Wydział
Informatyki
POLITECHNIKA BIAŁOSTOCKA

Programowanie aplikacji WWW w technologii .NET

Tworzenie widoków w Razor

Ireneusz Mrozek

Materiały przygotowane w ramach projektu „PB 5.0 – dostosowanie oferty dydaktycznej Politechniki Białostockiej do potrzeb nowoczesnej gospodarki oraz zielonej i cyfrowej transformacji” (nr umowy FERS.01.05-IP.08-0327/23-00) w ramach programu Fundusze Europejskie dla Rozwoju Społecznego 2021-2027 współfinansowanego ze środków Europejskiego Funduszu Społecznego Plus.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską
NCBR Narodowe Centrum Badań i Rozwoju

Razor

Razor to silnik szablonów używany do tworzenia dynamicznych widoków w aplikacjach ASP.NET. Pozwala na łączenie kodu HTML z C# w jednym pliku, umożliwiając dynamiczne generowanie stron. Wspiera tworzenie widoków w ASP.NET MVC oraz Razor Pages.

Różnice między HTML a widokami Razor:

- Razor integruje C# z HTML, pozwalając na dynamiczne generowanie treści.
- Tradycyjne pliki HTML są statyczne, podczas gdy Razor pozwala na generowanie dynamicznych treści na serwerze przed wysłaniem do przeglądarki.

Podstawowe zalety Razor:

- **Prosta składnia**: Razor używa uproszczonej, łatwej w użyciu składni do wbudowania C# w HTML.
- **Wydajność**: Renderowanie widoków Razor jest szybkie i efektywne dzięki integracji z serwerowym przetwarzaniem C#.
- **Integracja z C#**: Umożliwia korzystanie z pełnej mocy języka C# i funkcji .NET bez potrzeby rozdzielania logiki na różne warstwy.

Podstawowe elementy składni

Wstawianie kodu C#:

```
<h1>Hello, @Model.UserName!</h1>
```

Blok kodu:

```
@{
    var currentDate = DateTime.Now;
}
<p>Today is: @currentDate.ToString("D")</p>
```

Wyrażenia:

```
<p>The sum of 5 and 10 is: @(5 + 10)</p>
```

Komentarze:

```
@* This is a Razor comment *@
```

Pętle

@foreach

```
<ul>
@foreach (var item in Model.Items)
{
    <li>@item</li>
}
</ul>
```

@for

```
<ul>
@for (int i = 0; i < Model.Items.Length; i++)
{
    <li>@Model.Items[i]</li>
}
</ul>
```

Instrukcja warunkowa @if … else

Instrukcja warunkowa @if

```
@if (Model.IsActive)
{
    <p>User is active.</p>
}
else
{
    <p>User is inactive.</p>
}
```

Atrybut <text>....</text>

Tag <text> w Razor jest pomocny do wstawiania tekstu (bez HTML'u). <text> wstawia surowy tekst bez tworzenia dodatkowych elementów HTML i jest używane głównie w kontekście kodu Razor, gdzie potrzebujesz wstawić tekst, ale nie chcesz, aby był interpretowany jako HTML.

```
@for (int i = 0; i < 5; i++){
        <text>Element numer @i</text><br />
}
```

Komentarze

```
@{
    /* comment 1 */
    // comment 2
    @* comment 3 *@
    var numbers = Enumerable.Range(1, 10);
}

@foreach(var number in numbers){
    @number
}
```

Tag Helpers

Tag Helpers to funkcje, które rozszerzają standardowe tagi HTML w Razor o dodatkową funkcjonalność. Dzięki nim można bezpośrednio w HTML kontrolować renderowanie elementów, korzystając z logiki C#. Działają poprzez atrybuty z prefiksem `asp-`, co pozwala na łatwiejsze tworzenie dynamicznych elementów bez konieczności ręcznego pisania kodu C# w widokach. Integrują C# z HTML, tworząc czytelny, zwięzły i łatwiejszy w utrzymaniu kod.

Popularne Tag Helpers

asp-for:

```
<form method="post">
    <div>
    <label asp-for="firstName"></label>
    <input asp-for="firstName" />
    </div>
    <br/>
    <div>
    <label asp-for="dateOfBirth" ></label>
    <input asp-for="dateOfBirth" />
    </div>
</form>
```

```
public string firstName { get; set; }
public string lastName { get; set; }

[DataType(DataType.Date)]
[Display(Name ="Data urodzenia")]
public DateTime dateOfBirth { get; set; }

public void OnGet()
{
    dateOfBirth = DateTime.Now;
}
```

Popularne Tag Helpers

asp-page:

```
<a asp-page="Index">Go to Index</a>
```

asp-action:

```
<a asp-action="Index">Go to Index</a>
```

asp-route:

```
<a asp-route-id="1" asp-route-name="Product">Product Details</a>
```

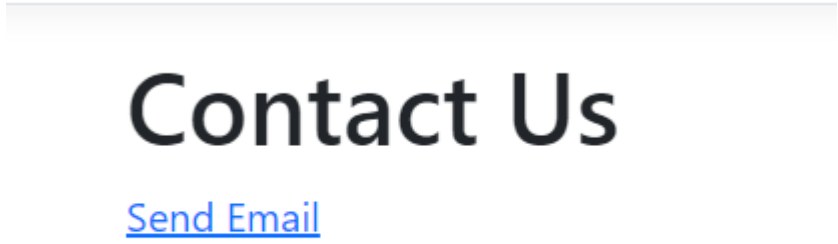
Tworzenie własnego Tag Helpers

```
using Microsoft.AspNetCore.Razor.TagHelpers;
namespace razor
{
    [HtmlTargetElement("email")]
    public class EmailTagHelper : TagHelper
    {
        public string Address { get; set; }
        public string LinkText { get; set; }

        public override void Process(TagHelperContext context, TagHelperOutput output)
        {

            output.TagName = "a";
            output.Attributes.SetAttribute("href", $"mailto:{Address}");
            output.Content.SetContent(LinkText);

        }
    }
}
```

Tworzenie własnego Tag Helpers

```
@page
@model razor.Pages.createtaghelpersModel
@addTagHelper *, razor


<h1>Contact Us</h1>


<email address="example@example.com" link-text="Send Email"></email>
```

```
...
<h1>Contact Us</h1>


<a href="mailto:example@example.com">Send Email</a>
```



Przykłady stosowania Tag Helpers

Formularze

```
<form asp-action="Submit">
        <input asp-for="Name" />
        <button type="submit">Submit</button>
</form>
```

Linki

```
<a asp-controller="Home" asp-action="About">About</a>
```

Walidacja

```
<span asp-validation-for="Name" class="text-danger"></span>
```

Strona wzorcowa (Layout)

Jedną z podstawowych cech serwisu internetowego jest jego ujednolicony wygląd.

Wybrane podejścia do ujednoliconego wyglądu:

Kopiowanie wspólnych części do kolejnych stron serwisu. Problemem staje się modyfikacja części wspólnej.

Dołączanie do każdej strony wspólnego pliku lub kontrolki użytkownika. Pojawia się problem z projektowaniem wyglądu strony. Środowiska nie zawsze zapewniają możliwość podglądu „składanej” strony.

Wykorzystanie stron wzorcowych.

Layout to bazowy szablon, który definiuje wspólną strukturę dla wielu stron w aplikacji. Umożliwia spójną prezentację, zawierając wspólne elementy, takie jak nagłówki, stopki czy menu. Wykorzystanie stron wzorcowych zmniejsza powtarzalność kodu. I ułatwia zarządzanie stylem i układem strony.

Atrybut Layout

```
@page
@model WebApplication4.IndexModel
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index1</title>
</head>
<body>
</body>
</html>
```

Wartość `null` przypisana do atrybutu Layout oznacza, iż strona jest „samowystarczalna” i zawiera w sobie wszystkie niezbędne elementy.

Strona wzorcowa - struktura

Strona wzorcowa (Layout Page) jest szablonem dla wszystkich stron odnoszących się do niej.

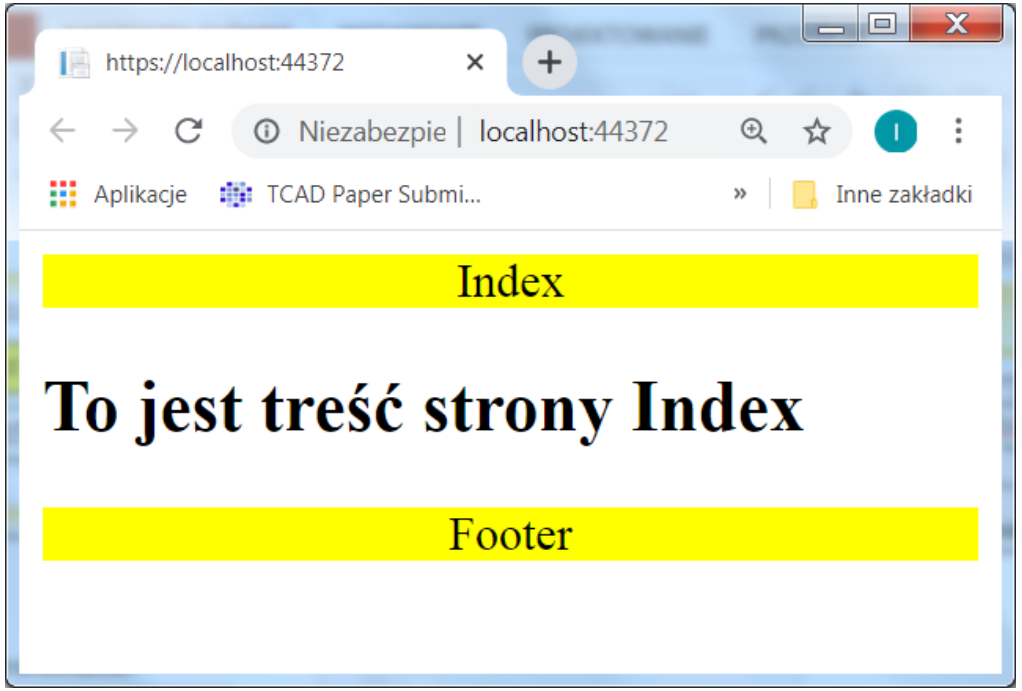
```
<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
</head>
<body>
    <div>
        @RenderBody()
    </div>
</body>
</html>
```

Metoda `@RenderBody()` wstawia w miejsce jej występowania zawartość widoku, który jest powiązany z daną stroną wzorcową.

Powiązanie widoku ze stroną wzorcową

Index.cshtml

```
@page
@model IndexModel
@{
    Layout = "_Layout";
    ViewBag.Title = "Index";
}
<h2>To jest treść strony Index</h2>
```



_Layout.cshtml

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport"
content="width=device-width" />
</head>
<body>
    <div style="text-align:center; background-
color:yellow">
        @ViewBag.title
    </div>
    <div>
        @RenderBody()
    </div>
    <div style="text-align:center; background-
color:yellow">
        Footer
    </div>
</body>
</html>
```

Sekcje

Funkcja `@RenderBody()` pozwala określić w stronie wzorcowej miejsce umieszczenia zawartości strony podstawowej.

Strona podstawowa może zostać podzielona na określone sekcje. Umiejscowienie poszczególnych sekcji w stronie wzorcowej można określić z wykorzystaniem funkcji `@RenderSection()`.

Sekcje

Fragment pliku
`_Layout.cshtml`

```
…
@RenderSection(„Sekcja1", required: false)
…
```

Parametr `Sekcja1` – nazwa sekcji

Parametr `required false` - wskazuje, że sekcja nie jest obowiązkowa i nie musi występować w pliku z treścią.

Fragment pliku z treścią np. `index.cshtml`

```
…
@section Sekcja1 {
    // content here
}…
```

W przypadku, gdy sekcja nie jest zdefiniowana można użyć treści domyślnej:

```
…
@if(IsSectionDefined(„Sekcja1"))
{
    @RenderSection(„Sekcja1")
}
else
{
    // treść domyślna
}
…
```

_ViewStart

Framework .NET Core przetwarzając stronę zawsze najpierw poszukuje pliku o nazwie `_ViewStart.cshtml`.

Plik `_ViewStart.cshtml` zawiera kod, który jest wykonywany na początku przetwarzania każdej strony Razor.

Plik `_ViewStart.cshtml` ma wpływ na wszystkie strony Razor znajdujące się w tym samym folderze co plik `_ViewStart.cshtml` lub w dowolnym jego podfolderze.

Plik `_ViewStart` ma charakter hierarchiczny. Pliki znajdujące się w podfolderach wykonują się po plikach znajdujących się w folderach nadrzędnych.

_ViewStart

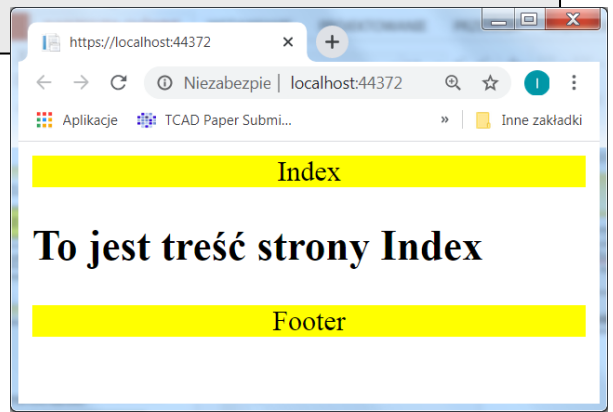
Najczęściej plik `_ViewStart` wykorzystywany jest do zdefiniowania strony wzorcowej

Index.cshtml

```
@page
@model IndexModel
@{
    ViewBag.Title = "Index";
}
<h2>To jest treść strony Index</h2>
```

_ViewStart.cshtml

```
@{
  Layout = "_Layout";
}
```



_Layout.cshtml

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width"
/>
</head>
<body>
    <div style="text-align:center; background-
color:yellow">
        @ViewBag.title
    </div>
    <div>
        @RenderBody()
    </div>
    <div style="text-align:center; background-
color:yellow">
        Footer
    </div>
</body>
</html>
```

Atrybut Layout

Wartości atrybutu `Layout` w konkretnej stronie:

- Wskazanie na konkretną stronę wzorcową (nadpisywane są wówczas ustawienia z `_ViewStart`)
- Atrybut pominięty - wartość jest ustalana na podstawie pliku `_ViewStart`
- Wartość `null` - framework zakłada, że strona jest „samowystarczalna”.

Strony wzorcowe zazwyczaj noszą nazwę `_Layout.cshtml`. Znak „_” na początku nazwy sprawia, iż nie mogą zostać wykonane bezpośrednio.

```
@{
    Layout = „/Themes//Winter/_Layout.cshtml";
}
```

Katalogi poszukiwania stron wzorcowych

1. Katalog bieżący

2. Katalogi nadrzędne, aż do folderu `/`

3. Zarejestrowane foldery, domyślnie `Pages/Shared` i `Views/Shared`.

Przykład: Odwołując się do strony umieszczonej w folderze: `Pages/Students/` plik `_Layout.cshtml` poszukiwany będzie w następujących katalogach:

1. `Pages/Student`

2. `Pages`

3. `Pages/Shared`

4. `Views/Shared.`

Widoki częściowe

Widok częściowy to plik .cshtml, który zawiera fragment HTML, CSS oraz opcjonalnie kod C#. Widoki te są zazwyczaj wykorzystywane do renderowania powtarzalnych elementów na stronie, takich jak:

- Nagłówki,
- Stopki,
- Elementy nawigacji,
- Formularze,
- Komponenty powiadomień,
- listy produktów, itp.

Zalety

- Ponowne wykorzystanie kodu
- Czystość i organizacja kodu
- Poprawa wydajności.

Widoki częściowe

_MyPartialView.cshtml

```
<div>
    <h2>@Model.Title</h2>
    <p>@Model.Description</p>
</div>
```

Page1.cshtml

```
@page
@model MyApp.Pages.IndexModel

<h1>Strona Główna</h1>

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

Page1.cshtml.cs

```
public class Page1Model : PageModel
{
    public string Title { get; set; }
    public string Description { get; set; }

    public void OnGet()
    {
        Title = "STRONA 1";
        Description = "To jest opis strony
1";
    }
}
```

Widoki częściowe – przekazanie modelu

```
public class PartialViewModel
{
    public string Title { get; set; }
    public string Description { get; set; }
}
```

```
@model
razor.Pages.PartialExample.PartialViewModel

<div>
    <h2>@Model.Title</h2>
    <h3>@Model.Description</h3>
</div>
```

```
public class Page3Model : PageModel
{
    public PartialViewModel PartialViewModel {
get; set; }
    public void OnGet()
    {
      PartialViewModel = new PartialViewModel();
      PartialViewModel.Title = "STRONA 3";
      PartialViewModel.Description = "Opis
strony 3";
    }
}
```

```
@page
@model
razor.Pages.PartialExample.Page3Model
@{
    Layout = "_Layout";
}
@Html.Partial("Partials/_MyPartialWithModel
", Model.PartialViewModel)
```

Przekazywanie danych z PageModel do widoku

```
public class IndexModel : PageModel
{
    public string Message { get; set; }

    public void OnGet()
    {
        Message = "Witaj w Razor Pages!";
    }
}
```

```
<p>@Model.Message</p>
```

Wyświetlanie list i tabel

Wyświetlenie listy

```
<ul>
    @foreach (var item in Model.Items)
    {
        <li>@item.Name</li>
    }
</ul>
```

Wyświetlenie tabeli

```
<table>
    <thead>
        <tr>
            <th>Nazwa</th>
            <th>Cena</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in
Model.Products)
        {
            <tr>
                <td>@item.Name</td>

<td>@item.Price.ToString("C")</td>
            </tr>
        }
    </tbody>
</table>
```