

ASP.NET MVC

Vježba 5: Razor, model binding

Sadržaj

Načela pisanja view predložaka i razor sintaksa	3
Plan izrade aplikacije	4
Index – lista elemenata	5
Details – pregled detalja.....	6
Model binding	8
Razlike između POST i GET metode.....	8
Povezivanje vrijednosti forme i parametara akcije	9

Načela pisanja view predložaka i razor sintaksa

Razor engine je mehanizam ASP.NET MVC-a kojim se pomoću C# koda olakšava generiranje HTML koda. Razor sintaksa se sastoji od nekoliko ključnih koncepata¹:

- C# naredba počinje ključnim znakom @
- Unutar svakog view-a (kojem je proslijeđen neki model) dostupan je model objekt preko naredbe **@Model** (nakon čega slijedi svojstvo ili funkcija). Treba razlikovati definiciju modela (prva linija cshtml datoteke) koja je napisana u obliku **@model** (nakon čega slijedi tip modela)
- Kada se otvori blok vitičastim zagradama { i }, direktno unutar bloka nije potrebno koristiti znak @
- Ključne naredbe **@:** i **<text>** služe za ispis čistog teksta unutar bloka koda

Koristeći razor, unutar viewa moguće je pisati bilo kakav C# kod. No, slijedeći pravila ispravnog koncipiranja aplikacije, jedina logika koja je dopuštena je **jednostavno grananje** i **iteracija** po kolekciji. Po tom pitanju razor sintaksa je vrlo jednostvana:

The image shows a snippet of Razor view code with several annotations in blue boxes:

- Definicija tipa modela:** Points to the line `@model QuizManager.Web.Models.ContactSuccessViewModel`.
- Primjer isprepletanja HTML koda i C# koda. Kod korištenja if naredbe obavezno je koristiti vitičaste zagrade { i }.** Points to the `@if (Model.LastAccess != null)` block.
- Pristupanje svojstvima modela:** Points to the `@Model.LastAccess` property access within the if block.

The code snippet is as follows:

```
@model QuizManager.Web.Models.ContactSuccessViewModel

@{
    ViewBag.Title = "Contact";
}

<hgroup class="title">
    <h1>@ViewBag.Title.</h1>
    <h2>@ViewBag.Message</h2>
</hgroup>

<section class="contact-data">
    <header>
        <h3>Kontakt podaci uspješno spremljeni!</h3>
    </header>
    <div>
        IP: @Model.IPAddress
    </div>
    @if (Model.LastAccess != null)
    {
        <div>
            Last access: @Model.LastAccess
        </div>
    }
    <div>
        Browser: @Model.Browser
    </div>
    <h4>Poslani kontakt podaci</h4>
    <div>
        Ime i prezime: @Model.ContactData.Ime @Model.ContactData.Prezime
    </div>
    <div>
        Email: @Model.ContactData.Email
    </div>
    <div>
        <pre>
            @Model.ContactData.Poruka
        </pre>
    </div>
</section>
```

Controller se brine o tome da kreira model, napuni ga s podacima te ga proslijeđuje u View, koji onda s obzirom na podatke u modelu prezentira informacije korisniku.

¹ Kratki pregled razor sintakse može se naći ovdje: <http://haacked.com/archive/2011/01/06/razor-syntax-quick-reference.aspx/>

Plan izrade aplikacije

Tokom narednih vježbi, napraviti ćemo aplikaciju za konzultante – gdje će oni moći voditi evidenciju o svojim klijentima i sastancima s njima.

Neke od funkcionalnosti su: popis klijenata, detalji o jednom klijentu, popis sastanaka sa klijentom, detalji jednog sastanka, unos datoteke (file upload), kalendarski pregled nadolazećih sastanaka (po svim klijentima), pretraga klijenata/sastanaka.

Svaka nadolazeća vježba će se velikim dijelom naslanjati na prijašnje vježbe, tako da je potrebno razumjeti obrađeno gradivo kako bi se vježbe mogle napraviti.

Zadatak 5.1

Kreirati novi controller za prikaz informacijama o klijentima. U zadacima s UI koristiti Bootstrap principe uz pomoć bootstrap dokumentacije i ranijih vježbi.

1. Dodati novi controller u **Vjezba.Web/Controllers** i nazvati ga **ClientController**.
 - a. Desni klik na folder **Controllers** -> **Add** -> **Controller**
 - b. Odabrati **MVC empty controller**
 - c. Nazvati ga **ClientController**
2. Automatski je kreirana akcija **Index**, te ćemo nju koristiti za pozivanje i iscrtavanje tabličnog prikaza svih klijenata. Akciji se pristupa preko URL adrese: **/Client/Index**.
3. Unutar akcije **Index**, kreirati model koji je tipa **List<Client>**. Napuniti listu s klijentima na način da se pozove **MockClientRepository.Instance.All()**;
 - a. Proslijediti model view-u koji se zove **Index.cshtml**
4. Kreirati view **Index.cshtml** u mapi **Vjezba.Web/Views/Client**.
 - a. Postaviti na vrhu predloška (prva linija u cshtml datoteci) da se koristi model koji je tipa **List<Client>**. Koristiti ranije opisanu direktivu **@model**.
 - b. U viewu ispisati samo jednu rečenicu: „U sustavu postoji N klijenata.“, gdje je N broj klijenata do kojeg je moguće doći naredbom **@Model.Count**. Navedeni tekst prikazati Bootstrap komponentom „alert“.

Index – lista elemenata

Česta je situacija da u aplikaciji trebamo napraviti tablicu podataka. U tom slučaju, model koji se proslijeđuje u view je kolekcija ili kompleksniji objekt koji unutar sebe ima kolekciju koju ćemo u konačnici prikazati. U donjem primjeru je to lista gradova, koji se prikazuju svaki u svom <div> elementu:

```
@model List<Vjezba.Web.Models.Mock.City>
```

```
<h2>Pregled gradova</h2>
```

```
@foreach (var city in Model)
```

```
{
```

```
    <div>
```

```
        Naziv grada: @city.Name
```

```
    </div>
```

```
}
```

Drukčiji view, gdje je model kolekcija objekata

Češći način prikaza je koristeći HTML <table> element, koji će se koristiti u narednom zadatku.

Zadatak 5.2

Proširiti kreirani **Index.cshtml** na način da uz ukupan broj klijenata prikazuje i podatke o klijentima tablično koristeći Bootstrap komponentu za tablični prikaz. Ekran treba izgledati po prilici ovako:

Vjezba.Web Home About Contact			
U sustavu postoji 157 klijenata.			
Client	Address	Email	City
Auberon Markey	186 Scott Avenue	amarkey0@umich.edu	Adorjan
Robin Eck		reck1@dot.gov	
Henderson Hartzog	8799 Golf View Avenue	hhartzog2@facebook.com	Oklahoma City
Quinn Tarrant	9896 Monica Terrace	qtarrant3@wordpress.org	Figueiró dos Vinhos
Lily D'Alwis	0 Starling Road	ldalwis4@cbslocal.com	Oklahoma City
Hoyt Reppaport	19 Esker Place	hreppaport5@yandex.ru	Roche Terre
Benita Busen	529 Evergreen Junction	bbusen6@cdc.gov	Muzi
Torrin Eaglesham		teaglesham7@uol.com.br	
Justine Beaglehole	40567 Twin Pines Drive	jbeaglehole8@icq.com	Rudong
Fritz Strapp	173 Loeprich Drive	fstrapp9@jugem.jp	Oklahoma City
Beale McCandless		bmccandlessa@google.co.uk	
Celeste Rew	673 7th Road	crew@technorati.com	Auas
Tory Karlolczak		tkarlolczak@virginia.edu	
Honey Squibbs		hsquibbsd@about.com	
.lere Cuthbon	8 Bonner Court	icuthbon@facebook.com	Dennila

Details – pregled detalja

Uz listu, prilagođeni pregled detalja za neki podatak se također veoma često koristi u poslovnim aplikacijama. Na Index stranici se prikazuju jednostavniji/kompaktniji podaci koji daju samo dio informacija za svaki element, dok se na stranici detalja prikazuju detaljniji podaci – u kontekstu aplikacije za evidenciju klijenata, to bi moguće bili slika, ime, prezime, adresa, sumarni podaci o sastancima, popis sastanaka, itd.

Zadatak 5.3

Kreirati novu akciju u `ClientController` controller-u koja dohvaća i prikazuje podatke samo jednog klijenta.

1. Kreirati akciju **Details** u **ClientController** klasi
 - a. Akcija Details prima jedan parametar naziva id, tipa int
 - b. Unutar akcije details, koristiti funkciju **MockRepository** objekta **FindByID** koja dohvaća samo jednog klijenta s obzirom na proslijeđeni id.
 - c. Proslijediti dohvaćeni objekt kao model u view
2. Kreirati view **Details.cshtml** i smjestiti ga u ispravnu mapu. Navedeni view prima objekt tipa `Client` (samo jedan klijent)
 - a. Osigurati da se, ukoliko je proslijeđen nepostojeći id (akcija se pozove kao `/Client/Details/999`), ili ako se ne proslijedi parametar (akcija se pozove kao `/Client/Details`), da se korisniku prikaže ispravna poruka. Konzultirati se s omiljenim internetskim pretraživačem kako omogućiti da parametar akcije kontrolera može biti opcionalan.

Rješenje treba izgledati po prilici ovako:

Details

Client

ID	2
Ime i prezime	Robin Eck
Email	reck1@dot.gov
Spol	F
Adresa	
Tel.	
Grad	

[Back to List](#)

Zadatak 5.4

Omogućiti da se odabirom elementa s liste klijenata (akcija i view Index) prikažu detalji o odabranom elementu.

1. Modificirati **Index.cshtml** na način da se omogući prijelaz na prikaz detalja za željeni redak (dodati posebni link koji će se izgenerirati za svaki redak u tabličnom prikazu)
 - a. Naziv klijenta treba biti link na detalje o klijentu
2. Modificirati **Details.cshtml** na način da se doda bootstrap komponenta „krušne mrvice“ ☺ koja nudi opciju za povratak na prethodni ekran

U sustavu postoji 157 klijenata.



Client	Address	Email	City
Auberon Markey	186 Scott Avenue	amarkey0@umich.edu	Adorjan
Robin Eck		reck1@dot.gov	

Details

[Popis klijenata](#) / [Auberon Markey](#)

Client

ID	1
Ime i prezime	Auberon Markey
Email	amarkey0@umich.edu
Spol	M
Adresa	186 Scott Avenue
Tel.	464-599-3923
Grad	Adorjan

Model binding

Model binding je jedan od osnovnih koncepata MVC paradigme, te ćemo u narednom poglavlju proučiti što je to i kako funkcionira.

Razlike između POST i GET metode

Akcije controllera dijelimo na sljedeće dvije bitne skupine:

- **GET akcije** – obrađuju zahtjeve koji kao rezultat vraćaju podatke za pregled, i ne šalju POST parametre na server. Ukoliko neku akciju želimo ograničiti samo da prima zahtjeve poslane GET metodom, potrebno je dodati anotaciju **[HttpGet]**
- **POST akcije** – služe za obradu podataka koji su unešeni preko forme. Ukoliko neku akciju želimo ograničiti samo da prima zahtjeve poslane POST metodom, potrebno je dodati anotaciju **[HttpPost]**

Primjer i pojašnjenje GET i POST akcije za kontakt formu:

```
public ActionResult Contact()
{
    ViewBag.Message = "Your contact page";
    return View();
}

[HttpPost]
public ActionResult Contact(FormCollection formData)
{
    //Obrada podataka

    return View("Contact");
}
```

Dvije različite akcije (iako se jednako zovu i jednaki im je URL):

- Ukoliko se ne navede eksplicitno HTTP METHOD, podrazumjeva se GET (gornja akcija)
- Donja akcija će se zvati samo ukoliko se radi o POST metodi

Potrebno je prilagoditi i HTML:

```
<h3>Pošaljite nam upit!</h3>
<form action="/Home/Contact" method="post">
  <div>
    Ime: <input type="text" name="ime" />
    Prezime: <input type="text" name="prezime" />
  </div>
</form>
```


Povezivanje vrijednosti forme i parametara akcije

Ovakav način korištenja MVC nije ono što ju čini tako dobrom:

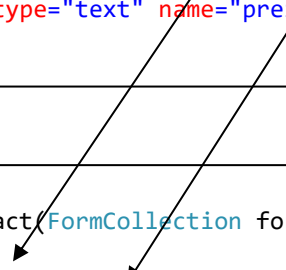
```
<h3>Pošaljite nam upit!</h3>
<form action="/Home/Contact" method="post">
  <div>
    Ime: <input type="text" name="ime" />
    Prezime: <input type="text" name="prezime" />
  </div>
```

HomeController.cs

```
[HttpPost]
public ActionResult Contact(FormCollection formData)
{
    var ime = formData["ime"];
    var prezime = formData["prezime"];

    //Obrada podataka

    return View();
}
```



Naime, na ovaj način uviđamo sljedeći niz problema:

- velika je mogućnost pogreške jer se sva imena varijabli i ostalih ulaza prenose preko stringova
- potrebno je pisati vlastiti kod za pretvorbu raznih tipova
- posebni načini rukovanja bool vrijednostima

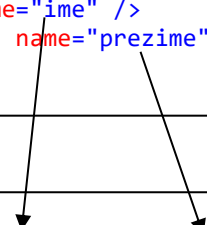
Međutim, moguće je ponešto optimizirati ovaj način prijenosa – primjerice, ukoliko akcija controllera prima parametar naziva „ime“, tipa „string“, možemo (bez promjena u View datoteci) automatski učitati navedene parametre u jednostavne varijable:

```
<h3>Pošaljite nam upit!</h3>
<form action="/Home/Contact" method="post">
  <div>
    Ime: <input type="text" name="ime" />
    Prezime: <input type="text" name="prezime" />
  </div>
```

HomeController.cs

```
[HttpPost]
public ActionResult Contact(string ime, string prezime)
{
    //Obrada podataka

    return View();
}
```



Naime, preko atributa „name“ u HTML jeziku se definira naziv tog parametra u akciji, kao što smo ih dohvaćali i preko **formData**.

Zadatak 5.5

Modificirati Client/Index view na način da se doda forma za pretragu.

1. U view **Client/Index.cshtml** dodati **<form>** element
 - a. Podesiti metodu **GET**
 - b. Podesiti ispravan „**action**“ atribut na način da se pri slanju forme aktivira akcija controller-a Index
 - c. Dodati **<input>** polje u formu za unos teksta
 - i. Postaviti atribut name=„**query**“
 - d. Dodati **<input type=„submit“>** u formu (aktivira slanje forme)
2. U akciji **Index** u controlleru potrebno je dodati parametar koji će sadržavati vrijednost unesenu na formi (nazvati ga ispravnim imenom)
 - a. Prikazati samo one klijente koje u svojem nazivu imaju uneseni pojam
3. Podesiti da forma izgleda kao dolje na slici (osnovni bootstrap principi)
 - a. Kako bi to postigli, koristiti klasu „**form-inline**“ na form elementu, te **form-group** i **form-control**, prema uputstvima na službenim stranicama za bootstrap radni okvir

Zadatak 5.5 - GET forma s jednim parametrom

Client	Address	Email	City
Auberon Markey	186 Scott Avenue	amarkey0@umich.edu	Adorjan
Aube Shemilt		ashemilt21@soundcloud.com	

© 2019 - Vježba Web

Zadatak 5.6

Modificirati Client/Index view na način da se doda identična forma za pretragu u zasebni tab ali s metodom **POST**.

1. Isto kao u gornjem zadatku, dodati formu u tab komponentu (bootstrap)
 - a. Podesiti metodu **POST**
 - b. Umjesto query, nazvati input polje „**queryName**“
 - c. Dodati dodatno polje u formu u koje se unosi string, nazvati ga **queryAddress**
2. U controlleru Client kreirati akciju Index (istog imena), ali s naznakom da procesira samo zahtjeve koji su došli POST metodom
 - a. U ovoj akciji filtrirati klijente koje u svojem imenu imaju uneseni pojam **queryName**, a u svojoj adresi imaju uneseni pojam **queryAddress**
3. Korištenjem osnovnih bootstrap principa, postići da forma izgleda slično kao na slici dolje

Zadatak 5.5

Zadatak 5.6

Zadatak 5.6 - POST forma s dva parametra

Pretraga po imenu

drive

Submit

Client	Address	Email	City
Justine Beaglehole	40567 Twin Pines Drive	jbeaglehole8@icq.com	Rudong
Fritz Strapp	173 Loeprich Drive	fstrapp9@jugem.jp	Oklahoma City
Menard Bullerwell	7 Amoth Drive	mbullerwellg@wikipedia.org	Belas
Dar Dootson	0 Westend Drive	ddootsons@psu.edu	Belas
Marcy Andrivot	58579 Melby Drive	mandrivotx@ox.ac.uk	Ustrzyki Dolne
Bertie Erasmus	947 I akeland Drive	herasmus20@businessinsider.com	Mazakiai

Gore navedeni način povezivanja parametara s unesenim vrijednostima na formi je nešto bolji od prijašnjeg, no i dalje ima nedostataka:

- Broj parametara u metodi može drastično rasti (primjerice, forma na kojoj se unosi više od 10 vrijednosti)
- Nije se smanjila mogućnost pogreške krivog naziva varijable

Umjesto toga, moguće je ostaviti ASP.NET MVC-u da automatski popuni polja u objektu prema imenima HTML input polja. Naime, ukoliko se kao parametar akcije očekuje kompleksni objekt (vlastito kreirani model), ASP.NET MVC mehanizam će pokušati kreirati instancu tog objekta, te popuniti njegova svojstva s obzirom na imena polja na formi. To je ilustrirano sljedećim programskim odsječkom:

```
<h3>Pošaljite nam upit!</h3>
<form action="/Home/Contact" method="post">
  <div>
    Ime: <input type="text" name="ime" />
    Prezime: <input type="text" name="prezime" />
  </div>
```

```
public class ContactModel
{
    public string Ime { get; set; }
    public string Prezime { get; set; }
}
```

```
[HttpPost]
public ActionResult Contact(ContactModel model)
{
    var ime = model.Ime;
    var prezime = model.Prezime;

    //Obrada podataka

    return View();
}
```

Zadatak 5.7

Dodati treću formu na istu stranicu, te dopustiti unos dodatnih parametara pretrage. Također, kreirati novu klasu `ClientFilterModel` u koju će se povezivati vrijednosti s forme.

1. Kreirati klasu **`ClientFilterModel`** u datoteci **`Vjezba.Web/Models/ClientModels.cs`**
 - a. Primjetiti kako je moguće staviti više klasa u istu datoteku, primjerice kao što je inicijalno stavljeno u `AccountViewModels.cs` ili `ManageViewModels.cs` datoteci
2. Kreirati još jednu formu na pregledu klijenata, koja dopušta unos sljedećih parametara:
 - a. Dio naziva klijenta
 - b. Dio emaila
 - c. Dio adrese
 - d. Dio naziva grada
 - e. Osigurati da se gornja polja ispravno povezuju na svojstva **`ClientFilterModel`** klase
3. Kreirati akciju u controlleru koja ima prototip **`public ActionResult AdvancedSearch(ClientFilterModel model)`**.
 - a. Po potrebi dodati ograničenja na tip metode
 - b. Uzeti u obzir da ne moraju nužno sva polja biti popunjena²
4. Korištenjem osnovnih bootstrap principa, podesiti da forma izgleda kao na slici dolje

[Zadatak 5.5](#) [Zadatak 5.6](#) [Zadatak 5.7](#)

Zadatak 5.7 - POST forma, binding na model

Client	Address	Email	City
Fabiano Sorsby	18 Center Center	fsorsby31@barnesandnoble.com	Muzi
Larissa Torra	5403 Thompson Trail	ltorra3r@yellowbook.com	Muzi

© 2019 - Vjezba.Web

Iako gornji način rješava problem potencijalno velikog broja parametara, svejedno ne rješava sljedeći problem:

- Mogućnost pogreške u nazivu parametara

Kako bi riješili taj problem, koristimo razor funkcionalnost koja se bazira na stablima izraza kako bi se napravilo HTML input polje na način da se specifično povezuje sa željenim poljem u modelu. Primjerice, kako bi u viewu napravili input polje za unos imena, možemo isto polje definirati ovakvim izrazom (uz obvezno definiranje tipa modela na vrhu):

² Ovo možda može pomoći: <http://stackoverflow.com/questions/3682835/if-condition-in-linq-where-clause>

Contact.cshtml

```
@model QuizManager.Web.Models.ContactModel

...

<section class="contact-form">
    <header>
        <h3>Pošaljite nam upit!</h3>
        <form action="/Home/Contact" method="post" class="form-inline">
            <div>
                Ime: @Html.EditorFor(p => p.Ime, new
                    { htmlAttributes = new {
                        @class = "form-control",
                        placeholder = "Pretraga po nazivu" }
                    })

                Prezime: <input type="text" name="prezime" />
            </div>
        </div>
    </div>
```

Gornji način se može koristiti samo u slučaju da polja odgovaraju direktno poljima modela, stoga je prvo potrebno razumjeti koncept korisničke kontrole/djelomičnog pogleda (partial view) kako bi se gorenavedena funkcionalnost mogla implementirati na ekran za pregled/pretragu klijenata.