**Введение в Razor Pages**

**Что такое Razor Pages. Первый проект с .NET CLI**

Начиная с версии 2.0 в ASP.NET Core была добавлена такая платформа, как **Razor Pages**. Razor Pages предоставляют технологию, альтернативную фреймворку MVC(Model-View-Controller). В тоже время Razor Pages в плане организации считается проще чем MVC.

Приложение в Razor Pages организовано в виде страниц Razor. Каждая страница представляет пользовательский интерфейс и связанную с ним логику, причем определение пользовательского интерфейса от связанной с ним логики, что облегчает разработку приложения и его тестирование. Для определения пользовательского интерфейса применяется движок Razor, а для описания связанной со страницей логики - C#.

Для начала работы с Razor Pages создадим новый проект ASP.NET Core с .NET CLI. Для создания приложений на основе Razor Pages в .NET CLI имеется готовый шаблон - **webapp, razor**.

Итак, определим каталог для проекта Razor Pages. Например, в моем случае это будет каталог **C:\dotnet\razorpages\helloapp**. Перейдем к этому каталогу в терминале с помощью команды **cd**:

cd C:\dotnet\razorpages\helloapp

Затем для создания проекта в этой папке выполним команду **dotnet new webapp**:

C:\dotnet\razorpages\helloapp>dotnet new webapp

Шаблон "Веб-приложение ASP.NET Core" успешно создан.

Этот шаблон содержит технологии сторонних производителей, кроме Майкрософт. Дополнительные сведения см. в разделе https://aka.ms/aspnetcore/8.0-third-party-notices.

Идет обработка действий после создания...

Восстановление C:\dotnet\razorpages\helloapp\helloapp.csproj:

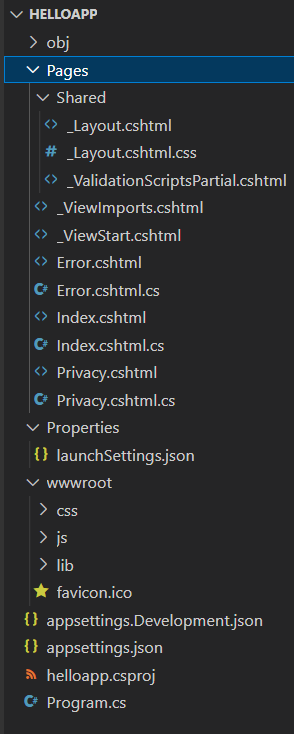
Определение проектов для восстановления...

Восстановлен C:\dotnet\razorpages\helloapp\helloapp.csproj (за 431 ms).

Восстановление выполнено.

C:\dotnet\razorpages\helloapp>

После выполнения команды будет создан следующий проект:



Частично, этот тип проекта будет повторять общую структуру проектов ASP.NET Core. Главным же отличием данного проекта от других типов проектов ASP.NET Core будет наличие папки **Pages**. Эта папка содержит все страницы Razor, которые есть в проекте по умолчанию. Вкратце организация проекта:

* **Properties**: содержит настройки проекта, в частности, файл **launchSettings.json**, который определяет настройки запуска проекта
* **wwwroot**: эта папка предназначен для хранения статических файлов. По умолчанию здесь уже есть ряд скриптов javascript и файлов css, в частности, файлы фреймворка bootstrap и библиотек валидации.
* **Pages**: содержит все страницы Razor. По умолчанию здесь имеются следующие файлы:
  + **\_Layout.cshtml**: мастер-страница, в которую вставляются страницы Razor
  + **\_ViewStart.cshtml**: задает мастер-страницу
  + **\_ViewImports.cshtml**: определяет директивы Razor, которые добавляются на каждую страницу Razor
  + **\_ValidationScriptsPartial.cshtml**: частичное представление, которое подключает js-скрипты валидации на стороне клиента
  + **Index.cshtml**, **Error.cshtml** и **Privacy.cshtml**: собственно страницы Razor, которые определяют визуальную часть страницы и логику обработки запроса.
* **appsettings.Development.json**: хранит конфигурацию приложения для стадии разработки
* **appsettings.json**: хранит конфигурацию приложения
* **helloapp.csproj**: главный файл проекта, который определяет его конфигурацию
* **Program.cs**: файл, который определяет класс Program, с которого начинается работа приложения. То есть это входная точка в приложение.

Каждая страница Razor представляет файл с расширением **.cshtml** и содержит смесь кода html и конструкций C#. С каждой страницей Razor связан файл отделенного кода логики на C#. Например, с файлом **Index.cshtml**, который определяет визуальную часть с помощью синтаксиса Razor (HTML + C#), связан файл **Index.cshtml.cs**, который определяет логику страницы или ее поведение с помощью кода C#.

Аналогично обстоит дело с другими страницами - **Error.cshtml** и **Privacy.cshtml**, с которыми связаны соответственно файлы кода **Error.cshtml.cs** и **Privacy.cshtml.cs**

### Подключение Razor Pages

Как работают Razor Pages? Чтобы разобраться в этом, откроем файл **Program.cs**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | var builder = WebApplication.CreateBuilder(args);    // подключение сервисов razor pages  builder.Services.AddRazorPages();    var app = builder.Build();    if (!app.Environment.IsDevelopment())  {      app.UseExceptionHandler("/Error");      app.UseHsts();  }    app.UseHttpsRedirection();  app.UseStaticFiles();    app.UseRouting();    app.UseAuthorization();    app.MapRazorPages();    app.Run(); |

Здесь надо отметить два момента. Во-первых, для использования Razor Pages необходимо подключить соответствующие сервисы:

|  |  |
| --- | --- |
| 1 | builder.Services.AddRazorPages(); |

Для подключения сервисов Razor Pages применяется метод **AddRazorPages**.

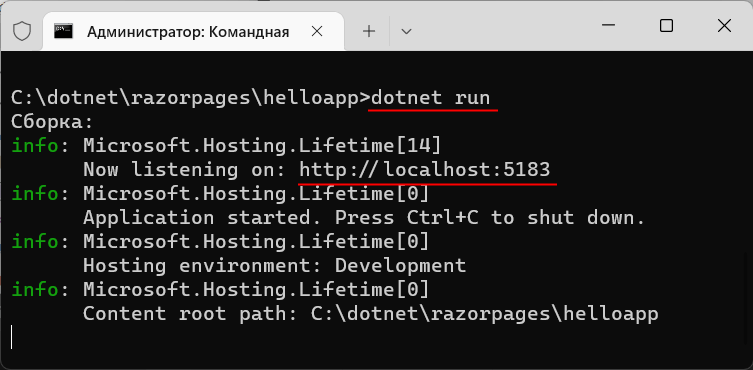
Другим важным моментом является маршрутизация или как Razor Pages сопоставляются с запросами. В частности, в файле Program.cs также можно найти следующий вызов:

|  |  |
| --- | --- |
| 1 | app.MapRazorPages(); |

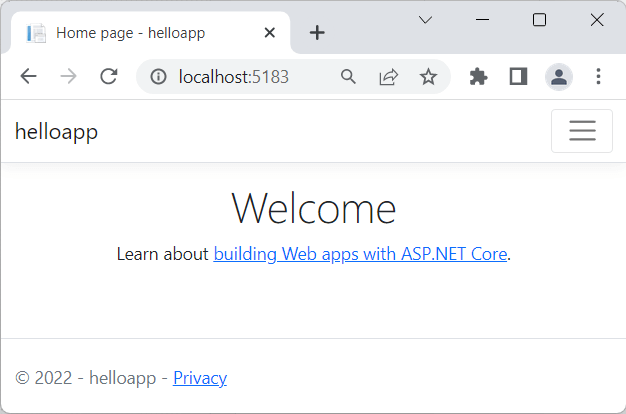
Данный вызов добавляет поддержку системы маршрутизации, которая позволяет сопоставить строку запроса URL с определенной страницей Razor на основании ее расположения в проекте в папке **Pages**. Например, запросу */Index* будет соответствовать страница */Pages/Index.cshtml*. Затем на основе страницы Razor будет генерироваться html-страница, отправляемая в ответ клиенту.

Запустим проект с помощью команды

dotnet run

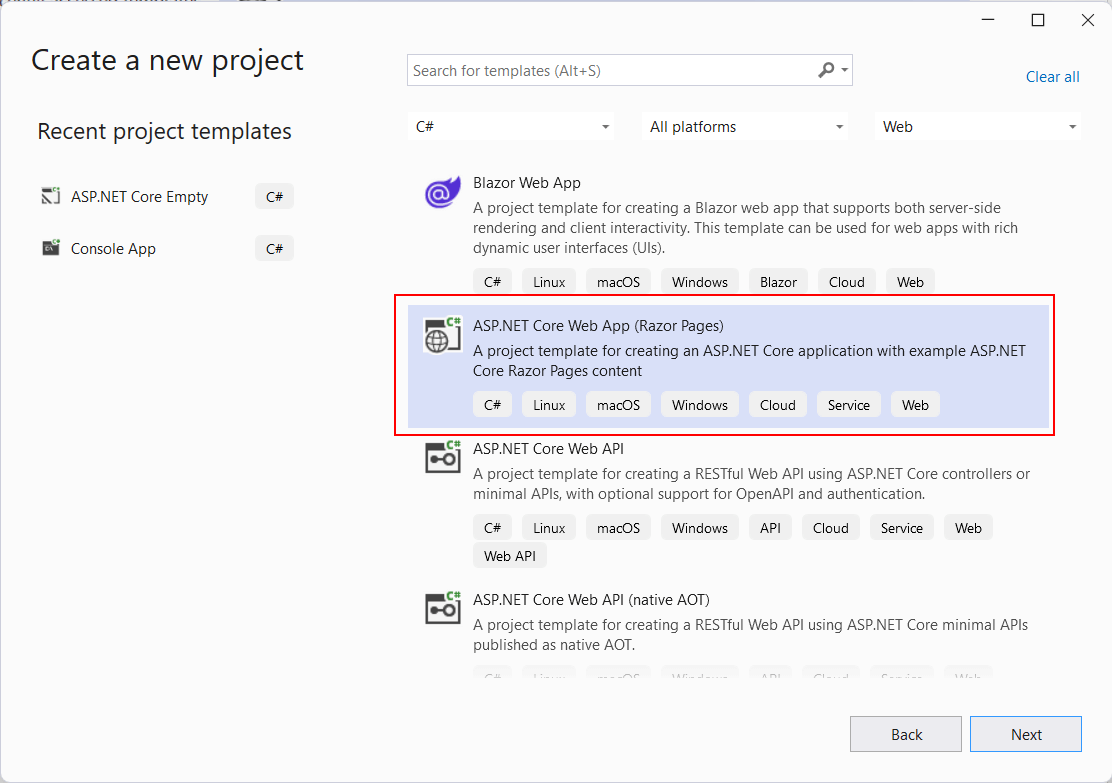


Консоль отобразит адрес зарущенного приложения (в моем случае это http://localhost:5183). Обратимся по этому адресу, и сработает запрос к странице Razor по умолчанию - странице **Index.cshtml**, на основе которой и будет сгенерирована html-страница, которую мы увидим в своем веб-браузере:

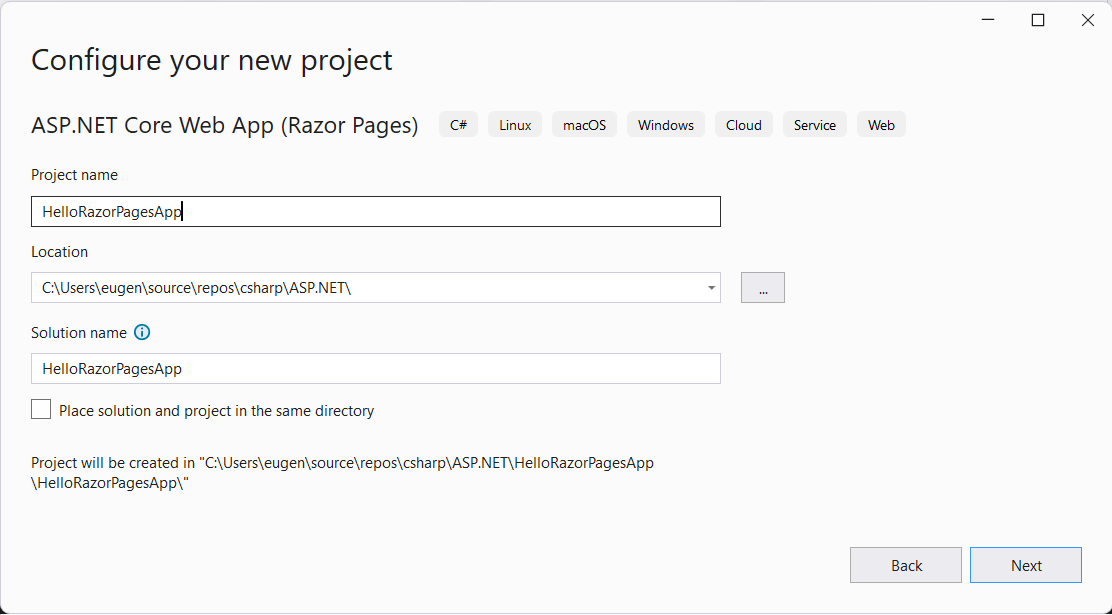


## Первый проект в Visual Studio

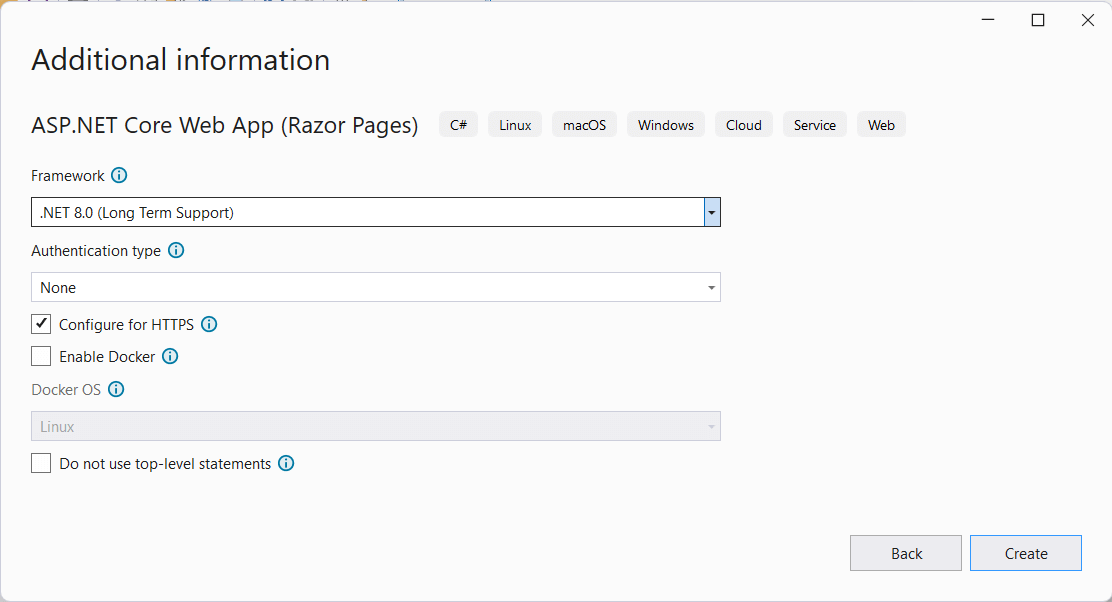
Для создания приложений на основе Razor Pages в Visual Studio имеется готовый шаблон - **ASP.NET Core Web App (Razor Pages)**. Выберем его в качестве шаблона проекта:



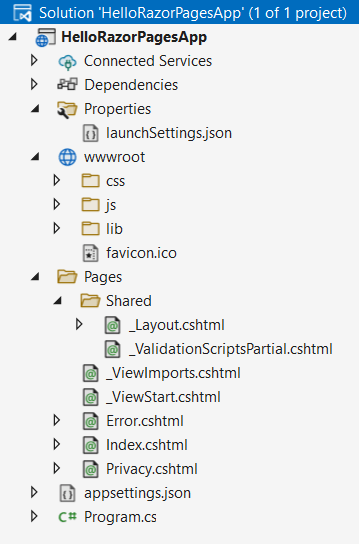
На следующем шаге необходимо будет указать имя проекта и его расположение. Допустим, проект будет называться **HelloRazorPagesApp**:



Далее можно будет указать версию .NET и еще ряд настроек. Оставим все настройки по умолчанию и нажмем на кнопку Create для создания проекта:



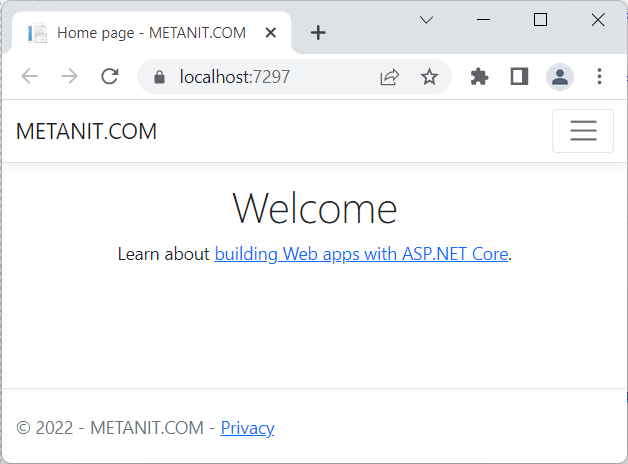
После создания проект будет иметь следующую структуру:



Эта та же структура, которая создается с помощью .NET CLI:

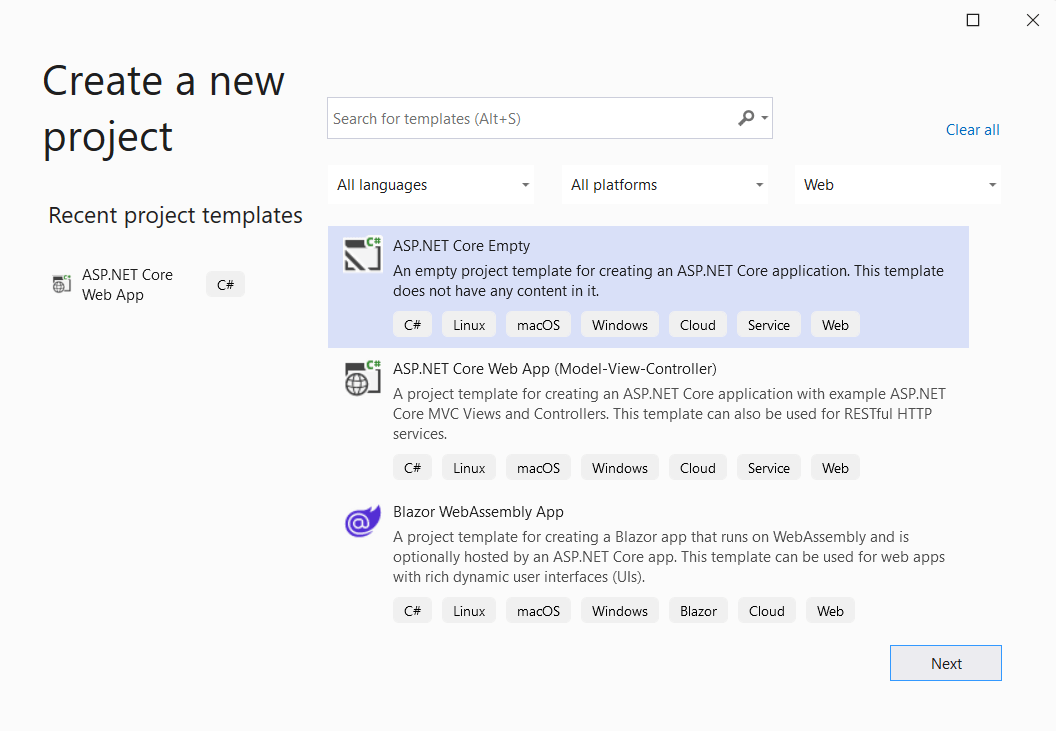
* **Dependencies**: все добавленные в проект пакеты и библиотеки
* **Properties**: содержит настройки проекта, в частности, файл **launchSettings.json**, который определяет настройки запуска проекта
* **wwwroot**: эта папка предназначен для хранения статических файлов. По умолчанию здесь уже есть ряд скриптов javascript и файлов css, в частности, файлы фреймворка bootstrap и библиотек валидации.
* **Pages**: содержит все страницы Razor. По умолчанию здесь имеются следующие файлы:
  + **\_Layout.cshtml**: мастер-страница, в которую вставляются страницы Razor
  + **\_ViewStart.cshtml**: задает мастер-страницу
  + **\_ViewImports.cshtml**: определяет директивы Razor, которые добавляются на каждую страницу Razor
  + **\_ValidationScriptsPartial.cshtml**: частичное представление, которое подключает js-скрипты валидации на стороне клиента
  + **Index.cshtml**, **Error.cshtml** и **Privacy.cshtml**: собственно страницы Razor, которые определяют визуальную часть страницы и логику обработки запроса.
* **appsettings.json**: хранит конфигурацию приложения
* **Program.cs**: файл, который определяет класс Program, с которого начинается работа приложения. То есть это входная точка в приложение.

И если мы запустим проект на выполнение, то сработает запрос к странице Razor по умолчанию - странице **Index.cshtml**, на основе которой и будет сгенерирована html-страница, которую мы увидим в своем веб-браузере:



## Добавление RazorPages в пустой проект

Visual Studio предоставляет готовый шаблон для создания проекта Razor Pages. Однако иногда может возникнуть необходимость добавить функциональность Razor Pages в другие типы проектов или вообще создать проект с нуля. Поэтому рассмотрим, как добавлять функциональность и страницы RazorPages в пустой проект ASP NET Core. Если мы работаем в Visual Studio, то создадим по типу **ASP NET Core Empty**



Если мы работаем через .NET CLI, то создадим проект по типу **webapp**, выполнив в терминале команду

dotnet new webapp

После создания проекта первым делом подключим в него функциональность Razor Pages. Для этого откроем файл **Program.cs**. По умолчанию он имеет примерно следующее содержимое:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | var builder = WebApplication.CreateBuilder(args);  var app = builder.Build();    app.MapGet("/", () => "Hello World!");    app.Run(); |

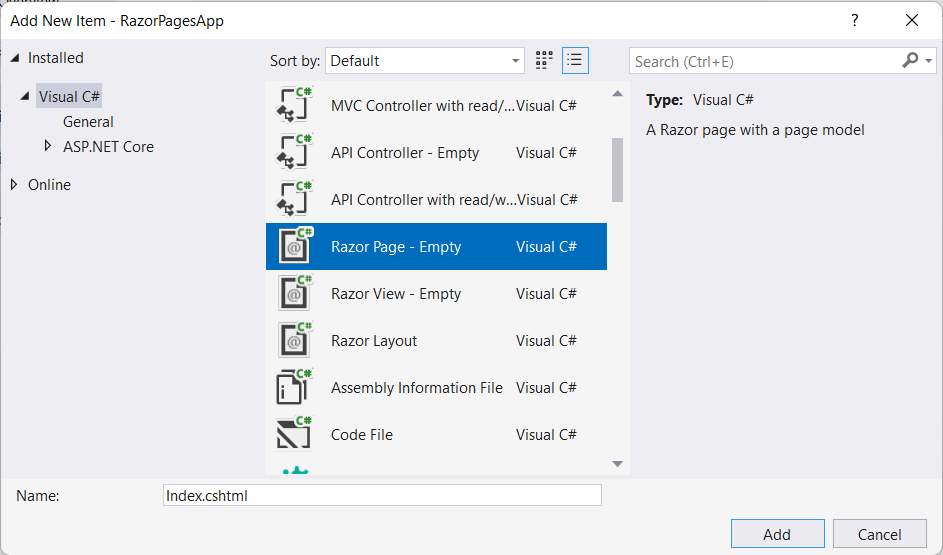
Изменим его следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | var builder = WebApplication.CreateBuilder(args);    // добавляем в приложение сервисы Razor Pages  builder.Services.AddRazorPages();    var app = builder.Build();    // добавляем поддержку маршрутизации для Razor Pages  app.MapRazorPages();    app.Run(); |

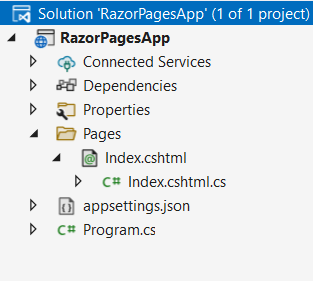
### Добавление страницы Razor Page

Теперь добавим страницу Razor. И в начале определим в проекте папку **Pages** для хранения Razor Pages. Далее в эту папку добавим новую страницу Razor.

Если мы работаем в Visual Studio, то для добавления нажмем на данную папку правой кнопкой мыши и выберем в контекстном меню пункт **Add -> New Item**. Далее среди шаблонов выберем шаблон **Razor Page - Empty** и назовем новый файл **Index.cshtml**:



После создания этой страницы в проект в папку Pages будут добавлены два файла - сама страница **Index.cshtml** и связаный с ней файл кода **Index.cshtml.cs**.



Если мы НЕ работаем в Visual Studio, то надо в папке Pages создать текстовый файл Index.cshtml.

При добавлении в Visual Studio файл **Index.cshtml** по умолчанию выглядит следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4 | @page  @model RazorPagesApp.Pages.IndexModel  @{  } |

Фактически это пустая страница. Директива **@page** указывает, что это страница Razor. А директива **@model** - в данном случае это класс привязанного к странице кода IndexModel. Согласно условностям класс модели называется по имени страницы плюс суффикс "Model".

А файл **Index.cshtml.cs** при добавлении в Visual Studio содержит простейшее определение модели **IndexModel**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | using Microsoft.AspNetCore.Mvc;  using Microsoft.AspNetCore.Mvc.RazorPages;    namespace RazorPagesApp.Pages  {      public class IndexModel : PageModel      {          public void OnGet()          {          }      }  } |

В классе модели определен метод **OnGet()**. Этот метод обрабатывает GET-запросы. В данном случае он пуст. В реальности в данном случае он нам не понадобится.

В дальнейшем мы подробно разберем обработку запросов в Razor Pages. А пока определим на странице какое-нибудь простейшее содержимое. Для этого изменим файл **Index.cshtml** следующим образом:

|  |  |
| --- | --- |
| 1  2  3 | @page    <h2>Hello METANIT.COM!</h2> |

Запустим проект. И нам отобразится определенный на странице заголовок:

