Razvoj Web Aplikacija

Predavanje 10

Modeli

- MVC Model – model pogleda (engl. ViewModel)
- Klijentska validacija modela
- Serverska validacija modela
- Mapiranje modela
  - Automapper
  - Višeslojna arhitektura

MVC model - ViewModel

Model

- U MVC-u *model* može značiti:
  - Podaci koji se šalju od klijenta prema akciji ili iz akcije klijentu
    ➔ To zovemo model pogleda ili **viewmodel**
  - Podaci koji se dohvaćaju iz baze podataka i obrađuju u aplikaciji ili šalju iz aplikacije u bazu podataka
    ➔ To zovemo model baze podataka ili **DAL model** (data access model)
- Postupak pretvaranja podataka poslanih s klijenta preko parametra akcije naziva se povezivanje modela (model binding), isto kao u Web API
- U troslojnoj (3-tier) arhitekturi može postojati i treća vrsta modela: model poslovnog sloja ili **BL model**
- Obično pretvaramo jedan model u drugi, i to nazivamo **mapiranje**
  - Dvoslojna aplikacija: viewmodel ⇔ DAL model
  - Troslojna aplikacija: viewmodel ⇔ BL model ⇔ DAL model

Model pogleda (viewmodel)

- Viewmodel ne bi trebao imati puno funkcionalnosti
- Bez programske logike!
- Namijenjeno samo za prijenos prikazanih informacija
- Međutim, može bit složen
- Primjer: podaci o računu i stavke računa u istom modelu

Primjeri modela

- Primjer modela 1:

```
public class EmailNotification
{
    public int Id { get; set; }
    public string Receiver { get; set; }
    public string Subject { get; set; }
    public string Body { get; set; }
}
```

- Primjer modela 2:

```
public class EmailBatch
{
    public DateTime? SentAt { get; set; }
    public IEnumerable<EmailNotification> Notifications { get; set; }
    public int Errors { get; set; }

}
```

- Model također može biti kolekcija:

```
IEnumerable<EmailNotification>
```

Provjera valjanosti - validacija

Provjera valjanosti (validacija) modela

- Atributi validacije modela

```
public class EmailNotification
{
    public int Id { get; set; }
    [EmailAddress]
    public string Receiver { get; set; }
    [StringLength(128)]
    public string Subject { get; set; }
    public string Body { get; set; }
}
```

- Svi podržani atributi validacije – pogledajte Web API
- Validacija modela je automatska
  - Ponekad bismo možda željeli unutar kontrolera ponovno validirati model

```
ModelState.ClearValidationState(nameof(EmailNotification));
if (!TryValidateModel(emailNotification, nameof(EmailNotification)))
{
    return View(emailNotification);
}
```

Model – serverska validacija

- Serverska (server-side) validacija

```
if (!ModelState.IsValid)
    return View(emailNotification);
```

- Tipična obrada serverske validacije
  - Ako validacija modela ne uspije, klijentu vrati prikaz ispunjen neispravno unesenim podacima
  - Ako validacija modela uspije, vrati rezultat preusmjeravanja (*Post-Redirect-Get* uzorak ili PRG)

- DEMO

Model – klijentska validacija

- Postoji i validacija na strani klijenta, koja je prema zadanim postavkama automatska
  - Međutim, da bi to funkcioniralo, moramo uključiti JavaScript koji radi tu automatizaciju

```
@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

  - MVC koristi posebnu JavaScript biblioteku koja se naziva "nenametljiva" (*unobtrusive*) provjera valjanosti
  - **Unobtrusive** ➔ Implementacija jednostavne validacije na strani klijenta bez potrebe zagađivanja vlastite implementacije dodatnim validacijskim provjerama
  - Drugim riječima, JS koristi iste atribute kao i serverska validacija u C# i koristi minimalistički pristup dok istovremeno maksimalno koristi HTML atribute
  - *RequiredAttribute*: data-val-required
  - *RangeAttribute*: data-val-range, data-val-range-min, data-val-range-max

- DEMO

Model i pogled (view)

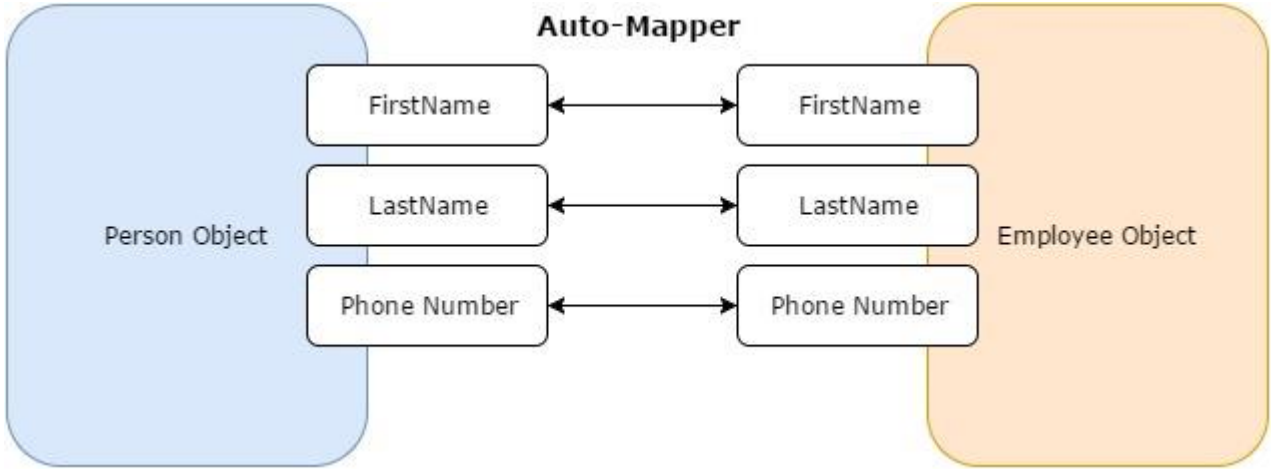
- Slanje viewmodela u view
  - Korištenjem ViewData
  - Korištenjem ViewBag
  - Korištenjem strogo tipiziranog view-a
  - Što će se dogoditi ako promijenimo dio modela?

- DEMO

Mapiranje modela

Mapiranje modela

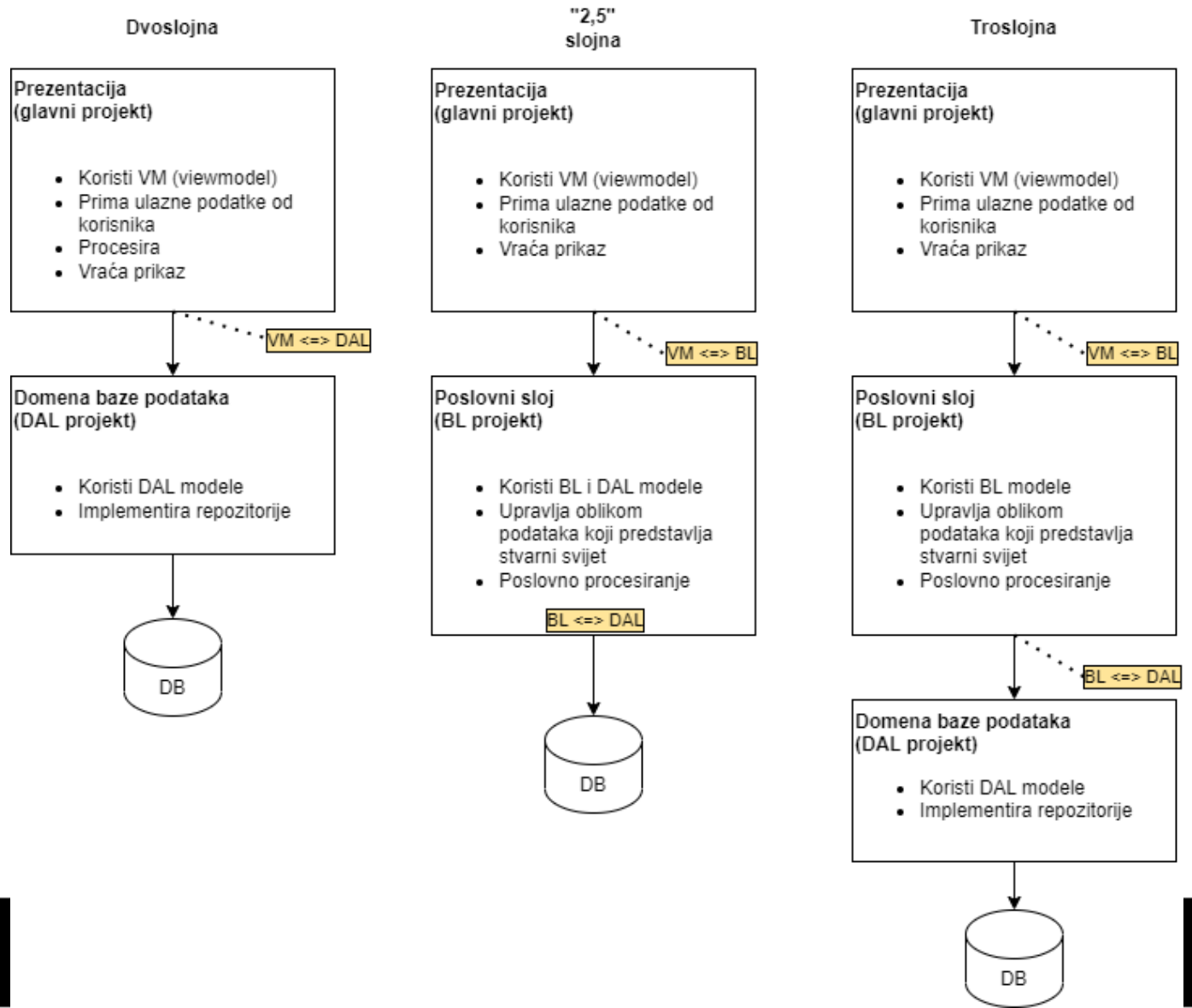
- Model možemo ručno mapirati iz jednog sloja u drugi (npr. viewmodel ➔ BL) ali to zahtijeva ručno kodiranje
- Puno modela ➔ puno ručnog kodiranja
- Automapper to može učiniti za nas



Automapper

- Što radi Automapper:
  - Na temelju konvencije i konfiguracije, koristi instancu objekta vrste A za stvaranje instance objekta vrste B
  - Konvencija: podaci su kopirani iz instance A u instancu B prema nazivima svojstava (A.FirstName ➔ B.FirstName)
  - Konfiguracija: A.Birthday ➔ mapiran u B.DOB
- Automapper treba instalaciju i konfiguraciju da bi omogućio mapiranje:
  - Instaliranje Automapper paketa (glavni projekt)
  - Kreiranje Automapper profila s mapiranjem (može biti glavni projekt, može biti i drugi projekt ako se koristi dvoslojna ili troslojna arhitektura)
  - Registrirajte dostupne Automapper profile u DI kontejneru (glavni projekt)
- Izvođenje stvarnog mapiranja (bilo koji projekt):
  - Koristite IMapper sučelje (interface)
  - Koristite metodu *Map<CiljaniTip>(izvornaInstanca)*
  - Ako mapirate kolekciju, koristite *Map<IEnumerable<CiljaniTip>>(izvornaKolekcija)*

Mapiranje modela u višeslojnoj arhitekturi



Hvala na pažnji!