

# Современные платформы прикладной разработки

---

RAZOR PAGES

# Общая информация

---

RAZOR PAGES

# Razor pages

**Razor Pages** - это новый аспект ASP.NET Core, который делает кодирование ориентированных на страницы сценариев более простым и продуктивным.

# Razor pages

```
services.AddRazorPages();
```

---

```
app.MapControllerRoute(  
    name: "default",  
    pattern: "{controller=Home}/{action=Index}/{id?}"  
);  
app.MapRazorPages();
```

# Razor pages

Каждая страница Razor представляет собой пару файлов:

- файл `.cshtml`, содержащий разметку HTML с кодом C # с использованием синтаксиса Razor.
- файл `.cshtml.cs` – модель страницы - , содержащий код C #, который обрабатывает события страницы.

# Razor pages

---

Файлы страниц располагаются по пути:

/Pages

/Areas/[название области]/Pages

# Разметка страницы и обработка запросов

---

RAZOR PAGES

# Директива @page

@page превращает файл в действие MVC - это означает, что он обрабатывает запросы напрямую, без прохождения через контроллер.

@page должна быть первой директивой Razor на странице.

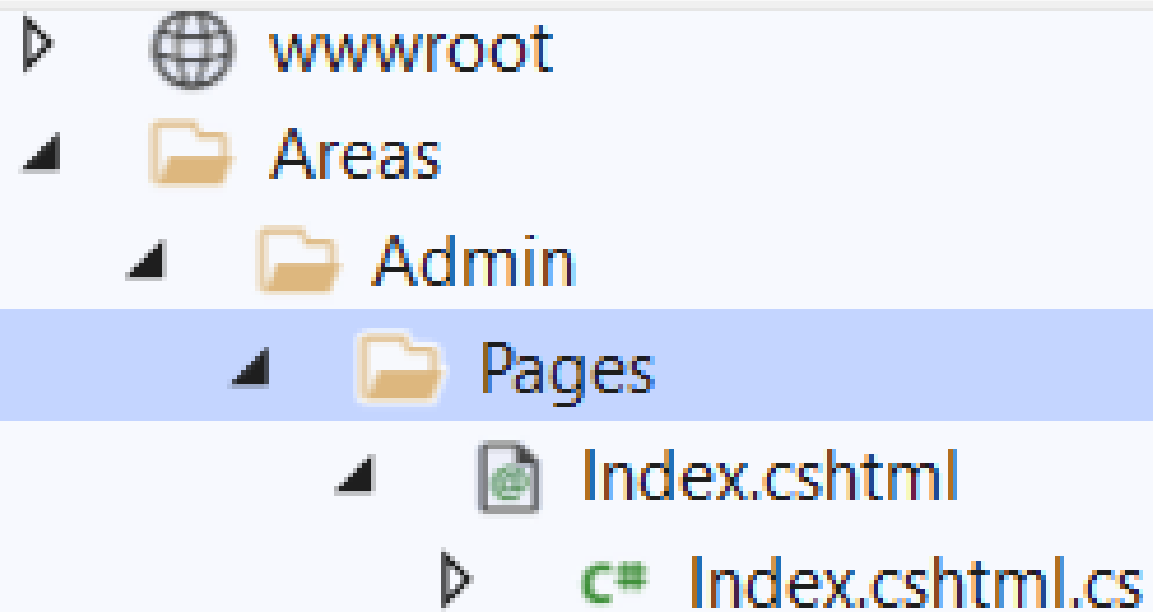
@page влияет на поведение других конструкций Razor.



# Директива @model

---

Директива @model определяет класс модели страницы (описывается в файле .cshtml.cs )



# Index.cshtml

```
@page  
@model XXX.Areas.Admin.Pages.IndexModel  
@{  
}
```

# Index.cshtml.cs

```
namespace XXX.Areas.Admin.Pages
{
    public class IndexModel : PageModel
    {
        public void OnGet()
        {
        }
    }
}
```

# Чтение данных модели

```
public class IndexModel : PageModel
{
    public decimal Price { get; set; }
    public string Name { get; set; }
    public void OnGet()
    {
        Name = "parsley";
        Price=1.2M;
    }
}
```

<p>  
Name: @Model.Name; price: @Model.Price  
</p>



# Обработка запросов

Для обработки запросов в модели страницы используются методы, имя которых совпадает с типом запроса, например:

```
public void OnGet()  
public async Task OnGetAsync()  
public async Task<IActionResult> OnGetAsync(int? id)  
  
public async Task<IActionResult> OnPostAsync(int? id)  
public IActionResult OnPost(int? id)
```

# Несколько обработчиков на одной странице

```
public async Task<IActionResult> OnPostEditAsync()  
{ . . . }  
public async Task<IActionResult> OnPostCreateAsync()  
{ . . . }
```



```
<a asp-page="/Books/Edit"  
    asp-page-handler="Create">Сохранить</a>
```

<https://localhost:44329/Books/Edit?handler=Create>

# Привязка данных

Для привязки данных к свойствам модели страницы используется атрибут ***[BindProperty]***

Привязка осуществляется для всех методов, кроме Get.

Если нужна привязка по Get, то дополнительно указывается:

***[BindProperty(SupportsGet = true)]***

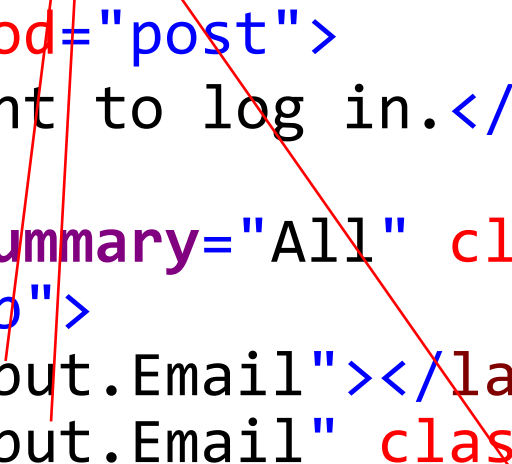


# Привязка данных

[BindProperty]

```
public InputModel Input { get; set; }
```

```
<form id="account" method="post">
  <h4>Use a local account to log in.</h4>
  <hr />
  <div asp-validation-summary="All" class="text-danger"></div>
  <div class="form-group">
    <label asp-for="Input.Email"></label>
    <input asp-for="Input.Email" class="form-control" />
    <span
      asp-validation-for="Input.Email" class="text-
danger"></span>
```



The diagram illustrates the data binding process. A red rounded rectangle highlights the `Input` property in the `InputModel` class. Three red arrows originate from this rectangle: one points to the `asp-for="Input.Email"` attribute in the `<label>` tag, another points to the `asp-for="Input.Email"` attribute in the `<input>` tag, and the third points to the `asp-validation-for="Input.Email"` attribute in the `<span>` tag. This visualizes how the `Input` property is mapped to the form elements.

# ViewData и TempData

[ViewData]

```
public string Title { get; } = "About";
```

[TempData]

```
public string Message { get; set; }
```

---

См. страницу Razor

`Identity/Manage/Register.cshtml`

# Маршрутизация страниц

---

RAZOR PAGES

# Маршрутизация страниц

Первая строка страницы может содержать директиву `@page` с указанием шаблона маршрута к данной странице, например:

```
@page "{id:int}"
```

# Маршрутизация страниц

Ограничение маршрутизации "{id: int}" указывает странице принимать запросы к странице, которые содержат данные о маршруте int.

Если запрос к странице не содержит данных о маршруте, которые можно преобразовать в int, среда выполнения возвращает ошибку HTTP 404 (не найдена).

# Страница Pages/Customers/Edit

---

Строка **./Index** - это относительное имя страницы.

Используется для создания URL-адресов на страницу, например, Pages/Customers/Index.cshtml.

Url.Page ("./Index ", ...)

<a asp-page="./Index"> Назад к списку</a>

RedirectToPage ("./Index ")

# Страница Pages/Customers/Edit

Строка **/Index** - это абсолютное имя страницы.

Используется для создания URL-адресов на страницу, например, Pages/Index.cshtml.

Url.Page ("/Index ", ...)

<a asp-page="/Index">На главную страницу</a>

RedirectToPage ("/Index ")



# Pages vs MVC

---

ANDREW LOCK: *ASP.NET CORE IN ACTION, SECOND EDITION*

Если вы новичок в ASP.NET Core, используйте Razor Pages для приложений, генерируемых на стороне сервера, и используйте платформу MVC для создания веб-API.

Вместо PageModel и обработчика страниц MVC использует концепцию контроллеров и методов действий. Они почти полностью аналогичны своим аналогам Razor Pages.

С другой стороны, контроллеры MVC используют явные модели представлений для передачи данных в представление Razor, а не предоставляют данные как свойства сами по себе (как Razor Pages делают с моделями страниц).

В MVC один контроллер может иметь несколько методов действия. Каждое действие обрабатывает отдельный запрос и генерирует другой ответ. Группировка нескольких действий в контроллере несколько произвольна, но обычно она используется для группировки действий, связанных с определенной сущностью.

К сожалению, вы часто можете обнаружить, что ваши контроллеры становятся очень большими и раздутыми, со многими зависимостями.

Еще одна ловушка контроллеров MVC — это то, как они обычно организованы в вашем проекте. Большинству методов действий в контроллере потребуется связанное представление Razor и модель представления для передачи данных в представление. Подход MVC традиционно группирует классы по типу (контроллер, представление, модель представления), в то время как подход Razor Page группирует по функциям — все, что связано с конкретной страницей, размещается в одном месте.

---

Razor Pages учитывает тот факт, что вы создаете страничное приложение, и оптимизирует рабочий процесс, объединяя все, что связано с одной страницей.

Если вы не хотите отображать представления — Razor Pages лучше всего подходит для приложений на основе страниц, где вы визуализируете представление для пользователя. Если вы создаете веб-API, вместо этого вам следует использовать контроллеры MVC.



При преобразовании существующего приложения MVC в ASP.NET Core. Если у вас уже есть приложение ASP.NET, использующее MVC, вероятно, не стоит преобразовывать существующие контроллеры MVC в Razor Pages. Имеет больше смысла сохранить существующий код и, возможно, рассмотреть возможность разработки нового приложения с помощью Razor Pages.

Когда вы делаете много частичных обновлений страниц — можно использовать JavaScript в приложении, чтобы избежать полной навигации по страницам, обновляя только часть страницы. Этот подход, находящийся на полпути между полностью отрисовкой на стороне сервера и приложением на стороне клиента, может быть проще реализовать с помощью контроллеров MVC, чем Razor Pages.