

Tallinna Ülikool  
Informaatika Instituut

**Objektorienteeritud veebilahenduste loomise  
õppematerjal ASP.NET MVC raamistiku abil**

Seminaritöö

Autor: Raimo Virolainen  
Juhendaja: Jaagup Kippar

Autor:.....“ .....“2013  
Juhendaja:.....“ .....“2013  
Instituudi direktor:.....“ .....“2013

Tallinn 2013

## **Autorideklaratsioon**

Kinnitan, et käesolev seminaritöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud.

.....

(kuupäev)

.....

(allkiri)

## Sisukord

Sissejuhatus .....	4
1. Olemasolevad materjalid .....	5
2. ASP.NET MVC tutvustus .....	7
3. ASP.NET MVC alternatiivid.....	8
3.1 ASP.NET Web Forms.....	8
3.2 Codeigniter.....	9
4. Struktuuri analüüs.....	10
5. Andmete kasutamine rakenduses .....	12
6. ASP.NET MVC vahendite analüüs .....	13
6.1 Marsruutimine.....	15
6.2 Model binding.....	16
6.3 Valideerimine.....	17
6.4 Optimeerimine.....	19
6.4.1 Puhverdamine .....	19
6.4.2 Bundling ja minification.....	20
7. Mis osutus raskeks? .....	22
8. Õppematerjali kasutamine .....	24
9. ASP.NET MVC tugevused ja nõrkused .....	25
10. Millal kasutada .....	27
11. Kokkuvõte .....	28

## Sissejuhatus

Käesoleva seminaritöö eesmärgiks on tutvustada ASP.NET MVC raamistikku ja tema võimalusi. Seminaritöö käigus luuakse näidisrakendus, mille abil see saavutatakse.

Seminaritöö teema valik tulenes autori huvi objektorienteeritud programmeerimise ja *Model-View-Controller* mustri vastu. MVC mustri tundmine on tänapäeva veebiarendaja jaoks oluline. Samuti leidub väga vähe eestikeelset materjali ASP.NET MVC raamistiku kohta. Õppematerjali loomisel on oluline, et loodud õppematerjal oleks loetav ja arusaadav algaja tasemel.

Autor tahaks mainida, et loodud õppematerjal ei anta ülevaadet kõikidest ASP.NET MVC raamistiku võimalustest ega eesmärgiks ei ole luua ka täielikult valmis rakendus, vaid pigem anda ülevaade olulistemast ASP.NET võimalustest ja kuidas nende kasutamine aitab rakendusi luua ja kuidas nende kasutamine suuremates projektides kasulikuks võib osutuda.

Siinne õppematerjal on koostatud ASP.NET MVC raamistiku neljandal versioonil. Õppematerjali läbimise eeldusteks on installeeritud Microsoft Visual Studio 2012 ning objektorienteeritud programmeerimise tundmine C# programmeerimiskeeles (Jaagup Kippar, 2011).

## 1. Olemasolevad materjalid

### 1. <http://asp.net/mvc>

ASP.NET MVC raamistiku ametlik leht. Seal on palju põhjalikke ingliskeelseid juhendeid. Seal on võimalik ASP.NET MVC osade kohta eraldi juhendeid leida. Autor toob välja kolm põhjalikumad rakenduse loomise juhendit.

#### 1) <http://www.asp.net/mvc/tutorials/mvc-4/getting-started-with-aspnet-mvc-4-and-visual-studio-2011/intro-to-aspnet-mvc-4>

Juhend on lihtne ja arusaadav ning algajatele mõeldud. Siinses seminaritöös koostatud õppematerjal on sarnane eelmainitud juhendiga. Erinevus on järjestuses, kus siinses töös on valideerimine rakendusse toodud enne andmebaasi andmete salvestamist. Samuti on autor põhjalikumalt kirjeldanud ning kasutanud ASP.NET MVC raamistiku kompositsiooni vahendit (ingl. k. *Layout feature*). Käesolev töö on ka mahukam. Lisatud on autentimise, turvalisuse ja optimeerimise juhendid.

#### 2) <http://www.asp.net/mvc/tutorials/mvc-music-store>

*Music Store* juhend on loodud ASP.NET MVC raamistiku kolmanda versiooniga, seega on vähesel määral aegunud. Juhendi ülesehitus on sarnane eelneva juhendiga, seega erinevused käesoleva õppematerjaliga on samad. Erinevus tuleb rakenduse funktsionaalsuses, kus *Music Store* juhendis luuakse e-poe tüüpi rakendus. Seega juhend on kasulik sama tüüpi veebirakenduste loomisel. Autori arvates on ka *Music Store* juhend raskemini loetav ning sobiks paremini lugejale, kes on juba tuttav ASP.NET MVC raamistikuga.

#### 3) <http://www.asp.net/mvc/tutorials/getting-started-with-ef-using-mvc>

Juhend on pigem mõeldud tutvustamiseks Entity raamistikku kui ASP.NET MVC raamistikku. Juhend on mõeldud õppematerjaliks keerulistemate andmemudelite loomiseks. Seal kasutatakse küll ära ASP.NET MVC raamistiku vahendeid, kuid on eelkõige mõeldud tutvustamiseks Entity raamistiku ja kuidas keeruliste andmemudelite pealt andmebaasi luua.

### 2. [http://www.w3schools.com/aspnet/mvc\\_intro.asp](http://www.w3schools.com/aspnet/mvc_intro.asp)

Väga minimalistlik juhend tutvustamaks ASP.NET MVC raamistiku komponente. Näited on toodud samuti MVC raamistiku kolmanda versiooni näitel, seega on mingil määral aegunud. Autori arvates on materjal algajate jaoks raskesti arusaadav ning kuna juhendis ei looda näidiskoodi ja kuna juhend on nõnda minimalistlik, siis ei anta ASP.NET MVC raamistikust ja tema võimalustest piisavalt head ülevaadet.

3. <http://www.codeproject.com/Articles/468408/ASP-NET-MVC-4-Setup-Environment>

Tegemist on väga põhjaliku sissejuhatava osaga ASP.NET MVC raamistikku. Juhendis on palju teooriat, kuid väga vähe praktilisi näiteid. Juhend ei anna täielikku ülevaadet MVC raamistikust. Juhend annab hea ülevaate MVC projekti struktuurist, kuid MVC raamistiku vahendite ja nende kasutamise kohta ei ole juhendis piisavalt informatsiooni. Autori arvates sobib materjal pigem sissejuhatavaks, illustreerivaks ja teoreetiliseks näiteks mingi muu materjali kõrvale.

## 2. ASP.NET MVC tutvustus

ASP.NET MVC on üks kolmest arendusmeetoditest ASP.NET veebirakenduste loomiseks. ASP.NET MVC on Microsofti poolt loodud avatud lähtekoodiga raamistik veebirakenduste arendamiseks, kasutades MVC (*Model View Controller*) arhitektuuri mustrit. ASP.NET MVC on alternatiiviks traditsioonilisele ASP.NET Web Forms mustrile. Raamistik on kergekaaluline ja kergesti testitav ning on ehitatud ASP.NET raamistiku peale. ASP.NET MVC raamistiku esimene versioon tuli välja 2007 aasta lõpus ning 17. oktoobril 2013. aastal tuli välja viies versioon raamistikust. Käesolev töö on koostatud kasutades neljandat versiooni.

MVC arhitektuuri muster jaotab rakenduse kolmeks osaks:

1. Model: On rakenduse osa, kus kirjeldatakse loogikat rakenduses kasutatavate andmete kohta. Näiteks õppematerjalis loodud ürituste mudel sisaldab ürituse omaduste defineerimist andes omadustele tüüpi ning kirjeldades kuvamise loogikat ja valideerimise loogikat. Samuti sisaldab andmemudel andmetele ligipääsu loogikat näiteks kuidas andmebaasiga suhelda.
2. View: Vaated on rakenduse osad, mis vastutavad kasutajaliidese kuvamise eest. Vaated kuvavad rakenduses kasutatavaid andmeid kasutajatele ning sisaldavad HTML koodi, mille abil kasutajaliides esitatakse.
3. Controller: Kontrollor on rakenduse osa, mis käsitleb suhtlust kasutajaga. Samuti kuidas manipuleerida andmemudelit ning milliseid vaateid kuvada. Ehk siis rakenduses on kontrollor see, mis juhib rakenduse käitumist. Kui vaade on see, mis kasutajatele kuvatakse, siis kontrollor on see, kelle poole kasutaja pöördub, et vaadet kuvada.

Põhimõtte mustri kasutamise taga on, et rakenduse jagamine osadeks vähendab keerukust (ingl. k. „*Separation of concerns*“). See tähendab, et me saame MVC komponente arendada ja testida üksteisest eraldi. (Microsoft ASP.NET Team, 2009)

### 3. ASP.NET MVC alternatiivid

Nagu eelpool mainitud on ASP.NET MVC üks kolmest ASP.NET platvormil veebirakenduste loomiseks mõeldud raamistik. Teised on ASP.NET Web Pages ja ASP.NET Web Forms, mida saab lugeda ASP.NET MVC alternatiivideks. ASP.NET MVC raamistik on 2007. aasta lõpus loodud ning on avatud lähtekoodiga alates 2012. aastast. Alternatiiviks ASP.NET MVC raamistikule on ka Monorails raamistik. Monorails on varasem avatud lähtekoodiga MVC raamistik .NET veebirakenduste loomiseks, mida on arendatud alates 2003. aastast.

MVC arhitektuuri mustrit kasutatakse ka teistes programmeerimis keeltes ja ka neid võib lugeda ASP.NET MVC raamistiku alternatiivideks. PHP MVC raamistikudest võib mainida järgmisi: Codeigniter, CakePHP, Yii, Symphony jt. Käesolevas töös kasutab autor võrdluste toomiseks ASP.NET Web Forms ja Codeigniter raamistikke.

#### 3.1 ASP.NET Web Forms

Web Forms all mõistetakse veebilehekülgi, mille poole kasutajad veebilehitsejast pöörduvad. Need veebileheküljed on kirjutatud kombineerides HTML-i, serveri komponente ja serveri koodi.

Veeb töötab HTTP protokollil peal, mis on olekuvaba protokoll. See tähendab, et iga päring serveri poole on sõltumatu eelnevatest päringutest. Rakendus ootab kasutaja sisendit ning kasutaja iga sisestus ja interaktsioon käituvad uue päringuna (*GET/POST*) serveri poole. ASP.NET Web Forms üritas probleemi lahendada simuleerides olekuga mudelit veebiarenduses. Seda tehti kasutades kontseptsioone nagu *self postback* (saata vormi andmeid samale lehele) ning *ViewState* (säilitab väärtused *postback*-i ajal). (Marla Suresh, 2013)



## **3.2 Codeigniter**

Codeigniter on PHP raamistik, mis kasutab MVC arhitektuuri mustrit. Samuti on Codeigniteris lisatud kaasa kasulikke teeke, abivahendeid ja muid ressursse, mis muudavad koodi kirjutamise lihtsamaks ja kiiremaks. Codeigniter on avatud lähtekoodiga ning kergekaaluline (vähemahuline).

## 4. Struktuuri analüüs

ASP.NET MVC ja ASP.NET Web Forms raamistikud on mõlemad ehitatud ASP.NET raamistiku ehk System.Web nimeruumi otsa. Samuti kasutavad mõlemad IIS-it (Internet Information Server) ja ASP.NET päringuliini (ingl. k. *request pipeline*). Kuna nad mõlemad kasutavad samu ASP.NET mooduleid ja muid komponente (näiteks konfiguratsioon), siis nende struktuur on sarnane. (Scott Vandervort, 2011)

Traditsioonilises ASP.NET Web Forms veebirakenduses viidatakse URL-iga ühele kindlale failile serveris. Need failid on üldjuhul lainediga .aspx ja sisaldavad HTML-i ning koodi, mida käivitatakse päringu tegemisel. ASP.NET MVC raamistik murrab seose füüsilise faili ja URL-i vahel. MVC raamistik kaardistab URL-id selliselt, et nad viitavad kontrolleri klassidele. Kontrollerid klassid käsitlevad sissetulevaid kasutaja poolt tehtavaid päringuid, täidavad klassi kirjapandud loogikat ning esitavad väljundina üldjuhul vaate.

Kõige põhilisemaks erinevuseks kahe raamistiku vahel ongi MVC mustri põhimõttes. Kui Web Forms raamistikus on HTML ja rakenduse kood kombineeritud ühte faili, mis päringut tehes kasutajale kuvatakse, siis MVC raamistikus on see jagatud kihtide (mudel, vaade ja kontroller) vahel. Web Forms raamistikus on küll arendaja jaoks kood eraldi failis (.aspx fail sisaldab .aspx.cs klassi ning .aspx.designer.cs klassi), kuid serveri jaoks on kogu kood ühes failis. Samamoodi peab kogu failiga käiv kood olema .aspx.cs klassis, mis omakord võib põhjustada pikka ja keerulist koodi suuremate projektide korral.

Erinevused Web Forms ja MVC raamistiku vahel esinevad ka vaadete loomisel. Web Forms raamistikus on võimalik kasutada ainult Web Forms süntaksit koodi kirjutamisel HTML-is. Seega kood peab olema HTML-is märkide „<%“ ja „%>“ vahel. MVC raamistik toetab erinevate vaatemootorite (ingl. k. *view engine*) kasutamist, seega saab kasutada erinevaid süntakse. MVC raamistikus saab kasutada ka Web Forms süntaksi, kuid vaikimis kasutatakse Razor süntaksit. Lisaks selle saab vajadusel lisada ka teiste vaatemootorite kasutamist, millest populaarsemad on Spark ja Nhaml.

Õppematerjali koostades selgus, et ASP.NET MVC projektis on „Controllers“ ja „Models“ kaustad konventsiooniks, kuhu kontrollreid ja mudeleid luuakse. Mudelid ja kontrollerid ei

pea paiknema nendes kaustades. Kõik kontrollid ja mudelid, mis asuvad vastavalt projektinimi.Controllers ja projektinimi.Models nimeruumi sees saab projekti raames kasutada. Saab ka kasutada teiste projektide kontrollereid ja mudeleid lisades „Reference“ kausta viite projektist.

ASP.NET MVC raamistikul on erinevus MVC mustri kasutamisel võrreldes CodeIgniter raamistikuga. Kui ASP.NET MVC raamistikus on mudel klass, kus defineerime andmeid mida rakenduses kasutatakse, siis CodeIgniter-is vastutab mudel ka andmebaasiga suhtlemise eest. ASP.NET MVC raamistikus toimub andmete andmebaasist pärimine ja salvestamine kontrollis, kuid CodeIgniter raamistikus toimub sama tegevus mudelis ning kontrolli kaudu laetakse mudel. CodeIgniter raamistikus peavad üldjuhul kõik kontrollid ja mudelid asuma vastavates kaustades.

## 5. Andmete kasutamine rakenduses

Siinses õppematerjalis kasutatakse rakenduses andmete salvestamiseks Entity nimelist raamistiku. Entity raamistik võimaldab arendajatel luua andmebaase andmete põhjal, erinevalt traditsioonilisest andmete töötlemisest andmebaasi skeemiga sobitamiseks. Entity raamistik toetab andmebaas enne (ingl. k. *database first*) kui ka kood enne lähenemist (ingl. k. *code first*) andmete kasutamisel. Eesmärk on vähendada arendajatel andmetele juurdepääsuga ja korrashoiuga seotud koodi kirjutamist andmetele-orienteeritud rakendustes. Käesolevas õppematerjalis kasutatakse kood enne lähenemist. Kuna Entity raamistik on .NET raamistik komponent, siis nii MVC kui ka Web Forms raamistikud saavad Entity raamistiku kasutada.

Näide andmebaas enne lähenemisest Web Forms raamistikus. (Jana Abhijit, 2010)

Näide kood enne lähenemisest andmebaasi loomisel MVC raamistikus. (Tom Dykstra, 2013)

## 6. ASP.NET MVC vahendite analüüs

Õppematerjali loomisel kasutatakse ära erinevaid vahendeid, mida .NET raamistik ja tema laiend ASP.NET MVC raamistik pakuvad. ASP.NET Web Forms raamistik toetab serveripoolseid komponente (ingl. k. *server-side controls*), mida saab projektile lisada ning mis vähendavad HTML koodi kirjutamist. Nende kasutamine aitab kaasa rakenduste kiirele arendamisele, ilma et arendajat peaks muretsema genereeritava HTML-i pärast. Arendajatel on vähem kontrolli HTML-i üle, mis võib osutada problemaatiliseks kliendipoolsete tehnoloogiate (JavaScript, JQuery jne) kasutamisel. ASP.NET MVC raamistiku üheks tugevuseks on täielik kontroll HTML koodi üle, mis kasutajatele kuvatakse. See ei tähenda, et kogu HTML tuleb käsitsi kirjutada. MVC raamistikus on erinevaid abilisi, mis vähendavad vajamineva koodi kirjutamist ning lihtsustavad koodi lugemist. (Marla Sukesh, 2013)

Üheks kasulikuks abiliseks MVC raamistikus on *Html.ActionLink* abiline, mille abil saab genereerida linke rakenduses. Kuna MVC raamistikus URL-iga ei viidata konkreetse faili peale, siis ei saa kasutada traditsioonilist linki raamistikus, vaid tuleb viidata kontrolleri ja meetodile. *Html.ActionLink* abiline kasutab marsruutimisreegleid, et sobiv link rakenduses genereerida.

```
<!-- Traditsiooniline link HTML-is, mis MVC rakenduses ei toimi -->
  <a href="~/Views/Events/Create.cshtml">Loo uus üritus</a>

  <!-- Korrektne viis kuidas MVC rakenduses linki luua, kus Events on kontrolleri ja
  Create on kontrolleri meetod -->
  <a href="Events/Create">Loo uus üritus</a>

  <!-- Lingi genereerimine kasutades Html.ActionLink abilist, kus Create on
  kontrolleri meetod ja Events on kontrolleri -->
  @Html.ActionLink("Loo uus üritus", "Create", "Events")
```

### Koodinäide 1. *Html.ActionLink* abiline

Järgmiseks on MVC raamistikus mitu kasulikku HTML vormiga seotud abilist. Põhiliselt on nende ülesanneteks vormi loomine ning vormi andmete kuvamine rakenduses. Vaatame kõigepealt traditsioonilist Web Forms raamistiku vormi.

```

<form id="form1" runat="server">
    <label for="Text1">Nimi: </label>
    <input id="Text1" type="text" />
    <asp:Button id="button1" Text="Sisesta" runat="server" OnClick="submit" />
</form>

```

### Koodinäide 2. HTML vorm Web Forms raamistikus

Vormis defineeritakse väljatüübid ja nimed vastavalt sellele, milliseid andmeid rakendus ootab. MVC raamistikus defineeritakse andmemudel ning väljatüübid luuakse andmemudeli põhjal.

```

@model ProjektiNimi.Models.MudeliNimi
@using (Html.BeginForm()) {
    @Html.ValidationSummary(true)
    <fieldset>

        <div class="editor-label">
            @Html.LabelFor(model => model.Nimi)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Nimi)
            @Html.ValidationMessageFor(model => model.Nimi)

        <p>
            <input type="submit" value="Sisesta" />
        </p>
    </fieldset>
}

```

### Koodinäide 3. HTML abilised MVC raamistikus

Tegemist on tüüpilise vormi esitamisega vaates. Oluline on, et eelnevalt oleks loodud andmemudel, mida vormi esitamiseks kasutatakse ning vaade tuleb öelda millist mudelit kasutatakse. Vormi loomiseks kasutatakse *Html.BeginForm* abilist ning kõik mis jääb abilise sisse kuvatakse kasutajale HTML *form* märgendi sees. *Html.ValidationSummary* abilisega lisatakse vormile valideerimine ning valideerimisel tekkinud veateateid kuvatakse *Html.ValidationMessageFor* abilisega. *Html.LabelFor* ja *Html.EditorFor* abilised vaatavad üle mudeli ning leiavad parima viisi, kuidas vastavalt andmete nime ja väljatüüpi esitada. Selleks kasutatakse mudelis andmete annotatsiooni väljadele nime ja tüübi määramiseks. Ka valideerimist saab andmete annotatsiooniga paika panna.

Kasutades Web Forms raamistikus serveripoolseid komponente saab sama loogikat ka Web Forms rakenduses luua, kuid sellega on väiksem kontroll genereeritava HTML koodi üle. (Erik Reitan, 2012)

Need on ainult mõned abilised, mida MVC raamistikus saab kasutada. HTML kuvamisega seotud abilisi võib leida System.Web.Mvc.Html nimeruumist

## 6.1 Marsruutimine

Marsruutimine on üheks oluliseimaks vahendiks MVC raamistikus. Eelnevalt mainis autor, et MVC raamistikus ei viita URL-id konkreetsele failile serveris. Marsruutimissüsteem selle saavutabki. Marsruutimissüsteem defineerib kuidas rakenduses URL-e kaardistatakse. Marsruudid suunavad kasutajaid soovitud kontrollereite ja kontrolleri meetodi poole. MVC raamistikus defineeritakse meile vaikimisi marsruut.

```
routes.MapRoute(  
    name: "Default",  
    url: "{controller}/{action}/{id}",  
    defaults: new { controller = "Home", action = "Index", id =  
        UrlParameter.Optional }  
);
```

### Koodinäide 4. Vaikimisi marsruut

Marsruudil on kolm atribuuti ehk siis nime, mustri ja vaikimisi väärtuste atribuudid (*name*, *url* ja *defaults*). *Name* atribuudiga määratakse marsruudi nime. *Url* atribuudiga defineerime URL mustri, mis võib sisaldada erinevaid sõnalise väärtuse ja parameetri kohahoidja kombinatsioone. Muutuvaid väärtusi hoitakse URL mustri defineerimisel loogisulgude sees. *Defaults* atribuudiga annab parameetritele vaikimisi väärtused. MVC raamistiku marsruutimissüsteem on võimekas ja lubab vastavalt vajadusele erinevaid marsruute defineerida. Mida suuremaks ja keerulisemaks rakendused muutuvad seda keerulisemaid marsruute on vaja defineerida.

Marsruutide defineerimisel on oluline jälgida marsruutide järjekorda. Sissetulevat URL sobitatakse marsruutidega ning kasutatakse esimest marsruuti, mille muster sobib sissetuleva URL-iga. Ehk teisisõnu kui sisestatud URL sobib kahe arendaja poolt defineeritud marsruudi mustriga, siis kasutatakse eespool defineeritud marsruuti. (John Atten, 2013)

Lisaks sellele, et marsruutimissüsteem muudab URL-id lihtsamini loetavaks ja kasutatavamaks, muudab ka otsingumootorite jaoks paremini leitavaks. URL-ide optimeerimine on üheks osaks SEO-st (ingl. k. *Search Engine Optimization*), mille abil saab veebilehete muuta otsingumootorites paremini leitavaks. Marsruutid loomiselt tasub

tähelepanu pöörata ka SEO põhimõtetele. Sinn Käesolevas õppematerjalis loodud marsruutide kohta võiks tuua välja kaks SEO põhimõtet. Tähelepanu tuleks pöörata URL struktuurile. Mida lühemad on URL-id seda parem, kasutada loetavaid sõnu ja pigem vältida pikkade identifikaatorite kasutamist. Kirjavahemärkide kasutamisel URL-is kasutada pigem sidekriipsu alljoone asemel sõnade eristamiseks. (Google, Url Structure)

MVC raamistikus on võimalik luua ka marsruute, mida ignoreeritakse. Näiteks kui rakenduses esineb mõne muu programmeerimiskeele koodi, mida .NET käitus (ingl. k. *runtime*) ei oska kasutada, siis tasuks marsruudiga ignoreerida antud failide poole pöördumist URL-i kaudu.

Alates ASP.NET raamistiku neljandast versioonist on Web Forms raamistikus ka võimalik marsruute luua. Võimalik on arendaja vajadustele vastavaid URL-e kasutada, kuid URL-iga pöördutakse endiselt kindla faili poole serveris. (Microsoft MSDN)

## 6.2 Model binding

*Model binding* on ASP.NET MVC omadus, mis aitab andmeid päringutest kätte saada, täita automaatselt kontrolleri meetodite parameetreid ning kannab hoolt omaduste kaardistamise ja omaduste andmetüübi muutmise eest. MVC raamistikus on mitu viisi kuidas andmeid päringutest kätte saada. Esiteks on võimalik saada marsruudist andmeid kui meil on defineeritud marsruudis parameetri kohahoidja. Eelpool defineeritud *Default* nimelisest marsruudist on võimalik sisestatud URL-ist saada kätte *Id* parameeter. Andmeid on võimalik saada veel *QueryString* parameetritest ehk *Request.QueryString* kolleksioonist URL-ist.

<http://localhost/Events?Asukoht=Tallinn>

Näiteks antud URL-ist on võimalik saada asukoha parameetri väärtus „Tallinn“.

Ning veel üheks viisiks on andmeid vormist kätte saada POST meetodil ehk *Request.Form* kolleksioonist, mida rakendusse on vormist sisestatud.

Neid andmeid on võimalik kasutada meetodite parameetritena ning meetodite kaudu vajalikud andmed vaatesse esitada. (Jess Chadwick, 2012)



```
public ActionResult Details(long id)
{
    var Event = db.Events.find(id)
    return View(Event);
}
```

#### Koodinäide 5. Model binding URL-st saadud Id parameetriga

Antud näite puhul oodab meetod URL-st id parameetrit. Kuigi URL-ist saadakse parameeter *string* andmetüübina, siis *model binding* määrab parameetri *long* andmetüübiks nagu on nõutud. Sama on võimalik teha ka kompleksandmetüüpide korral, kus siis ASP.NET MVC raamistik automaatselt täidab ära kompleksandmetüübi väljad sobitades kokku päringust tulnud vastava väljade nimedega parameetrite väärtused.

```
public ActionResult Create(Models.Events Event)
{
    return View();
}
```

#### Koodinäide 6. Model binding komplekstüüpi andmemudeliga

Antud näites luuakse andmemudelid isend ning täidetakse isendi väljad sobitades kokku andmemudelis defineeritud väljad vormi samanimelistesse väljadesse sisestatud andmetega.

Õppematerjali tegemise käigus selgus, et kompleksandmetüüpide sidumine vormist saadud andmetega võib osutada problemaatiliseks kui mudelis on selliseid andmeid, millele saab väärtust anda, kuid millele kasutajad ei tohiks saada väärtust anda. Probleemi saab lahendada kasutades *Bind* atribuudi *Exclude* omadust, mille abil on võimalik objekti omadusi *model binding*-ust välja jätta.

## 6.3 Valideerimine

Mudelis saab määrata kuidas objekti omadusi kasutajatele esitatakse ning millist andmetüüpi omadused on. Seda kõike saab teha kasutades andme annotatsioone. Samuti saab ka valideerimise reeglid andme annotatsioonidega kirja panna. Valideerimise lisamiseks kasutatakse *Html.ValidationSummary* abilist, mis lisatakse vormile. Valideerimise vigade kuvamiseks vormis kasutatakse *Html.ValidationMessageFor* abilist. Andmete korrektsuse kontrollimiseks kasutatakse kontrolleriis *ModelState.IsValid* meetodit, mis kontrollib sisestatud andmete õigsust kasutades defineeritud mudelit. Õppematerjali tegemisel selgus, et valideerimise lisades tuleb jagada vastav kontrolleri meetod kaheks osaks. Esimene osa

tegeleb vormi kuvamisega ning teine osa kontrollib sisestatud andmeid. Seda saab saavutada MVC raamistikus kasutades *HttpGet* ja *HttpPost* atribuute.

```
[HttpGet]
public ActionResult Create()
{
    // Vormi kuvamine
    return View();
}

[HttpPost]
public ActionResult Create(models.Events Event)
{
    // Andmete valideerimine
    if (ModelState.IsValid)
    {
        //Andmebaasi salvestamine
        return RedirectToAction("Index");
    }
    return Create();
}
```

#### Koodinäide 7. Valideerimise lisamine Create meetodile

MVC rakenduses on võimalik implementeerida serveripoolset kui ka kliendipoolset valideerimist. Selleks, et kliendipoolset valideerimist kasutada tuleb rakenduses konfiguratsiooni failis lubada kliendipoolse valideerimise kasutamist, mis vaikimisi rakenduse luues ka on lubatud.

```
<appSettings>
  <add key="ClientValidationEnabled" value="true" />
  <add key="UnobtrusiveJavaScriptEnabled" value="true" />
</appSettings>
```

#### Koodinäide 8. Kliendipoolse valideerimise lubamine

Kliendipoolse valideerimise kasutamiseks tuleb vormi esitamise vaates lisada JQuery valideerimise teek.

Andme annotatsioonide abil on võimalik elementaarset valideerimist lisada. Võib esineda juhtumeid kui on vaja lisada valideerimist, mida ei ole andmeannotatsioonidega võimalik, siis MVC raamistikus on võimalik ka luua vastavalt vajadusele valideerimise reegleid, kasutades *ValidationAttribute* klassi. Siinses õppematerjalist kasutatakse ainult elementaarset valideerimist ning lihtsuse mõttes jäeti välja kohandatud valideerimisreeglite defineerimine. (Rick Anderson, 2012)

ASP.NET Web Forms raamistikus on võimalik samuti andmemudeleid luua ning ka valideerimist luua andmeannotatsioonide kasutamisega. Kuna CodeIgniter-i raamistikus rakenduses kasutatavaid andmeid mudelis selliselt ei defineerita, siis valideerimine toimub teisiti. CodeIgniter raamistikus on võimalik kasutada vormi valideerimise klassi ning tema kaudu defineerida valideerimise reeglid. (Ellislab, Form Validation)

## 6.4 Optimeerimine

Veebirakenduse optimeerimise eesmärgiks on tagada kasutajale parim kasutajakogemus (ingl. k. *UX* ehk „*User Experience*“). Siinses õppematerjalis kasutatakse rakenduse optimeerimiseks puhverdamise ning failide kokkuliitmise ja minimeerimise meetodeid (ingl. k. *bundling and minification*).

### 6.4.1 Puhverdamine

Puhverdamine on andmete salvestamine teatud perioodiks, et neid ei peaks serverist uuesti pärima. Puhverdamisega tagatakse, et andmete pärimine serverist toimiks võimalikult efektiivselt ehk üleliia päringuid serverisse ei tehtaks. Siinses õppematerjalis kasutatakse tervete veebilehtede puhverdamist ning ka veebilehe mingi kindla osa puhverdamist. Samuti kirjeldatakse kuhu puhver salvestatakse ning kuidas seda muuta. MVC raamistikus on puhverdamist lihtne lisada, kasutades *OutputCache* atribuuti. Paari koodireaga saab rakenduses puhverdamise määrata. Puhvrite kasutamisega tuleb olla ettevaatlik, enne kasutamist kindlasti ülemõelda kas ja kuidas andmeid puhverdada. Vale puhvri määramine võib oluliselt mõjutada rakenduse töökäiku. Näiteks kujutades ette olukorda, kus meil on kontrolleri meetod, mis kuvab sisselogitud kasutaja nime. Määrates kümneminutilise kestvusega puhvri ja puhvri salvestamise asukohaks serveri, siis kui esimene sisselogitud kasutaja meetodi poole pöördub, siis kuvatakse tema nimi ning järgmise sisselogitud kasutaja pöördumisel kuvatakse esimese kasutaja nimi kuna kasutatakse puhvrit. Samuti tasub mõelda rakenduse kasutajate koguse peale. Siinses õppematerjalis kasutatakse mikropuhverdamist. Mikropuhverdamine on andmete puhverdamine väga lühikeseks ajaks (üks sekund). Kui on rakendus, millele tehakse igal hetkel suur hulk päringuid, siis mikropuhverdamine võimaldab päringute arvu serverisse vähendada. Näiteks kujutades ette olukorda, kus kontrolleri meetod kuvab andmebaasist andmeid ning kui sada kasutajat pöörduvad üheaegselt meetodi poole,

siis tehaks andmebaasi ka sada päringut. Mikropuhvri määramisel meetodile saab päringute arvu vähendada ühe peale. Õppematerjalis loodud rakenduses ei ole otstarbekas mikropuhverdamist kasutada, kuna kui rakendusel on vähe kasutajaid, siis luuaks iga sekundi tagant uus puhver, mis ei ole vajalik. (Microsoft ASP.NET Team, 2009)

Puhvrite kasutamisel tuleb läbi mõelda kas ja kuidas on otstarbekas puhvrit kasutada. Puhvrite kasutamine on otstarbekas staatiliste ja vähemuutuvate andmete esitamiseks. Pidevalt muutuvate andmete korral ei pruugi olla puhverdamine kasulik, kuna kasutaja võib saada aegunud andmeid. Samuti kui rakenduses kuvatakse erinevatele kasutajatele erinevaid andmeid, siis ei ole arukas puhvrit serverisse salvestada, vaid puhver tuleks salvestada kliendi veebilehitsejasse.

#### 6.4.2 Bundling ja minification

Bundling ja minification meetodite eesmärgiks on tagada, et kliendi ja serveri vahel võimalikult vähe ja võimalikult väiksed andmed liiguks. Enamus peamisest veebilehitsejatest võimaldavad kuute kuni kaheksat üheaegset ühendust hostinimele ehk kui kuute kuni kaheksat päringut töödeltakse, siis ülejäänud lisatakse järjekorda. Järelikult mida vähem faile kliendi ja serveri vahel liigub, seda kiiremini veebileht kasutajale esitatakse. *Bundling* vahendi kasutamine võimaldab mitu faili üheks failiks kombineerida ja *minification* võimaldab kasutada erinevaid koodi optimeerimisvahendeid muutes faili mahu võimalikult väikeks. Näiteks eemaldatakse failidest ebavajalikud tühemikud (ingl. k. *whitespace*), kommentaarid ning muudetakse muutjate nimed ühe tähemärgi pikkuseteks. Enamasti kasutatakse neid JavaScript-i ja CSS failide esitamisel rakenduses. Õppematerjali tegemisel selgus, et *bundling* ja *minification* vahendite kasutamine muudab märgatavalt veebilehe laadimise aega. Antud näide on toodud näidisrakenduse põhjal. (Rick Anderson, 2012)

Tabel 1. Veebilehe laadimise aja võrdlus

	Ilma bundling ja minificationita	Kasutades bundling ja minification
Päringuid	17	10
Andmemah	1126,4 KB	188,0 KB

Laadimise aeg	1700 ms	801 ms
---------------	---------	--------

*Bundling* ja *minification* vahendid paiknevad System.Web.Optimization nimeruumis ja on kasutatavad ka Web Forms raamistikus. CodeIgniter raamistikus selline võimalus puudub.

## 7. Mis osutus raskeks?

Õppematerjalis kasutatakse ära erinevaid MVC raamistiku abilisi lihtsustamaks koodi kirjutamist ja lugemist. Raskeks osutus leida milliseid parameetreid ja kuidas nendele abilistele saab anda. Nende kohta võib küll leida definitsiooni ja süntaksi, kuid näiteid nende kasutamisest MVC raamistiku ametlikul lehel puuduvad. See võib algajatele raskeks osutada. Sama kehtib ka teiste MVC raamistiku vahendite ja meetodite kohta.

Õppematerjalis loodud rakenduses kasutatakse ära modelbinding võimalust päringutest andmete kättesaamiseks. Nagu eelpool mainitud võib modelbinding kasutamine olla ohukohaks kompleksandmetüüpide korral kui mudelis on objektid millele saab väärtusi anda, kuid millele kasutajad ei tohiks väärtusi anda. Lahenduseks leiti *Bind* atribuudi *Exclude* omadus, mille abil saab objekte mudelist välja jätta. Raskeks muutus *Bind* atribuudi *Exclude* omaduse kasutamine kui pärast andmete sisestamist on vaja väljajäetud objekti kasutada.

```
public ActionResult Edit([Bind(Exclude="EventsCreator")]Events Event)
{
    if (ModelState.IsValid)
    {
        if(Event.EventCreator == User.Identity.Name){
            db.Entry(Event).State = EntityState.Modified;
            db.SaveChanges();
            return RedirectToAction("Index");
        }
    }
    return View(Event);
}
```

### Koodinäide 9. Soovitud kood muutmise funktsionaalsuse lisamiseks

Näiteks loodud rakenduses ürituse muutmisega seotud meetodis *Edit* oleks tarvis välja jätta ürituse looja objekt *EventsCreator*, kuid siis ei saa kontrollida kas autenditud kasutaja on ürituse loonud või mitte. Samuti kasutades *Bind* atribuudi *Exclude* omadust jäetakse küll objekt sisestatud andmetest välja, mis tähendab et objektile määratakse null väärtus. See tähendab, et ürituse muutmise meetodis *Edit* oodatakse tehtud muudatusi ja ei taheta et saaks muuta ürituse looja objekti *EventsCreator*, kuid pärast andmete postitamist muudetakse objekti väärtus nulliks, kuna kasutatakse *Bind* atribuudi *Exclude* omadust. Autor lahendas

probleemi eemaldades autentitud kasutaja ja ürituse looja kontrolli ja jättes kontrolli ainult muudatuse vormi kuvamisele ning andes muutmisel ürituse loojale uuesti väärtuse.

```
public ActionResult Edit([Bind(Exclude="EventsCreator")]Events Event)
{
    if (ModelState.IsValid)
    {
        db.Entry(Event).State = EntityState.Modified;
        Event.EventCreator = User.Identity.Name;
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(Event);
}
```

#### Koodinäide 10. Töötav muutmise meetodi kood

Kõige suuremaks probleemiks rakenduse loomisel osutus kuupäeva valideerimine. Täpsemini kuupäeva kliendipoolne valideerimine eesti kuupäeva formaadis ehk kuupäev.kuu.aasta. Selgus, et andmeannotatsiooniga kuupäevale lisatud formaat ei olnud mõeldud valideerimiseks, vaid kuupäevad kuvamiseks rakenduses. Serveripoolne valideerimine toimis korralikult, kuid kliendipoolne mitte. Eesti formaati sisestatud kuupäevad andsid valideermisel veateate. Autor lahendas probleemi lisades Globalize JavaScript-i teegi, mille abil eesti formaati kuupäevad valideeruvad. Siinkohal mainiks autor, et tegemist ei pruugi olla parima lahendusega probleemi lahendamiseks, kuid ainus lahendus, mis autoril õnnestus leida.

Üheks ohukohaks on asjaolu, et rakenduses kõik *public* tüüpi kontrolleri meetodid on URL-i kaudu väljakutsutavad. Autor leidis probleemile järgmised lahendused. Turvalisuse atribuutidega nagu *Authorize* on võimalik liigipääsu lubada ainult autentitud kasutajatele. URL-i kaudu meetodite väljakutsumist saab piirata kolmel viisil. Esiteks muuta kontrolleri meetod *private* tüüpi. Teiseks luua kontrolleri meetod tütarmeetodina (ingl. k. *child action*), mis tähendab, et meetod on väljakutsutav ainult vaadetest. Ning kolmandaks on marsruudiga ignoreerida sisestatud URL-i. Vastavalt vajadusele saab neid lahendusi kasutada piiramaks ligipääsu kontrolleri meetoditele.

## 8. Õppematerjali kasutamine

Siinses töös loodud õppematerjali kasutas autor 30. oktoobril 2013. aastal .NET raamistiku aine tudengite peal neljas tunnis. Nelja tunniga jõuti läbida umbes pool õppematerjalist. Täpsemini jõuti õppematerjalis ürituste muutmise ja kustutamise funktsionaalsuste lisamiseni. Enamus läbitud õppematerjalist tegi autor ka tunnis kaasa. Iseseisvalt lisasid tudengid rakendusele anonüümsete kasutajate ligipääsu piiramise, mis edukalt õnnestus. Kõige enam probleeme tekkis kontrollrite ja meetodite nimedega, kus siis URL-i kaudu kutsuti vale nimega kontrollereid, mida ei eksisteerinud ( ehk *Events* kontrolleri asemele pöörduiti *Event* kontrolleri poole). Tagasidest selgus, et õppematerjalis koodinäidete allkirjas võiks olla ka kaustanimi, kus fail asub, mida autor ka õppematerjalis muutis. Üldiselt oli õppematerjal loetav, kuid selgus et täpsemini arusaamiseks tuleks MVC raamistikku natukene kasutada. Samuti oli õppematerjali läbimise tempo liiga kiire, mis tähendab et käesoleva õppematerjali läbimiseks kuluks vähemalt kaheksa õppetundi.



## 9. ASP.NET MVC tugevused ja nõrkused

ASP.NET MVC raamistiku ehk MVC mustri kasutamine, mis jaotab loogika rakenduses kolme komponendi mudeli, vaate ja kontrolleri vahel, millest igüks realiseerib kindlat funktsionaalsust (ingl. k. *sepration of concerns*). See vähendab rakenduse keerukust. Rakenduse keerukus ei suurene projekti suurenedes. Sama ei saa öelda Web Forms raamistiku projekti kohta, kus kogu veebilehega kooskäiv kood on ühes failis.

Lisaks sellele, et MVC muster vähendab keerukust muudab MVC mustri kasutamine ka testimist lihtsamaks, võrreldes Web Forms raamistikuga. Näiteks Web Forms raamistiku rakenduses kasutatakse ühte klassi nii väljundi kuvamiseks kui ka kasutajalt sisendi saamiseks. Automaattestide kirjutamine Web Forms rakendusele võib osutuda keeruliseks. MVC raamistikus seevastu on rakenduse loogika jaotatud MVC komponentide vahel, mis võimaldab komponente eraldi testida. Kontrollerid ei ole seotud ühegi vaatega, mis tähendab et neid saab korduvkasutada ja testida. Käesolevas õppematerjalis testimist ei kasutata. (Dino Esposito, 2009)

ASP.NET MVC raamistik ei toeta serveripoolseid komponente (ingl. k. *server controls*), mis küll võib aeglustada rakenduse arendamist võrreldes Web Forms raamistikuga, kuid annab täieliku kontrolli kasutajale genereeritava HTML-i üle. Web Forms raamistikus kasutatakse *ViewState* vahendit, millega saavutatakse olekulisus. See saavutatakse lisades igale vormile *hidden* tüüpi välja *ViewState*, milles hoitakse kõikide vormi elementide viimaseid väärtusi. See võib osutuda väga mahukaks, mis vähendab veebilehtede laadimise aega. MVC raamistik *ViewState* vahendit ei toeta, mis vähendab veebilehtede suurust ja seega nende laadimise aega. *ViewState* vahendit on võimalik MVC raamistikus simuleerida *TempData* meetodiga, mis hoiab väärtusi serveris ühe päringu vältel. (Marla Suresh, 2013)

ASP.NET MVC raamistiku marsruutimissüsteem võimaldab rakenduses kasutada vajadusele vastavaid URL-e, muutes need ka otsingumootorite jaoks sõbralikeks.

MVC raamistiku nõrkuseks on rohkema programmeerimiskogemuse vajadus. Seoses serveripoolsete komponentide puudumisega tuleb rohkem koodi kirjutada. Lihtsamate

rakenduste loomine MVC raamistikus võtab üldjuhul kauem aega ja vajab rohkem koodi kirjutamist kui Web Forms raamistikus.

## 10. Millal kasutada

Valides ASP.NET raamistiku ja PHP raamistiku kasutamise vahel, siis üldjuhul tehakse valik seoses keskkonnaga, kus veebirakendus tööle hakkab. Kui server kasutab Linux operatsioonisüsteemi ja Apache veebiserverit, siis üldjuhul on kasulikum kasutada PHP raamistiku. ASP.NET MVC raamistikku on samuti võimalik Linux-i serveris tööle panna, kuid vajab vajalikke moodulite lisamist. Kui võrrelda konkreetselt CodeIgniter-i raamistikuga, siis CodeIgniter on väga väiksemahuline (võtab serveris kõigest ~4 MB ruumi). MVC raamistiku on mõistlik kasutada kui server, kus rakendus tööle hakkab, jookseb Windows operatsiooni süsteemil ning kasutab IIS serverit.

Valides ASP.NET MVC ja Web Forms raamistike vahel tuleks mõelda järgmistele faktoritele. Esiteks tuleks arvestada projekti mahukust. Väikeste projektide puhul võib olla kasulikum kasutada Web Forms raamistikku, kuna serveripoolsed komponentid muudavad rakenduse arendamise lihtsaks ja kiireks ning nõuab vähem programmeerimisoskusi. Suuremate projektide puhul oleks arukam kasutada MVC raamistikku, kuna MVC muster aitab vähendada rakenduse keerukust. Järgmisena kui testimine on vajalik, siis MVC raamistiku kasutamine on kasulikum, kuna testimist on MVC raamistikus kergem teostada. Samuti meeskonna kasutamine rakenduse arendamisel on MVC raamistiku kasuks, kuna MVC komponente on võimalik arendada ja testida üksteistest eraldi. Viimaseks võib välja tuua JavaScript-ide kasutamine. Mida rohkem JavaScript-i kasutatakse, seda otstarbekam oleks kasutada MVC raamistikku, kuna on olemas täielik kontroll genereeritava HTML koodi üle, mis lihtsustab JavaScriptid-e kasutamist. (Marla Suresh, 2013)

## **Kokkuvõte**

Käesoleva seminaritöö eesmärgiks oli koostada ASP.NET MVC raamistiku tutvustav õppematerjal, mis on mõeldud algajatele, kellel puudub varasem kokkupuude ASP.NET MVC raamistikuga. Õppematerjali käigus loodi rakendus tutvustamiseks ASP.NET MVC võimalusi ning kuidas nende abil erinevaid funktsionaalsusi on võimalik realiseerida. Autori jaoks kõige huvitavamateks võimalusteks osutusid andmete andmebaasi salvestamiseks mõeldud Entity raamistiku kasutamine ning veebirakenduse optimeerimise vahendid. Analüüsi tulemusena selgitati välja ASP.NET MVC raamistiku tugevused ja nõrkused ning millal oleks kõige mõistlikum ASP.NET MVC raamistiku kasutada.

## Kasutatud materjalid

Jana Abhijit (3. detsember 2010. a.) The Entity Framework 4.0 and ASP.NET. Kasutamise kuupäev: 30. oktoober 2013. a. Allikas: asp.net/web-forms: <http://www.asp.net/web-forms/tutorials/getting-started-with-ef>

Rick Anderson (28. august 2012. a.) Adding Validation to the Model. Kasutamise kuupäev: 15. oktoober 2013. a. Allikas: asp.net/mvc: <http://www.asp.net/mvc/tutorials/mvc-4/getting-started-with-aspnet-mvc4/adding-validation-to-the-model>

Rick Anderson (23. august 2012. a.) Bundling and Minification. Kasutamise kuupäev: 14. oktoober 2013. a. Allikas: asp.net/mvc: <http://www.asp.net/mvc/tutorials/mvc-4/bundling-and-minification>

John Atten (22. august 2013. a.) Customizing routes in ASP.NET MVC. Kasutamise kuupäev: 31. oktoober 2013. a. Allikas: codeproject.com: <http://www.codeproject.com/Articles/641783/Customizing-Routes-in-ASP-NET-MVC>

Jess Chadwick (Veebruar 2012. a.) The Features and Foibles of ASP.NET MVC Model Binding. Kasutamise kuupäev: 31. oktoober 2013. a. Allikas: MSDN Magazine: <http://msdn.microsoft.com/en-us/magazine/hh781022.aspx>

Tom Dykstra (30. juuli 2013. a.) Creating an Entity Framework Data Model for an ASP.NET MVC Application. Kasutamise kuupäev: 30. oktoober 2013. a. Allikas: asp.net/mvc: <http://www.asp.net/mvc/tutorials/getting-started-with-ef-5-using-mvc-4/creating-an-entity-framework-data-model-for-an-asp-net-mvc-application>

Ellislab Inc. CodeIgniter user-guide: Form Validation. Kasutamise kuupäev: 31. oktoober 2013. a. Allikas: ellislab.com: [http://ellislab.com/codeigniter%20user-guide/libraries/form\\_validation.html](http://ellislab.com/codeigniter%20user-guide/libraries/form_validation.html)

Dino Esposito (Juuli 2009. a.) Comparing Web Forms and ASP.NET MVC. Kasutamise kuupäev: 31. oktoober 2013. a. Allikas: MSDN Magazine: <http://msdn.microsoft.com/en-us/magazine/dd942833.aspx>

Google. Url Structure. Kasutamise kuupäev: 11. oktoober 2013. a. Allikas support.google.com: <https://support.google.com/webmasters/answer/76329>

Jaagup Kippar (2011). Andmebaasipõhiste veebirakenduste arendamine Microsoft Visual Studio ja SQL server'i baasil – C#. Allikas: tlu.ee: <http://minitorn.tlu.ee/~jaagup/kool/java/loeng/dotnet/ctrell.docx>

Microsoft ASP.NET Team (27. jaanuar 2009. a.) ASP.NET MVC Overview. Kasutamise kuupäev: 1. November 2013. aasta. Allikas asp.net/mvc: <http://www.asp.net/mvc/tutorials/older-versions/overview/asp-net-mvc-overview>

Microsoft ASP.NET Team (27. jaanuar 2009. a.) Improving Performance with Output Caching(C#). Kasutamise kuupäev: 25. oktoober. 2013. a. Allikas asp.net/mvc: <http://www.asp.net/mvc/tutorials/older-versions/controllers-and-routing/improving-performance-with-output-caching-cs>

Microsoft MSDN. Walkthrough: Using ASP.NET Routing in a Web Forms Application. Kasutamise kuupäev: 30. oktoober 2013. a. Allikas: msdn.microsoft.com: <http://msdn.microsoft.com/en-us/library/dd329551.ASPX>

Erik Reitan (14. august 2012. a.) Getting Started with ASP.NET 4.5 Web Forms. Kasutamise kuupäev: 31. oktoober 2013. a. Allikas: asp.net/web-forms: (<http://www.asp.net/web-forms/tutorials/aspnet-45/getting-started-with-aspnet-45-web-forms/introduction-and-overview>)

Marla Sukesh (7. Veebruar 2013. a.) WebForms vs. MVC. Kasutamise kuupäev: 30. oktoober 2013. a. Allikas: Codeproject.com: <http://www.codeproject.com/Articles/528117/WebForms-vs-MVC>

Scott Vandervort (11. mai 2011. a.) ASP.NET MVC versus Web Forms Smackdown. Kasutamise kuupäev: 31. oktoober 2013. a. Allikas: blogspot.com: <http://scottsjewels.blogspot.com/2011/05/aspnet-mvc-versus-web-forms-smackdown.html>

# **L I S A**

**ASP.NET MVC raamistiku õppematerjal**

## Sisukord

Sissejuhatus ASP.NET MVC-sse .....	3
1. Projekti loomine .....	4
1.1 MVC veebirakenduse struktuur .....	4
1.2 Projekti loomine kasutades malli .....	7
2. Näidise rakenduse loomine .....	9
2.1 Mudeli loomine .....	9
2.2 Marsruutimine .....	9
2.3 Razor .....	11
2.4 Mudeliga suhtlemine .....	12
2.5 Kasutajate suhtlus veebirakendusega .....	14
2.6 Model binding .....	16
2.7 Ülesanne 1 .....	18
3. Rakendusele funktsionaalsuste lisamine .....	19
3.1 Valideerimine .....	19
3.2 Andmebaasi salvestamine .....	22
3.3 Andmebaasist andmete kuvamine .....	24
3.4 Andmemudeli muutmine ja andmebaasi uuendamine .....	26
3.5 Uue marsruudi loomine .....	29
3.6 Turvalisus ja autentimine .....	30
3.7 Ürituse muutmise ja kustutamise funktsionaalsuse lisamine .....	34
3.8 Cross-Site Request Forgery .....	38
3.9 Üritusel osalemise funktsionaalsus .....	39
3.10 Partial tüüpi vaate .....	42



3.11	Terviklik kujundus ja paigutus.....	45
3.11.1	Layout loomine.....	47
3.12	Veebirakenduse optimiseerimine.....	50
3.12.1	Puhverdamine.....	50
3.12.2	Bundling ja minification.....	54
3.13	Kuupäeva valideerimine.....	57

## Sissejuhatus ASP.NET MVC-sse

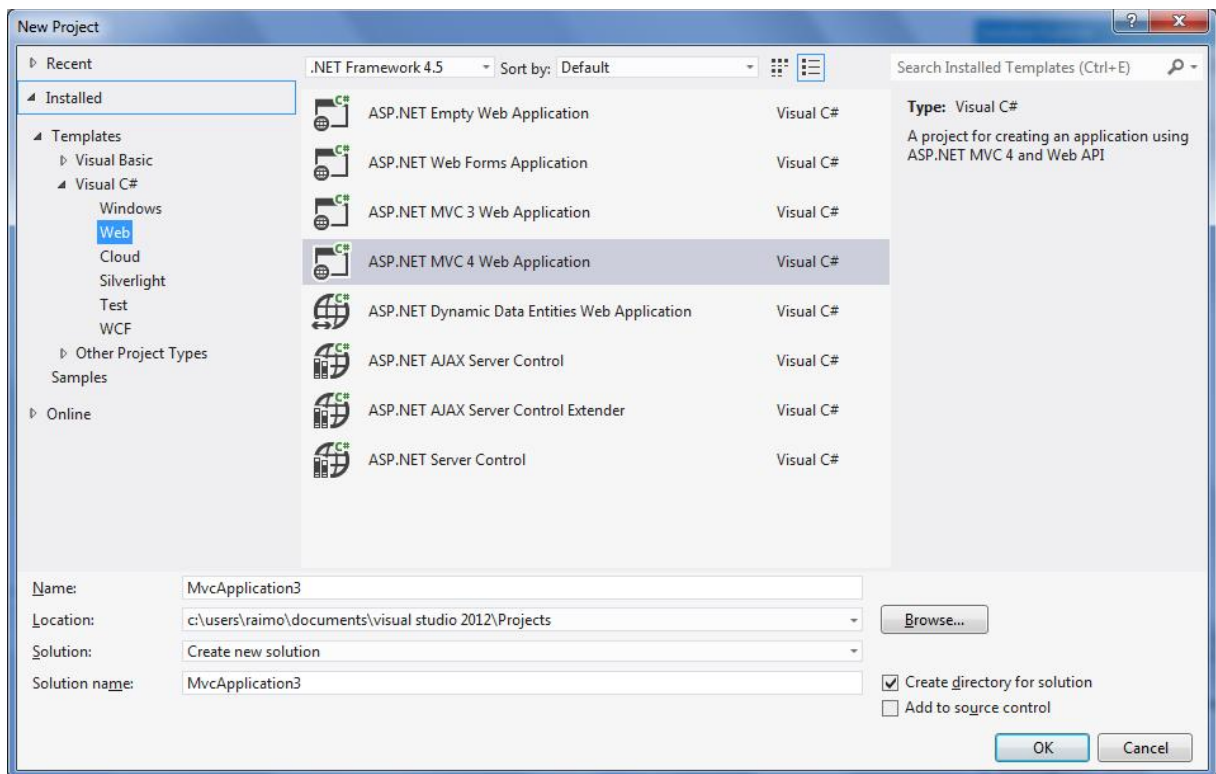
Mis asi see ASP.NET MVC siis on? Kõige lihtsamini öelduna on tegu veebirakenduste loomiseks mõeldud raamistikuga, mis kasutab *Model-View-Controller* mustrit. See arhitektuuri muster jaotab rakenduse kolmeks põhiliseks osaks: *Model*, *View* ja *Controller*. Igalühel nendest on oma kindel ülesanne. Põhimõtte sellise mustri taga on, et kõik neid osi saab arendada ja testida üksteisest eraldi ja kombineerituna loovad väga elastse ja jõulise rakenduse.

- *Model*: Selliste klasside kogum, mis kirjeldavad rakenduse poolt kasutatavaid andmeid ja loogikat kuidas andmeid muuta.
- *View*: Defineerib kasutajaliidese kuvamise.
- *Controller*: Klasside kogum, mis käsitleb suhtlust kasutajaga. Töötleb veebilehitsejast tulnud päringuid, otsib välja mudeli andmed ning esitab vaateid.

Täpsemalt MVC arhitektuurist arusaamiseks loome MVC rakenduse. ASP.NET MVC rakenduse loomiseks kasutame Visual Studio Express 2012.

# 1. Projekti loomine

Kasutades Visual Studio Express loome uue projekti valides menüüst *FILE->New Project*.

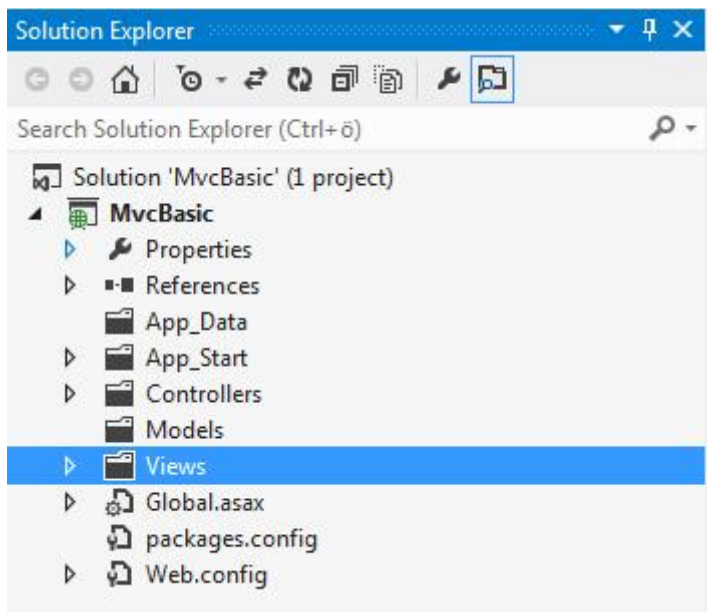


Joonis 1. Uue projekti loomine

Valime *ASP.NET MVC 4 Web Application*. Projekti nimi ei ole hetkel oluline, kuna tegemist on minimaalse MVC raamistiku käivitamiseks vajamineva projektiga. Järgnevalt aknast valime *Empty* malli ning loome projekti.

## 1.1 MVC veebirakenduse struktuur

Projekti loomisel näeme *Solution Explorer*-is projekti struktuuri.



**Joonis 2. Rakenduse struktuur**

*Properties* – Sisaldab *AssemblyInfo.cs* faili, mis omakorda sisaldab üksiasju, mis on seotud *assembly* versiooni informatsiooniga ja muu informatsiooniga *assembly* kohta.

*References* – Sisaldab *assembly*-si, mida projektis kasutatakse.

*App\_Data* – Sisaldab andmesalvestamisega seotud faile.

*App\_Start* – Sisaldab staatilisi klasse, mida rakenduse käivitamisel väljakutsutakse.

*Controllers* – Sisaldab kontrolleri klasse.

*Models* – Sisaldab mudeli klasse.

*Views* – Sisaldab vaateid.

*Global.asax* – Globaalne ASP.NET klass, mille kood rakenduse käivitamisel käivitatakse.

*Packages.config* – Seda faili kasutatakse NuGet pakketid jälgimiseks.

*Web.config* – Põhiline ASP.NET veebirakenduse sätete ja konfiguratsiooni fail.

Esiolgu on meie jaoks olulised kolm kausta: *Models*, *Views* ja *Controllers*. Loo uue kontrolleri, selleks parem hiireklõps „Controller“ kausta peal ja *Add->Controller*. Anname kontrolleri nimeks „Test“ koos sufiksiga „Controller“, et täisnimeks jääks „TestController“. Malli hetkel ei lisa, nii et selleks väljaks jääb *Empty MVC Controller*. ASP.NET MVC genereerib minimaalse koodi, et kontrolleri toimiks. Kontrolleri on tavaline klass, mis implementeerib *System.Web.Mvc.ControllerBase* liidest. Kõikide kontrolleri nimesid

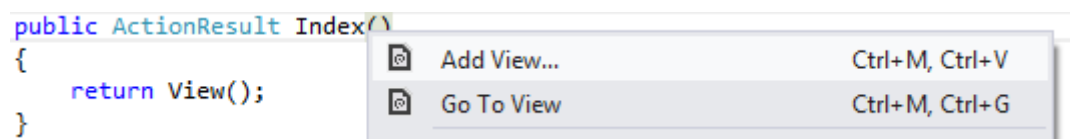
peavad sisaldama *Controller* järelliidest. Kontrolleri klassi sees asuvad *Controller action*-id, mis käsitlevad rakenduse poole pöördumisi. Täpselt nagu .NET mõistes on kontrolleriid tavalised klassid on ka *Controller action*-id tavalised meetodid, mis sisaldavad loogikat kuidas sissetulevaid päringuid töödelda ja mida väljastada.

Esitatakse vaade vastavalt kontrolleri meetodi nimele, ehk kui kontrolleri meetodi nimi on *Index*, siis esitatakse ka *Index* vaade. Saab ka alljärgneval viisil esitatava vaate nime muuta, kuid koodi lihtsuse eesmärgil ei ole see kõige parem.

```
public ActionResult Index()
{
    return View("About");
}
```

#### Koodinäide 1. Vaadete esitamine

Vaate loomiseks parem hiireklõps kontrolleri meetodi nime peale ja *Add view*.



#### Joonis 3. Vaate loomine

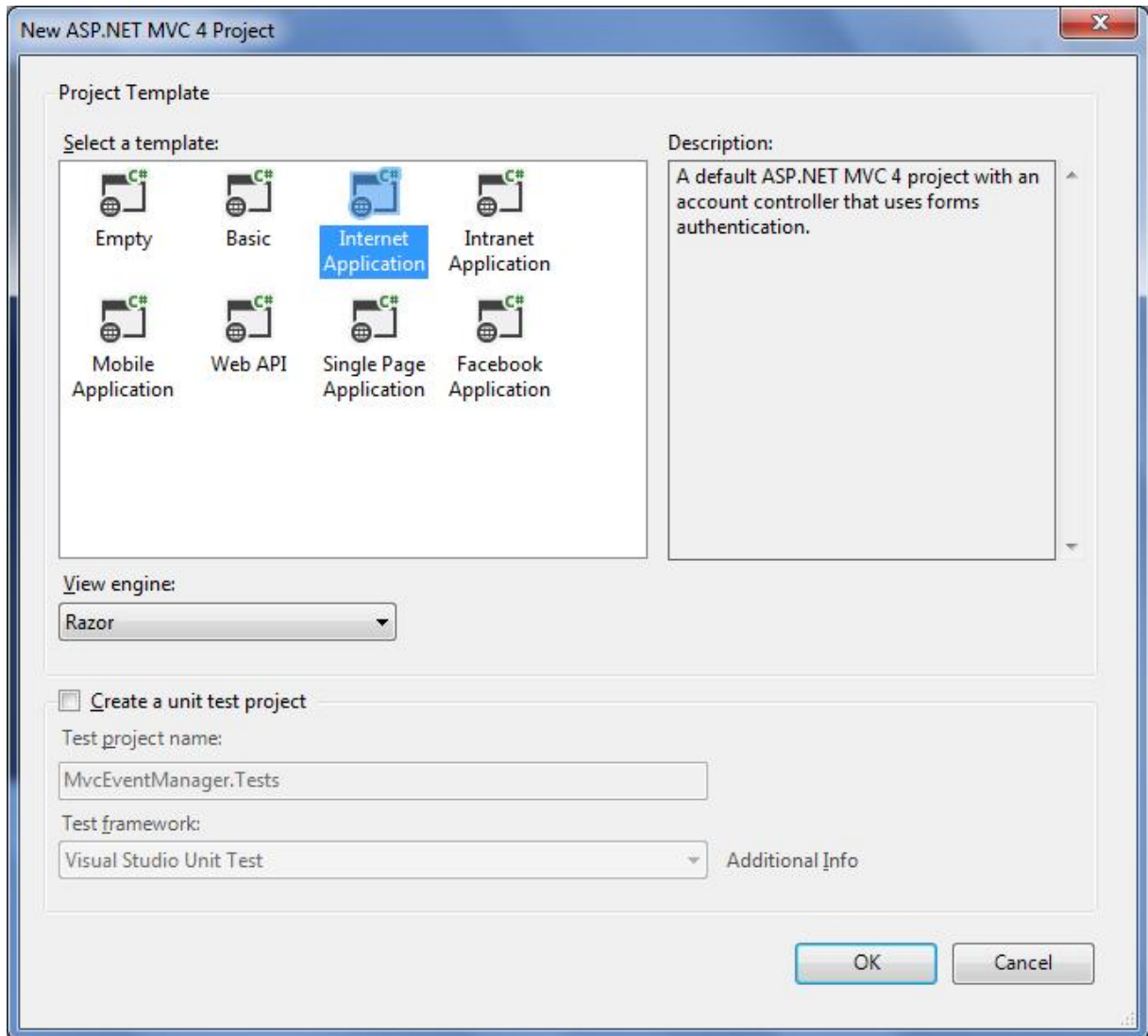
Vaate loomise aknas jätame kõik vaikimisi valikud. Teisi võimalusi kasutame ära hiljem.

„Views“ kaustas asuvad rakenduse kuvamise ja kasutajaliidesega seotud failid (HTML). Iga kontrolleri kohta on üks vaate kaust. Vaatades „Views“ kausta näeme, et loodud *Index* vaade tehti „Test“ kausta sisse kontrolleri nime järgi. „Shared“ kaustas asuvad kuvamisega seotud failid, mida kõik kontrolleriid saavad kasutada.

Lisame *Test* kontrolleriidesse veel ühe meetodi nimega „Teine“ ning lisame selle meetodile ka vaate nagu eelnevalt. Käivitame projekti valides menüüribalt *Debug->Start Debugging* või vajutades klaviatuuril klahv F5. Navigeerida saab loodud lehtedele järgmiselt: *localhost:port/Test/Index* ning *localhost:port/Test/Teine*.

## 1.2 Projekti loomine kasutades malli.

Loome uue ASP.NET MVC 4 veebirakenduse ja anname projektile nime. Kuna selle õppematerjali käiguse loome ürituste loomise veebirakenduse, siis anname nimeks „MvcEventManager“.



Joonis 4. Projekti malli valimine

Projekti malliks valime *Internet Application*. Selle malli kasutamine lisab meie projektile mõned abistavad kliendi- ja serveripoolsed teegid (näiteks JQuery JavaScript Library), samuti serveripoolse andmete juurdepääsu raamistiku nimega Entity. Kõiki installeeritud lisasi näeb projektis *packages.config* failis. Lisaks sellele genereeritakse paar kontrollerit ja vaadet, mis lisavad liikmelisuse ja autentimise funktsionaalsused.

*View engine* võib jääda Razor, millest täpsemalt räägime vastavas peatükis. Projektile saab lisada ka komponentide testimist. Hetkel seda ei lisa, kuna seda saab hiljem projektile juurde lisada kui see on vajalik.

Kui projekt on valmis vaatame, mis ASP.NET MVC meje jaoks loonud on. Selleks valida menüüst *Debug->Start Debugging* või vajutada lihtsalt F5.

## 2. Näidisrakenduse loomine

### 2.1 Mudeli loomine

Uue mudeli loomiseks parem hiireklõps „Models“ kausta peal ja *Add->Class*. Anname klassile nime „Events“ ja lisame klassi vajutades *Add*. Lisame klassile mõned üritusi defineerivad omadused.

```
public class Events
{
    public long Id { get; set; }
    public string Title { get; set; }
    public string Description { get; set; }
    public DateTime Date { get; set; }
}
```

**Koodinäide 2. Models/Events.cs – Mudeli loomine**

Esiteks *Id* väli, mis on tavaline identifikaator eristamaks erinevaid üritusi. Seejärel *Title* väli, et anda üritusele pealkiri, *Description* väli, et anda üritusele kirjeldus ja *Date* väli, et anda üritusele toimumise kuupäev.

### 2.2 Marsruutimine

Traditsiooniliselt viidatakse URL-iga ühele failile failisüsteemis. Näiteks:

`http://minudomeen.com/info/kontakt.html`

viitaks failile nimega „kontakt.html“, kaustas *~/info*, mis asub domeeni *minudomeen.com* juurkataloogis. ASP.NET MVC raamistik sisaldab pindlikku URL marsruutimissüsteemi, mis võimaldab defineerida rakenduse raames URL kaardistamise reeglid. See murrab seose füüsilise faili ja URL-i vahel ja mis lubab URL-i rakenduses kohandada vastvalt vajadusele. ASP.NET MVC tavaks on kaardistada URL-id selliselt, et nad näitaks kindlas kontrollerris kindla meetodi peale.



http://minudomeen.com/controllerName/controllerActionName

ASP.NET MVC saavutatakse see registreerides marsruutimise mallid, mis seavad paika kuidas sissetulevad URL-id kindla kontrolleri ja meetodi pealse suunatakse. Seda tehakse rakenduse käivitamisel läbi *Application\_Start* meetodi, mis asub *Global.asax* klassis. *Application\_Start* meetod kutsub välja *RegisterRoutes* meetodi, mis asub eraldi klassis *RouteConfig*, mis asub „App\_Start“ kaustas.

```
public class RouteConfig
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new { controller = "Home", action = "Index", id =
                UrlParameter.Optional }
        );
    }
}
```

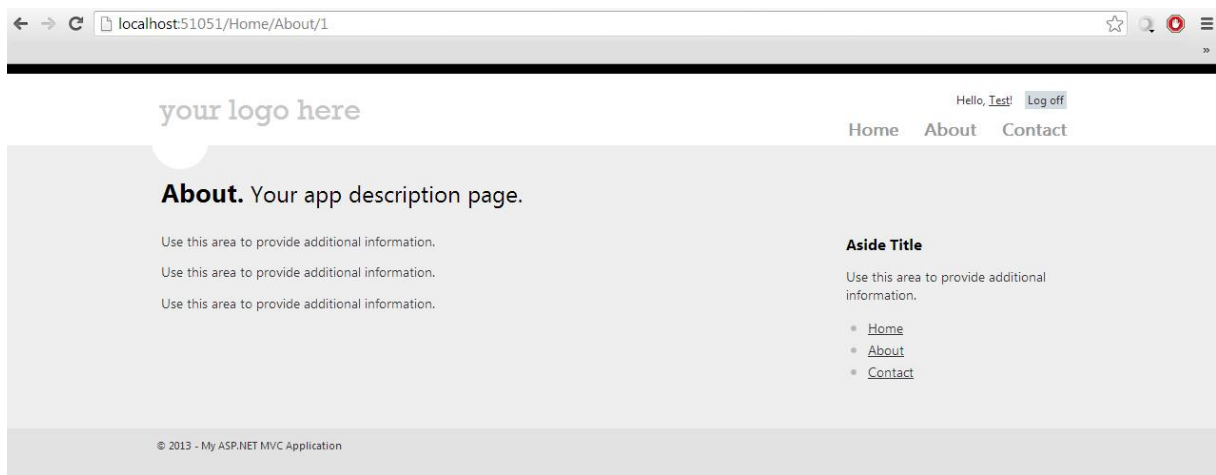
### Koodinäide 3. App\_Start/RouteConfig.cs – Vaikimisi marsruut

Meetod *RegisterRoutes* loob uue marsruudi nimega „Default“ ja defineerib URL mustri, mis jagab URL-i kolmeks osaks: *Controller*, *Action* ja *Id*. Seejärel anname nendele osadele vaikimisi väärtused, mis lähevad kasutusele kui kasutaja pole neid URL-i sisestanud.

Pannes projekti kohalikus masinas tööle ja sisestades addressi reale järgnev:

http://localhost:port/Home/About/1

Sellega kuvatakse meile *Home* kontrolleri *About* meetodi poolt esitatav vaade. Kuigi me hetkel *Id*-d ei kasuta on see siiski korrektne marsruut ja toimib. Kui *Id* ära jätta toimib see samuti, sest definitsioonis on see määratud valikuliseks. Kui aga addressi realt ka *About* ära jätta, siis kuvatakse *Index* meetod, sest marsruudi definitsioonis on see määratud vaikimisi meetodiks.



Joonis 5. Rakenduse esitamine veebilehitsejas

## 2.3 Razor

Vaate mootor (ingl. k. *View engine*) vastutab HTML-i esitamise eest vaatest veebilehitsejasse. ASP.NET MVC 4 raamistikus on Microsofti poolt loodud võimekas vaate mootor Razor. Razor-i kasutamine on lihtne ja ta lihtsustab oluliselt koodi kirjutamist ja lugemist.

Razor-iga koodiosa kirjutamiseks HTML-is kasutatakse „@“ sümbolit. Oluline on asjaolu, et Razor ei nõua, et koodiploki oleks lõputähis.

```

@{
    //üherealine kommentaar koodiblokki sees. @{ tähistatakse koodiblokki algust.
    string näide = "tekst";
}

<h1>Razor näide</h1>
@*
    Kommentaariblokki näide.
*@
<h2>
    Tere @name, praegu on @DateTime.Now.Year aasta.
</h2>

@*
    Razor näide mitmerealise koodinäite puhul (if statement).
    Lihtteksti kasutamiseks koodi sees on kaks võimalust: <text> või @:
*@
<p>
    @if (products.Count == 0){
        <text>Tooted on otsas</text>
    }else {
        @: Hetkel tooteid veel alles: @products.Count
    }
</p>

```

**Koodinäide 4. Razor-i kasutamine**

## 2.4 Mudeliga suhtlemine

Andmeid saab kasutada kolme põhilise meetodi abil: *ViewData*, *ViewBag* ja *TempData*. Kõik kolm on nii kontrolleri kui vaate atribuudid. Üldiselt kasutatakse neid vähesemahuliste andmete saatmiseks ühest kindlast asukohast teise, kas siis kontrollerist vaatesse või vaatest vaatesse. *ViewData* ja *ViewBag* on sarnased kuid paari olulise erinevusega. *ViewData* on sõnastikutüüpi objekt, mille poole saab pöörduda kasutades sõnesid võtmetena. *ViewBag* objekt on ümbris *ViewData* objektile, mis lubab luua dünaamilisi atribuute. *ViewData* nõuab andmetüübi teisendamist kompleksandmetüüpide korral, *ViewBag* aga mitte. *TempData* on väga sarnane *ViewData*-le, erinevusega selles, et *TempData* on mõeldud olema serveris väga vähe aega. Kui sa annad *TempData*-le andmeid ühe pöördumisega, siis andmed on seal ainult kuni järgmise pöördumiseni. Seega põhiliselt kasutatakse seda ainult veebilehtede ümbersuunamisel.

Loodud on mudel, mis defineerib ürituse omadused. Loome kontrollerile meetodi, kus loome uue isendi *Events* mudelist. Lisame *EventsController.cs* klassi uue meetodi nimega „Event“.

```

public ActionResult Event()
{
    var Event = new MvcEventManager.Models.Events()
    {
        Title = "See on pealkiri",
        Description = "See on kirjeldus",
        Date = DateTime.Now,
    };

    ViewBag.Event = Event;
    return View();
}

```

#### Koodinäide 5. Controllers/EventsController.cs – Meetodi loomine

Loome ka vaate eelpool mainitud viisil. Lisame HTML koodi, et üritus kuvada.

```

@{
    var Event = ViewBag.Event;
}

<div class="Event">

<h2>@Event.Title</h2>
    <div class="details">
        <p>Kirjeldus: @Event.Description</p>
        <p>Toimumise aeg: @Event.Date.ToString("g")</p>
    </div>
</div>

```

#### Koodinäide 6. Views/Events/Event.cshtml – Event meetodile vaate loomine ViewBag meetodiga

*ViewBag* on efektiivne, kuid on veel parem viis kuidas andmeid saata. Alternatiiviks *ViewBag* meetodile on saata mudel otse vaatesse läbi *View Helper* abilise.

```

public ActionResult Event()
{
    var Event = new MvcEventManager.Models.Events()
    {
        Title = "See on pealkiri",
        Description = "See on kirjeldus",
        Date = DateTime.Now,
    };
    return View(Event);
}

```

#### Koodinäide 7. Controllers/EventsController.cs – Mudeli saatmine vaatesse

Muudame ka vaadet, täpsustades ära millist mudeli tüüpi kasutatakse. Lisades rea „@model *MvcEventManager.Models.Events*“ seostuvad andmeobjektidele ka andmetüübid nagu *Events* mudelis defineeritud.

```

@model MvcEventManager.Models.Events
@{
    var Event = Model;
}

<div class="Event">

<h2>@Event.Title</h2>
    <div class="details">
        <p>Kirjeldus: @Event.Description</p>
        <p>Toimumise aeg: @Event.Date.ToString("g")</p>
    </div>
</div>

```

**Koodinäide 8. Views/Events/Event.cshtml – Vaate sidumine mudeliga**

## 2.5 Kasutajate suhtlus veebirakendusega

Selleks, et kasutajad saaksid veebirakenduses üritusi luua loome *Events* kontrolleri uue meetodi nimega „Create“.

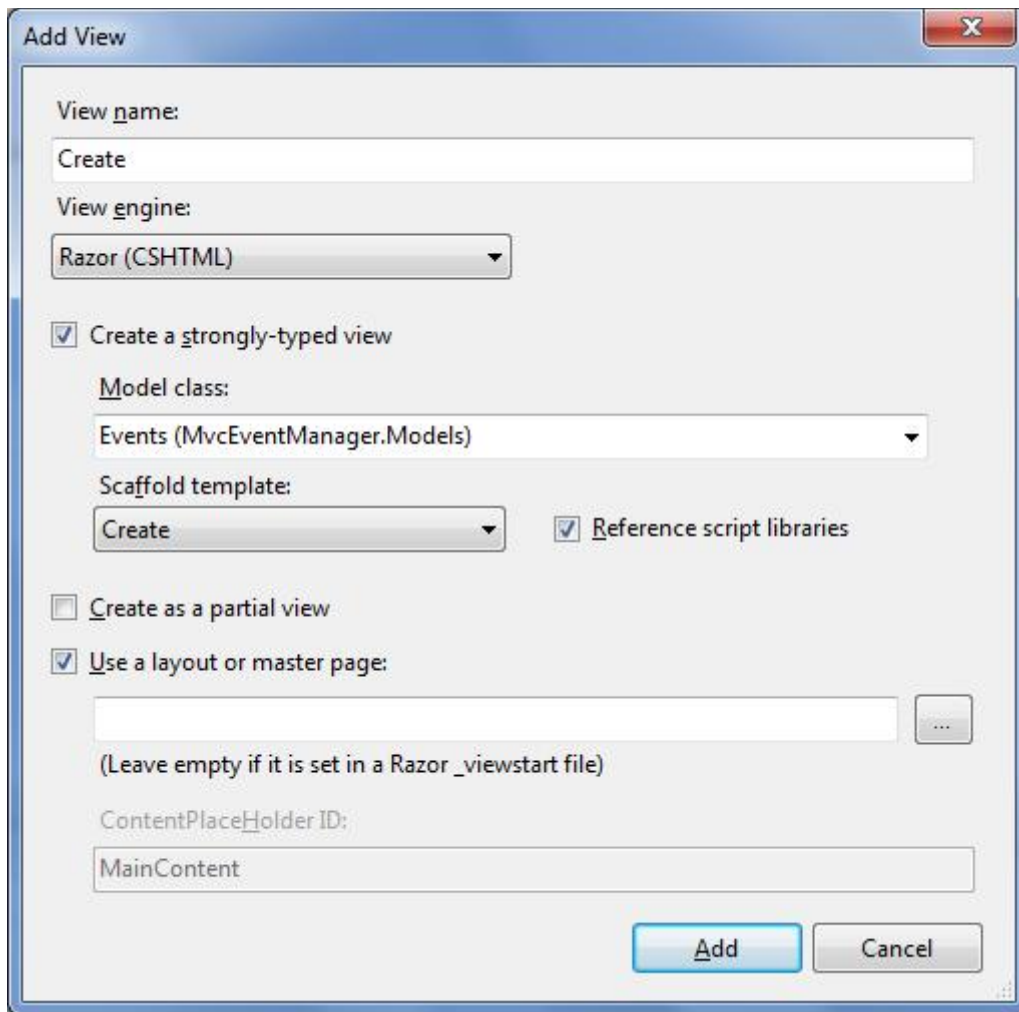
```

public ActionResult Create()
{
    return View();
}

```

**Koodinäide 9. Controllers/EventsController.cs – Create meetodi loomine**

Lisame *Create* meetodile ka vaate. Selleks parem hiireklõps meetodi nime peal ja *Add View*. Valikutes märgistame välja *Create a strongly-typed view*, mudeli klassiks määrame *Events* klassi ja *Scaffold template* väljas valime *Create* malli.



Joonis 6. Vaate lisamine kasutades Create malli

ASP.NET MVC raamistik genereerib HTML vormi koodi. Vormi loomiseks kasutatakse *Html.BeginForm* abilist. Kõik, mis jääb selle abilise sisse kuvatakse kasutajale HTML vormi sees. ASP.NET MVC raamistik vaatab üle mudeli ja genereerib vastavate objektide väljad, kasutades *Html.LabelFor* ja *Html.EditorFor* abilisi.

```

@using (Html.BeginForm()) {
    @Html.AntiForgeryToken()
    @Html.ValidationSummary(true)

    <fieldset>
        <legend>Events</legend>

        <div class="editor-label">
            @Html.LabelFor(model => model.Title)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Title)
            @Html.ValidationMessageFor(model => model.Title)
        </div>

        ...      <!-- teised atribuutide väljad -->

        <p>
            <input type="submit" value="Create" />
        </p>
    </fieldset>
}

```

**Koodinäide 10.** Views/Events/Create.cshtml – Create malliga loodud vaade

*Html.LabelFor* ja *Html.EditorFor* genereerivad vastavad väljad kasutades andmete annotatsioone mudelis. Kui seda pole tehtud siis vaikimisi kuvatakse kõikideks väljeks tekstiväljad ning märgistus mudelis määratud muutuja nime järgi. *Html.ValidationMessageFor* abiline kuvab valideerimisel tekkinud vead. Valideerimise reeglid defineeritakse samuti andme annotatsioonidega mudelis.

## 2.6 Model binding

Model binding on ASP.NET MVC omadus, mis aitab andmeid päringutest kätte saada, täita automaatselt kontrolleri meetodite parameetreid ning kannab hoolt omaduste kaardistamise ja omaduste andmetüübi muutmise eest. Näide kuidas tavaliselt kasutaja sisestatud andmeid kätte saada:

```

public ActionResult Create()
{
    var event1 = new Event()
    {
        Title = Request["Title"],
        Description = Request["Description"],
        Date = DateTime.Parse(Request["Date"]),
        TicketPrice = Decimal.Parse(Request["TicketPrice"])
    };
    ...
}

```

#### Koodinäide 11. Koodinäide päringust andmete saamiseks

Järgneva näite tulemus on sama, kuid kasutades model binding omadust:

```

public ActionResult Create(string title, string description, DateTime date, decimal
ticketPrice)
{
    var event1 = new Event()
    {
        Title = title,
        Description = description,
        Date = date,
        TicketPrice = ticketPrice
    };
    ...
}

```

#### Koodinäide 12. Koodinäide *model binding*-u kasutamisest

Sama saame teha ka keeruliste andmetüüpide puhul. Seome ürituste veebirakenduses *Create* meetodi parameetrid mudeli klassiga.

```

public ActionResult Create(Models.Events Event)
{
    return View();
}

```

#### Koodinäide 13. Controllers/EventsController.cs – Meetodi seostamine mudeliga

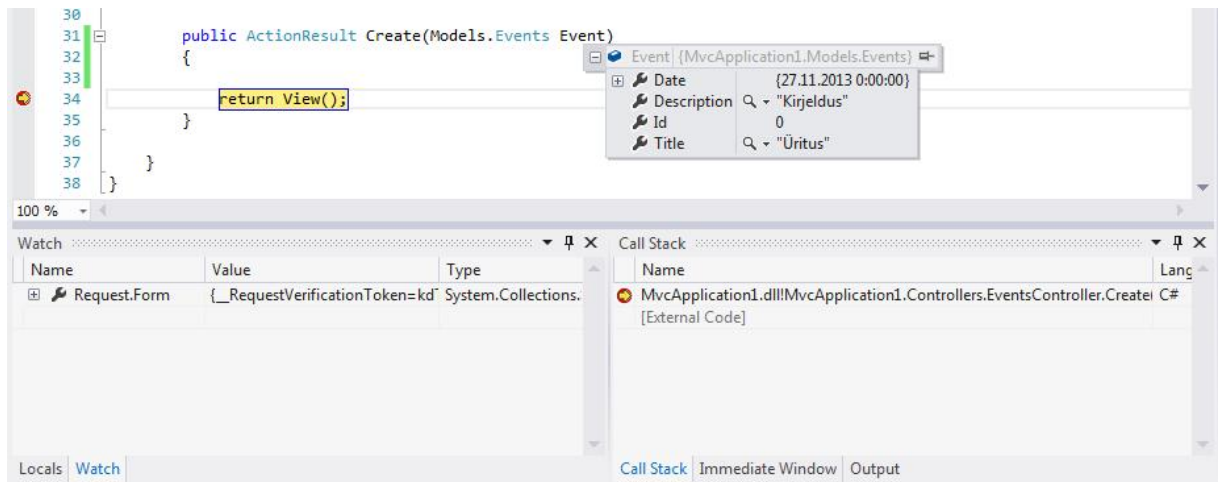
Et veenduda vormi andmete serverisse saatmise ja sisestatud andmete mudeliga seostamise eest, lisame katkestuspunkti projektis reale, mis esitab vaate.



```
30
31     public ActionResult Create(models.Events Event)
32     {
33
34         return View();
35     }
36
37 }
38 }
```

Joonis 7. EventsController.cs – Katkestuspunkti lisamine

See järel käivitame projekti silumisrežiimis ja üritame andmeid postitada. Liikudes kursoriga sisestatud mudeli *Event* peale ja vajutades plussmärgi peale näeme sisestatud andmeid, mis on seotud mudelis defineeritud objektidega.



Joonis 8. Näide vormist saadud andmete kohta

## 2.7 Ülesanne 1

Luaa Blogi rakendus MVC raamistikus

1. Luaa blogi mudel vajalikke andmetega (Pealkiri, kuupäev, sisu jne).
2. Luaa blogi kontrolleri koos kuvamise ja postituste loomise meedotitega ja vaadetega.
3. Marsruudi loomine, mille abil saab kuupäevade kaudu navigeerida. Näiteks: Archive/2010/08.

## 3. Rakendusele funktsionaalsuste lisamine

### 3.1 Valideerimine

Selleks, et kasutajad ebakorrektsaid andmeid ei saaks rakendusse sisestada on vaja andmed valideerida. Seda saab väga lihtsalt saavutada lisades andmetele annotatsiooni mudeli klassis. Andmete annotatsioon lubab kirjeldada reegleid, mida me mudel objektidele tahame anda. ASP.NET MVC hoolitseb selle eest, et reegleid täidetakse ja vajalikud sõnumid kasutajatele kuvatakse. Selleks, et annotatsioone kasutada saaks tuleb lisada nimeruum `System.ComponentModel.DataAnnotations`.

```
public class Events
{
    [Required]
    public long Id { get; set; }

    [Required]
    [DataType(DataType.Text)]
    [StringLength(maximumLength: 100, MinimumLength = 4)]
    public string Title { get; set; }

    [Required]
    [DataType(DataType.MultilineText)]
    public string Description { get; set; }

    [Required]
    [DataType(DataType.DateTime)]
    [DisplayFormat(DataFormatString = "{0:dd.MM.yyyy}", ApplyFormatInEditMode =
true)]
    public DateTime Date { get; set; }
}
```

#### Koodinäide 14. Models/Events.cs – Mudelile annotatsioonide lisamine

Selleks, et valideeritud vormi sisestada ja hiljem ka andmebaasi salvestada on vaja muuta *Create* meetodit.

```

public ActionResult Create(Models.Events Event)
{
    return View();
    if (ModelState.IsValid){
        //Andmebaasi lisamine
        return RedirectToAction("Index");
    }
    return Create();
}

```

**Koodinäide 15. Controllers/EventsController.cs – Sisestatud andmete valideerimine**

*ModelState.IsValid* omadus kontrollib, kas mudel on valideeritud. Kui valideerimine ebaõnnestub kuvatakse vaade uuesti ning lisatakse valideerimise veateated, kasutades *Html.ValidationMessageFor* abilist vaadetes.

Selline lahendus ei kompileeru. *Create* meetod tuleb jagada kaheks: üks, mis kuvaks ürituse loomise vormi ja teine, mis kontrollib sisestatud andmete korrektsust ja saadab need serverisse. Kaheks jaotamisel ei oska aga ASP.NET MVC käitus (ingl. k. *runtime*) valida kumba kasutada. Selleks lisame meetoditele *HttpGet* ja *HttpPost* atribuudid.

```

[HttpGet]
public ActionResult Create()
{
    return View();
}
[HttpPost]
public ActionResult Create(Models.Events Event)
{
    if (ModelState.IsValid){
        //Andmebaasi lisamine
        return RedirectToAction("Index");
    }
    return Create();
}

```

**Koodinäide 16. Controllers/EventsController.cs – Create meetodi jagamine kaheks**

„Views“ kaustas *Create.cshtml* failis viimastel ridadel on ASP.NET MVC genereerinud JQuery teegid.

```
@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

#### Koodinäide 17. Views/Events/Create.cshtml – JQuery teekide kasutamine

Sellega muudetakse valideerimine kliendipoolseks ehk nõuetele mittevastavaid andmeid serverisse kontrolliks ei saadeta, vaid valideerimine toimib kliendi veebilehitsejas. Seda saame lihtsalt katsetada. Kommenteerides välja JQuery teegid ning käivitades rakenduse. Avame veebilehitseja arendaja tööriistada (Chrome veebilehitsejas *Customize->Tools->Developer tools* või *Ctrl+Shift+I*) ning valime *Network* riba. Ebakorreksete andmete sisestamisel näeme, et laetakse terve leht uuesti. Eemaldades kommentaarid JQuery teegi kasutamise ümbert ning seejärel ebakorreksete andmete sisestamisel näeme, et andmeid serverisse ei saadeta.

The screenshot shows a web form titled "Create" with three input fields: "Title", "Description", and "Date". Each field has a red error message next to it: "The Title field is required.", "The Description field is required.", and "The Date field is required." Below the form is a table with columns: Name, Path, Method, Status, Text, Type, Initiator, Size, Content, Time, Latency, and Timeline. The table is currently empty. At the bottom of the screenshot, the Chrome DevTools Network tab is visible, showing a message: "No requests captured. Reload the page to see detailed information on the network activity." The Network tab also shows a filter for "All" and a list of categories: Documents, Stylesheets, Images, Scripts, XHR, Fonts, WebSockets, and Other.

#### Joonis 9. Kliendipoolse valideerimise kontrollimine

## 3.2 Andmebaasi salvestamine

Andmebaasi andmete salvestamiseks kasutame Entity raamistiku, mis lisati projektile *Internet Application* malli valimisel. Entity raamistik on .NET raamistiku osa, mis kasutab „kood esimesena“ (ingl. k. „*code first*“) lähenemist andmete poole pöördumiseks. Seega Entity raamistik genereerib andmebaasi kirjutatud andmemudeli põhjal. See välistab enamiku andmepöörduskoodi kirjutamise arendajate poolt.

Entity raamistiku abil andmebaasiga suhtlemiseks tuleb luua andmebaasi kontekst (ingl. k. *database context*). Tegemist on klassiga, mis implementeerib Entity raamistiku *DbContext* baasklassi. Selle klassi kasutamine lubab andmebaasi muuta ja sealt andmeid kätte saada.

Loome kausta „Models“ uue klassi nimega „EventsDataContext“. See klass tuleneb *DbContext* klassist ning failile tuleb lisada nimeruum System.Data.Entity. Klassi lisame *DbSet* andmetüübi, kus näitame millise mudeli andmeid andmebaasiga suhtlemisel kasutatakse. Entity raamistiku terminoloogias määrab *DbSet* ühe andmetabeli ja mudeli klassi objektid määravad andmetabeli rea.

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Web;

namespace MvcEventManager.Models
{
    public class EventsDataContext : DbContext
    {
        public DbSet<Events> Events { get; set; }
    }
}
```

**Koodinäide 18. Models/EventsDataContext.cs – Database context klassi loomine**

Ürituste andmebaasi lisamiseks muudame *Events* kontrolleri *Create* meetodi osa, mis tegeleb uue ürituse lisamisega. Loome *EventsDataContext*-ist uue isendi. Seejärel lisame *Event* isendi *Events* andmekogumikku loodud *EventsDataContext* isendis. Ning lõpuks kutsume välja *SaveChanges* meetodi, mis salvestab muudatused andmebaasi.

```

[HttpPost]
public ActionResult Create(Models.Events Event)
{
    if (ModelState.IsValid)
    {
        //Andmebaasi lisamine
        var db = new EventsDataContext();
        db.Events.Add(Event);
        db.SaveChanges();

        return RedirectToAction("Index");
    }
    return Create();
}

```

#### Koodinäide 19. Controllers/EventsController.cs – Ürituse isendi andmebaasi salvestamine

Andmebaasiga ühendamiseks kasutame *SQL Express LocalDB* tehnoloogiat. Selleks lisame juurkaustas olevasse *Web.Config* faili andmebaasiga ühendamise sõne. *ConnectionString* välja alt leiame, et üks ühendus on olemas, mille ASP.NET MVC genereeris projekti loomisel liikmelisuse võimaldamiseks. Teeme sellest koopia ja muutame omadused järgnevalt.

```

<add name="MvcEventManager.Models.EventsDataContext" connectionString="Data
Source=(LocalDb)\v11.0;Integrated
Security=SSPI;AttachDBFilename=|DataDirectory|\MvcEventManager.Models.EventsDataConte
xt.mdf" providerName="System.Data.SqlClient" />

```

#### Koodinäide 20. Web.Config – *ConnectionString*-i loomine

Projekti käivitamisel ja uue ürituse lisamisel loob Entity raamistik uue andmebaasi ja lisab sinna sisestatud andmed. Selleks, et Entity raamistik saaks andmebaasi uue tabeli luua, peab olema andmemudelil annotatsiooniga „Key“ tähistatud objekt, mis SQL andmebaasis primaarse võtme rolli täidab. Samuti võib andmemudelil olla objekt „Id“, mille Entity raamistik ilma annotatsioonita primaarse võtme rolli paneb. Pärast andmebaasi loomist Entity raamistikule ei meeldi muudatused andmemudelil. Kuna projekti arenedes mudel muutub ja selleks, et andmebaasi skeem uueneks vastavalt mudelis tehtud muudatustele, tuleb kasutada Entity raamistiku *Code First Migrations* võimalust. Selleks ülevalt menüüst *TOOLS->Library Package Manager->Package Manager Console*.

```

PM> enable-migrations -ContextTypeName MvcEventManager.Models.EventsDataContext

```

Pärast käsu käivitamist lisatakse projektile „Migrations“ kaust. Kaustas on konfiguratsiooni klass, kus saab määrata kuidas migratsioonid andmebaasi kontekstis käituvad. Lubame automaatsed migratsioonid muutes järgnevat rida.

```
AutomaticMigrationsEnabled = true;
```

**Koodinäide 21. Migrations/Configuration.cs – Automaatsete migratsioonide lubamine**

Konfiguratsiooni klassis on ka `Seed` meetod, mis käivitub iga migratsiooni käivitamisel. Selle meetodiga saab andmebaasi algandmeid lisada. *Package Manager* konsoolis rea *Update-database* kirjutamisel loob Entity raamistik andmebaasi, mida näeme „App\_Data“ kaustas, valides *Solution Explorer* tööriistaribalt *Show All Files*.

### 3.3 Andmebaasist andmete kuvamine

Hetkel kuvatakse meile rakenduses koodiga kirjapandud klassi isendid, kuid nüüd sooviks andmeid andmebaasist saada. Selleks muutame *Events* kontrolleri selliselt, et *Index* meetod kuvaks kõik olemasolevad üritused ja *Event* meetod kuvaks ühe konkreetse ürituse info.

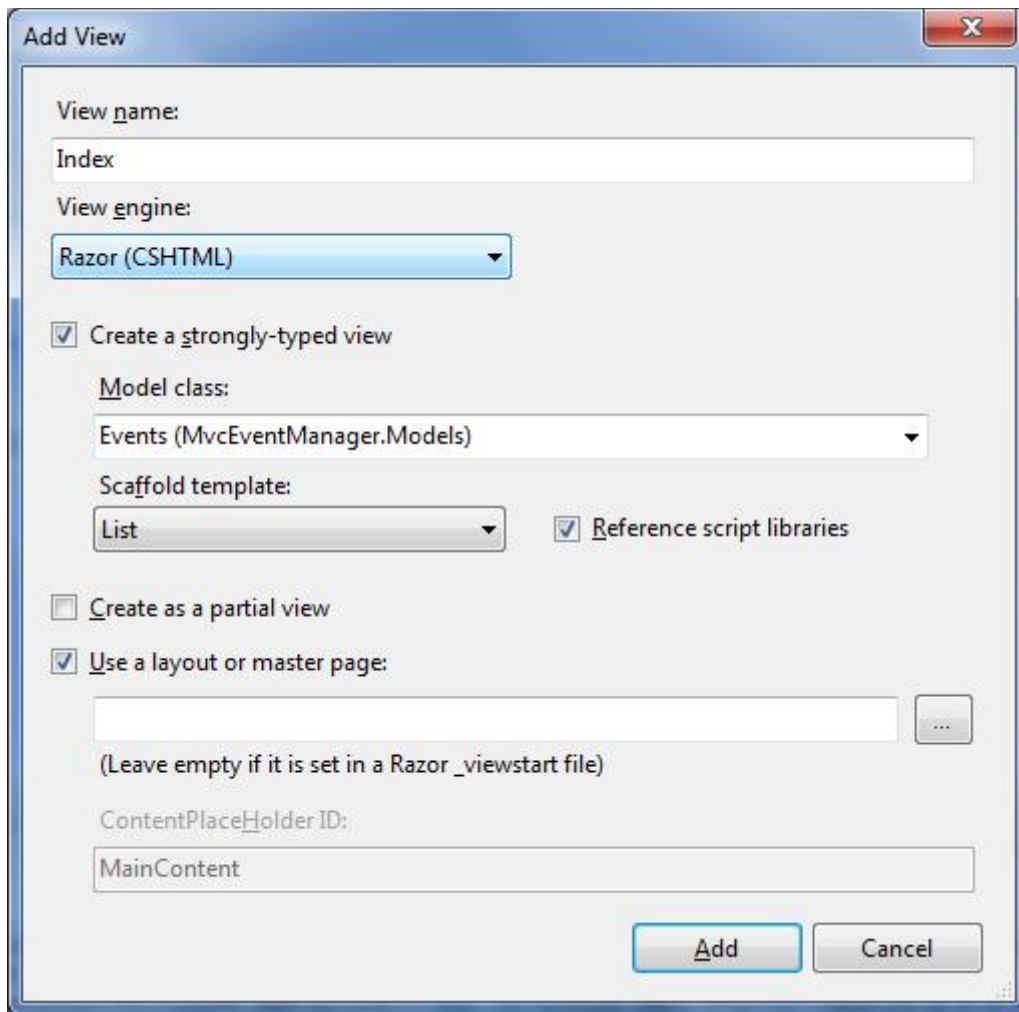
```
private EventsDataContext db = new EventsDataContext();
public ActionResult Index()
{
    var events = db.Events.ToArray();

    return View(events);
}
public ActionResult Event(long Id)
{
    var Event = db.Events.Find(Id);

    return View(Event);
}
```

**Koodinäide 22. Controllers/EventsController.cs – Andmebaasist andmete pärimine**

Kustutame ära kaustas „Views/Events“ asuva *Index.cshtml* faili ning loome uue vaate *Index* meetodile.



Joonis 10. Index vaate lisamine kasutades *List* malli

ASP.NET MVC genereerib HTML-i, kus on näha kasutusel paar abilist. ASP.NET MVC genereerib lingid, kasutades *Html.ActionLink* abilist. Hetkel toimib ainult *Create new* link, kuna meil on ürituste loomiseks meetod loodud. Samuti kasutades *Html.DisplayNameFor* ja *Html.DisplayFor* abilisi leiab ASP.NET MVC parima lahenduse kuidas kuvada vastavalt välja nimi ja välja tüüp. Seda tehakse annotatsioonide abil mudelis. Vaikimisi määrab ASP.NET MVC väljatüübiks tekstivälja ja välja nimeks mudeli objektide nimed. Kui aga annotatsioonis on määratud väljatüüp ja on antud väljale nimi, siis kasutab ASP.NET MVC neid.



### 3.4 Andmemudeli muutmine ja andmebaasi uunedamine

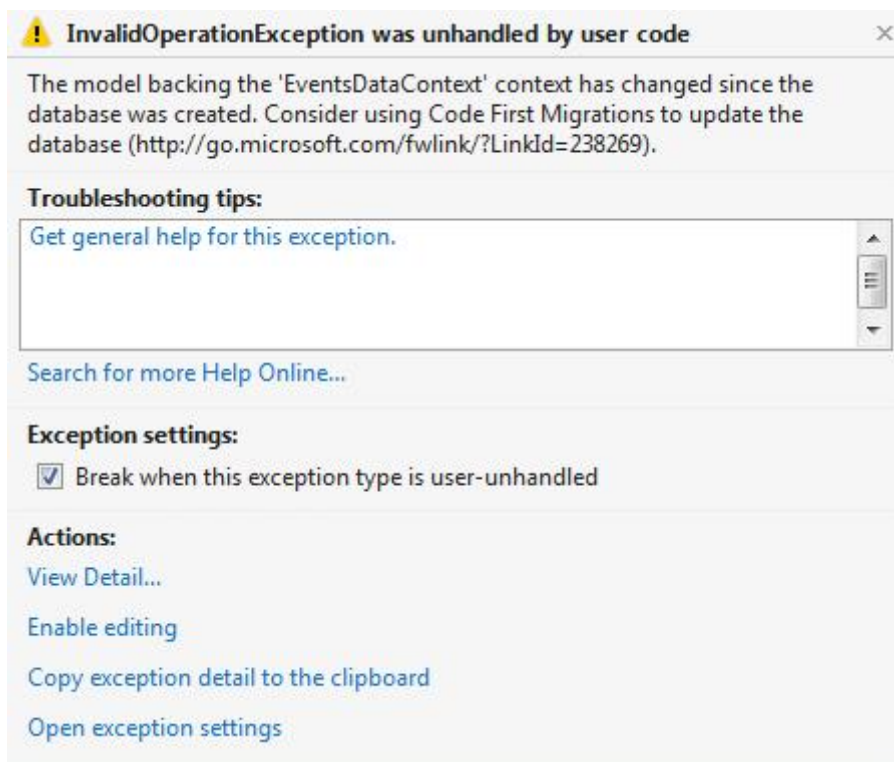
Muudame andmemudelit, lisades ürituse klassile kaks objekti juurde. Lisame asukoha objekti ja üritusele pildi lisamise objekti.

```
[Required]
[DataType(DataType.Text)]
[StringLength(maximumLength: 300)]
public string Location { get; set;}

[DataType(DataType.ImageUrl)]
[Display(Name = "Image Url")]
public string ImageUrl { get; set; }
```

Koodinäide 23. Models/Events.cs – Mudelile omaduste lisamine

Projekti käivitamisel saame veateate.



Joonis 11. Andmemudeli muutuse veateade

Veateatest näeme, et andmemudelit on muudetud ning nüüd tuleb muuta ka andmebaasi skeemi. Selleks kirjutame *Package Manager* konsoolis „Add-Migration AddLocation“, kus „AddLocation“ on migratsioonile antud nimi, mille hetkel tuletame sellest, et lisame mudelile asukoha objekti. Seejärel *Update-Database* muudatuste andmebaasi salvestamiseks. Add-

*Migration* abil saame oma andmemudeli uuendustest ülevaadet pidada. Migratsiooni lisamisel tekib sellekohta uus fail „Migrations“ kausta. Kuna automaatne migratsioon on lubatud oleks võinud *Add-Migration* käsu ka ära jätta. *Add-Migration* vaatab andmemudeli üle ja võrdleb seda olemasoleva andmebaasi skeemiga. Kui andmemudelis on midagi teistmoodi, siis paneb muutused andmebaasi uuendamiseks kirja ja *Update-Database* käivitab need muudatused.

Kui avame andmemudeli muudatuse kohta loodud migratsiooni faili näeme, et andmebaasi tuleb lisada *Location* ja *ImageUrl* veerud.

```
public partial class AddLocation : DbMigration
{
    public override void Up()
    {
        AddColumn("dbo.Events", "Location", c => c.String(nullable: false,
            maxLength: 300));
        AddColumn("dbo.Events", "ImageUrl", c => c.String());
    }

    public override void Down()
    {
        DropColumn("dbo.Events", "ImageUrl");
        DropColumn("dbo.Events", "Location");
    }
}
```

#### **Koodinäide 24. Muudatused andmebaasi lisamiseks**

Lisatud väljade esitamiseks vaadetes muudame *Index*, *Event* ja *Create* vaadet, lisades vastavad väljad.

```

<div class="editor-label">
    @Html.LabelFor(model => model.Location)
</div>
<div class="editor-field">
    @Html.EditorFor(model => model.Location)
    @Html.ValidationMessageFor(model => model.Location)
</div>

<div class="editor-label">
    @Html.LabelFor(model => model.ImageUrl)
</div>
<div class="editor-field">
    @Html.EditorFor(model => model.ImageUrl)
    @Html.ValidationMessageFor(model => model.ImageUrl)
</div>

```

**Koodinäide 25. Views/Events/Create.cshtml – Asukoha ja pildilingi väljade lisamine Create vaatesse**

```

<table>
    <tr>
        ... //olemas olevate väljade kuvamine
        <th>
            @Html.DisplayNameFor(model => model.Location)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.ImageUrl)
        </th>
        <th></th>
    </tr>

    @foreach (var item in Model) {
        <tr>
            ... //olemas olevate väljade kuvamine
            <td>
                @Html.DisplayFor(modelItem => item.Location)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.ImageUrl)
            </td>
        </tr>
    }
</table>

```

**Koodinäide 26. Views/Events/Index.cshtml - Asukoha ja pildilingi väljade lisamine Index vaatesse**

```
<div class="Event">
<h2>@Event.Title</h2>
  <div class="details">
    <p>Kirjeldus: @Event.Description</p>
    <p>Toimumise aeg: @Event.Date.ToString("g")</p>
    <p>Ürituse toimumise asukoht: @Event.Location</p>
    <p>Ürituse pilt: @Event.ImageUrl</p>
  </div>
</div>
```

**Koodinäide 27. Views/Events/Event.cshtml - Asukoha ja pildilingi väljade lisamine Event vaatesse**

### 3.5 Uue marsruudi loomine

Marsruudid defineerivad, kuidas sisestatud URL kindla kontrolleri ja meetodi peale suunatakse. Marsruudid tunnevad ära URL-e mustri järgi, mis jagab URL-i segmentideks ning segmente eraldatakse kaldkriipsuga. Iga segment võib sisaldada erinevaid sõnaliste väärtuste kui ka parameetri kohahoidjate kombinatsioone. Parameetri kohahoidjaid defineeritakse marsruudis loogsulgude sees. ASP.NET MVC tunneb ära spetsiaalsed parameetri kohahoidjad nagu „{controller}“ ja „{action}“, mille abil sobiv kontrolleri ja meetod leitakse. URL mustreid luues võime igasuguseid sõnaliste ja parameetri kohahoidjate kombinatsioone kasutada, peasi et parameetrite kohahoidjad on eraldatud kas eraldajaga (kaldkriips) või vähemalt ühe tähemärgilise sõnalise väärtusega.

Loome uue marsruudi selliselt, et URL sisaldaks ka ürituse nime. Selleks muudame *RouteConfig.cs* faili.

```

public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute(
        name: "Events",
        url: "Events/Details/{id}/{title}",
        defaults: new { controller = "Events", action = "Event" }
    );

    routes.MapRoute(
        name: "Default",
        url: "{controller}/{action}/{id}",
        defaults: new { controller = "Home", action = "Index", id =
            UrlParameter.Optional }
    );
}

```

**Koodinäide 28. App\_Start/RouteConfig.cs – Uue marsruudi loomine**

Lõime uue marsruudi nimega „Events“, mis lubab kasutada URL järgmisel kujul:

`http://localhost:port/Events/Details/Id/ÜrituseNimi`

Kuna URL ei viita kindlale failile failisüsteemis, siis see võimaldab vajadustele kohaldatavaid URL mustreid luua, mida näeme ka URL mustri „Events/Details/{id}/{title}“ defineerimisel. URL mustrit defineerimisel ei pea muutma kontrolleri nimesid.

Marsruutide registreerimise järjekord on oluline. Sisestatud URL-i võrreldakse defineeritud marsruutide mustritega ja peatutakse esimese sobiva URL mustri juures. Loodud mustrit kasutame rakenduses ära hiljem ning hetkel võib mustri välja kommenteerida.

### 3.6 Turvalisus ja autentimine

ASP.NET MVC lubab vaikumisi kõikidel kasutajatel kõiki kontrollereid ja nende meetodeid kasutada. Rakenduses tuleks piirata anonüümsete kasutajate võimalusi. Selleks on ASP.NET MVC raamistikus *Authorize* atribuut. Selle lisamine kontrolleri meetodile lubab ainult autenditud kasutajatel liigipääsu antud meetodile.

```
[Authorize]
public ActionResult Index()
{
    return View();
}
```

#### Koodinäide 29. Ligipääsu piiramise atribuudi lisamine

Samuti saab ka autenditud kasutajate ligipääsu kontrolleri meetoditele piirata. Täpsustades, millistel kasutajatel või millistel kasutaja rollidel on ligipääs meetodile.

```
[Authorize(Roles = "Administrator", Users = "Jaan, Kati") ]
public ActionResult Index()
{
    return View();
}
```

#### Koodinäide 30. Ligipääsu piiramise atribuudi võimalused

Ligipääsu kontrolleritele ja nende meetoditele saab ka globaalselt rakenduse piires määrata. Kasutades *AuthorizeAttribute* filtrit saavad kõikidele meetodile ligi ainult autenditud kasutajad. Anonüümsetele kasutajatele saab ligipääsu lubada *AllowAnonymous* atribuudi abil, määrates seda atribuuti kontrollerite meetoditele või kontrolleri klassile. *AuthorizeAttribute* filtri kasutamiseks tuleb filter lisada *RegisterGlobalFilters* klassi *FilterConfig.cs* failis.

```
public class FilterConfig
{
    public static void RegisterGlobalFilters(GlobalFilterCollection filters)
    {
        filters.Add(new HandleErrorAttribute());
        filters.Add(new AuthorizeAttribute());
    }
}
```

#### Koodinäide 31. App\_Start/FilterConfig.cs – Globaalse ligipääsupiirangu määramine

Ürituste rakenduses peaks ürituste lisamine toimuma ainult autenditud kasutajate poolt. Lisame *AuthorizeAttribute* filtri eelmainitud viisil ning täpsustame kuhu on anonüümsetele kasutajatel ligipääs. *Home* kontrolleris võivad anonüümised kasutajad kõikidele meetoditele ligi pääseda, seega anname tervele klassile *AllowAnonymous* atribuudi.

```
[AllowAnonymous]
public class HomeController : Controller
{
    ...
}
```

**Koodinäide 32. Controllers/HomeController.cs – Anonüümse ligipääsu võimaldamine Home kontrolleri**

*Events* kontrolleri tahame, et ainult autenditud kasutajad saaksid lisada uusi üritusi, seega lubame anonüümsete kasutajate ligipääsu *Index* ja *Event* meetoditele.

```
public class EventsController : Controller
{
    ...
    [AllowAnonymous]
    public ActionResult Index()
    {
        ...
    }
    [AllowAnonymous]
    public ActionResult Event(long id)
    {
        ...
    }
    [HttpGet]
    public ActionResult Create()
    {
        ...
    }
    [HttpPost]
    public ActionResult Create(models.Events Event)
    {
        ...
    }
}
```

**Koodinäide 33. Controllers/EventsController.cs – Anonüümse ligipääsu võimaldamine meetoditele**

Üritustel peaks olema ka olema looja ehk siis kasutaja, kes ürituse rakendusse lisab. Lisame andmemudelisse uue objekti ürituse looja kohta.

```

public class Events
{
    ...

    [DataType(DataType.Text)]
    public string EventCreator { get; set; }

}

```

#### Koodinäide 34. Models/Events.cs – Ürituse looja lisamine mudelisse

Ürituse looja nimi tuleb sisse logitud kasutaja nimest, seega ei pea me ürituse lisamise vaatesse uut välja looma. Kuid me ei taha ka, et kasutaja saaks teisi abivahendeid kasutades ürituse looja väljale andmeid sisestada. Selleks jätame ürituse looja *Post* meetodist välja, kasutades *Bind* atribuudi *Exclude* omadust. Ürituse looja väljale anname väärtuse kasutades *User.Identity.Name* omadust, mis annab hetkel sisselogitud kasutaja nime. Andmebaasi muudatuste tegemiseks *Package Manager* konsoolis kirjutame rea *Update-Database*.

```

[HttpPost]
public ActionResult Create([Bind(Exclude="EventCreator")]Models.Events Event)
{
    if (ModelState.IsValid)
    {
        //Andmebaasi lisamine
        var db = new EventsDataContext();
        Event.EventCreator = User.Identity.Name;
        db.Events.Add(Event);
        db.SaveChanges();

        return RedirectToAction("Index");
    }

    return Create();
}

```

#### Koodinäide 35. Controllers/EventsController.cs – Looja lisamine üritusele

Ürituse lisades antakse igale üritusele kaasa ka kasutaja, kes selle ürituse tegi. Ürituste *Index* vaates kuvatakse meile ka *Edit*, *Details* ja *Delete* lingid. *Edit* ja *Delete* lingid võiks kuvada ainult siis kui sisselogitud kasutaja on selle ürituse loonud. Selleks kontrollime kõigepealt, et kasutaja oleks sisselogitud. Seejärel kontrollime kas ürituse looja on sama, mis sisselogitud kasutaja. Kui sisselogitud kasutaja on ürituse looja, siis kuvatakse vastavad lingid. *Details* lingi kaotame ära ja asendame ürituse pealkirja lingiga konkreetse ürituse peale. Nüüd on hea võimalus kasutada ära ka eelvalt loodud marsruuti, mis lisab URL-ile ka ürituse pealkirja. Selleks eemaldame kommentaarid marsruudilt ning lisame ürituse pealkirja lingile ka pealkirja atribuudi, milles tühemikud asendame sidekriipsuga.



```

@foreach (var item in Model) {
    <tr>
        <td>
            @Html.ActionLink(item.Title, "Event", new { id = item.Id, title =
Server.HtmlDecode(item.Title).Replace(" ", "-") })
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Description)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Date)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Location)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.ImageUrl)
        </td>
        <td>
            @if(User.Identity.IsAuthenticated){
                if(item.EventCreator == User.Identity.Name){
            <td>
                @Html.ActionLink("Edit", "Edit", new { id=item.Id }) |
                @Html.ActionLink("Delete", "Delete", new { id=item.Id })
            </td>
                }
            }
        </td>
    </tr>
}
}

```

**Koodinäide 36. Views/Events/Index.cshtml – Vaate kohandamine autentimise kasutamisel**

### 3.7 Ürituse muutmise ja kustutamise funktsionaalsuse lisamine

Ürituse lisanud kasutaja võiks saada ka oma lisatud üritusi muuta ja kustuda. Selleks lisame muutmise ja kustutamise meetodid kontrollerrisse.

```

[HttpGet]
public ActionResult Edit(long id = 0)
{
    Events Event = db.Events.Find(id);
    if (Event == null)
    {
        return HttpNotFound();
    }
    if (Event.EventCreator == User.Identity.Name)
    {
        return View(Event);
    }
    TempData["error"] = "Kasutaja pole ürituse looja";
    return RedirectToAction("Index");
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Edit(Events Event)
{
    if (ModelState.IsValid)
    {
        db.Entry(Event).State = EntityState.Modified;
        Event.EventCreator = User.Identity.Name;
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(Event);
}

```

**Koodinäide 37. EventsController.cs – Ürituse muutmise funktsionaalsuse lisamine**

```

[HttpGet]
public ActionResult Delete(long id = 0)
{
    Events Event = db.Events.Find(id);
    if (Event == null)
    {
        return HttpNotFound();
    }
    if (Event.EventCreator == User.Identity.Name)
    {
        return View(Event);
    }
    TempData["error"] = "Kasutaja pole ürituse looja";
    return RedirectToAction("Index");
}

[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public ActionResult DeleteConfirmed(long id)
{
    Events Event = db.Events.Find(id);
    if (Event.EventCreator == User.Identity.Name)
    {
        db.Events.Remove(Event);
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    TempData["error"] = "Kasutaja pole ürituse looja";
    return RedirectToAction("Index");
}

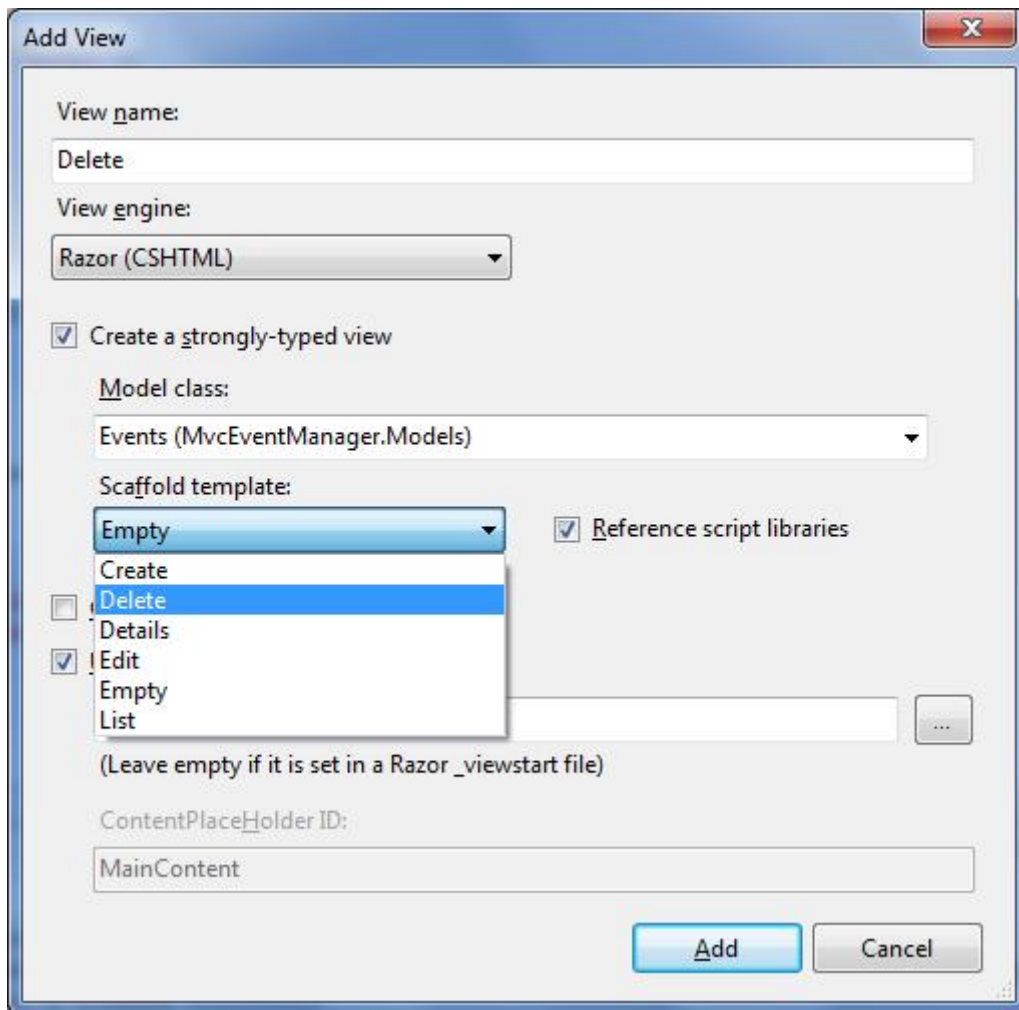
```

#### Koodinäide 38. Controllers/EventsController.cs – Ürituse kustutamise funktsionaalsuse lisamine

Sarnaselt *Create* meetodile on *Edit* ja *Delete* meetoditel *HttpGet* ja *HttpPost* atribuudid. *Edit* meetodi poole pöördumisel otsitakse andmebaasist vastava *Id*-ga üritust. *Id* tuleb veebilehitseja aadressirealt. Kui andmebaasist leitakse üritus ja sisselogitud kasutaja on ürituse loonud, siis kuvatakse vorm muudatuste tegemiseks. *Edit* meetodi *HttpPost* atribuudi osa määrab ära, et antud ürituse instantsi on muudetud, andes *db.Entry(Event)* oleku väärtuseks *Modified*. *SaveChanges* meetodi kasutamine salvestab uuendused andmebaasi. Oluline on, et uuendatakse kõiki andmebaasi veerge, isegi neid mida kasutaja ei muutnud. Kuna andmemudel is ürituse looja objekti kasutaja muuta ei tohiks, vaid see peaks automaatselt sisselogitud kasutaja olema, siis tuleb määrata ürituse looja uuesti, sest vormi kaudu seda sisestada ei tohi. Seejärel kontrollitakse kasutades *ModelState.IsValid* meetodit, kas vorm on valideeritud. Kui sisestatud andme on korrektsed tehakse vastavad muudatused ning kui ei ole, siis kuvatakse vorm uuesti. Kustutamise meetodi *Delete* väljakutsumisel kuvatakse vastava *Id*-ga ürituse kustutamise kinnituse vaade. Sarnaselt muutmise meetodile

kontrollitakse, et ürituse lisanud kasutaja oleks sama, mis sisselogitud kasutaja ning kinnitamisel vastav üritus andmebaasist kustutatakse. Kasutades *TempData* objekti kuvame veateate kui kasutaja soovib muuta või kustuda üritust, mida tema ei ole loonud.

Loome muudatamisega ja kustutamisega seotud vaated. Mõlemalt puhul loome tugevatüübilised vaated ning valime malliks vastavalt *Edit* ja *Delete*.



Joonis 12. Vaadete loomine Edit ja Delete meetoditele

Muutmise vaates kustutame ära ürituse loojaga seotud read, sest nagu eelpool mainitud ei tohiks kasutaja saada seda muuta.

```

<div class="editor-label">
    @Html.LabelFor(model => model.EventCreator)
</div>
<div class="editor-field">
    @Html.EditorFor(model => model.EventCreator)
    @Html.ValidationMessageFor(model => model.EventCreator)
</div>

```

**Koodinäide 39. Views/Events/Edit.cshtml – Kustutamisele kuuluvad read Edit vaates**

*TempData* kaudu veateate kuvamiseks lisame *Index* vaatele järgneva rea sinna, kus tahame et veateade kuvatakse.

```

@if (TempData["error"] != null)
{
    <div class="error-message">@TempData["error"]</div>
}

```

**Koodinäide 40. Views/Events/Index.cshtml – TempData abil veateate kuvamine**



**Joonis 13. Ürituse loojaga seotud veateate kuvamine rakenduses**

### 3.8 Cross-Site Request Forgery

ASP.NET MVC vaadete genereerimisel, mis kasutavad vorme, lisatakse ka *Html.AntiForgeryToken* abiline. See abiline hoiab ära *CSRF* (*Cross Site Request Forgery*) rünnakuid. Lühitalt öelduna on *CSRF* rünnakud sellised, kus pahatahtlik veeb saadab päringuid rünnaku vastu nõrgale veebilehele sinna sisselogitud kasutaja läbi. *Html.AntiForgeryToken* abiline loob vormi *Hidden* tüüpi välja, mille väärtuseks genereerib

juhusliku koodi. Genereeritakse ka teine kood, mis saadetakse serverisse küpsisena. Lisades kontrolleri meetodile *ValidateAntiForgeryToken* atribuudi, kontrollitakse et vormis ja küpsises olev kood oleksid samad. Lisame atribuudi *Create* meetodile. Vaatele abilist ei pea lisama, sest see on juba olemas.

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create([Bind(Exclude="EventCreator")]Models.Events Event)
{
    ...
}
```

**Koodinäide 41. Controllers/EventsController.cs – ValidateAntiForgeryToken atribuudi lisamine**

Käivitame projekti ja navigeerime ürituse loomise lehele ning vaatame veebilehitsejast lähtekoodi.

```
<form action="/Events/Create" method="post">

<input name="__RequestVerificationToken" type="hidden"
value="VfQ7XSXQr8hoxWfPZwaJx9xYeUthR_c7bCnMQUyR7KTwmpDeuMLrkgbX0_ei2RjkRctoo3lY
_oC1lyF1L4p0PZcsv_tXikjdOM3xT3O6cFA1" />
```

ASP.NET MVC on genereerinud hidden tüüpi välja nimega „\_\_RequestVerificationToken“ ja millele on antud juhuslik väärtus. Selle koodi valideerimisel saabki kindlaks teha, et post meetodiga tulnud päringud on tulnud enda serverist.

### 3.9 Üritusel osalemise funktsionaalsus.

Järgmiseks funktsionaalsuseks sooviks, et autentitud kasutajad saaksid ennast üritusel osalejaks määrata. Loomeme andmemudelisse uue klassi nimega „Participants“. Klassile lisame *Id*, *EventId* ja *Participant* objektid. Nende kaudu saame teada, millised osalised üritustel on.

```
public class Participants
{
    public long Id { get; set; }
    public long EventId { get; set; }
    public string Participant { get; set; }
}
```

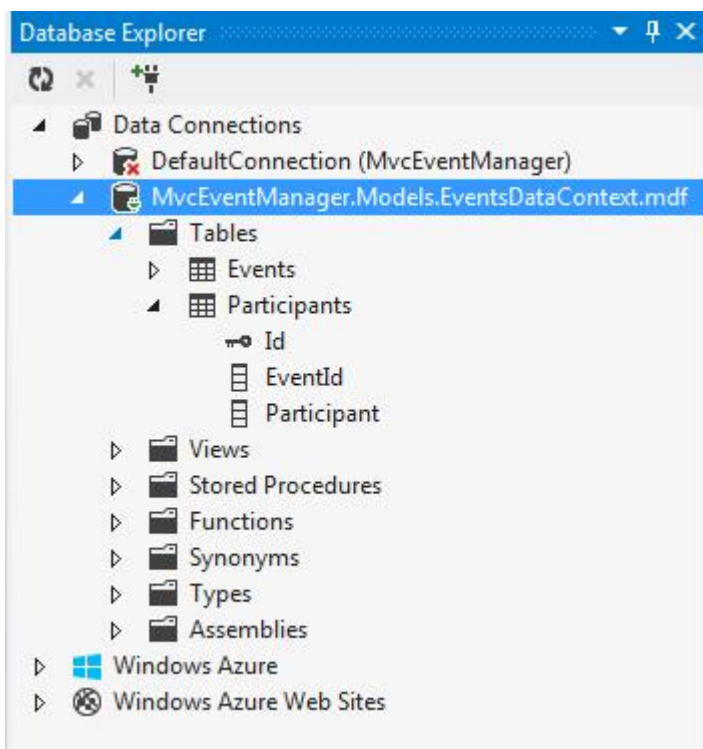
**Koodinäide 42. Models/Events.cs – Osaliste klassi loomine mudelisse**

Uus klass peaks andmebaasis olema eraldi tabelina, siis loome ka uue *DbSet* andmeobjekti. Selleks muudame olemasolevat *EventDataContext* klassi.

```
public class EventsDataContext : DbContext
{
    public DbSet<Events> Events { get; set; }
    public DbSet<Participants> Participants { get; set; }
}
```

**Koodinäide 43. Models/EventsDataContext.cs – Osaliste DbSet andmeobjekti loomine**

Teeme muudatused ka andmebaasis, käivitades *Package Manager* konsoolist update-database käsu. Avame „App\_Data“ kaustast oma andmebaasi ja vaatame, mis muudatused tehti.



**Joonis 14. Osaliste tabel andmebaasis**

Näeme, et lisati uus tabel eelpool mainitud väljadega. Järgmiseks lisame kontrolleri meetodi, mis hakkab kasutajaid üritustele osalejateks määrama.

```

public ActionResult Participate(long Id)
{
    if (User.Identity.IsAuthenticated)
    {
        IEnumerable<Participants> currentParticipants = db.Participants.Where(p
=> p.EventId == Id).ToList();
        var alreadyParticipating = currentParticipants.Where(p => p.Participant
== User.Identity.Name);

        Participants participant = new Participants();
        participant.EventId = Id;
        participant.Participant = User.Identity.Name;

        if (alreadyParticipating.Count() == 0)
        {
            db.Participants.Add(participant);
            db.SaveChanges();
            return RedirectToAction("Index");
        }
        else
        {
            TempData["error"] = "Kasutaja juba osaleb";
            return RedirectToAction("Event", new { id = Id });
        }
    }
    return RedirectToAction("Event", new { id = Id });
}

```

#### Koodinäide 44. Controllers/EventsController.cs – Üritusel osalemise funktsionaalsuse lisamine

*Participate* kontrolleri meetodile anname kaasa ürituse *Id*. Seejärel kontrollime, et kasutaja oleks autenditud. Loome objekti *currentParticipants*, mis otsib andmebaasi *Participants* tabelist sissetuleva *Id*-ga üritusel kõik osalevad kasutajad. Seejärel loome *alreadyParticipating* objekti, mis kontrollib, kas autenditud kasutaja on juba ürituse osalejate nimekirjas. Kui ei ole, siis lisatakse osaleja isend andmebaasi.

Järgmisena tuleks lisada *Event* vaatele üritusel osalemise link ning *Event* vaates võiks kuvada ka üritusel osalevate kasutajate nimekiri. Kuna reaga „*@model MvcEventManager.Models.Events*“ on mudel seotud *Events* mudeliga, siis hetkel seda teha ei saa. Selleks, et vaates mitut mudelit kuvada tuleb luua vaatemudel. Loome selle uue klassina *Events* andmemudelisse.



```

public class EventsViewModel
{
    public Events Event { get; set; }
    public IEnumerable<Participants> Participant { get; set; }
}

```

#### Koodinäide 45. Models/Events.cs – Vaatemudeli loomine

Seejärel muudame *Event* vaadet, lisades üritusele osalemise lingi, veateate kuvamise ning üritusele osalejate nimekirja kuvamise.

```

@model MvcEventManager.Models.EventsViewModel
@if (TempData["error"] != null)
{
    <div class="error-message">@TempData["error"]</div>
}

<div class="Event">

<h2>@Model.Event.Title</h2>
    <div class="details">
        <p>Kirjeldus: @Model.Event.Description</p>
        <p>Toimumise aeg: @Model.Event.Date.ToString("g")</p>
        <p>Ürituse toimumise asukoht: @Model.Event.Location</p>
        <p>Ürituse pilt: @Model.Event.ImageUrl</p>

    </div>
    @if(Model.Participant.Count() != 0){
    <p>Üritusel osalejad:
    @foreach(var item in Model.Participant){
        <span>@item.Participant | </span>
    }
    </p>
    }
    @Html.ActionLink("Osale", "Participate", new { id=Model.Event.Id })
</div>

```

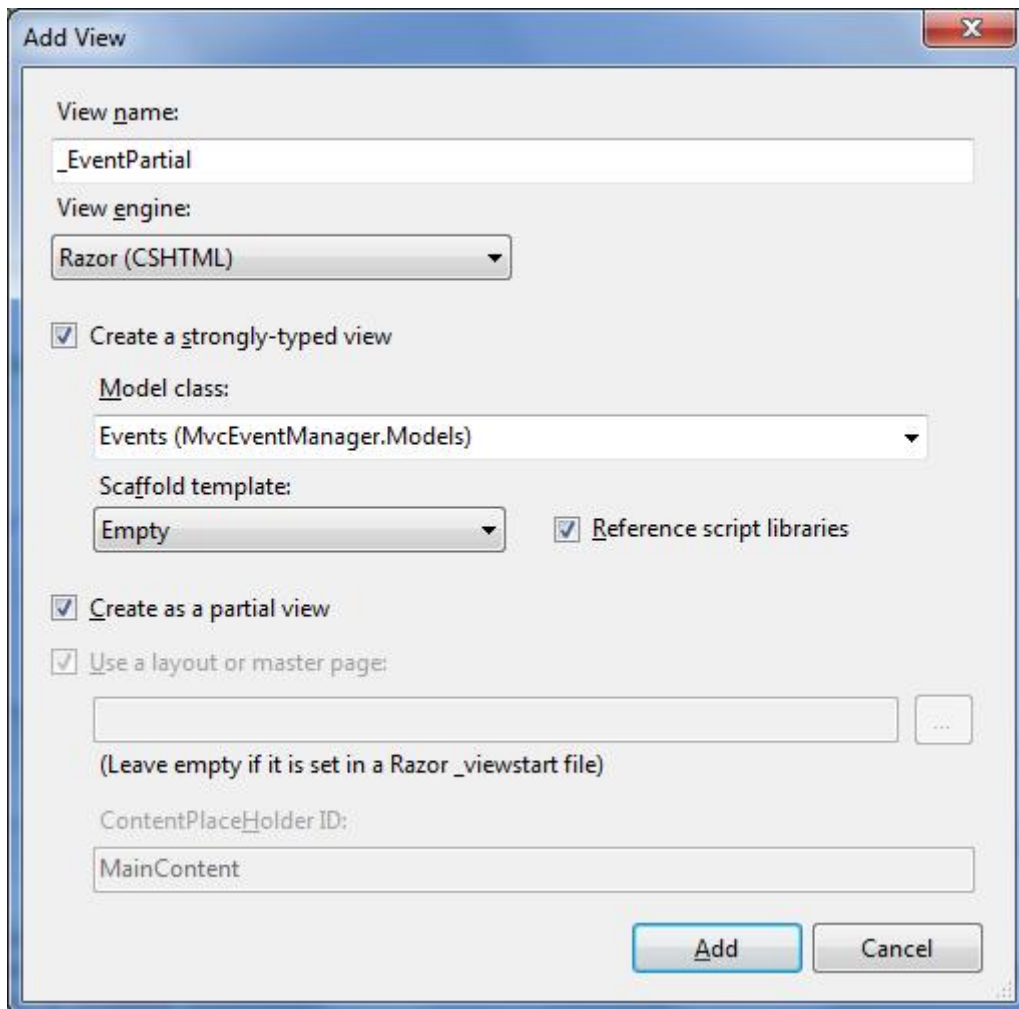
#### Koodinäide 46. Views/Events/Event.cshtml – Vaatemudeli kasutamine vaates

Nagu näha seoti mudel seekord loodud *EventsViewModel* vaatemudeliga. Üritusel osalejate nimekiri kuvatakse ainult siis kui seal on osalejaid.

### 3.10 Partial tüüpi vaate

Kujutame ette olukorda, kust tahame oma ürituse andmemudelile veel objekte juurde lisada. Me peaks erinevatele vaadetele objektide kohta uued väljad looma ja see tähendaks mitme faili muutmist. Kui me vaatame *Edit* ja *Create* vaadet, siis suur osa koodi on seal sama. Me

saame kasutada *Html.Partial* abilist, mille abil saame oma koodi lihtsamaks ja kergemini loetavaks muuta. Loo uue vaate nimega *\_EventPartial* „Views/Shared“ kausta. Alljooon on konventsiooniks tähistamiseks vaateid, mis ei ole mõeldud eraldi kuvamiseks, vaid mida teised vaated kasutavad. Loo tugeva-tüübilise vaate ning valime malliks *Events* mudeli. Samuti märgistame ära *Create as partial view* välja.



Joonis 15. Partial tüüpi vaate loomine

Lisame *\_EventPartial* faili *Edit* ja *Create* vaadete ühise osa.

```

@model MvcEventManager.Models.Events
<div class="editor-label">
    @Html.LabelFor(model => model.Title)
</div>
<div class="editor-field">
    @Html.EditorFor(model => model.Title)
    @Html.ValidationMessageFor(model => model.Title)
</div>

<div class="editor-label">
    @Html.LabelFor(model => model.Description)
</div>
<div class="editor-field">
    @Html.EditorFor(model => model.Description)
    @Html.ValidationMessageFor(model => model.Description)
</div>

<div class="editor-label">
    @Html.LabelFor(model => model.Date)
</div>
<div class="editor-field">
    @Html.EditorFor(model => model.Date)
    @Html.ValidationMessageFor(model => model.Date)
</div>

<div class="editor-label">
    @Html.LabelFor(model => model.Location)
</div>
<div class="editor-field">
    @Html.EditorFor(model => model.Location)
    @Html.ValidationMessageFor(model => model.Location)
</div>

<div class="editor-label">
    @Html.LabelFor(model => model.ImageUrl)
</div>
<div class="editor-field">
    @Html.EditorFor(model => model.ImageUrl)
    @Html.ValidationMessageFor(model => model.ImageUrl)
</div>

```

**Koodinäide 47. Views/Shared/\_EventsPartial.cshtml – Create ja Edit vaate ühine osa**

Seejärel kommenteerime kopeeritud osa esialgu mõlemast vaatest välja. Seejärel lisame *Html.Partial* abilise, mis meile ühise vaate *\_EventPartial* kuvab.

```
<fieldset>
  <legend>Events</legend>

  @Html.Partial("_EventPartial")
  <p>
    <input type="submit" value="Create" />
  </p>
</fieldset>
```

**Koodinäide 48. Views/Events/Create.cshtml ja Views/Events/Edit.cshtml – Partial vaate esitamine**

Käivitage projekti ja veendume, et abiline toimib ning võib väljakommenteeritud koodi ära kustuda.

### 3.11 Ülesanne 2

1. Luua blogi koos muutmise ja kustutamise funktsionaalsustega
2. Lisada blogi postitustele kommenteerimise funktsionaalsus (kommentaari mudel)
3. Piirata rakenduses anonüümsete kasutajate ligipääsu. Anonüümsed kasutajad ei tohiks saada kommenteerida blogi postitusi.

### 3.12 Terviklik kujundus ja paigutus

Veebis on sama asja kuvamiseks mitmetel lehtedel ainult üks viis, kuvada sama HTML igal lehel. Korduvat koodi on aga väga raske lugeda ja muuta. Vaatades oma veebilehte näeme, et meile kuvatakse palju rohkem HTML-i kui meie loodud vaadetes seda on. Seda lisa HTML-i kutsutakse ASP.NET MVC *layout*-iks. *Layout* koosneb HTML-ist, mida terve veebirakenduse raames kasutatakse. *Layout* on vaade, kus on ära defineeritud veebilehe üldine ülesehituslik mall ning millele teised vaated viitavad. Seega kogu korduv HTML, mis igal veebilehel on, paikneb kõik ühes failis. Avame „Views/Shared“ kaustas oleva *\_Layout.cshtml* faili ning uurime sisu.

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>@ViewBag.Title - My ASP.NET MVC Application</title>
    <link href="~/favicon.ico" rel="shortcut icon" type="image/x-icon" />
    <meta name="viewport" content="width=device-width" />
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
  </head>
  <body>
    <header>
      <div class="content-wrapper">
        <div class="float-left">
          <p class="site-title">@Html.ActionLink("your logo here", "Index",
"Home")</p>
        </div>
        <div class="float-right">
          <section id="login">
            @Html.Partial("_LoginPartial")
          </section>
          <nav>
            <ul id="menu">
              <li>@Html.ActionLink("Home", "Index", "Home")</li>
              <li>@Html.ActionLink("About", "About", "Home")</li>
              <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
            </ul>
          </nav>
        </div>
      </div>
    </header>
    <div id="body">
      @RenderSection("featured", required: false)
      <section class="content-wrapper main-content clear-fix">
        @RenderBody()
      </section>
    </div>
    <footer>
      <div class="content-wrapper">
        <div class="float-left">
          <p>&copy; @DateTime.Now.Year - My ASP.NET MVC Application</p>
        </div>
      </div>
    </footer>

    @Scripts.Render("~/bundles/jquery")
    @RenderSection("scripts", required: false)
  </body>
</html>

```

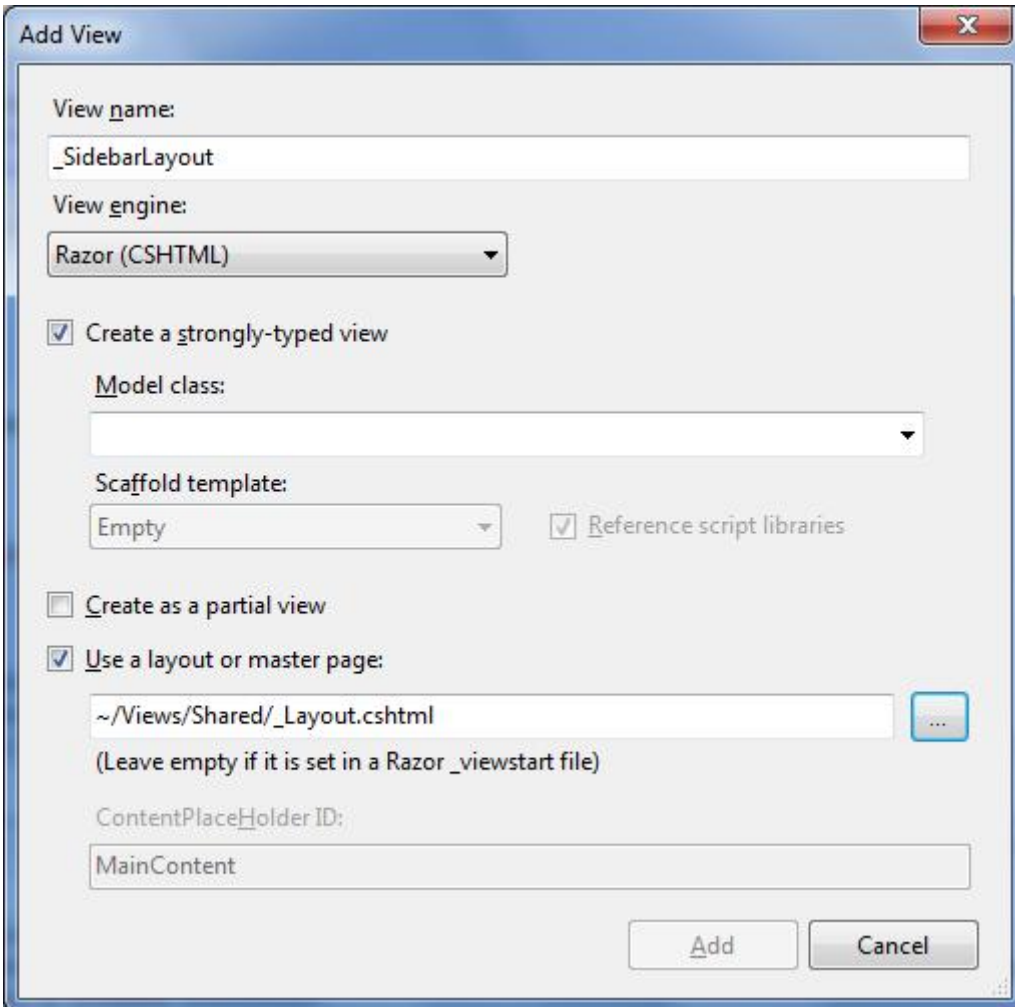
#### Koodinäide 49. Views/Shared/\_Layout.cshtml. Veebirakenduse terviklik kujundus

HTML *Body* sisse jäävad kolm osa: *header*, *div id*-ga „body“ ning *footer*. Päise osas on näeme, et meile on loodud logo, menüü ning sisselogimis moodul. Kõige olulisem kogu *Layout* osas on *RenderBody* meetod. Selle meetodi kutsumine hakkabki kuvama loodud vaateid vastavasse kohta.

Oma vaadete kaustas näeme ka faili `_ViewStart.cshtml`. Seda vaadet rakendatakse kõikidele kaustas ning alamkaustades olevatele vaadetele. Ehk millal iganes Razor vaatemootor vaadet käivitab, käivitab ta kõigepealt `_ViewStart.cshtml` faili, mis laeb `_Layout.cshtml` faili ning seejärel kuvab vaate.

### 3.12.1 Layout loomine

Uue `layout`-i lisamiseks loome uue vaate „Views/Shared“ kausta. Paneme nimeks „\_SidebarLayout“ ning jätame vaikimisi valikud. Määrame ära ainult, et kasutaks `_Layout.cshtml` faili `layout`-ina. Selle võib ka tühjaks jätta, kuna meil on `viewstart` failis see juba määratud.



The screenshot shows the 'Add View' dialog box with the following configuration:

- View name: `_SidebarLayout`
- View engine: `Razor (CSHTML)`
- Create a strongly-typed view
  - Model class: (empty)
  - Scaffold template: `Empty`
  - Reference script libraries
- Create as a partial view
- Use a layout or master page
  - Path: `~/Views/Shared/_Layout.cshtml`
  - (Leave empty if it is set in a Razor `_viewstart` file)
- ContentPlaceHolder ID: `MainContent`

Buttons: `Add` and `Cancel`

Joonis 16. Layout-i loomine

Loome küljeriba struktuuri, kus muudame *content* sektsiooni peamises *layout* failis. Lisades *RenderBody* meetodi *content* sektsiooni ning küljeriba kuvamise *RenderSection* meetodi abil *sidebar* sektsiooni.

```
@{
    ViewBag.Title = "_SidebarLayout";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<div class="sidebar">
    @RenderSection("sidebar")
</div>
<div class="content">
    @RenderBody()
</div>
```

**Koodinäide 50. Views/Shared/\_SidebarLayout.cshtml – Küljeribaga layout-i defineerimine**

Kujundame küljeriba, muutes „Content“ kaustas *Site.css* faili. Lisame faili lõppu alljärgneva.

```
.sidebar {
    width: 10em;
    margin: 1em;
    padding: 1em;
    min-height: 30em;
    float: right;
    background: #7ac0da;
}
```

**Koodinäide 51. Content/Site.css – Küljeriba kujundus**

Selleks, et vaated *layout*-i kasutaks tuleb seda vaadetele öelda. Lisame *Home* kontrolleri *Index* vaadetele küljeriba, mis kuvaks ürituste lingi.

```
@{
    ViewBag.Title = "Home Page";
    Layout = "~/Views/Shared/_SidebarLayout.cshtml";
}

@section sidebar {
    @Html.ActionLink("Üritused", "Index", "Events")
}
```

**Koodinäide 52. Views/Home/Index.cshtml – Küljeriba esitamine**

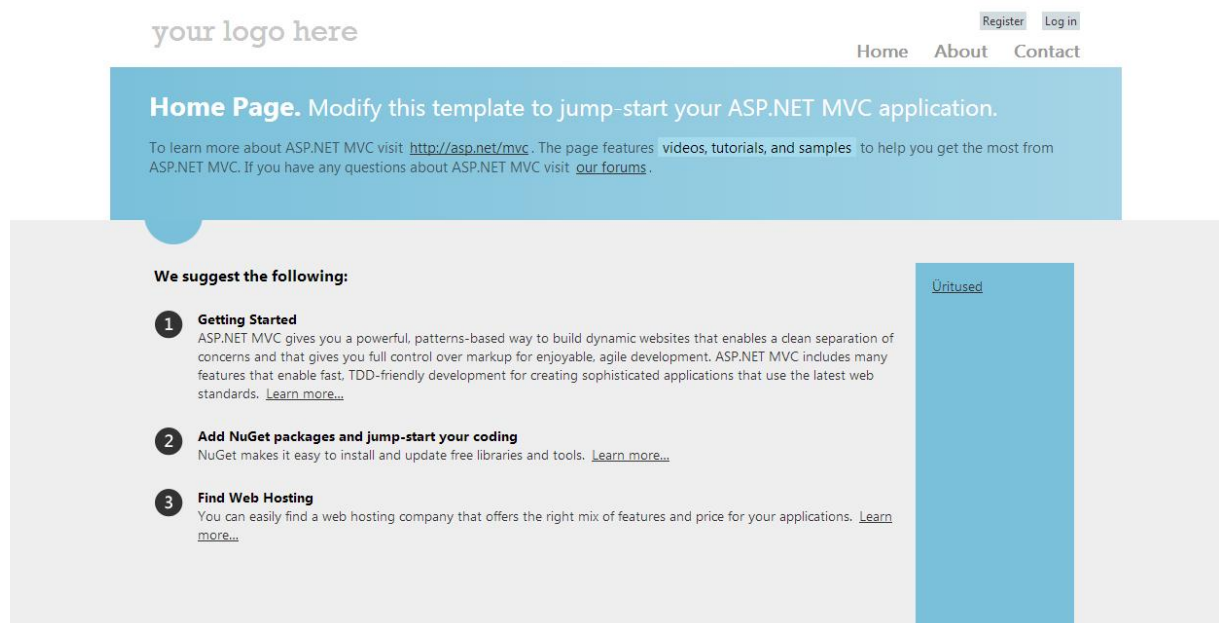
Kui projekti käivitame saame veateate, et sektsioonid *Featured* ja *Scripts* on defineeritud, kuid neid ei renderdata. Kuna nad on defineeritud *\_Layout.cshtml* failis ja nende definitsioon

ei kandu edasi, siis peame nad defineerima ka `_SidebarLayout.cshtml` failis. Me ei nõua, et need sektsiooni sisu omavad, seega anname `required` parameetrile väärtuse `false`.

```
@{
    ViewBag.Title = "_SidebarLayout";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
@section Featured{
    @RenderSection("Featured", required: false)
}
@section Scripts{
    @RenderSection("Scripts", required: false)
}
<div class="sidebar">
    @RenderSection("sidebar")
</div>
<div class="content">
    @RenderBody()
</div>
```

**Koodinäide 53. Views/Shared/\_SidebarLayout.cshtml – Sektsioonide defineerimine**

Käivitades projekti nüüd näeme oma vaates tulemust.



**Joonis 17. Küljeriba esitamine veebilehitsejas**



### 3.13 Veebirakenduse optimeerimine

Veebirakenduse optimeerimise eesmärgiks on tagada kasutajale parim kasutajakogemus (ingl. k. *UX* ehk „*User Experience*“). Meie tahame, et meie veebirakendus reageeriks kasutaja päringule nii kiiresti kui võimalik. Seejärel, et päringu vastus saadetakse kliendile nii kiiresti kui võimalik ning kasutajale esitatakse veebileht nii kiiresti kui võimalik.

Projekti loomisel *Internet Application* malli kasutamisel lisati meie projektile erinevaid pakette. Lisatud pakette näeme *packages.config* failis. Sarnaselt näeme projektile installeeritud pakette kui valime menüüst *PROJECT->Manage NuGet Packages*. NuGet on paketihaldur Microsofti .NET raamistikus. Selle abil on väga lihtne pakette lisada ja neid hallata. Samamoodi on pakette lihtne uuendada ilma projektis muudatusi tegemata. Meie projektile on lisatud pakette, mis aitavad meid veebilehe optimeerimisel. Kasutame Microsofti veebi optimeerimise raamistiku ning *webgrease* tööristakomplekti.

#### 3.13.1 Puhverdamine

Üheks viisiks, et veebirakendus võimalikult optimaalselt andmebaasiga suhtleks kasutame ära puhverdamist. Puhverdamine on andmete salvestamine teatud perioodiks, et neid ei peaks serverist uuesti pärima.

Lisades väljundi puhverdamise oma projektile saame puhverdada terveid lehekülgi oma projektis. Küll tasub mõelda kus on seda otstarbekas kasutada. Oma projekti näitel *Events* kontrolleri *Index* meetod kuvab meile olemasolevad üritused andmebaasist. Lehe uuesti laadimisel saadetakse uus päring ja saadakse uued andmed andmebaasist. Väljundi puhverdamisega saame aga salvestada andmed uuesti kasutamiseks nii, et kasutaja ei pea igakord uuesti andmed pärima. Kui aga meie veebirakenduses palju kasutajaid ja üritusi lisandub pidevalt juurde, siis ei ole võibolla otstarbekas väljundi puhverdamist kasutada, sest kasutaja võib saada aegunud andmeid.

Oma projektis näeme, et väljundi puhverdamist oleks arukas kasutada *Home* kontrolleri kõikidel meetoditel, kuna seal olev info muutub väga harva. Seega lehe uuesti laadimisel laetakse puhvrise salvestatud vaade. Lisame *OutputCache* atribuudi tervele klassile *HomeController* failis ning lisame kellaja kuvamise, et saaks veenduda puhvri toimimises.

```

[OutputCache(Duration=5)]
[AllowAnonymous]
public class HomeController : Controller
{
    public ActionResult Index()
    {
        ViewBag.Message = DateTime.Now;

        return View();
    }
}

```

**Koodinäide 54. Controllers/HomeController.cs – Puhvri lisamine Home kontrolleri**

Tulemusena näeme, et kellaeg muutub iga 5 sekundi tagant. *OutputCache* atribuudi välja kommenteerimisel muutub kellaeg peale igat lehekülje laadimist.

Vaikimisi väljundi puhverdamise kasutamisel salvestatakse puhver kolme asukohta: serverisse, veebilehitsejasse ja puhverserverisse. Lisades *Location* omaduse *OutputCache* atribuudile saab kontrollida kuhu puhver salvestatakse. *Location* omadusele saab anda väärtuseid: *Any*, *Client*, *Downstream*, *Server*, *None* ja *ServerAndClient*. Vaikimisi on väärtus *Any*.

Eelnevalt mainisin, et tuleb olla ettevaatlik puhverdamisel. Kui meie veebirakendust kasutavad paljud inimesed ja me ei taha aegunud andmeid kuvada, kas siis tasub väga lühikese ajalist puhverdamist teha? Seda kutsutakse mikropuhverdamiseks ja tasub küll. Kujutame ette, et meie rakenduses laevad sada inimest korraga *Events* kontrolleri *Index* meetodi. Kui puhverdamist ei ole, siis tehakse andmebaasi sada päringut ja saadetakse sada vastust. Kui meil on aga puhverdus, tehakse ainult üks päring. Lisame oma rakenduses ühe sekundilise kestvusega *OutputCache* atribuudid *Events* kontrolleri *Index* ja *Event* meetoditele.

```

[OutputCache(Duration = 1)]
[AllowAnonymous]
public ActionResult Index()
{
    ...
}
[OutputCache(Duration = 1)]
[AllowAnonymous]
public ActionResult Event(long id)
{
    ...
}

```

**Koodinäide 55. Controllers/EventsController.cs – Mikropuhverduse lisamine Index ja Event meetoditele**

Staatiliste andmete kuvamiseks mingil veebilehel osal ei ole arukas kasutada terve veebilehe puhverdamist. Ütleme näiteks, et tahame oma ürituste vaates küljeribal kuvada staatilisi andmeid. Näiteks kui meil oleks andmemudelil defineeritud ürituste kategooria või ürituste asukoht selliselt, et kasutaja nendele väärtust anda ei saa, küll aga saab valida olemasolevate väärtuste hulgast sobiva. Kuna selliseid objekte pole, siis tahame kuvada küljeribas kolme esimese ürituse pealkirja lingina. Selleks kasutama ära ASP.NET MVC tütarmeetodi võimalust. Lisame kontrollersse uue tütarmeetodi nimega „firstThree“, mis hakkab esimest kolme üritust andmebaasist kuvama. Kuna iga *public* tüüpi meetodit saab URL-i kaudu välja kutsuda, siis kasutame *ChildActionOnly* atribuuti, mida saab välja kutsuda ainult vaatest.

```

[AllowAnonymous]
[ChildActionOnly]
public ActionResult firstThree()
{
    ViewBag.first = db.Events.OrderBy(x => x.Id).Take(3).ToArray();
    return PartialView();
}

```

**Koodinäide 56. Controllers/EventsController.cs – Tütarmeetodi loomine**

Tulemuse väljastame osalisse vaatesse, mille loome järgmisena. Valikutest valida, et vaade luuakse *partial* tüüpi vaadena (*Create as a partial view*).

```
<h2>Esimesed 3 üritust</h2>
  @foreach(var a in ViewBag.first){
    <p>@Html.ActionLink((string)a.Title,"Event", new{id=a.Id})</p>
  }
```

**Koodinäide 57. Views/Events/firstThree.cshtml – Esimese kolme ürituse pealkirja esitamine**

Seejärel muudame *Events* kontrolleri *Index* vaadet, lisades sellele küljeriba kuvamise. Küljeriba kuvamise sektsioonis kasutame *Html.Action* abilist, mille abil kuvame *firstThree* meetodi *Events* kontrollerris.

```
@{
  ViewBag.Title = "Index";
  Layout = "~/Views/Shared/_SidebarLayout.cshtml";
}
@section sidebar{
  @Html.Action("firstThree", "Events")
}
```

**Koodinäide 58. Views/Events/Index.cshtml – firstThree tütarmeetodi esitamine küljeribal**

Käivitame projekti ja vaatame tulemust.



**Joonis 18. Tütarmeetodi esitamine veebilehitsejas**

Lisame nüüd väljundi puhvri *firstThree* meetodile. Lisame samuti kellaaja kuvamise, et saaks kontrollida puhvri toimimist tervest vaatest eraldi.

```

[OutputCache(Duration = 1)]
[AllowAnonymous]
public ActionResult Index()
{
    var Event = db.Events.ToArray();
    ViewBag.message = DateTime.Now;
    return View(Event);
}
[AllowAnonymous]
[ChildActionOnly]
[OutputCache(Duration = 3600)]
public ActionResult firstThree()
{
    ViewBag.first = db.Events.OrderBy(x => x.Id).Take(3).ToArray();
    ViewBag.message = DateTime.Now;
    return PartialView();
}

```

#### Koodinäide 59. Controllers/EventsController.cs – Puhvri lisamine tütarmeetodile

Lisame *ViewBag.message* koodirea *Index.cshtml* faili ning *firstThree.cshtml* faili. Käivitamisel näeme, et kellaag küljeribal muutub ainult iga tunni aja tagant ning Index vaatele lisatud kellaag muutub iga sekundi järel, mis tähendab et meie poolt lisatud puhverdamine toimib.

### 3.13.2 Bundling ja minification

Selleks, et kliendi ja serveri vahel võimalikult vähe andmeid liiguks kasutame ära *bundling* ja *minification* võimalusi. Enamus peamisest veebilehitsejatest võimaldavad kuute üheaegset ühendust hostinimele. See tähendab, et kui kuute päringut töödeltakse, siis ülejäänud listakse järjekorda. *Bundling* on võimalus ASP.NET raamistikus, mis võimaldab failide kombineerimist üheks failiks. Vähem faile aitab parandada veebilehe laadimiseks kuluvat aega. *Minification* on failide mahu vähendamiseks, kasutades selleks erinevaid koodi optimeerimise vahendeid. Näiteks eemaldab failidest ebavajalikud tühemikud (ingl. k. *whitespace*), kommentaarid ning muutes muutjate nimed ühe tähemärgi pikkuseks.

Selleks et veebilehe laadimise aega testida lisame „Content“ kausta koos koopiat *Site.css* failist ja nimetame need ümber (nt. *test1.css*, *test2.css* jne.). Samuti lisame „Scripts“ kausta kolm koopiat *jquery-{version}.js* failist ja nimetame need samamoodi ümber (nt. *test1.js* jne.). *Bundle* loomiseks avame „App\_Start“ kaustas oleva *BundleConfig.cs* faili. Loomes uue *bundle* lisades järgmised read.

```
bundles.Add(new ScriptBundle("~/bundles/test").Include(
    "~/Scripts/test1.js",
    "~/Scripts/test2.js",
    "~/Scripts/test3.js"));
bundles.Add(new StyleBundle("~/Content/test").Include(
    "~/Content/test*"));
```

**Koodinäide 60. App\_Start/BundleConfig.cs – Skripti ja stiili bundle-ite loomine testimiseks**

Lõime kaks *bundle* objekti, mõlemad nimega „test“, kuid vastavalt „bundles“ ja „Content“ kaustadesse. Skripti failid lisasime ükshaaval, kuid CSS failide lisamiseks kasutasime „\*“ sümbolit, mis lisab kõik „Content“ kaustas olevad sõnaga „test“ algavad failid. Kui vaadata ASP.NET MVC raamistiku poolt loodud *bundle* objekte, siis näeme ka seal sarnast kasutusviisi.

```
bundles.Add(new ScriptBundle("~/bundles/jquery").Include(
    "~/Scripts/jquery-{version}.js"));
```

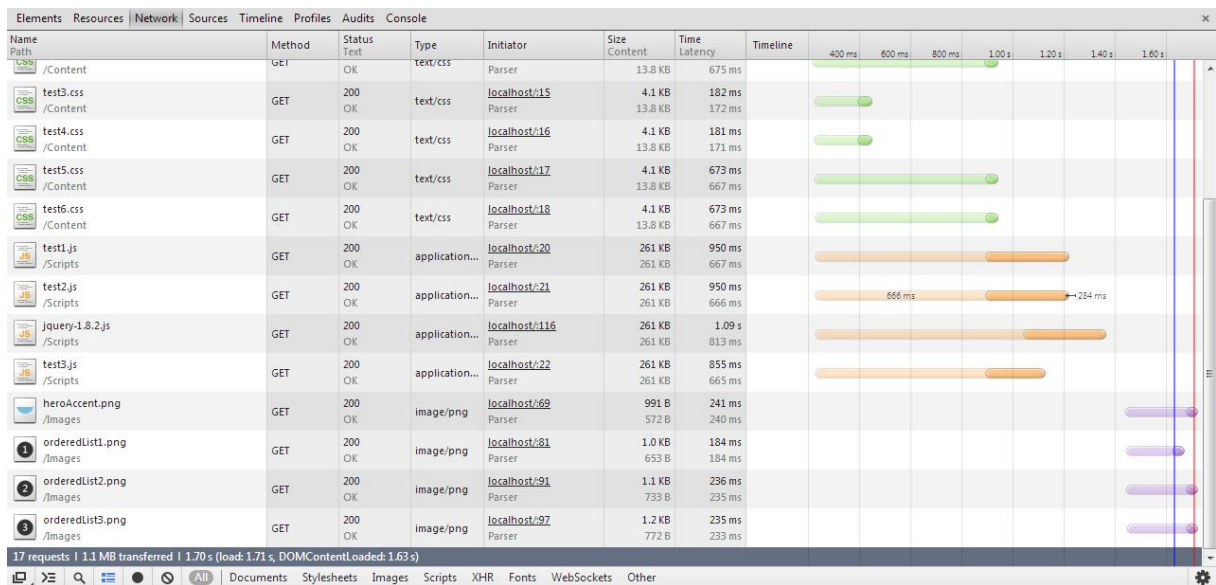
**Koodinäide 61. App\_Start/BundleConfig.cs – JQuery teegi bundle**

JQuery *bundle* lisamisel näeme looksulugude sees version tähistust, mis tähendab et kui me peaks projektis JQuery NuGet paketti uuendama, siis lisatud *bundle* toimib ikka. Selleks, et veebileht *bundle* objekte kasutaks tuleb need vaadetesse lisada. Lisame *\_Layout.cshtml* faili *bundle* esitamiseks koodi.

```
@Styles.Render("~/Content/test")
@Scripts.Render("~/bundles/test")
```

**Koodinäide 62. Views/Shared/\_Layout.cshtml – Bundle-ite esitamine**

Käivitame rakenduse ja avame veebilehitseja arendaja tööriistad (Chrome veebilehitsejas *Customize>Tools>Developer tools*). Valime *Network* riba ning paremalt alt nurgast muudame ka sätteid. Märgistame valiku *Disable Cache (while DevTools is open)*, et andmeid puhvrists ei võetaks. Laeme lehe uuesti ja vaatame tulemust.



17 requests | 1.1 MB transferred | 1.70 s (load: 1.71 s, DOMContentLoaded: 1.63 s)

### Joonis 19. Veebirakenduse laadimine ilma bundle-ite kasutamiset

Veebilehele tehti 17 päringut kogumalus 1.1 MB ning laadimine võttis aega 1.70 sekundit. Avame ka veebilehe lähtekoodi.

```
<link href="/Content/test1.css" rel="stylesheet"/>
<link href="/Content/test2.css" rel="stylesheet"/>
<link href="/Content/test3.css" rel="stylesheet"/>
<link href="/Content/test4.css" rel="stylesheet"/>
<link href="/Content/test5.css" rel="stylesheet"/>
<link href="/Content/test6.css" rel="stylesheet"/>
```

```
<script src="/Scripts/test1.js"></script>
<script src="/Scripts/test2.js"></script>
<script src="/Scripts/test3.js"></script>
```

Näeme, et failid laetakse ikka ükshaaval. Selle põhjuseks on see, et meie rakendus töötab silumisrežiimis (ingl. k. *Debug mode*). Silumisrežiim muudab arenduse mugavamaks, kuid *bundling* ja *minification* võimalused ei ole lubatud. Selleks, et kontrollida *bundling* ja *minification* toimimist võime silumisrežiimi maha võtta muutes *Web.config* failis reale *compilation debug* väärtuseks *false*.

```
<compilation debug="false" targetFramework="4.5" />
```

Teine võimalus on lisada *BundleConfig.cs* faili lõppu rida, mis võimaldab optimeerimise ja mis tühistab *Web.config* failis defineeritu.

```
BundleTable.EnableOptimizations = true;
```

### Koodinäide 63. App\_Start/BundleConfig.cs – Optimiseerimise lubamine

Käivitame rakenduse uuesti ning uurime tulemust.

10 requests | 188 KB transferred | 801 ms (load: 827 ms, DOMContentLoaded: 790 ms)

Joonis 20. Veebilehe laadimise aeg kasutades bundling ja minification meetodeid

Tabel 1. Veebilehe laadimise aegade võrdlus

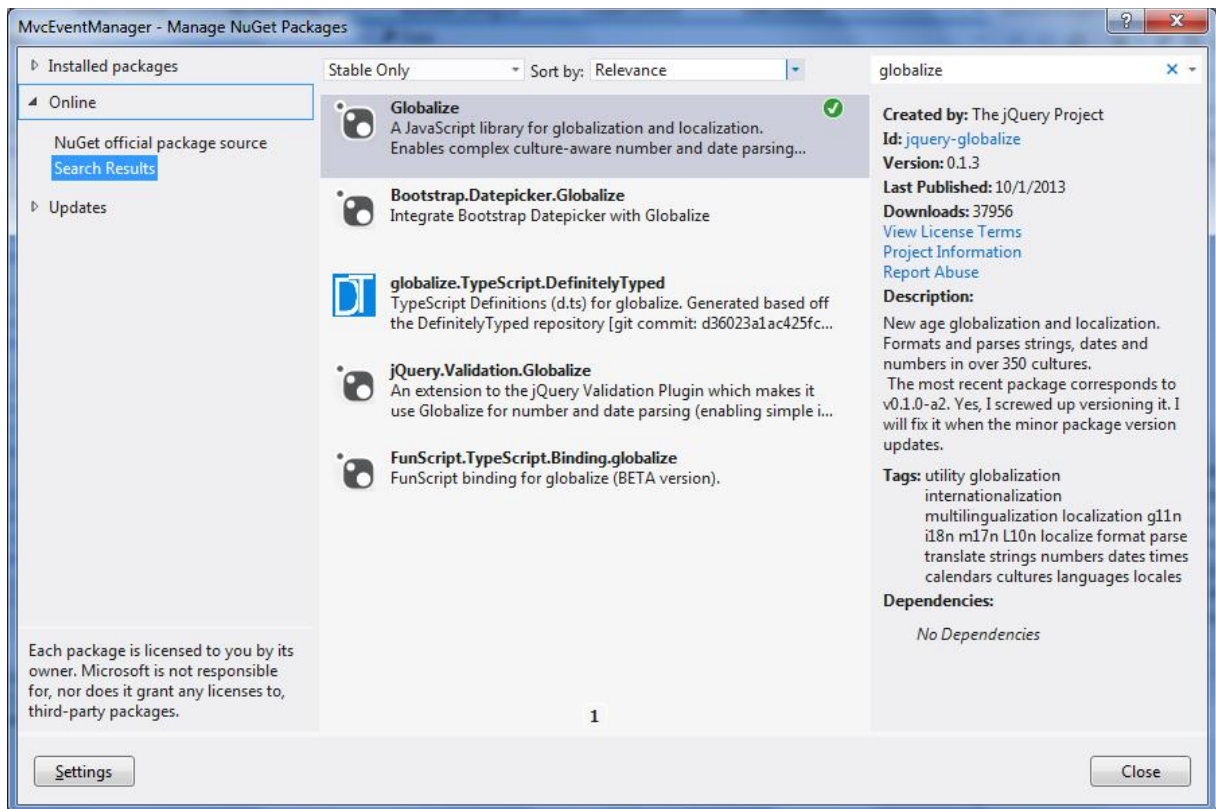
	Ilma bundling ja minificationita	Kasutades bundling ja minification
Päringuid	17	10
Andmemaht	1126,4 KB	188,0 KB
Laadimise aeg	1700 ms	801 ms

Näeme märkimisväärset muudatust – andmete maht vähenes ligi kuus korda ning laadimise aeg vähenes rohkem kui kaks korda.

### 3.14 Kuupäeva valideerimine

Hetkel valideeruvad kuupäevad ainult kuu-kuupäev-aasta formaadis. Serverpoolne valideerimine aksepteerib ka kuupäev-kuu-aasta formaati, kuid kliendipoolne mitte. Kuupäevale lisatud formaadi andme annotatsioon kehtib ainult kuupäeva esitamisele, kuid mitte valideermisele. Selleks, et eesti formaadis kuupäev valideeruks lisame rakendusele Globalize paketti. Selle menüüribalt *PROJECT->Manage Nuget Packages*. Otsime Globalize nimelist paketti ning installeerime selle oma projekti.





## Joonis 21. Nuget Package Manager

Projekti „Scripts“ kausta tekkis „globalize“ kaust. Looime uue *bundle*-i nimega „global“, mis vajalikud JavaScript-i failid meie rakendusele lisab.

```
bundles.Add(new ScriptBundle("~/bundles/global").Include(
    "~/Scripts/globalize/globalize.js",
    "~/Scripts/globalize/cultures/globalize.culture.et-EE.js"));
```

## Koodinäide 64. App\_Start/BundleConfig.cs – Globalize bundle-i loomine

Looime „Scripts“ kausta uue JavaScript-i faili, vajutades „Scripts“ kausta peale parema hiireklõpsuga ning *Add->New Item*. Valime loetelust *JavaScript File* ning anname nimeks „valformat.js“. Lisame faili *valformat.js* järgneva koodi.

```

$(function () {
    $.validator.methods.date = function (value, element) {
        Globalize.culture("et-EE");
        // you can alternatively pass the culture to parseDate instead of
        // setting the culture above, like so:
        // parseDate(value, null, "en-AU")
        return this.optional(element) || Globalize.parseDate(value) !== null;
    }
});

```

#### Koodinäide 65. Scripts/Valformat.js – Globalize-iga kultuuri kasutamine

Renderdades antud skriptid *Create* vaate sektsioonis *Scripts*, valideeruvad nüüd ka eesti formaadis kuupäevad. Selleks, et kasutajad ei saaks vales formaadis kuupäevi sisestada ning ka kasutajaliidese mugavamaks tegemiseks kasutame JQuery tööriista *Datepicker*. Selleks lisame veel kaks *bundle*-it.

```

bundles.Add(new ScriptBundle("~/bundles/datepicker").Include(
    "~/Scripts/jquery.ui.core.min.js",
    "~/Scripts/jquery.ui.datepicker.min.js",
    "~/Scripts/DatePickerReady.js",
    "~/Scripts/valformat.js",
    "~/Scripts/datepicker.js"));
bundles.Add(new StyleBundle("~/Content/datepicker").Include(
    "~/Content/themes/base/jquery.ui.core.css",
    "~/Content/themes/base/jquery.ui.datepicker.css",
    "~/Content/themes/base/jquery.ui.theme.css"));

```

#### Koodinäide 66. App\_Start/BundleConfig.cs – Datepicker skripti ja stiili bundle-i loomine

*ScriptBundle*-isse lisasime ka eelpool loodud *valformat.js* faili ning *datepicker.js* faili, mis määrab väljadele *Datepicker* tööriista kasutamise ning mille loome järgmisena. Lisame „Scripts“ kausta uue JavaScript-i faili nimega „datepicker.js“ ning lisame koodi.

```

$(document).ready(function () {
    $('#Date').datepicker
    ({
        dateFormat: 'dd.mm.yy',
    });
});

```

#### Koodinäide 67. Scripts/Datepicker.js – Date tüüpi väljadele datepicker-i kasutamise määramine

Muudame *Create* ja *Edit* vaadet lisades skriptide kasutamise vastavasse sektsiooni.

```
@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")

    @Scripts.Render("~/bundles/datepicker")
    @Styles.Render("~/Content/datepicker")
    @Scripts.Render("~/bundles/global")
}
```

**Koodinäide 68. Views/Event/Create.cshtml ja Views/Event/Edit.cshtml – Skriptide renderdamine Create ja Edit vaadetele**

Muudame vaadet *\_Layout.cshtml* lisades lehekülje lõppu peale jaluse kuvamist *JQueryui* renderdamise.

```
@Scripts.Render("~/bundles/jqueryui")
```

**Koodinäide 69. Views/Shared/\_Layout.cshtml – JQuery UI renderdamise lisamine**

Käivitades rakenduse ning üritades uut üritust lisada kuvatakse meile kuupäev lisamisel nüüd *Datepicker* tööriist.

## Create

Title

Description

Date

October 2013						
Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

[Back to List](#)

Joonis 22. Datepicker veebilehitsejas

## **Kokkuvõte**

Käesolevas õppematerjalis loodi ürituste rakendus koos mitmete funktsionaalsustega. Loodi ürituste loomise, muutmise, kustutamise ja osalemise funktsionaalsused. Anti ülevaade ASP.NET MVC raamistiku võimalustest ja kuidas nende kasutamisega erinevaid funktsionaalsusi on võimalik realiseerida. Loodedavasti tekkis arusaam MVC muustrist ja kuidas muistri kasutamine rakenduste loomisel kasuks tuleb. Autor soovib edu rakenduse edasi arendamisel ning uute võimaluste avastamisel.

## Kasutatud kirjandus

Rick Anderson (15. august 2012. a.) Intro to ASP.NET MVC 4. Kasutamise kuupäev: 1. oktoober 2013. a. Allikas: asp.net/mvc: <http://www.asp.net/mvc/tutorials/mvc-4/getting-started-with-aspnet-mvc4/intro-to-aspnet-mvc-4>

Rick Anderson (23. märts 2012. a.) Securing your ASP.NET MVC 4 App and the new AllowAnonymous attribute. Kasutamise kuupäev 17. oktoober 2013. a. Allikas: msdn.com: <http://blogs.msdn.com/b/rickandy/archive/2012/03/23/securing-your-asp-net-mvc-4-app-and-the-new-allowanonymous-attribute.aspx>

Rick Anderson (14. märts 2013. a.) XSRF/CSRF Prevention in ASP.NET MVC and Web Pages. Kasutamise kuupäev: 17. oktoober 2013. a. Allikas: asp.net/mvc: <http://www.asp.net/mvc/overview/security/xsrfcsrf-prevention-in-aspnet-mvc-and-web-pages>

Rick Anderson (28. august 2012. a.) Adding Validation to the Model. Kasutamise kuupäev: 15. oktoober 2013. a. Allikas: asp.net/mvc: <http://www.asp.net/mvc/tutorials/mvc-4/getting-started-with-aspnet-mvc4/adding-validation-to-the-model>

Rick Anderson (23. august 2012. a.) Bundling and Minification. Kasutamise kuupäev: 18. oktoober 2013. a. Allikas: asp.net/mvc: <http://www.asp.net/mvc/tutorials/mvc-4/bundling-and-minification>

John Atten (22. august 2013. a.) Customizing routes in ASP.NET MVC. Kasutamise kuupäev: 7. oktoober 2013. a. Allikas: codeproject.com: <http://www.codeproject.com/Articles/641783/Customizing-Routes-in-ASP-NET-MVC>

Jess Chadwick (Veebruar 2012. a.) The Features and Foibles of ASP.NET MVC Model Binding. Kasutamise kuupäev: 9. oktoober 2013. a. Allikas: MSDN Magazine: <http://msdn.microsoft.com/en-us/magazine/hh781022.aspx>

Dino Esposito (Juuli 2009. a.) Comparing Web Forms and ASP.NET MVC. Kasutamise kuupäev: 7. oktoober 2013. a. Allikas: MSDN Magazine: <http://msdn.microsoft.com/en-us/magazine/dd942833.aspx>

Microsoft ASP.NET Team (27. jaanuar 2009. a.) ASP.NET MVC Overview. Kasutamise kuupäev: 6. Oktoober 2013. aasta. Allikas asp.net/mvc: <http://www.asp.net/mvc/tutorials/older-versions/overview/asp-net-mvc-overview>

Microsoft ASP.NET Team (27. jaanuar 2009. a.) Improving Performance with Output Caching(C#). Kasutamise kuupäev: 25. oktoober. 2013. a. Allikas asp.net/mvc: <http://www.asp.net/mvc/tutorials/older-versions/controllers-and-routing/improving-performance-with-output-caching-cs>

Steven Smith (3. mai 2004. a.) ASP.NET Micro Caching: Benefits of a One-Second Cache. Kasutamise kuupäev: 20. oktoober. 2013. a. Allikas: aspalliance.com: [http://aspalliance.com/251\\_ASPNET\\_Micro\\_Caching\\_Benefits\\_of\\_a\\_OneSecond\\_Cache.all](http://aspalliance.com/251_ASPNET_Micro_Caching_Benefits_of_a_OneSecond_Cache.all)

Marla Suresh (7. Veebruar 2013. a.) WebForms vs. MVC. Kasutamise kuupäev: 11. oktoober 2013. a. Allikas: Codeproject.com: <http://www.codeproject.com/Articles/528117/WebForms-vs-MVC>

Scott Vandervort (11. mai 2011. a.) ASP.NET MVC versus Web Forms Smackdown. Kasutamise kuupäev: 11. oktoober 2013. a. Allikas: blogspot.com: <http://scottsjewels.blogspot.com/2011/05/aspnet-mvc-versus-web-forms-smackdown.html>