.NET Okruženje

Vjezba 8: CRUD, validacija, padajući izbornik

Sadržaj

Metoda TryUpdateModel

Do sada smo spominjali u kontekstu povezivanja modela s podacima s forme jedino mogućnost automatskog povezivanja na način da se direktno podaci s forme spremaju u model (akcija Create). Međutim, takav način nam ne odgovara uvijek – pogotovo kod uređivanja podataka iz baze. Naime, podsjetimo se kako izgleda proces kompletnog prikaza edit forme i spremanja podataka natrag u bazu korištenjem repository obrasca:

1. Poziva se akcija Edit
    a. Dohvaća se podatak iz baze (primjerice, kompanija sa konkretnom ID vrijednošću)
    b. Prikazuje se forma s već popunjenim podacima iz baze (model Client)
2. Kod slanja forme na server, poziva se akcija Edit (HttpPost) koja kao parametar prima model Client
    a. Spremamo tog klijenta u bazu podataka

Koji je problem u gornjem scenariju? Što ako klijent ima neka polja koja se ne prikazuju na formi, što će biti s tim podacima?

Zbog gornjeg problema, sigurnije je koristiti TryUpdateModel funkciju, i to na sljedeći način (isti scenarij kao gore, malo modificiran):

1. Poziva se akcija Edit
    a. Dohvaća se podatak iz baze (primjerice, klijent sa konkretnom ID vrijednošću)
    b. Prikazuje se forma s već popunjenim podacima iz baze (model Client)
2. Kod slanja forme na server, poziva se akcija Edit (HttpPost) koja ~~kao parametar prima model Client~~
    a. Dohvaća se trenutno aktualni klijent iz baze
    b. Nad tim dohvaćenim klijentom poziva se TryUpdateModel funkcija koja podatke s forme zapiše u model
    c. Spremamo tog klijenta u bazu podataka

Naravno, najoptimalnije bi bilo korištenje potpuno drugog objekta kao modela koji se šalje u view (recimo ClientViewModel), i onda premapiranje pojedinih polja u entitet Client, no takva arhitektura također zna biti pomalo problematična jer za većinu entiteta iz baze nam je dovoljan i gore opisan scenarij[1].

[1] Dobar video o korištenju UpdateModel funkcije (iz starije verzije ASP.NET MVC, ali primjenjivo), kao i atributa ActionName (iako koristi neke od stvari koje nisu obrađene kao komunikacija s bazom podataka, i sl.): https://www.youtube.com/watch?v=uXwmyuvrn1E (iako je glas malo čudan ☺)

Atribut ActionName

Kod izrade akcije za uređivanje ili za kreiranje novih objekata, nailazimo na problem da bi htjeli imati dvije ovakve akcije (primjerice, za Edit akciju):

ClientController.cs

```
public ActionResult Edit(int id)
{
    ...
}

[HttpPost]
public ActionResult Edit(int id)
{
    this.UpdateModel(...);
    ...
}
```

Ove dvije funkcije su identične i ne mogu se prevesti u C# jeziku

Kako bi ipak zadržali svojstvo da URL tih akcija bude identičan (razlikuje se samo po metodi GET ili POST), možemo koristiti ovako nešto:

ClientController.cs

```
[ActionName("Edit")]
public ActionResult EditGet(int id)
{
    ...
}

[HttpPost]
[ActionName("Edit")]
public ActionResult EditPost(int id)
{
    this.UpdateModel(...);
    ...
}
```

Zadatak 8.1

Napraviti mogućnost uređivanja postojećih podataka za klijenta. Trenutno je napravljena mogućnost kreiranja novog klijenta, te treba postojeći mehanizam proširiti mogućnošću osvježavanja podataka za odabranog klijenta.

1. Napraviti odgovarajuće akcije u controlleru ClientController (**Edit** i **[HttpPost]Edit**). Prilikom obrade rezultata dobivenih s edit forme ([HttpPost]Edit() akcija), potrebno je napraviti sljedeće:
    a. Dohvatiti željeni objekt iz baze podataka pretragom po ID-u
    b. Pozvati metodu **TryUpdateModelAsync**() iz controllera
        i. EF prati promjene nad dohvaćenim objektom, te već pozivom TryUpdateModel() će sve promjene biti zabilježene i dovoljno je samo nakon tog pozvati SaveChanges()

   c. Spremiti promjene i preusmjeriti na stranicu **Index** (koristiti `return RedirectToAction() poziv`)
   d. Po potrebi koristiti ActionName atribute
2. Kreirati **Edit.cshtml** pogled
   a. Koristiti **Html.HiddenFor**() ili **odgovarajući tag helper** za spremanje podatka kao što je Id koji nam treba kako bi znali urediti ispravni objekt
3. Modificirati **Index.cshtml** i **Details.cshtml** kako bi se s tih stranica moglo doći do forme za uređivanje (potrebno je proslijediti parametar id u linku)
   a. Na **Create** i **Edit** potrebno je dodati bootstrap kontrolu „krušne mrvice“, kako bi se moglo vratiti na pregled svih klijenata (slično kao što postoji na detaljima klijenta).
   b. Modificirati **Index** view kako bi prikazivao klijente poredano po **ID** polju
   c. U tabličnom prikazu, najčešće je to dodatna kolona s akcijom „edit“
4. Izdvojiti zajednički kod iz **Create.cshtml** i **Edit.cshtml** u djelomični pogled **_CreateOrEdit.cshtml**.
5. Forma treba biti napravljena slijeđenjem standardnih bootstrap principa

Validacija

Validacija je jedan od bitnijih segmenata svake aplikacije. Osnovni koncepti validacije su:

- Onemogućiti spremanje podataka koji nisu konzistentni
  - Validacija na klijentu (prije slanja na server)
  - Validacija na serveru
  - Validacija na razini baze podataka
- Prikazati adekvatnu poruku korisniku u slučaju pogreške

ASP.NET MVC framework nudi out-of-the-box riješenje za validaciju, ali također omogućava naprednu prilagodbu validacijskih mehanizama na svim razinama (klijentu, serveru) o kojoj će biti više govora u kasnijim predavanjima.

Anotacije modela

Osnovna metoda validacije je anotacijama pojedinih svojstava u modelu. Naravno, moguće je definirati validaciju i neovisno od modela, no s obzirom da se MVC paradigma uvelike temelji na modelima, gotovo uvijek je preporučljivo koristiti jasno definiranu klasu kao model prilikom izrade formi za unos i izmjenu podataka.

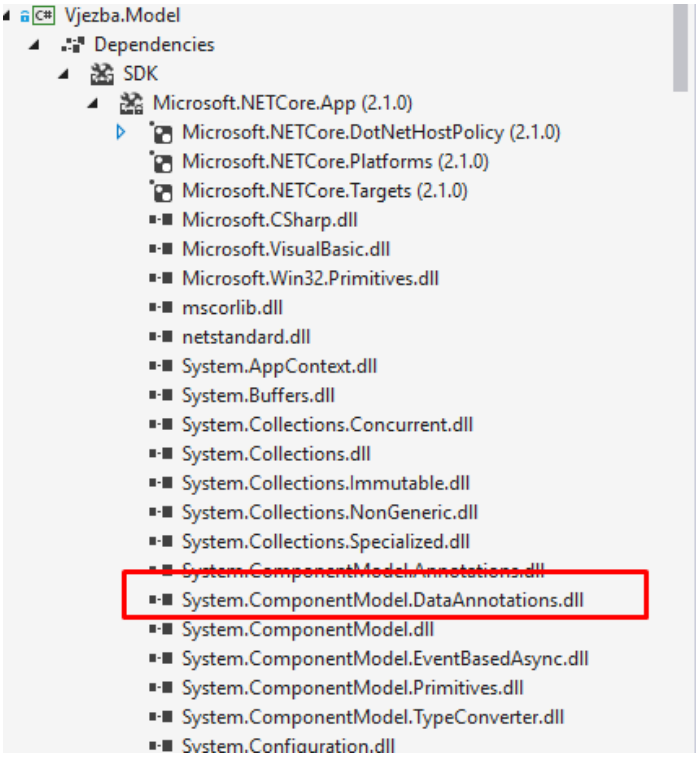
Primjena validacije se sastoji od sljedećih nekoliko koraka:

- Anotirati željena svojstva u modelu
- U viewu pozvati naredbu za prikaz validacijske poruke (ukoliko se koristi)
- U controlleru provjeriti je li model prošao validaciju

Validacija obveznog polja (required)

Jedna od najčešćih i najjednostavnijih tipova validacije je obvezno polje. U sljedećem primjeru, postavit ćemo da je obvezno polje pri unosu podataka o kvizovima naziv kviza (Title).

Pošto se model klasa prema kojoj prikazujemo view nalazi u QuizManager.Model projektu, potrebno je osigurati da taj projekt ima referencu na System.ComponentModel.DataAnnotations biblioteku. U .NET Core aplikaciji, automatski je definiran set međuzavisnosti (Dependencies), i najvjerojatnije se tamo i nalazi navedena biblioteka (slika desno).

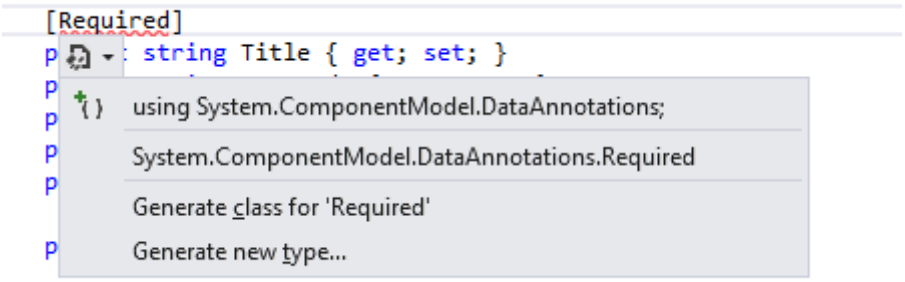


Nakon toga možemo dodati atribut Required na svojstvo u koje spremamo naslov kviza:

Quiz.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace QuizManager.Model
{
    public class Quiz
    {
        public int Id { get; set; }

        [Required]
        public string Title { get; set; }

        public string Keywords { get; set; }
        public string Author { get; set; }
        public DateTime DateCreated { get; set; }
        public QuizCategory Category { get; set; }

        public List<Question> Questions { get; set; }
    }
}
```

S obzirom da pri dodavanju [**Required**] atributa unutar klase Quiz nije poznat imenski prostor u kojem se ta klasa nalazi, moguće je ostaviti VS alatu da ju sam pronađe:



U idućem koraku, potrebno je omogućiti da se klijentu prikaže odgovarajuća poruka ukoliko validacija nije zadovoljena – u ovom slučaju, ukoliko ostavi prazno polje za naslov kviza:

_CreateOrEdit.cshtml

```
...

<div class="form-group">
    <label asp-for="Email" class="control-label"></label>
    <input asp-for="Email" class="form-control" />
    <span asp-validation-for="Email" class="text-danger"></span>
</div>

...
```

Potrebno je dodati i odgovarajuće javascript biblioteke kako bi klijentska validacija funkcionirala ispravno. U predlošku novog projekta koji je kreiran, osnovne validacijske skripte su dodane automatski unutar kontrole _ValidationScriptsPartial. Dodat ćemo sljedeći odsječak u _Layout.cshtml kako bi validacija bila dostupna na svim formama koje radimo u projektu:

_Layout.cshtml

```
    <footer class="border-top footer text-muted">
        <div class="container">
            &copy; 2021 - Vjezba.Web - <a asp-area="" asp-controller="Home" asp-
action="Privacy">Privacy</a>
        </div>
    </footer>
    <script src="~/lib/jquery/dist/jquery.min.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
    <partial name="_ValidationScriptsPartial" />
    <script src="~/js/site.js" asp-append-version="true"></script>
    @await RenderSectionAsync("Scripts", required: false)
</body>
```

Nakon toga, ukoliko pokrenemo aplikaciju, i pokušamo dodati novi kviz preko sučelja, dobit ćemo ispravnu validacijsku pogrešku. Međutim, na taj način se nismo u potpunosti osigurali, jer je moguće simulirati zahtjev koji se šalje s podacima forme i na serveru i dalje unosi krive podatke. Iz tog razloga, potrebno je prije obrade informacija na serveru, provjeriti prolaze li podaci validaciju:

QuizController.cs

```
[HttpPost]
public IActionResult Create(Quiz model)
{
    if(ModelState.IsValid)
    {
        this._dbContext.Quizes.Add(model);
        this._dbContext.SaveChanges();

        return RedirectToAction(nameof(Index));
    }

    return View(model);
}


[HttpPost, ActionName("Edit")]
public async Task<IActionResult> EditPost(int id)
{
    var quiz = this._dbContext.Quizes.FirstOrDefault(p => p.ID == id);
    var ok = await this.TryUpdateModelAsync(quiz);

    if(ok)
    {
        this._dbContext.SaveChanges();
        return RedirectToAction(nameof(Index));
    }

    return View(quiz);
}
```

Zadatak 8.2

Portebno je modificirati **Client** model, view **_CreateOrUpdate.cshtml** i controller **ClientController** na način da se ugradi validacija za obvezna polja.

1. Dodati ispravne anotacije za obvezna polja u modelu
2. Omogućiti klijentsku validaciju
3. Modificirati view na način da se korisniku prikazuju poruke u slučaju pogrešnog unosa
4. Osigurati da se na serveru također izvodi validacija.
    a. Za akcije **Edit** i **Create** je potrebno napraviti validaciju
    b. Slijediti gornja uputstva i implementirati **TryUpdateModel** metodu u slučaju **Edit** poziva, te **ModelState.IsValid** u slučaju **Create** poziva

Napomena: Općenito, za sve daljnje zadatke obvezno je koristiti navedenu validaciju za obvezna polja ukoliko je potrebno.

Ostale ugrađene validacije

Osim obveznog polja (koje je najčešći oblik validacije), moguće je i staviti ograničenje na minimalni ili maksimalni broj znakova za neko polje; te odrediti interval za minimalni ili maksimalni unešeni broj.

Također, moguće je staviti više validacija na isto polje (polje može biti obvezno, i može imati maksimalni broj znakova)[2].

Question.cs

```
namespace QuizManager.Model
{
    public class Question
    {
        [Required]
        [Range(1, 50)]
        public int Points { get; set; }

        [Required]
        [StringLength(2000, MinimumLength = 5)]
        public string QuestionText { get; set; }
    }
}
```

Također, moguće je definirati i proizvoljne poruke za prikaz korisniku. Naravno, postoji i mogućnost prikaza lokalizirane poruke, no o tome će biti više riječi u kasnijim poglavljima. Primjer prilagođene poruke:

Question.cs

```
namespace QuizManager.Model
{
    public class Question
    {
        [Required]
        [Range(1, 50, ErrorMessage = "Broj bodova mora biti između 1 i 50.")]
        public int Points { get; set; }

        [Required]
        [StringLength(2000, MinimumLength = 5)]
        public string QuestionText { get; set; }
    }
}
```

Zadatak 8.3

Implementirati gore navedene validacije na Client model.

1. Dodati cjelobrojno svojstvo WorkingExperience (godine radnog staža)
    a. Generirati migracijsku skriptu
    b. Može biti null
    c. Može biti u intervalu [0-100]
2. Dodati jednu validaciju za min/max broj znakova (FirstName, LastName, Email)
3. Dodati prilagođene poruke na neku od implementiranih validacija.
4. Osigurati da se greške prikazuju na korisničkom sučelju

Napomena: u ostalim zadacima nije potrebno ispisivati vlastitu prilagođenu poruku, već je dovoljno ostaviti „default“.

[2] Više o validaciji i anotacijama: https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-2.2

Padajući izbornik

Padajući izbornik (eng. combobox ili dropdown list) je nezaobilazna kontrola u bilo kojoj aplikaciji. Naime, koristi se u svakoj situaciji gdje trebamo osigurati unos polja koje je strani ključ (1-N veza). Primjerice, ako promotrimo Quiz klasu, strani ključ je kategorija, i za svaki kviz je potrbeno odabrati kategoriju:

Quiz.cs

```
namespace QuizManager.Model
{
    public class Quiz
    {
        ...

        public DateTime DateCreated { get; set; }

        public int? CategoryId { get; set; }

        ...
    }
}
```

Kako bi prikazali padajući izbornik za odabir kategorije, potrebno je:

- Modificirati controller i u željenoj akciji proslijediti sve moguće kategorije koje se mogu izabrati
- Modificirati view i osigurati da se za proslijeđene kategorije iscrtava padajući izbornik

Modifikacija controllera se svodi na spremanje svih mogućih kategorija u ViewBag (ili u model, ovisno o implementaciji):

QuizController.cs (dodavanje sadržaja za padajući izbornik)

```
var selectItems = new List<System.Web.Mvc.SelectListItem>();

//Polje je opcionalno
var listItem = new SelectListItem();
listItem.Text = "- odaberite -";
listItem.Value = "";
selectItems.Add(listItem);

foreach (var category in _db.QuizCategories)
{
    listItem = new SelectListItem();
    //Popuniti polja Text (ono što se prikazuje korisniku), Value (id)
    selectItems.Add(listItem);
}

ViewBag.PossibleCategories = selectItems;
```

Ovakav identični kod je potrebno izvršiti na ovim mjestima:

- Prilikom akcije **Create** (stvaranje novog)
- Prilikom akcije **[Post]Create** (procesiranje podataka s forme) u slučaju da validacije **ne prođe** i potrebno je ponovno prikazati isti View
- Prilikom akcije **Edit** (uređivanje postojećeg)
- Prilikom akcije **[Post]Edit** (procesiranje podataka s forme) u slučaju da validacija **ne prođe** i potrebno je ponovno prikazati isti View

Iz tog razloga, preporučljivo je taj kod izdvojiti u posebnu funkciju.

U view-u, potrebno je dodati odgovarajući poziv Html helper metode:

_CreateOrEdit.cshtml

```
<div class="form-group">
    <label class="control-label">Category</label>
    <select asp-for="CategoryID" asp-
items="ViewBag.PossibleCategories" class="form-control"></select>
</div>
```

Zadatak 8.4

Implementirati padajući izbornik kod uređivanja podataka o kompaniji na način da je moguće odabrati grad iz padajućeg izbornika, te source kod za punjenje mogućih vrijednosti izdvojiti u posebnu funkciju (primjerice, **FillDropDownValues**) i pozvati ju u odgovarajućim trenucima (kako je gore navedeno).

1. Mora biti moguće odabrati i „praznu vrijednost“
    a. Prazna vrijednost je SelectItem koji ima Text = „odaberite“ ili „“, a Value = „“