Razvoj Web Aplikacija

Predavanje 9

Koncepti jezika Razor

- Statičke i dinamičke HTML stranice
- Sintaksa Razor jezika
- Razor pogledi (engl. Razor Views)
- Prosljeđivanje podataka u pogled: ViewData, ViewBag
- Što je TempData?
- HTML pomagači (engl. HTML Helpers) i pomagači za oznake (engl. Tag Helpers)

Statička stranica vs MVC stranica

- Statička stranica
  - Zahtjev za stranicu: GET http://.../products/product_7.html
  - Server unutar mape **Products** nalazi stranicu **product_7.html** kao datoteku na disku
  - Server vraća sadržaj datoteke klijentu
- MVC (dinamička) stranica
  - Zahtjev za stranicu: GET http://.../products/product/7
  - Server nalazi kontroler s nazivom **Products**Controller i instancira ga
  - Server unutar instanciranog kontrolera nalazi metodu s nazivom **Product** i poziva ju
  - Metoda Product koristi View rezultat da bi našla predložak (template) **Product.cshtml**, te mu eventualno šalje podatke koje onda zovemo **model**
  - **Product.cshtml** generira HTML kod pomoću posebne Razor sintakse koristeći model
  - HTML kod server vraća korisniku
- Prednost statičke stranice: jednostavnost, lako održavanje
- Prednost dinamičke stranice: dinamičnost (baza podataka, servisi, integracija s drugim aplikacijama…)

Razor

- Razor je posebna markup sintaksa koja dijelom podsjeća na HTML, a dijelom na C#
- Razor mehanizam predložaka (Razor template engine) služi za generiranje koda kao što je HTML
- Također, moguće je osim HTML-a iz Razor datoteke generirati CSS, JavaScript, SVG, … čak i običan tekst
- Nalazi se u cshtml datotekama
  - Cshtml pogled (view) == Razor predložak
- Ukratko:
  - Akcijska metoda pronalazi pogled
  - Akcijska metoda predaje model pogledu
  - Pogled pretvara model u kod
- Karakteristike pogleda:
  - Pogled ne bi trebao sadržavati poslovnu logiku
  - Pogled treba generirati HTML kod na temelju koda predloška i poslanog modela

Razor

- Znak @ označava poziv Razor izraza, to jest prelazak u C# kod
  - Ako baš želimo koristiti znak @ a ne C# kod, trebamo navesti "escape" sekvencu @@
  - Primjer: `<p>Twitter: @@PatrikZvijezda</p>`
- Izvršavanje se automatski prebacuje između C# i HTML koda
- Od pogleda se kreira u C# klasa koja se prevodi (kompajlira) prilikom startanja aplikacije
  - Dio prefiksiran sa @ se prevodi u C# kod
  - Primjer: `<p>Ime: @ime</p>`
- HTML se može prebaciti u C# i obrnuto, Razor parser većinom sam zna kako to interpretirati

```
for(int i=0; i<10; i++) {
    <p>Broj: @i</p>
}
```

Razor

- Komentari počinju sekvencom @* a završavaju sekvencom *@
  - Primjer: `@* Komentar *@`

- Ako unutar bloka koda ne želimo generirati HTML već tekst, tada Razor ne zna kada se prebaciti iz C# u html
- Rješenje je korištenje <text> pseudo-oznake (pseudo-tag)
  - Nije riječ o oznaci koja se generira kao istoimena HTML oznaka, pa ju možemo nazvati "pseudo-oznaka"
  - Moguće je i korištenje @: sekvence u slučaju da se radi o jednoj liniji teksta

```
@* Neispravno *@
@for (int i = 0; i < 10; i++)
{
    Broj: @i
}
```

```
@* Ispravno *@
@for (int i = 0; i < 10; i++)
{
    <text>Broj: @i</text>
    @:Broj: @i
}
```

Kako izraditi Razor pogled?

- Pogledi su smješteni u mapi **Views**
  - Pogledi vezani uz pojedini kontroler nalaze se u podmapi s nazivom jednakim nazivu kontrolera
  - Pogledi koje mogu koristiti svi kontroleri nalaze se u podmapi **Shared**
- Najjednostavniji način za napraviti pogled
  - Desni klik unutar akcijske metode, **Add View**
  - Potrebno je upisati naziv pogleda i kliknuti **Add**
  - Predefinirani naziv pogleda trebao bi biti jednak nazivu akcijske metode
  - Moguće je i da akcija koristi pogled koji se zove drukčije od naziva akcije, ali se to rjeđe koristi
- Uz jedan kontroler može biti vezan neograničen broj akcija, pa tako i neograničen broj pogleda

Posebni pogledi

- **_ViewImports.cshtml**
  - često se koristi za specificiranje naziva namespace-ova koji se koriste za sve poglede
  - Primjer: `@using Task11.ViewModels`
  - Osim toga se može koristiti i za specificiranje default modela za sve poglede, specificiranje default ubačenih servisa za sve poglede itd.
  - Primjer: `@model Task11.ViewModels.DefaultModel`
- **_ValidationScriptsPartial.cshtml**
  - Koristi se za klijentsku validaciju

```
@section Scripts {
    Html.RenderPartial("_ValidationScriptsPartial")
}
```

- **_ViewStart.cshtml**
  - Koristi se za odabir koji će layout biti prikazan, to jest koji će se layout view koristiti

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

Posebni pogledi

Kako se generira rezultat pogleda X?

- Ako je za pojedini pogled X postavljen layout pogled (varijabla *Layout* ima vrijednost) tada se kod generiranja rezultata pogleda X prvo pokreće layout pogled
- Pogled X se pokreće na mjestu gdje se u layout pogledu nalazi **@RenderBody()**
- U slučaju da nije postavljen layout pogled ili je vrijednost poništena (varijabla *Layout* je *null*), generira se samo pogled X
- Rezultat layout pogleda je koherentan izgled svih pogleda koji ga koriste
  - Zajedničke CSS reference
  - Zajedničke JS reference
  - Zajednička Navigacija
- Moguće je i da dio pogleda koristi jedan layout pogled, a dio koristi neki drugi - višestruki layouti
- Posebna komponenta layout pogleda: **@RenderSection("sekcija1")**, rezultat poziva te funkcije je sekcija s istim imenom iz pogleda X koji se trenutno pokreće

```
@* unutar X.cshtml *@
@section sekcija1 {
    <h1>U sekciji "sekcija1"</h1>
}
```

Tipiziranost pogleda

Po kriteriju tipiziranosti postoje dvije vrste pogleda:

1. **Netipizirani** (engl. weakly typed) pogledi
   - Tip modela nije unaprijed poznat
2. **Strogo tipizirani** (engl. strongly typed) pogledi
   - Tip modela je unaprijed poznat
   - Pogled koristi model točno definiranog tipa za generiranje izlaza
   - Tip može biti bilo koja klasa

Tipiziranost pogleda

- Definiranje tipiziranog pogleda:

```
@* Model predstavlja objekt klase Product *@
@model MojProjekt.Models.Product

@* Model predstavlja kolekciju objekata klase Product *@
@model IEnumerable<MojProjekt.Models.Product>

@* ili *@
@using MojProjekt.Models
@model IEnumerable<Product>
```

- Prilikom deklariranja modela, piše se malim slovom: `@model`
- Prilikom pristupanja modelu, piše se velikim slovom `@Model`

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

Slanje podataka od kontrolera pogledu

- Kontroler može proslijediti podatke pogledu na sljedeće načine:

1. Pomoću **ViewData** ili **ViewBag** objekta
2. Pomoću **TempData** objekta
3. Pomoću **Session** objekta
4. Pomoću modela

- Kontroler poziva pogled da generira HTML kod tako što poziva metodu View iz bazne klase
  - Primjer: `return View();`
  - Bazna klasa je **Microsoft.AspNetCore.Mvc.Controller**
  - Kao parametar se može proslijediti naziv pogleda: `return View("AddNew");`
  - Ako se ne proslijedi naziv pogleda, uzima se pogled jednakog naziva kao akcijska metoda

ViewData i ViewBag

- **ViewData** predstavlja objekt tipa **ViewDataDictionary**, to jest riječnik kojem je ključ tipa **string**, a vrijednost tipa **objekt**
- Služi za slanje podataka između kontrolera i pogleda
  - Primjer: kontroler sprema podatak u riječnik, a pogled dohvaća i čita taj podatak
- **ViewBag** sadrži sve parove ključ-vrijednost kao i *ViewData*
  - predstavlja alternativni mehanizam pristupanju podataka iz *ViewData* objekta

```
@* Pogled pristupa ViewData podacima pomoću
ViewData riječnika i ViewBag objekta *@

<h1>@ViewData["pozdrav"]</h1>
<h1>@ViewBag.pozdrav</h1>
```

Objekt TempData

- Najčešće se koristi kod **preusmjeravanja** korisnika. U jednom zahtjevu se inicijalizira i dostupan je kod idućeg zahtjeva. Nakon što se podaci iz **TempData** objekta pročitaju, brišu se.
- Princip rada kod preusmjeravanja:
  1. Kontroler stavi podatke u **TempData** i pozove metodu **RedirectToAction()** (MVC u pozadini objekt **TempData** sprema u Session ili Cookie i prenese ga u novu akcijsku metodu - na taj način TempData preživljava preusmjeravanje)
  2. Nova akcijska metoda pozove metodu **View**
  3. U pogledu je **TempData** dostupan
  4. Nakon čitanja se **TempData** uklanja

Objekt Html

- ASP.NET MVC uključuje pomoćne metode koje služe za implementaciju često korištenih funkcionalnosti
  - Pomoćne metode su implementirane na objektu **Html**
  - Poznate kao **HTML helperi**
- Metode koje često koristimo:
  - **Html.ActionLink()** kreira <a> link na zadanu akcijsku metodu
  - **Html.BeginForm()** kreira <form> element
    - Uobičajeno je koristiti u kombinaciji s @**using** izrazom
    - Kada @**using** blok završava, generira se zatvoreni </form> tag
  - **Html.TextBox()** kreira <input type="text"> element
    - Atribut Name je važan
    - **Html.TextBoxFor(x => x.ImageUrl)** automatski generira name atribut

    ➔ <input type="text" name="ImageUrl">

Korištenje metoda objekta Html

- ActionLink()
  - Stvara <a href="…"> link
- TextBox() i TextBoxFor()
  - stvaraju <input type="text">
- DropDownList() i DropDownListFor()
  - stvaraju <select>
- HiddenField() i HiddenFieldFor()
  - stvaraju <input type="hidden">
- CheckBox() i CheckBoxFor()
  - stvaraju <input type="checkbox">
- DisplayNameFor()
  - Stvara naziv prema [DisplayName("moj naziv")] atributu u modelu
- EditorFor()
  - Stvara HTML unos prema zadanom tipu
- DisplayFor()
  - Stvara HTML prikaz prema zadanom tipu

Html helperi – prednosti i mane

- HTML helperi
  - Prednost: Stvaraju dosljedan kod
  - Prednost: Mogu se proširiti ili prilagoditi
  - Mana: Gledajući kôd ne vidite jasno kakav će točno HTML generirati html helper
  - Mana: Proizvoljni atributi podržani su korištenjem posljednjeg anonimnog objekta, što može nezgodno izgledati
  - ➔ `@Html.TextBoxFor(modelItem => item.Description, new { @class="form-control-lg form-icon-trailing", placeholder="Description", ... })`
- Rješenje ovih nedostataka je korištenje tag helpera

Korištenje tag helpera

- Kao ActionLink()
  - Tag helper
    ```
    <a asp-controller="FormTest" asp-action="Add" class="btn btn-primary" >Send data</a>
    ```
  - Izlaz
    ```
    <a class="btn btn-primary" href="/FormTest/Add">Send data</a>
    ```

- Kao TextBoxFor(), HiddenFieldFor(), CheckBoxFor()…
  - `<input asp-for="TestText" id="testText" class="form-control" type="text">`

  ➔ `<input id="testText" class="form-control" type="text" data-val="true" data-val-required="The TestText field is required." name="TestText" value="">`

  - `<input asp-for="TestHidden" class="form-control" type="hidden" value="Hidden value">`

  ➔ `<input class="form-control" type="hidden" value="Hidden value" data-val="true" data-val-required="The TestHidden field is required." id="TestHidden" name="TestHidden">`

Korištenje tag helpera

- Kao DropDownListFor():
  - <select **asp-for**="ImageUrl" **asp-items**="Model.ImageItems"></select>

- Kao DisplayNameFor():
  - <label **asp-for**="ImageUrl"></label>

- EditorFor() i DisplayFor() nemaju pandan u tag helperima!

Hvala na pažnji!