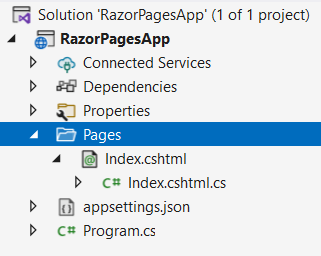
**Определение пользовательского интерфейса**

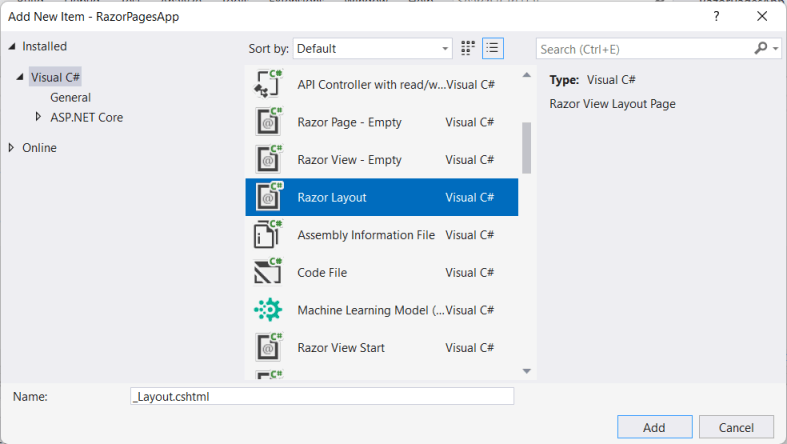
**Мастер-страницы layout**

Файлы **layout** или мастер-страницы позволяют определить единый шаблон для страниц Razor и применяются для создания единообразного, унифицированного вида приложения. Для определения интерфейса мастер-страницы также применяют код Razor и html, как обычные страницы razor, но при этом упрощают создание приложения. Например, можно определить на мастер-странице общие для всех остальных страниц Razor меню и другие элементы, а также подключить общие стили и скрипты. В итоге нам не придется на каждой отдельной странице прописывать путь к файлам стилей, а потом при необходимости его изменять.

Например, у нас есть страница Razor **Index.cshtml** и код связанной модели IndexModel в файле **Index.cshtml.cs**:



Создадим в папке **Pages** новый каталог, который назовем **Shared**. Далее добавим в этот каталог добавим новый файл, который назовем **\_Layout.cshtml**. Для добавления мастер-страниц в Visual Studio можно использовать шаблон файла **Razor Layout**:



После добавления файла **\_Layout.cshtml** изменим его код следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <!DOCTYPE html>  <html>  <head>      <meta name="viewport" content="width=device-width" />      <title>METANIT.COM | @ViewBag.Title</title>  </head>  <body>      <h2>@ViewBag.Title</h2>      <div><a href="/">Home</a> | <a href="About">About</a></div>      <div>          @RenderBody()      </div>  </body>  </html> |

Код мастер-страницы напоминает полноценную веб-страницу: здесь присутствуют основные теги <html>, <head>, <body> и так далее. И также здесь могут использоваться конструкции Razor. Например, через выражение @ViewBag.Title из каждой отдельной страницы Razor будет передаваться значение для заголовка веб-страницы.

Отличительной особенностью файлов layout является использование метода **@RenderBody()**. Этот метод будет вставлять содержимое страниц Razor, которые используют данную мастер-страницу. В итоге мы сможем легко установить для всех страниц единообразный стиль оформления.

Теперь определим папке **Pages** следующую страницу **Index.cshtml**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | @page    @{      ViewBag.Title = "Index";      Layout = "/Pages/Shared/\_Layout.cshtml";  }  <h3>Index Content</h3> |

Здесь задается значение ViewBag.Title, которое применяется на мастер-странице для вывода заголовка.

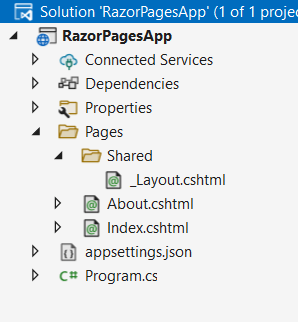
Кроме того, с помощью свойства **Layout** устанавливается используемая мастер-страница layout. В данном случае это файл по пути */Pages/Shared/\_Layout.cshtml*

Также добавим в папку **Pages** новую страницу **About.cshtml** со следующим кодом:

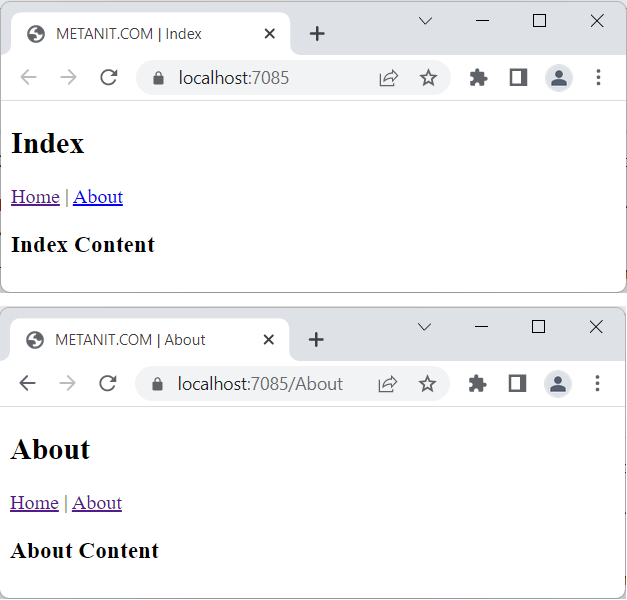
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | @page    @{      ViewBag.Title = "About";      Layout = "/Pages/Shared/\_Layout.cshtml";  }  <h3>About Content</h3> |

Оно выглядит аналогично странице Index.cshtml.

В итоге весь проект будет выглядеть следующим образом:



Запустим проект и в браузере при обращении к обоим страницам Razor мы лицезреем в браузере единообразую веб-страницу:

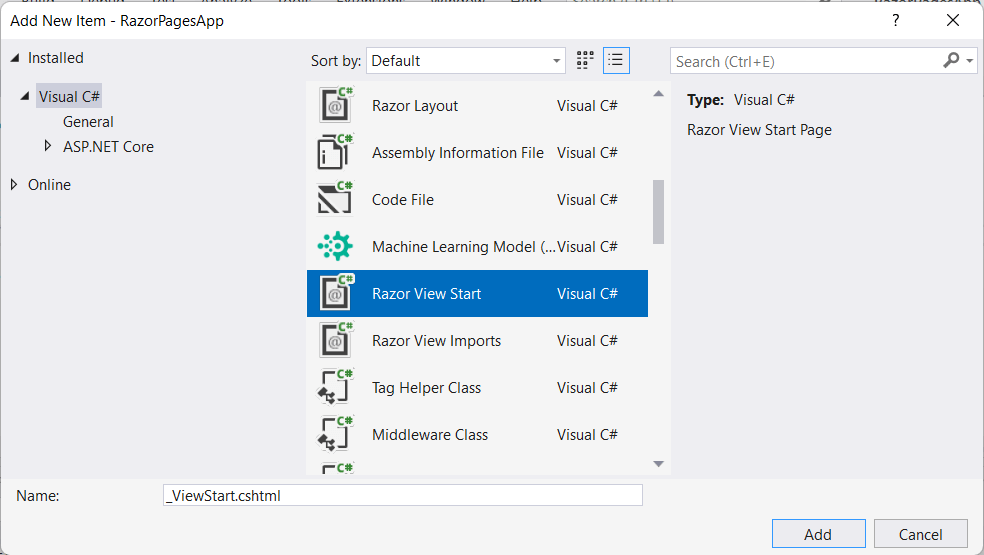
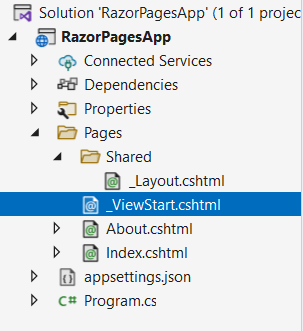


При необходимости мы можем определять одни и те же или разные мастер-страницы для разных страниц Razor.

### \_ViewStart.cshtml

Хотя выше приведенный код вполне успешно работает, у нас есть одна проблема - мы сталкиваемся с необходимостью на каждой странице явным образом прописывать, какую мастер-страницу layout будет применять страницы. Чтобы упростить данное действие, можно применять файлы **\_ViewStart.cshtml**

Итак, добавим в папку **Pages** новый файл, которое назовем **\_ViewStart.cshtml**. Для этого можно использовать элемент **Razor View Start**:

Код этого файла добавляется в самое начало кода преставлений при их запуске. Определим в файле **\_ViewStart.cshtml** следующий код (в принципе этот же код должен быть в добавляемом файле по умолчанию):

|  |  |
| --- | --- |
| 1  2  3 | @{      Layout = "\_Layout";  } |

Когда будет происходить рендеринг страницы Razor, то система будет искать мастер-страницу \_Layout по следующим путям:

/Pages/\_Layout.cshtml

/Pages/Shared/\_Layout.cshtml

Если в обеих папках: и в */Pages*, и в */Pages/Shared/* имеется файл с одинаковым именем, например, \_Layout.cshtml, то к странице применяется файл, который находится с ним в одной папке как более приоритетный.

После определения этого файла мы можем удалить из страниц **Index.cshtml** и **About.cshtml** подключение мастер-страницы. Например, код страницы **Index.cshtml**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @page    @{      ViewBag.Title = "Index";  }  <h3>Index Content</h3> |

### Переопределение мастер-страницы

Если вдруг мы хотим глобально по всему проекту поменять мастер-страницу на другой файл, который расположен в какой-то другой папке, например, в корне каталога Pages, то нам надо использовать полный путь к файлу в **\_ViewStart.cshtml**:

|  |  |
| --- | --- |
| 1  2  3 | @{      Layout = "~/Pages/Shared/\_Layout.cshtml";  } |

Код из *\_ViewStart.cshtml* выполняется до любого кода на странице Razor.

Естественно также мы можем переопределить мастер-страницу на каждой отдельной странице с помощью свойства Layout.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | @page    @{      ViewBag.Title = "Home Page";      Layout = "~/Pages/Shared/\_Layout.cshtml";  }  <h2>Страница Index.cshtml</h2> |

Мы можем вообще не использовать мастер-страницу, тогда на странице Razor свойству Layout надо присвоить значение null:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | @page    @{      Layout = null;  }    <!DOCTYPE html>  <html>  <head>      <meta name="viewport" content="width=device-width" />      <title>Home Page</title>  </head>  <body>      <h2>Страница Index.cshtml</h2>  </body>  </html> |

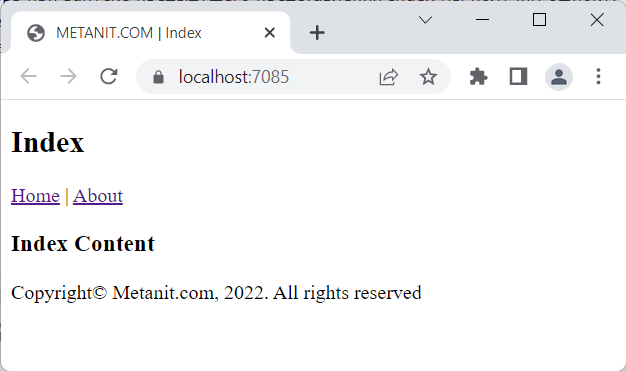
### Секции

Кроме метода RenderBody(), который вставляет освновное содержимое страниц Razor, файлы layout также могут использовать специальный метод RenderSection() для вставки секций. Мастер-страница может иметь несколько секций, куда страницы Razor могут поместить свое содержимое. Например, добавим к мастер-странице **\_Layout.cshtml** секцию с именем "footer":

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <!DOCTYPE html>  <html>  <head>      <meta name="viewport" content="width=device-width" />      <title>METANIT.COM | @ViewBag.Title</title>  </head>  <body>      <h2>@ViewBag.Title</h2>      <div><a href="/">Home</a> | <a href="About">About</a></div>      <div>          @RenderBody()      </div>      <footer>@RenderSection("Footer")</footer>  </body>  </html> |

Теперь при запуске предыдущей страницы Index мы получим ошибку, так как секция Footer не определена. По умолчанию страница должна передавать содержание для каждой секции мастер-страницы. Поэтому добавим вниз страницы **Index** секцию footer. Это мы можем сделать с помощью выражения @section:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | @page    @{      ViewBag.Title = "Index";  }  </h3>Index Content</h3>    @section Footer {     Copyright© Metanit.com, @DateTime.Now.Year. All rights reserved  } |



Но при таком подходе, если у нас есть куча страниц, и мы вдруг захотели определить новую секцию на мастер-странице, нам придется изменить все имеющиеся страницы Razor, что не очень удобно. В этом случае мы можем воспользоваться одним из вариантов гибкой настройки секций.

Первый вариант заключается в использовании перегруженной версии метода **RenderSection**, которая позволяет указать, что данную секцию не обязательно определять на странице Razor. Чтобы отметить секцию Footer в качестве необязательной, надо передать в метод в качестве второго параметра значение false:

|  |  |
| --- | --- |
| 1 | <footer>@RenderSection("Footer", false)</footer> |

Второй вариант позволяет задать содержание секции по умолчанию, если данная секция не определенана странице Razor:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <footer>      @if (IsSectionDefined("Footer"))      {          @RenderSection("Footer")      }      else      {          <span>Содержание элемента footer по умолчанию.</span>      }  </footer> |

## Файл \_ViewImports.cshtml

Файл **\_ViewImports.cshtml** позволяет по умолчанию подключить на страницы Razor некоторый функционал. Сначала рассмотрим ситуацию, с которой мы можем столкнуться. Пусть у нас в проекте есть некоторый класс **Person:**

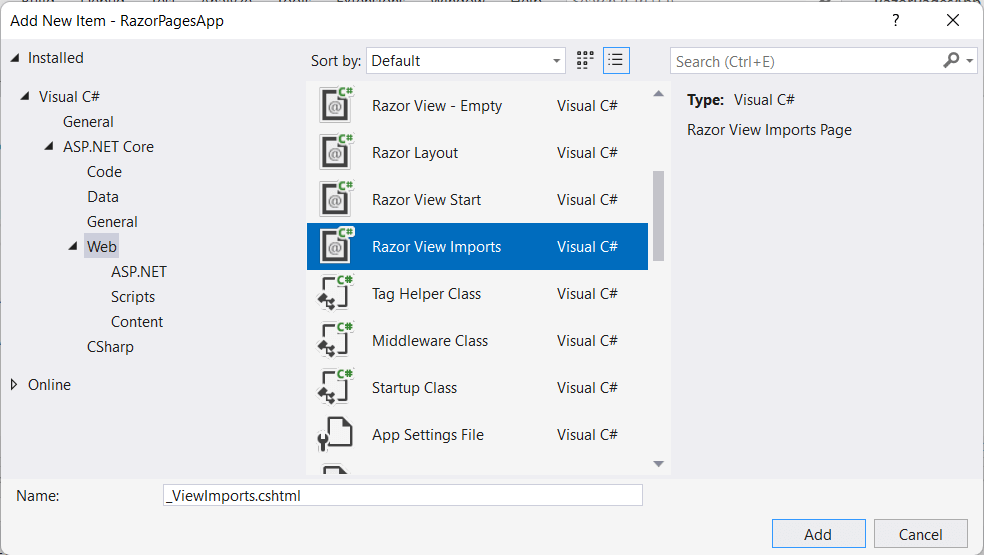
|  |  |
| --- | --- |
| 1  2  3  4 | namespace RazorPagesApp  {      public record class Person(string Name, int Age);  } |

Допустим, мы хотим использовать тип Person на странице **Index.cshtml**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | @page    @\* Подключаем пространство имен класса Person \*@  @using RazorPagesApp  @{      Person tom = new Person("Tom", 37);  }  <h2>Person Data</h2>  <p>Name: <b>@tom.Name</b></p>  <p>Age: <b>@tom.Age</b></p> |

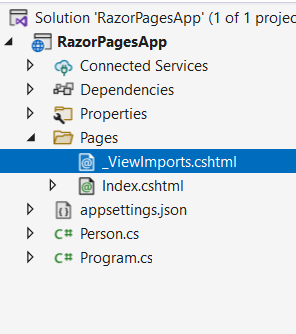
Чтобы использовать тип Person на странице Razor, мы вынуждены импортировать с помощью директивы **using** пространство имен, где этот тип определен. В данном случае ничего сложного нет, однако если у нас куча страниц, где мы хотим использовать этот же тип Person, то мы будем вынуждены определить то же самое выражение импорта на всех страницах. Это может создавать некоторые неудобства. Во-первых, мы повторяем один и тот же код. Во-вторых, если пространство имен изменится, то мы вынуждены будем менять все страницы. В-третьих, возможно, мы захотим подключить еще какие-то пространства имен, что увеличит работу, если будут какие-то изменения. Файл **\_ViewImports.cshtml** решает эту проблему

Итак, добавим в проект в папку **Pages** файл **\_ViewImports.cshtml**. Для его добавления в Visual Studio можно применять шаблон **Razor View Imports**:



Далее добавим в этот файл подключение пространства имен класса Person:

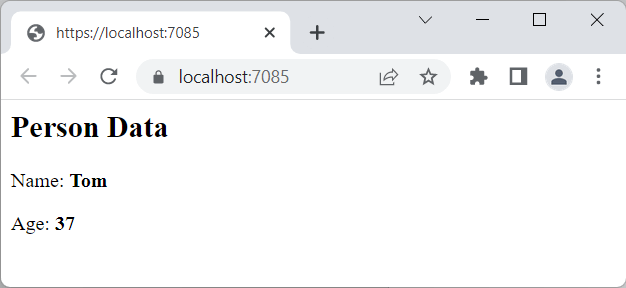
|  |  |
| --- | --- |
| 1  2 | @\* Подключаем пространство имен класса Person \*@  @using RazorPagesApp |



В этом случае функциональность пространства имен RazorPagesApps будет автоматически подключаться на все страницы Razor.

И тогда мы можем убрать из страницы **Index.cshtml** подключение пространства имен:

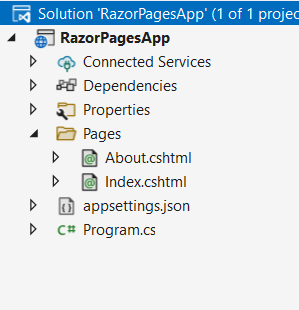
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | @page    @{      Person tom = new Person("Tom", 37);  }  <h2>Person Data</h2>  <p>Name: <b>@tom.Name</b></p>  <p>Age: <b>@tom.Age</b></p> |



## Введение в tag-хелперы

Для определения пользовательского интерфейса можно применять стандартные элементы html, а также конструкции Razor. Но кроме того для упрощения создания интерфейса ASP.NET Core предоставляет специальный инструмент, который называется **tag-хелперы**. Tag-хелперы представляют функциональность, предназначенную для генерации HTML-разметки. Tag-хелперы выглядят как обычные html-элементы или атрибуты, однако при работе приложения они обрабатываются движком Razor на стороне сервера и в конечном счете преобразуются в стандартные html-элементы.

Использовать tag-хелперы довольно просто. Допустим, у нас в проекте в папке **Pages** есть две страницы: **Index.cshtml** и **About**



На странице **Index.cshtml** определим следующий код:

|  |  |
| --- | --- |
| 1  2  3  4  5 | @page    @addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers    <a asp-page="About">About</a> |

Сначала на странице идет директива **addTagHelper**

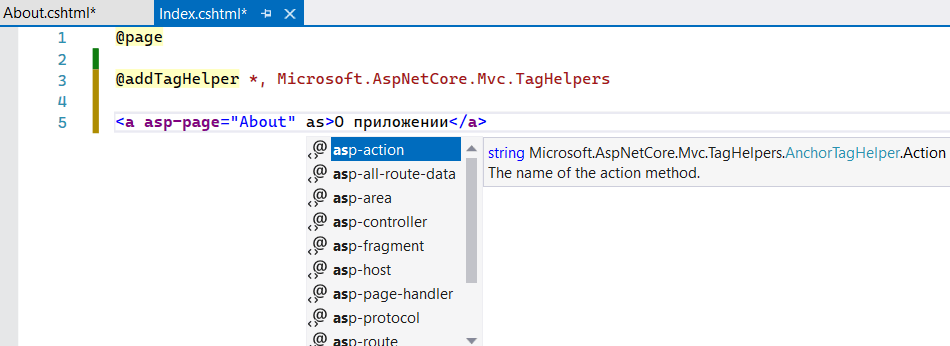
|  |  |
| --- | --- |
| 1 | @addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers |

Первый параметр директивы указывает на tag-хелперы, которые будут доступны на странице Razor, а второй параметр определяет библиотеку хелперов. В данном случае директива использует синтаксис подстановок - знак звездочки ("\*") означает, что подключаются все хелперы из библиотеки **Microsoft.AspNetCore.Mvc.TagHelpers**.

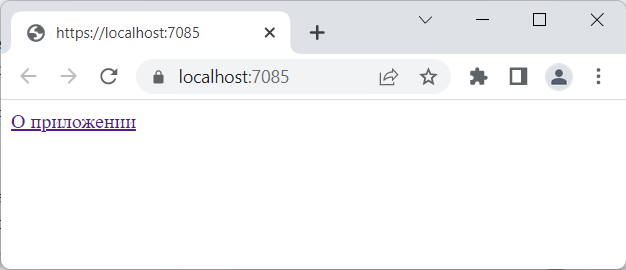
Далее идет собственно tag-хелпер:

|  |  |
| --- | --- |
| 1 | <a asp-page="About">О приложении</a> |

Внешне данный хелпер напоминает обычную ссылку - стандартный элемент html, однако это не элемент html. И если мы воспользуемся всплывающей подсказкой, то увидим, что кроме обычных для элемента <a /> он имеет ряд других:



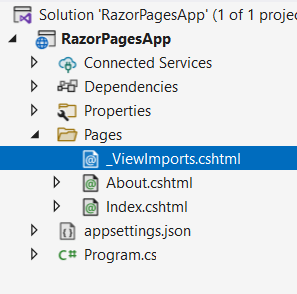
Данный хелпер создает ссылку, которая указывает на страницу About.cshtml. В итоге при запуске проекта вместа данного tag-хелпера будет сформирована гиперссылка, по нажатию на которую уйдет запрос на страницу About.cshtml:



### \_ViewImports.cshtml и @addTagHelper

Выше на странице Index.cshtml были подключены tag-хелперы. Но что, если нам надо подключить tag-хелперы на множество страниц Razor? Вместо того, чтобы прописывать директиву **@addTagHelper** на каждой отдельной странице, мы можем подключить все хелперы разом. Для этого, как и для подключения различных пространств имен, применяется файл **\_ViewImports.cshtml**.

Итак, добавим в проект в папку **Pages** новый файл **\_ViewImports.cshtml**:



В файле **\_ViewImports.cshtml** определим подключение tag-хелперов:

|  |  |
| --- | --- |
| 1 | @addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers |

После этого из кода страницы **Index.cshtml** можно удалить подключение tag-хелперов и оставить только создание ссылки:

|  |  |
| --- | --- |
| 1  2  3 | @page    <a asp-page="Contacts">О приложении</a> |

### Удаление tag-хелперов

Еще одна директива removeTagHelper удаляет ранее добавленные tag-хелперы. Ее применение аналогично:

|  |  |
| --- | --- |
| 1 | @removeTagHelper "\*, Microsoft.AspNetCore.Mvc.TagHelpers" |

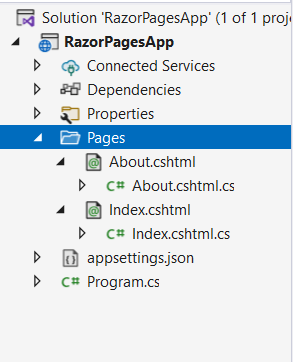
Данная директива может быть полезной, если мы, например, захотим ограничить применение хелперов в какой-то одной странице или группе страниц Razor. Эту директиву также можно определять в файле **\_ViewImports.cshtml**.

## Создание ссылок

Для создания ссылок на странице Razor можно естественно применять стандартный html-элемент <a >. Однако для упрощения генерации ссылок ASP.NET Core также предоставляет такой tag-хелпер, как **AnchorTagHelper**. Он может принимать ряд специальных атрибутов:

* **asp-controller**: указывает на контроллер, которому предназначен запрос
* **asp-action**: указывает на действие контроллера
* **asp-area**: указывает на действие область, в которой расположен контроллер или страница RazorPage (если они находятся в отдельной области)
* **asp-page**: указывает на RazorPage, которая будет обрабатывать запрос
* **asp-page-handler**: указывает на обработчик страницы RazorPage, которая будет применяться для обработки запроса
* **asp-host**: указывает на домен сайта
* **asp-protocol**: определяет протокол (http или https)
* **asp-route**: указывает на название маршрута
* **asp-all-route-data**: устанавливает набор значений для параметров
* **asp-route-[название параметра]**: определяет значение для определенного параметра
* **asp-fragment**: определяет ту часть хэш-ссылки, которая идет после символа решетки #. Например, "paragraph2" в ссылке "http://mysite.com/#paragraph2"

Рассмотрим некоторые из этих атрибутов. Допустим, в проекте есть две страницы - **Index.cshtml** и **About.cshtml**



### asp-page

Определим с помощью хелпера AnchorTagHelper на странице **Index.cshtml** ссылку на страницу **About.cshtml**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | @page  @addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers    @model RazorPagesApp.Pages.IndexModel    <h2>Index Page</h2>    <a asp-page="About">О сайте</a> |

В данном случае используется не элемент html <a />, а именно хелпер AnchorTagHelper. Его атрибут **asp-page** указывает на название страницы

### asp-page-handler

Атрибут **asp-page-handler** позволяет установить определенный обработчик страницы, на который будет идти запрос. Допустим, в файле **About.cshtml.cs** будет определен обработчик страницы:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | using Microsoft.AspNetCore.Mvc.RazorPages;    namespace RazorPagesApp.Pages  {      public class AboutModel : PageModel      {          public string Header { get; private set; } = "";          public void OnGet()          {              Header = "О компании";          }          public void OnGetSite()          {              Header = "О сайте";          }      }  } |

В дополнение к методу OnGet здесь определен обработчик страницы - метод OnGetSite(), то есть название обработчика - "Site". В каждом из методов изменяется значение свйоства Header.

На самой странице **About.cshtml** просто выведем значение свойства Header:

|  |  |
| --- | --- |
| 1  2  3  4 | @page  @model RazorPagesApp.Pages.AboutModel    <h2>@Model.Header</h2> |

А на странице **Index.cshtml** определим пару ссылок:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | @page  @addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers    @model RazorPagesApp.Pages.IndexModel    <h2>Index Page</h2>    <a asp-page="About">О компании</a> |  <a asp-page="About" asp-page-handler="Site">О сайте</a> |

Первая ссылка просто ведет на страницу About, а вторая - непосредственно на обработчик Site.

### asp-host и asp-protocol

AnchorTagHelper по умолчанию создает локальную ссылку, если же нам надо создать ссылку на другой домен, то мы можем применить атрибут **asp-host**:

|  |  |
| --- | --- |
| 1 | <a asp-page="About" asp-host="localhost.com" asp-protocol="https">О сайте</a> |

Кроме того, мы можем изменить стандартный протокол на https, использовав атрибут **asp-protocol**. Данный элемент в итоге создает следующую ссылку: *https://localhost.com/About*

### asp-route- и asp-all-route-data

А что если страница Razor должна получать какие-нибудь параметры, которые надо указать в ссылке. Например, пусть страница **About.cshtml** получает параметр id:

|  |  |
| --- | --- |
| 1  2  3  4 | @page "{id?}"  @model RazorPagesApp.Pages.AboutModel    <h2>About Person @Request.RouteValues["id"]</h2> |

В этом случае для создания ссылки можем использовать атрибут **asp-route-**:

|  |  |
| --- | --- |
| 1 | <a asp-page="About" asp-route-id="5">About Person 5</a> |

Если страница Razor принимает несколько параметров маршрута, например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @page "{id}/{name}/{age}"  @model RazorPagesApp.Pages.AboutModel    <p>Id: <b>@Request.RouteValues["id"]</b></p>  <p>Name: <b>@Request.RouteValues["name"]</b></p>  <p>Age: <b>@Request.RouteValues["age"]</b></p> |

то мы можем указать несколько атрибутов asp-route-:

|  |  |
| --- | --- |
| 1 | <a asp-page="About" asp-route-id="5"  asp-route-age="18" asp-route-name="Tom">Person 5</a> |

В данном случае будет формироваться ссылка

|  |  |
| --- | --- |
| 1 | https://localhost:7085/About/5/Tom/18 |

Чтобы не устанавливать все параметры по отдельности, можно применить атрибут **asp-all-route-data**:

|  |  |
| --- | --- |
| 1 | <a asp-page="About" asp-all-route-data='new Dictionary<string,string> { { "id", "5" }, {"name", "tom" }, { "age", "18" } }' >Person 5</a> |

asp-all-route-data в качестве значения принимает словарь с параметрами и их значениями. В результате будет генерироваться ссылка, аналогичная предыдущей.

### Url.Page

Кроме создания ссылок в коде Razor, ASP.NET Core позволяет создавать ссылки в коде C#. Для этого применяется метод **Url.Page()**, которому передаются необходимые параметры. Он имеет ряд версий:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | string? Url.Page (string? pageName, string? pageHandler, object? values, string? protocol, string? host, string? fragment);  string? Url.Page (string? pageName, string? pageHandler, object? values, string? protocol, string? host);  string? Url.Page (string? pageName, string? pageHandler, object? values, string? protocol);  string? Url.Page (string? pageName, string? pageHandler, object? values);  string? Url.Page (string? pageName, object? values);  string? Url.Page (string? pageName, string? pageHandler);  string? Url.Page (string? pageName); |

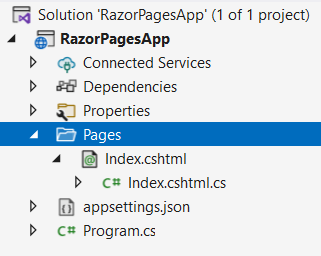
По сути этот метод все те же параметры, которые передаются в tag-хелпер AnchorTagHelper. Например, возьмем выше определенную страницу **About.cshtml**, которая принимает три параметра, и создадим ссылку на эту страницу в коде **Index.cshtml.cs**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | using Microsoft.AspNetCore.Mvc;  using Microsoft.AspNetCore.Mvc.RazorPages;    namespace RazorPagesApp.Pages  {      public class IndexModel : PageModel      {          public IActionResult OnGetAbout()          {              string? url = Url.Page("About", new { id = 1, name = "Tom", age = 37 });              return Redirect(url ?? "About");          }      }  } |

В данном случае в модели IndexModel определен обработчик страницы OnGetAbout, в котором будет создаваться ссылка "https://localhost:7085/About/1/Tom/37". И затем на эту ссылку будет идти переадресация. Стоит учитывать, что метод Url.Page может возвращать значение null. И в данном случае, если метод не смог сформировать ссылку, идет просто редирект на страницу About.cshtml.

## Работа с формами. Tag-хелперы форм

Для создания форм можно использовать стандартные элементы html. Однако для упрощения работы с формами ASP.NET Core также предоставляет группу tag-хелперов, с помощью которых можно определять формы html и их элементы. Рассмотрим применение подобных хелперов. Допустим, у нас есть страница Razor **Index.cshtml** и код связанной модели IndexModel в файле **Index.cshtml.cs**:



### FormTagHelper

Пусть в файле **Index.cshtml.cs** определен следующий код:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | using Microsoft.AspNetCore.Mvc;  using Microsoft.AspNetCore.Mvc.RazorPages;  namespace RazorPagesApp.Pages  {      public class IndexModel : PageModel      {          [BindProperty]          public Product Product { get; set; } = new("", 0, "");          public string Message { get; private set; } = "Добавление товара";            public void OnPost()          {              Message = $"Добавлен новый товар: {Product.Name} ({Product.Company})";          }      }        public record class Product(string Name, int Price, string Company);  } |

Здесь ожидается, что в post-запросе модель IndexModel будет получать данные в виде объекта Product. Для автоматической привязки данных запроса к объекту Person к свойству применяется атрибут BindProperty.

После получения объекта в запросе POST переустанавливается значение свойства Message.

Далее в файле **Index.cshtml** с помощью tag-хелперов определим форму для отправки данных:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | @page  @addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers  @model RazorPagesApp.Pages.IndexModel    <h2>@Model.Message</h2>  <form method="post" asp-antiforgery="true">      <p>          <label asp-for="Product.Name">Название</label><br />          <input type="text" asp-for="Product.Name" />      </p>      <p>          <label asp-for="Product.Price">Цена</label><br />          <input asp-for="Product.Price" />      </p>      <p>          <label asp-for="Product.Company">Производитель</label><br />          <input type="text" asp-for="Product.Company" />      </p>      <input type="submit" value="Отправить" />  </form> |

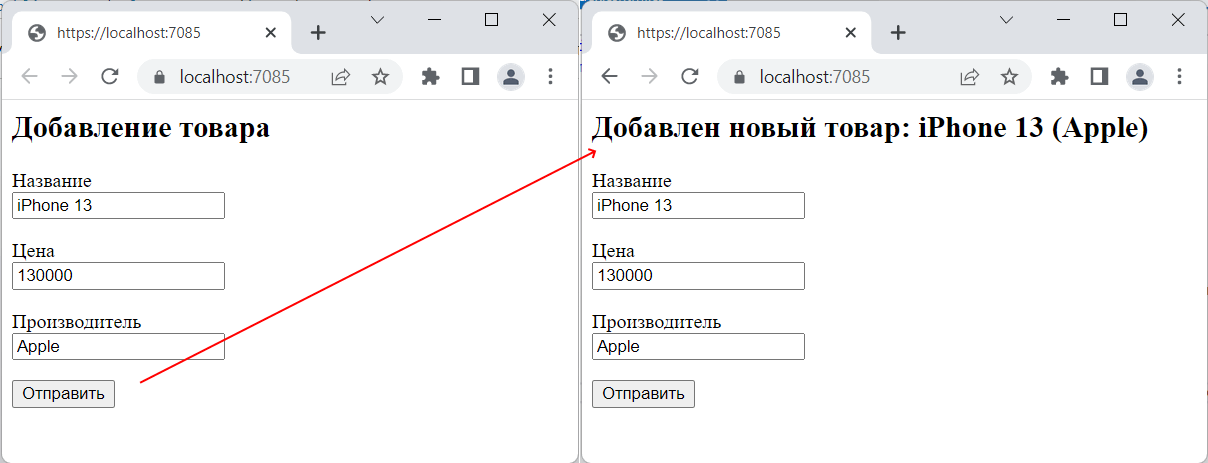
Тег-хелперы, используемые для создания форм, аналогичны соответствующим элементам html за тем исключением, что они добавляют дополнительную функциональность. Так, для создания формы используется класс **FormTagHelper**, представленный тегом form. Кроме стандартного атрибута method тег form также применяет атрибут **asp-antiforgery** указывает, будет ли для формы генерироваться antiforgery-токен. Если он имеет значение true, то токен будет генерироваться. Благодаря этому для модели IndexModel не нужно применять атрибут **[IgnoreAntiforgeryToken]**, поскольку по умолчанию для страниц Razor требуется наличие подобного токена

|  |  |
| --- | --- |
| 1  2 | //[IgnoreAntiforgeryToken] - атрибут не нужен  public class IndexModel : PageModel |

Все поля ввода, которые использовались выше на форме, имеют один общий атрибут **asp-for**, который указывает, для какого свойства модели создается элемент. Например:

|  |  |
| --- | --- |
| 1 | <input type="text" asp-for="Product.Name" /> |

Здесь атрибут **asp-for** указывает, что данный элемент будет применяться для ввода значения для свойства Product.Name. Таким образом, мы можем привязать поля ввода к определенным свойствам модели.



Класс **FormTagHelper** может принимать следующие атрибуты:

* **asp-controller**: указывает на контроллер, которому предназначен запрос
* **asp-action**: указывает на действие контроллера
* **asp-area**: указывает на название области, в которой будет вызываться контроллер для обработки формы
* **asp-antiforgery**: если имеет значение true, то для этой формы будет генерироваться antiforgery token
* **asp-route**: указывает на название маршрута
* **asp-all-route-data**: устанавливает набор значений для параметров
* **asp-route-[название параметра]**: определяет значение для определенного параметра
* **asp-page**: указывает на страницу RazorPage, которая будет обрабатывать запрос
* **asp-page-handler**: указывает на обработчик страницы RazorPage, который применяется для обработки запроса
* **asp-fragment**: указывает фрагмент, который добавляется к запрашиваемому адресу после символа #.

Например, если необходимо сделать запрос на другую страницу, то можно использовать атрибут **asp-page**:

|  |  |
| --- | --- |
| 1 | <form method="post" asp-page="Create"> |

В данном случае запрос идет на возможную страницу Create.cshtml.

С помощью атрибута **asp-page-handler** можно указать обработчик страницы, который будет обрабатывать запрос:

|  |  |
| --- | --- |
| 1 | <form method="post" asp-page="Create" asp-page-handler="Person"> |

В данном случае запрос идет на возможную страницу Create.cshtml и ее обработчику (по сути запрос будет обрабатываться методом OnPostPerson() при наличие такового).

Рассмотрим вкратце ряд основных хелперов, которые могут применяться для создания элементов форм.

### LabelTagHelper

**LabelTagHelper** использует тег label для создания метки:

|  |  |
| --- | --- |
| 1 | <label asp-for="Name"></label> |

### InputTagHelper

**InputTagHelper** создает поле ввода:

|  |  |
| --- | --- |
| 1 | <input asp-for="Name" /> |

### TextAreaTagHelper

**TextAreaTagHelper** используется для создания многострочного текстового поля textarea. Данный хелпер применяет только атрибут asp-for:

|  |  |
| --- | --- |
| 1 | <textarea asp-for="Name"></textarea> |

### SelectTagHelper

**SelectTagHelper** создает элемент списка <select>. Он имеет специальный атрибут **asp-items**, который указывает на объект IEnumerable<SelectListItem> - набор элементов, используемых для создания списка.

Рассмотрим одну из ситуаций, где может потребоваться подобный хелпер. Пусть в файле **Index.cshtml.cs** определен следующий код:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | using Microsoft.AspNetCore.Mvc;  using Microsoft.AspNetCore.Mvc.RazorPages;  using Microsoft.AspNetCore.Mvc.Rendering;    namespace RazorPagesApp.Pages  {      public class IndexModel : PageModel      {          // условные данные          static List<Company> companies { get; } = new()          {              new Company(1, "Apple"),              new Company(2, "Samsung"),              new Company(3, "Google")          };          public SelectList Companies { get; } = new SelectList(companies, "Id", "Name");          [BindProperty]          public Product Product { get; set; } = new("", 1000, 0);          public string Message { get; private set; } = "Добавление товара";            public void OnPost()          {              // получаем производителя товара              Company? company = companies.FirstOrDefault(c => c.Id == Product.CompanyId);              Message = $"Добавлен новый товар: {Product.Name} ({company?.Name})";          }      }      public record class Product(string Name, int Price, int CompanyId);      public record class Company(int Id, string Name);  } |

Здесь определены две сущности - Product и Company. Причем у класса Product определено свойство CompanyId, через которое объект Product будет связан с определенным объектом Company.

В классе модели IndexModel определены условные данные - статический список объектов Company, из которого будет выбираться производитель для добавления товара.

Но для отображения списка объектов в tag-хелпере SelectTagHelper нам нужен объект IEnumerable<SelectListItem>. В качестве такового обычно выступает объект класса **SelectList**.

Класс SelectList имеет ряд конструкторов:

|  |  |
| --- | --- |
| 1  2  3  4 | public SelectList (IEnumerable items);  public SelectList (IEnumerable items, object selectedValue);  public SelectList (IEnumerable items, string dataValueField, string dataTextField);  public SelectList (IEnumerable items, string dataValueField, string dataTextField, object selectedValue); |

Параметры конструктора:

* items задает набор элементов для списка
* selectedValue устанавливливает выделенное по умолчанию значение
* dataValueField устанавливливает свойство объекта, значение которого будет отправляться на сервер
* dataTextField устанавливливает свойство объекта, значение которого будет отображаться для данного элемента в списке

То есть в данном случае свойство

|  |  |
| --- | --- |
| 1 | public SelectList Companies { get; } = new SelectList(companies, "Id", "Name"); |

Что будет создаваться список из набора companies. В качестве значений, которые будет выбирать пользователь, будут использоваться значения свойства Id. А в качестве текста, который будет отображаться для элемента, будет применяться текст свойства Name.

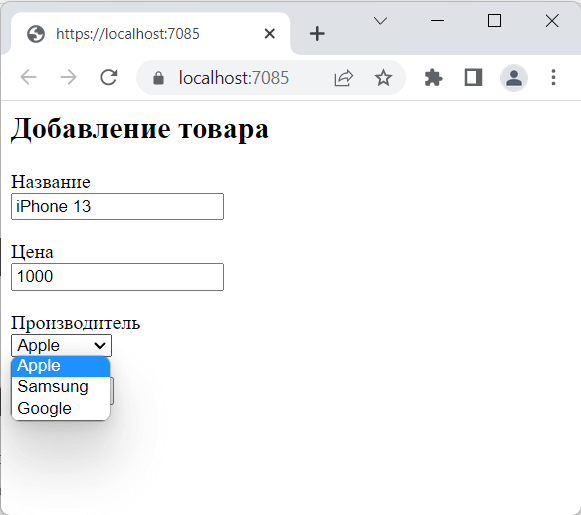
На странице **Index.cshtml** определим форму для ввода данных:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | @page  @addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers  @model RazorPagesApp.Pages.IndexModel    <h2>@Model.Message</h2>  <form method="post" asp-antiforgery="true">      <p>          <label asp-for="Product.Name">Название</label><br />          <input type="text" asp-for="Product.Name" />      </p>      <p>          <label asp-for="Product.Price">Цена</label><br />          <input asp-for="Product.Price" />      </p>      <p>          <label asp-for="Product.CompanyId">Производитель</label><br />          <select asp-for="Product.CompanyId" asp-items="Model.Companies"></select>      </p>      <input type="submit" value="Отправить" />  </form> |

Для создания выпадающего списка здесь применяется следующий tag-хелпер

|  |  |
| --- | --- |
| 1 | <select asp-for="Product.CompanyId" asp-items="Model.Companies"></select> |

Атрибуту **asp-items** передается объект SelectList, из которого будет создаваться список



При необходимости мы можем указать элемент, который будет отображаться по умолчанию:

|  |  |
| --- | --- |
| 1  2  3 | <select asp-for="Product.CompanyId" asp-items="Model.Companies">      <option selected="selected" disabled="disabled">Выберите компанию</option>  </select> |

### Обработка форм и обработчики страниц razor

Razor Pages также позволяет отправлять форму различным обработчикам одной и той же страницы. Для этого у тега формы можно указать нужный обработчик с помощью параметра **asp-page-handler**. Более того можно указать у отдельных кнопок, на какой обработчик они отправятся данные формы. Например, изменим код IndexModel:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | using Microsoft.AspNetCore.Mvc.RazorPages;    namespace RazorPagesApp.Pages  {      public class IndexModel : PageModel      {          // условные данные          IEnumerable<Person> people = new List<Person>         {              new ("Tom", 37),              new ("Sam", 28),              new ("Bob", 41),              new ("Tim", 25)          };          public IEnumerable<Person> DisplayedPeople { get; private set; } = new List<Person>();          public void OnGet()          {              DisplayedPeople = people;          }          public void OnPostGreaterThan(int age)          {              DisplayedPeople = people.Where(p => p.Age > age);          }          public void OnPostLessThan(int age)          {              DisplayedPeople = people.Where(p => p.Age < age);          }      }      public record class Person(string Name, int Age);  } |

Обработчик OnPostGreaterThan находит пользователей, у которых возраст больше определенного, а обработчик OnPostLessThan, наоборот, находит пользователей, у которых возраст меньше, чем определенный.

Изменим код страницы **Index.cshtml**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | @page "{handler?}"    @model PersonModel  @using RazorPagesApp.Models    <h2>Список пользователей</h2>  <form method="POST">      <input type="number" name="age" />      <input type="submit" asp-page-handler="GreaterThan" value="Старше" />      <input type="submit" asp-page-handler="LessThan" value="Младше" />  </form>  <table class="table">      <tr><th>Name</th><th>Age</th></tr>      @foreach(Person person in Model.DisplayedPeople)      {          <tr>              <td>@person.Name</td>              <td>@person.Age</td>          </tr>      }  </table> |

На странице определена форма, но в зависимости от того, на какую кнопку мы нажмем, введенное значение будет улетать тому или иному обработчику.

