Curs 2

Pagini Razor și sintaxa

Structura unei aplicatii Web – cu Razor

- In directorul wwwroot regasim continut de tip client-side CSS, JavaScript, imagini, si orice alt continut non-programatic

- Directorul Pages contine pagini Razor si fisiere suport. Fiecare pagina este o pereche a urmatoarelor fisiere:
  - Un fisier *.cshtml* care contine markup Razor, HTML si cod C#.
  - Un fisier *.cshtml.cs* care contine cod C# pentru a gestiona evenimentele la nivel de pagina.

- Fisierele suport au nume care incepe cu "_". Ex fisierul "*_Layout.cshtml*" configureaza elemente UI comune tuturor. Ex. acest fisier seteaza navigation menu in partea de sus a paginii si informatia despre copyright in partea de jos a paginii.

Pagini Layout

- Majoritatea site-urilor prezintă același conținut pe fiecare pagină sau într-un număr mare de pagini (ex. antet, subsol, bara meniu de navigare, scripturile, css etc.)
- Adăugarea aceluiași continut încalcă principiul DRY (Don’t Repeat Yourself) => trebuie să modificați aspectul antetului, trebuie să editați fiecare pagină
- Pagina Layout - acționează ca un șablon pentru toate paginile care fac referire la aceasta.
- Paginile care fac referire la pagina de Layout se numesc pagini de conținut. Paginile de conținut nu sunt pagini web complete. Acestea conțin doar conținutul care variază de la o pagină la alta.

Exemplu pagină de Layout foarte simplă:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8" />
<title></title>
<link href="/css/site.css" rel="stylesheet" type="text/css" />
</head>
    <body>
    @RenderBody()
    </body>
</html>
```

Fisiere configurare aplicatii Web – cu Razor

- **appSettings.json**
- Contine date de configurare, precum string-uri de conexiune

- **Program.cs**

Punctul de start al aplicatiei.

Sintaxa Razor

- Razor este o sintaxa de tip markup pentru a incorpora cod de tip server-side in pagini web
- Sintaxa Razor consta in Razor markup, C#, si HTML
- Limbajul Razor implicit este HTML. Interpretarea HTML din markup-ul Razor este similara cu interpretarea HTML dintr-un fisier HTML.
- Pentru a trece la limbaj C# - simbolul @ - Razor evalueaza expresiile C# si le randeaza in output HTML
- Cand simbolul @ este urmat de un cuvant cheie Razor – tranzitia se face la markup specific Razor

Expresii implicite/explicite Razor

- Expresii implicite

```
<p>@DateTime.Now</p>
<p>Last week: @DateTime.Now - TimeSpan.FromDays(7)</p>
```

Se afiseaza

```
<p>Last week: 7/7/2016 4:39:52 PM - TimeSpan.FromDays(7)</p>
```

- Expresii explicite ()

```
<p>Last week this time: @(DateTime.Now - TimeSpan. FromDays(7)) </p>
```

Continutul @() este evaluat si afisat

Expresii explicite

- Expresiile explicite pot fi utilizate pentru a concatena text cu un rezultat al unei expresii

```
@
{
 var joe = new { Name="Joe", Age=33 };
 }
```

Age@joe.Age

e interpretata ca o adresa de email , se afiseaza Age@joe.Age

Age@(joe.Age) se afiseaza Age33

Functii locale in blocuri de cod

- Se pot declara functii locale cu markup pentru a fi utilizate ca metode template

```
@{
void RenderName(string name)
 { <p>Name: <strong>@name</strong></p>
}
 RenderName(“John Keller");
 RenderName("Martin Johnson");
}
<p>Name: <strong>John Keller</strong></p>
<p>Name: <strong>Martin Johnson</strong></p>
```

Tranzitie implicita

- Limbajul implicit intr-un bloc de cod este C#, dar poate face implicit tranzitia la HTML

```
@{
var inCSharp = true;
<p>Now in HTML, was in C# @inCSharp</p>
 }
```

Tranzitie explicita

- Delimitata - Pentru a defini o sectiune dintr-un bloc de cod care trebuie sa afiseze HTML

```
@for (var i = 0; i < people.Length; i++)
{ var person = people[i];
<text>Name: @person.Name</text>
 }
```

- Inline - @:

```
@for (var i = 0; i < people.Length; i++)
{
var person = people[i];
@:Name: @person.Name
}
```

Structuri de control

- Conditionale

```
@if (value % 2 == 0)
 { <p>The value was even.</p> }
else
if (value >= 1337)
{ <p>The value is large.</p> }
else
{ <p>The value is odd and small.</p> }
```

Structuri de control - ciclice

```
@{
 var people = new Person[]
{ new Person("Weston", 33),
 new Person("Johnathon", 41),
... };
}
@for (var i = 0; i < people.Length;
i++)
{ var person = people[i];
<p>Name: @person.Name</p>
<p>Age: @person.Age</p>
 }
```

```
@foreach (var person in people)
{ <p>Name: @person.Name</p>
  <p>Age: @person.Age</p>
}
```

Clasa PageModel

- Scopul: de a oferi o separare clară între stratul UI (fișierul de vizualizare .cshtml) și logica de procesare a paginii.
- Beneficii:
  - Reduce complexitatea stratului UI, făcându-l mai ușor de întreținut.
  - Permite o mai mare flexibilitate pentru echipe, deoarece un membru poate lucra la vizualizare, în timp ce altul poate lucra la logica de procesare.

- Clasa PageModel este declarată într-un fișier de clasă separat - un fișier cu extensia .cs
- Prin conventie numele clasei este denumit dupa sablonul <NumePagina>Model - O clasă PageModel pentru About.cshtml va fi numită Model și va fi generată într-un fișier numit About.cshtml.cs.

PageModel

- gestionează o solicitare pentru o anumită pagină sau acțiune pentru o aplicatie web (echivalent cu rolul Controller-ului in MVC)
- O clasă Razor PageModel este o implementare a pattern-ului Page Controller-caracterizat prin faptul că există o mapare unu-la-unu între pagini și controlerele acestora.
- Rol:
  - a accepta input de la pagină
  - de a se asigura că toate operațiunile solicitate asupra modelului (date) sunt aplicate
  - de a determina vizualizarea corectă de utilizat pentru pagina rezultată.

Directive Razor

```
@page
@model IndexModel
@{
 }
```

- @page - permite ca pagina sa poata gestiona cereri
  - trebuie sa fie prima directiva care apare in fisier

- @model – specifica modelul transmis catre pagina Razor
  -reprezentat de clasa derivata din PageModel

Clasa PageModel

Mosteneste clasa de baza PageModel

Create.cshtml - Create.cshtml.cs

```
public class CreateModel : PageModel
{
  public IActionResult OnGet()
   {
   return Page();
   }

  [BindProperty]
  public Book Book { get; set; }
}
```

- Procesarea cererilor se realizeaza prin metode handler (On<verb> cu Async atașat opțional):
  - OnPost/OnPostAsync – ruleaza cand exista cereri POST (cand utilizatorul trimite un formular)
  - OnGet/OnGetAsync – initializeaza starea paginii
- Se pot adauga metode handler pentru orice verb HTTP (PUT, DELETE etc.)
- Sufixul Async este optional dar este folosit adeseori pentru a indica faptul că metoda este destinată să ruleze asincron
- Proprietatile clasei PageModel sunt disponibile ca si proprietati ale modelului in Pagina Razor
- Create.cshtml `<input asp-for="Book.Title" class="form-control" />`

Metode Handler in Pagini Razor

- OnGet si OnGetAsync reprezinta aceasi metoda handler-> nu pot exista ambele in aceeasi pagina.

Se va arunca exceptie: InvalidOperationException: Multiple handlers matched. The following handlers matched route data and had all constraints satisfied: OnGetAsync(), OnGet()

- Parametrii nu joacă nici un rol în dezambiguizarea între handlere bazate pe aceeași metodă HTTP, în ciuda faptului că compilatorul o va permite. Prin urmare, aceeași excepție va fi ridicată chiar dacă metoda OnGet preia parametri și metoda OnGetAsync nu
- Metodele de gestionare trebuie să fie publice și pot returna *void*, *Task* dacă sunt asincrone sau *IActionResult* (sau *Task<IActionResult>*).

HTTP este stateless

```
public class IndexModel : PageModel
{
public string Message { get; set; }
public void OnGet()
{
Message = "Get used";
}
public void OnPost()
{
Message = "Post used";
}
}
```

```
<h3>@Message</h3>
<form method="post"><button class="btn btn-default">Click to post</button></form>
<p><a href="/" class="btn btn-default">Click to Get</a></p>
```

Orice valoare inițializată în handlerul OnGet nu este disponibilă în handlerul OnPost

Metode Handler cu parametrii

- Delete.cshtml.cs

```
public IActionResult OnPost(int? id)
 {
 }
```

- Delete.cshtml

```
<form method="post">
     <input type="hidden" asp-for= "Book.id" />
     <input type="submit" value="Delete" class="btn btn-danger" /> |
     <a asp-page="./Index">Back to List</a>
 </form>
```

ActionResult

```
public IActionResult OnGet (int? id)
 {
     if (id == null || _context.Book == null)
     {
         return NotFound();
     }

     var book = _context.Book.FirstOrDefault (m => m.ID == id);
     else
     {
         Book = book;
     }
     return Page();
 }
```

ActionResult - tip de returnare a metodelor handler; sunt responsabile pentru generarea de răspunsuri și coduri de stare adecvate (clasa abstractă Microsoft.AspNetCore.Mvc.ActionResult sau interfața Microsoft.AspNetCore.Mvc.IActionResult)

-returnarea conținutului unei pagini Razor (PageResult), redirecționarea către o altă resursă (de exemplu, RedirectResult) sau pur și simplu returneaza un anumit cod de stare HTTP (ex. NotFoundResult, OkResult).

Pagina Layout

- Pages/Shared/_Layout.cshtml
- Furnizeaza un Layout unitar pentru intreaga aplicatie
- Permite containerului HTML pentru layout
  - Sa fie specificat intr-un singur loc
  - Sa fie utilizat in pagini multiple

- @RenderBody ()- placeholder care permite afisare continutului specific pentru fiecare pagina

Setarea Layout-ului

- Se realizeaza in Pages/_ViewStart.cshtml

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

- Seteaza pentru toate paginile din directorul Pages, layout-ul definit in Pages/Shared/_Layout.cshtml

ViewData si Layout

```
@{
    ViewData["Title"] = "Home page";
}
```

- Clasa de baza Pagemodel contine o proprietate de tip dictionar, care este utilizata pentru a trimite date la un View
- Obiectele sunt adaugate folosind perechi de tipul cheie/valoare
- Proprietatea Title este adaugata la dictionarul ViewData

ViewData si Layout

- Proprietatea title este folosita in Pages/Shared/_Layout.cshtml

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-
width, initial-scale=1.0" />
    <title>@ViewData["Title"] - Sample App</title>
```

ViewData

- container pentru a trimite date de la PageModel la pagina de continut
- dictionar de obiecte cu chei de tip string

```
public class IndexModel :
PageModel
{
public void OnGet()
{
ViewData["MyNumber"] = 42;
ViewData["MyString"] = "Hello
World";
}
}
```

```
@page
@model IndexModel
@{
}
<h2>@ViewData["MyString"]</h2>
<p>The correct number is @ViewData["MyNumber"]
</p>
```

ViewBag

Un wrapper asupra dictionarului ViewData

Ofera o alternativa pentru accesarea continutului din ViewData utilizand proprietati in locul cheilor de tip string

```
@page
@model IndexModel
@{
ViewBag.Title = "My Home Page";
}
```

```
<title>@ViewBag.Title</title>
<title>@ViewData["Title"]</title>
```