views/Home/Index.cshtml -->
@using (Html.BeginForm("Details", "Test", FormMethod.Post))
{
    @Html.TextArea("Text", (string)ViewData["Text"]);
}
```

# NON-TYPED HELPERS

`@Html.Hidden()` — создает элемент `<input type="hidden" />`:

```
@using (Html.BeginForm()){
    ViewData["Id"] = "111";
    @Html.Hidden("UserId", (string)ViewData["Id"])
}
<!-- создаст -->
<input id="UserId" name="UserId" type="hidden" value="111" />
```

`@Html.Password()` — создает элемент `<input type="password" />`:

```
@using (Html.BeginForm()){
    @Html.Password("UserPasswordd", null, new { placeholder = "Password" })
}
<!-- создаст -->
<input id="UserPassword" name="UserPassword" placeholder="Password" type="password" />
```



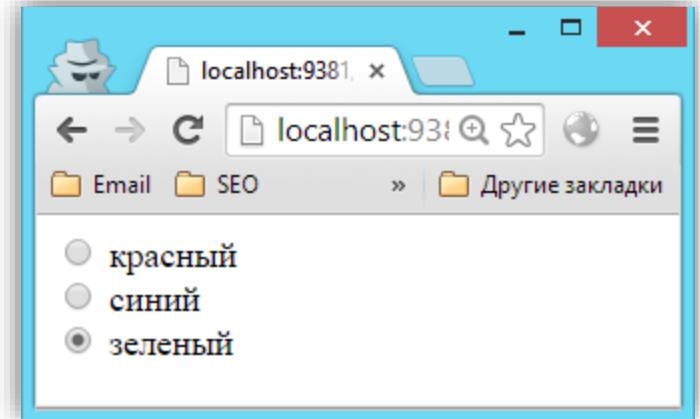
# NON-TYPED HELPERS

`@Html.RadioButton()` — создает элемент `<input type="radio" />`:

```
@using (Html.BeginForm()){
    @Html.RadioButton("color", "red")
    <span>красный</span><br />
    @Html.RadioButton("color", "blue")
    <span>синий</span><br />
    @Html.RadioButton("color", "green", true)
    <span>зеленый</span>
}
```

`<!-- преобразуется в -->`

```
<input id="color" name="color" type="radio" value="red" />
<span>красный</span><br />
<input id="color" name="color" type="radio" value="blue" />
<span>синий</span><br />
<input id="color" name="color" type="radio" value="green" checked="checked" />
<span>зеленый</span>
```



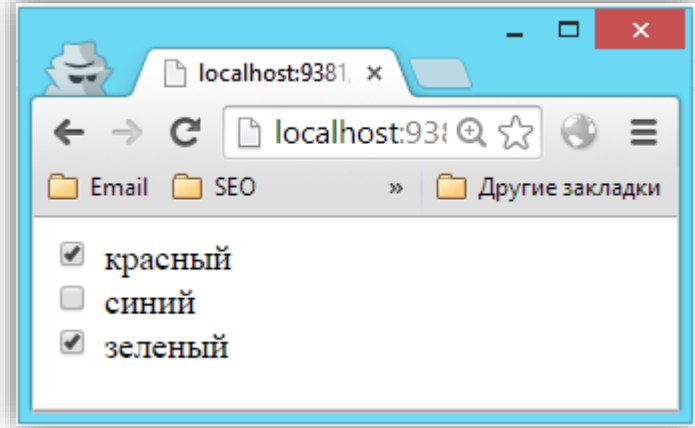
# NON-TYPED HELPERS

`@Html.CheckBox()` — создает элемент `<input type="checkbox" />`:

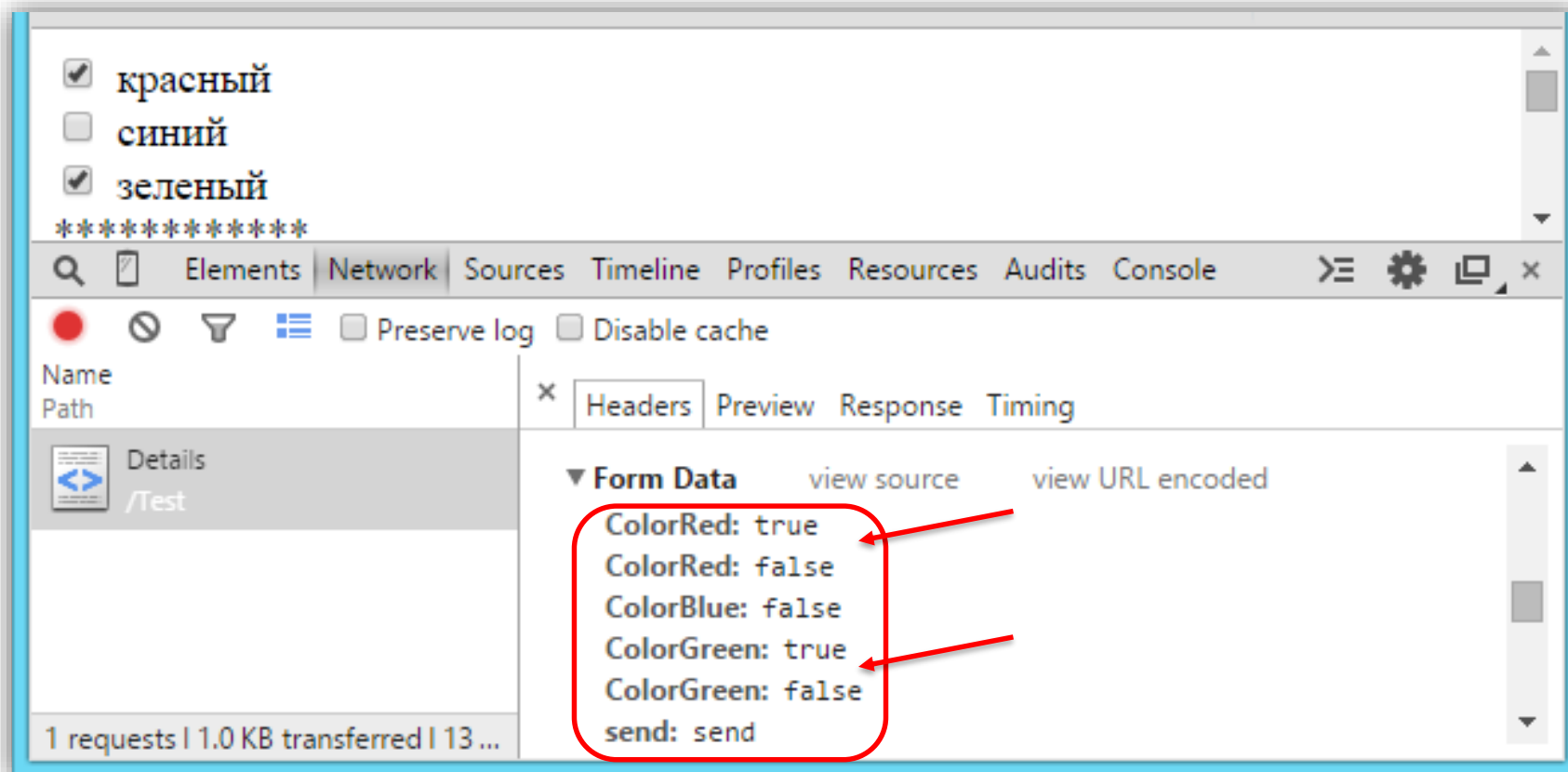
```
@using (Html.BeginForm()){
    @Html.CheckBox("ColorRed", true)
    <span>красный</span><br />
    @Html.CheckBox("ColorBlue")
    <span>синий</span><br />
    @Html.CheckBox("ColorGreen", true)
    <span>зеленый</span>
}
```

`<!-- преобразуется в -->`

```
<input id="ColorRed" name="ColorRed" type="checkbox" value="true" />
<input name="ColorRed" type="hidden" value="false" />
<span>красный</span><br />
<input id="ColorBlue" name="ColorBlue" type="checkbox" value="true" />
<input name="ColorBlue" type="hidden" value="false" />
<span>синий</span><br />
<input id="ColorGreen" name="ColorGreen" type="checkbox" value="true"/>
<input name="ColorGreen" type="hidden" value="false" />
<span>зеленый</span>
```



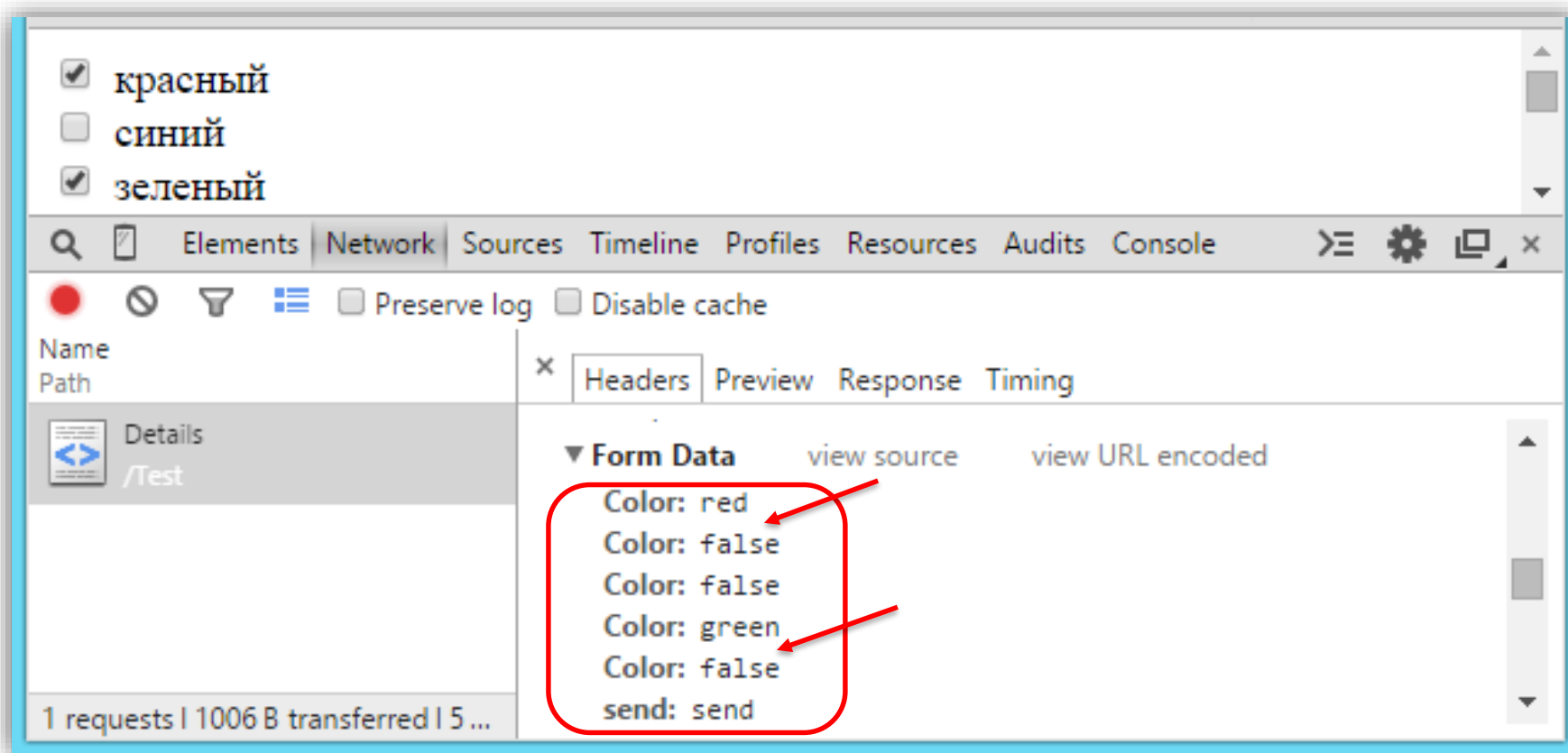
# NON-TYPED HELPERS



# NON-TYPED HELPERS

```
@using (Html.BeginForm()){
    @Html.CheckBox("Color", false, new { id = "ColorRed", value = "red" })
    <span>красный</span><br />
    @Html.CheckBox("Color", false, new { id = "ColorBlue", value = "blue" })
    <span>синий</span><br />
    @Html.CheckBox("Color", false, new { id = "ColorGreen", value = "green" })
    <span>зеленый</span>
}
<!-- преобразуется в -->
<input id="ColorRed" name="Color" type="checkbox" value="red" />
<input name="Color" type="hidden" value="false" />
<span>красный</span><br />
<input id="ColorBlue" name="Color" type="checkbox" value="blue" />
<input name="Color" type="hidden" value="false" />
<span>синий</span><br />
<input id="ColorGreen" name="Color" type="checkbox" value="green"/>
<input name="Color" type="hidden" value="false" />
<span>зеленый</span>
```

# NON-TYPED HELPERS



# NON-TYPED HELPERS

`@Html.Label()` — создает элемент `<label>`, который устанавливает связь между определенной меткой, в качестве которой обычно выступает текст, и элементом формы (`<input>`, `<select>`, `<textarea>`):

```
<input type="text" id="TextFirst">
```

```
@Html.Label("TextFirst", "Жми", new { style = "cursor: pointer" })
```

`<!-- преобразуется в -->`

```
<input type="text" id="TextFirst">
```

```
<label for="TextFirst" style="cursor: pointer">Жми</label>
```

# NON-TYPED HELPERS

`@Html.DropDownList()` — создает элемент `<select>`:

```
private List<SelectListItem> GetColors()
{
    List<SelectListItem> list = new List<SelectListItem>{
        new SelectListItem { Value = "Red", Text = "Красный" },
        new SelectListItem { Value = "Green", Text = "Зеленый", Selected = true },
        new SelectListItem { Value = "Blue", Text = "Синий" }
    };
    return list;
}

...
ViewBag.Colors = GetColors();
...

<!-- BO view -->
@Html.DropDownList("Colors", ViewBag.Colors as IEnumerable<SelectListItem>);
```

# NON-TYPED HELPERS

`@Html.ListBox()` — создает элемент `<select multiple="multiple">`:

```
private List<SelectListItem> GetColors()
{
    List<SelectListItem> list = new List<SelectListItem>{
        new SelectListItem { Value = "Red", Text = "Красный" },
        new SelectListItem { Value = "Green", Text = "Зеленый", Selected = true },
        new SelectListItem { Value = "Blue", Text = "Синий" }
    };
    return list;
}

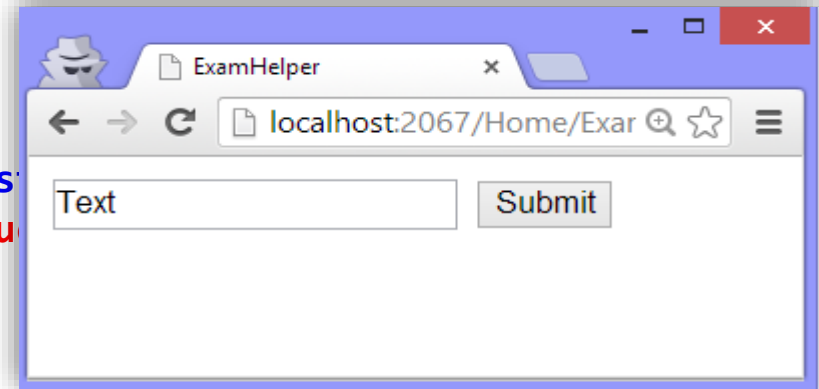
...
ViewBag.Colors = GetColors();
...

<!-- BO view -->
@Html.ListBox("Colors", ViewBag.Colors as IEnumerable<SelectListItem>);
```

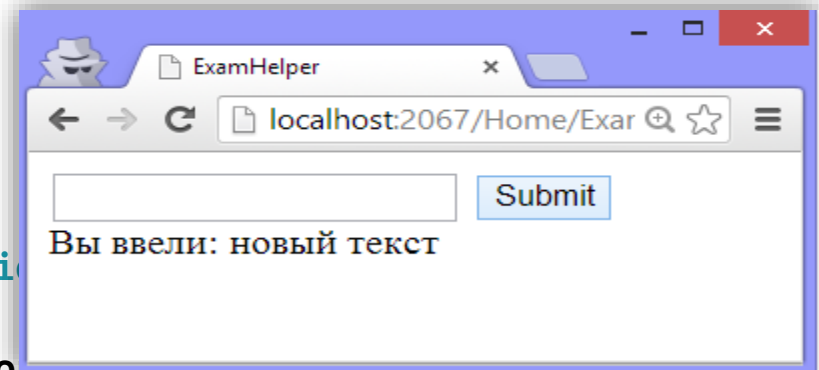


# VIEW-CONTROLLER DATA TRANSFER

```
<!-- ~/Views/Home/ExamHelper.cs -->
@using (Html.BeginForm())
{
    @Html.TextBox("Text", ViewBag.Text as string)
    <input type="submit" name="Button" value="Submit" />
    <div>@ViewBag.TextReturn</div>
}
```



```
<!-- ~/Controllers/HomeController.cs -->
[HttpGet]
public ActionResult ExamHelper()
{
    ViewBag.Text = "Text";
    return View();
}
[HttpPost]
public ActionResult ExamHelper(FormCollection form)
{
    ViewBag.TextReturn = "Вы ввели: " + form["Text"];
    return View();
}
```



**QUESTIONS?**

A light blue world map is centered in the background, showing the outlines of continents and countries. The map is semi-transparent, allowing the white text to stand out clearly.

**THANKS FOR YOUR  
ATTENTION!**