05 - MVC in action

Solution Explorer > Models > Add > Class > Damo naziv Book i dodelimo neke atribute.

```
namespace LibraryNT.Models
{
    public class Book
    {
        public int ID { get; set; }
        public string Name { get; set; }
    }
}
```

Pravimo random stranicu koja će prikazati podatke nekog filma. Putanja je **/books/random**.
*Potrebno je da napravimo **BooksController** sa metodom **Random**.*

Solution Explorer > Controller > Add > Controller > MVC 5 Controller - Empty
Na vrhu stavimo **using LibraryNT.Models;**

Unutra pravimo instancu **Book** klase. Obično podatke dobijamo iz baze podataka, ali za sad ćemo ih praviti sami.

BooksController.cs

LibraryNT | LibraryNT

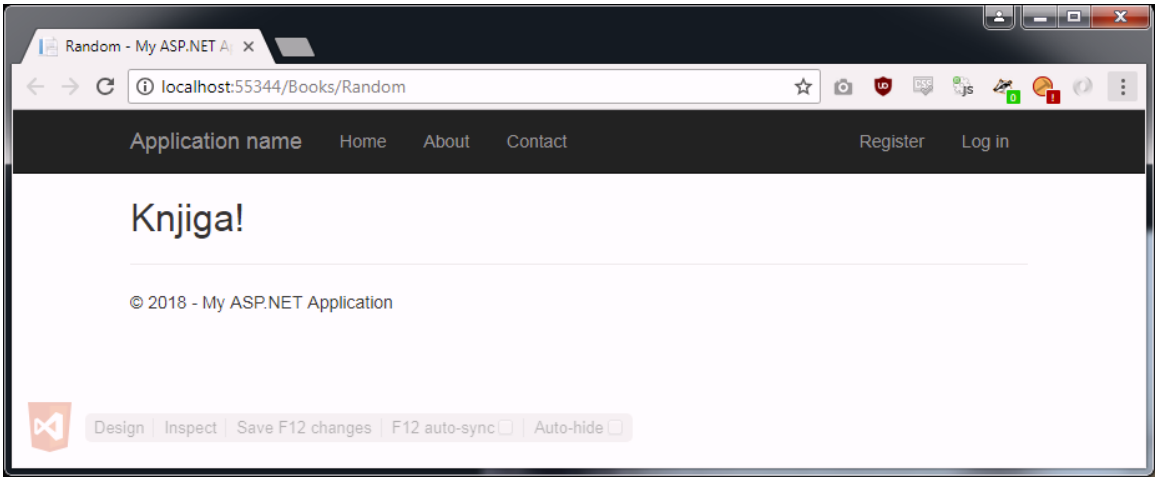
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using LibraryNT.Models;

namespace LibraryNT.Controllers
{
    public class BooksController : Controller
    {
        // GET: Books/Random
        public ActionResult Random()
        {
            var Book = new Book() { Name = "Knjiga!", ID = 1 };
            return View(Book);
        }
    }
}
```

Prosleđujemo Book model View-u koji će biti u Views/Books/Random.cshtml

Solution Explorer > Views > Movies *(već sam napravljen)* **> Ovde pravimo view sa nazivom metode. Add > View > Naziv Random i Empty.**
Odaberemo Layout ; Views > Shared > _Layout.cshtml.

Na vrhu strane napravimo referencu ka modelu. @model *ImeProjekta.Models.NazivModela*. Nakon ovoga možemo da pristupamo svojstima tog modela (ID, name itd.).

```
Random.cshtml
1
2   @model LibraryNT.Models.Book
3   @{
4       ViewBag.Title = "Random";
5       Layout = "~/Views/Shared/_Layout.cshtml";
6   }
7
8   <h2>@Model.Name</h2>
9
10
```



10 - Action results

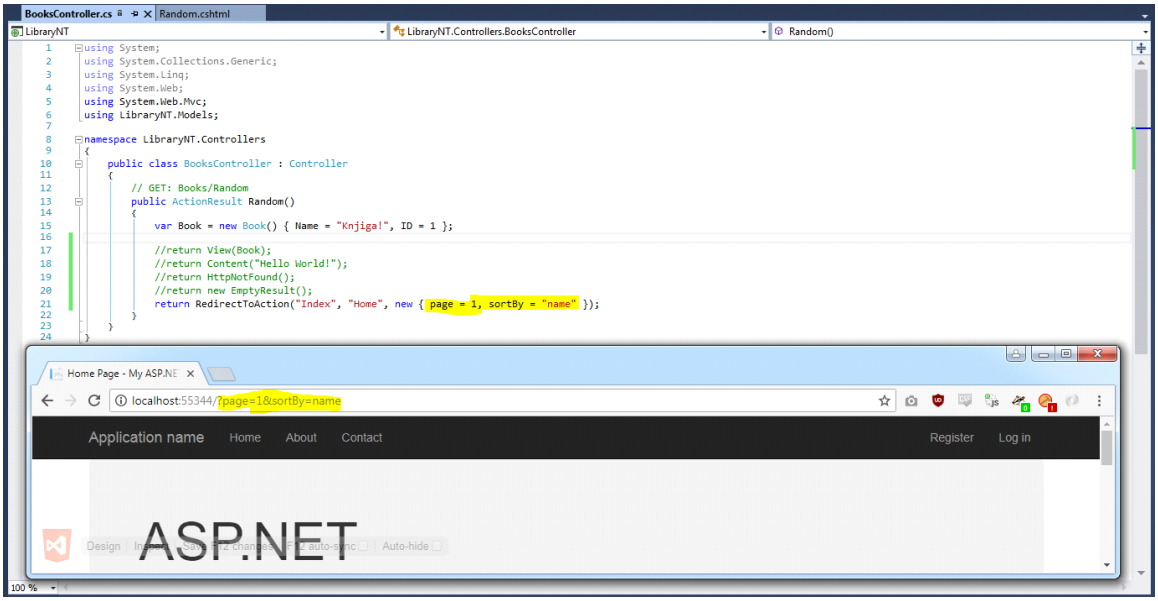
```
BooksController.cs
LibraryNT                                   LibraryNT
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Web;
5   using System.Web.Mvc;
6   using LibraryNT.Models;
7
8   namespace LibraryNT.Controllers
9   {
10      public class BooksController : Controller
11      {
12          // GET: Books/Random
13          public ActionResult Random()
14          {
15              var Book = new Book() { Name = "Knjiga!", ID = 1 };
16              return View(Book);
17          }
18      }
19  }
```

Ovde vidimo **View()** metodu u redu 16. Imamo još par metoda koje možemo da koristimo.

return **Content("Hello World!")**; - Vraća čisto Hello World! na HTML stranici.

return **HttpNotFound();** - Vraća 404 not found stranicu.
return **new EmptyResult();** - Kao void.
return **RedirectToAction("Index", "Home");** - Naziv metode, naziv kontrolera, [argumenti].
Argumente ubacujemo u anoniman objekat. new { ... }.

Kada odemo na Books/Random dobijemo ovo ispod.

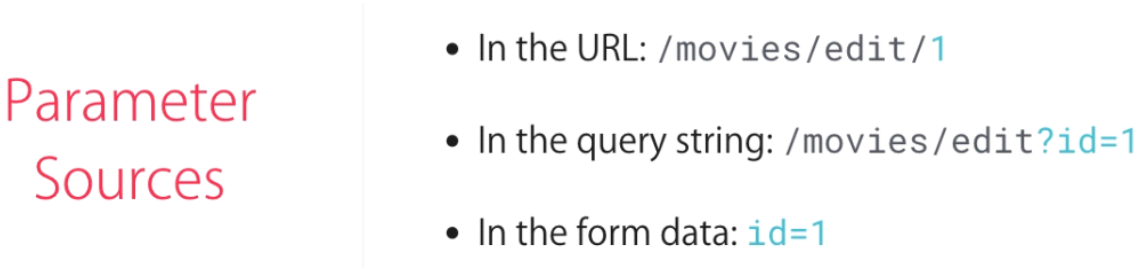


Uglavnom se koriste **View()**, **HttpNotFound()** i **Redirect()**

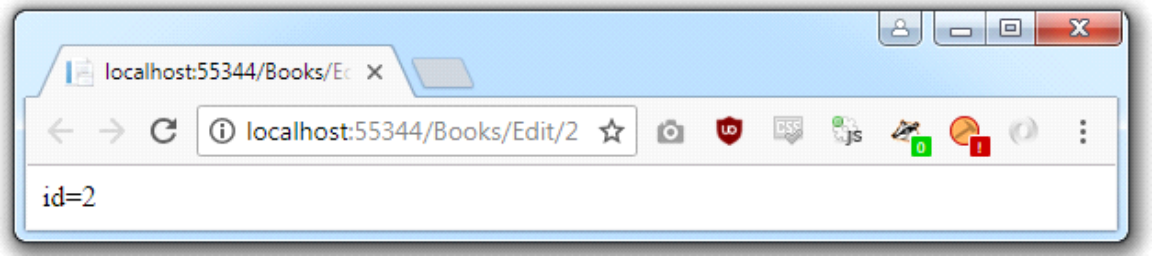
11 - Action parameters

Inputi za akcije. Kada request dođe u aplikaciji, MVC automatski mapira request data na parameter values za action metode. Dakle, ako action metoda ima parametre, MVC traži parametar sa istim nazivom u request data. Ako ga nađe, MVC automatski prosleđuje vrednost tog parametra ciljanoj akciji.

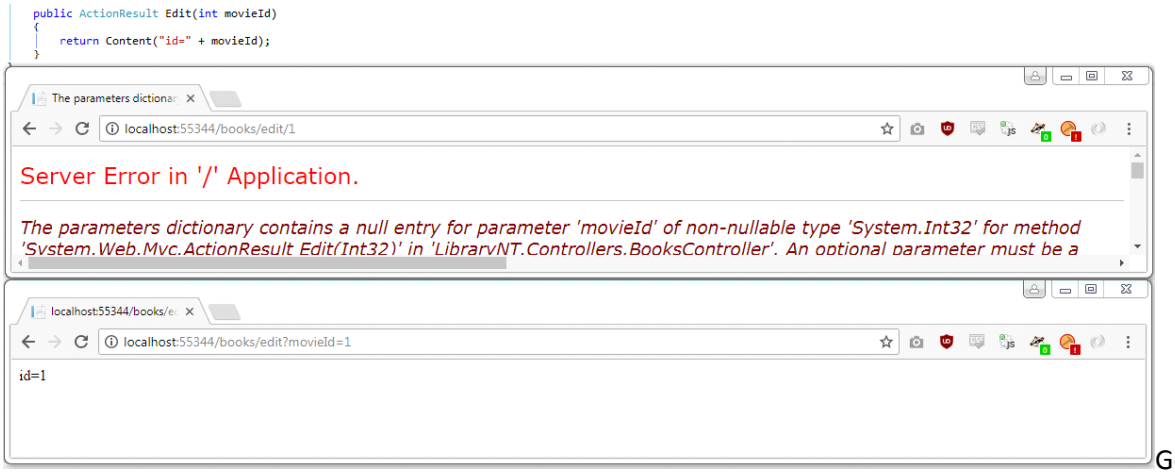
Izvori parametra mogu biti **URL**, **query string** i **forma**.



```
// Kreiramo ActionResult Edit sa parametrom id.
public ActionResult Edit(int id)
{
    return Content("id=" + id);
}
```



Moglo je i /Books/edit?id=2. ***Parametar mora da ima isti naziv i u metodi i url-u.***



Gornji izbacuje exception jer u **RouteConfig** postavljeno je da se parametar naziva **id**, a metoda ne zna šta je to.

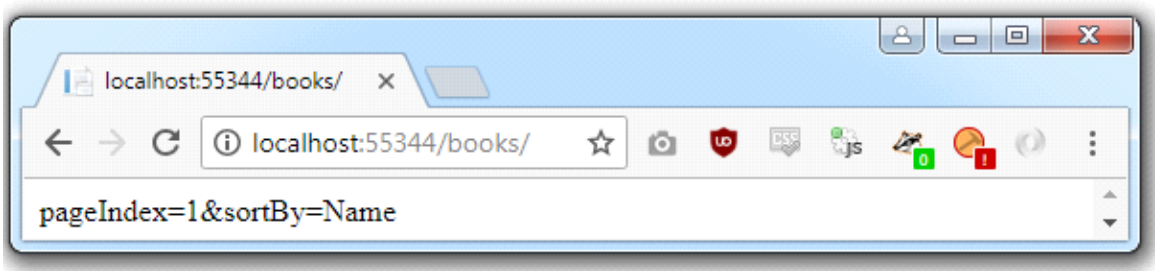
BooksController.cs

LibraryNT | LibraryNT.RouteConfig

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.Web.Mvc;
6  using System.Web.Routing;
7
8  namespace LibraryNT
9  {
10     public class RouteConfig
11     {
12         public static void RegisterRoutes(RouteCollection routes)
13         {
14             routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
15
16             routes.MapRoute(
17                 name: "Default",
18                 url: "{controller}/{action}/{id}",
19                 defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
20             );
21         }
22     }
23 }
24
```

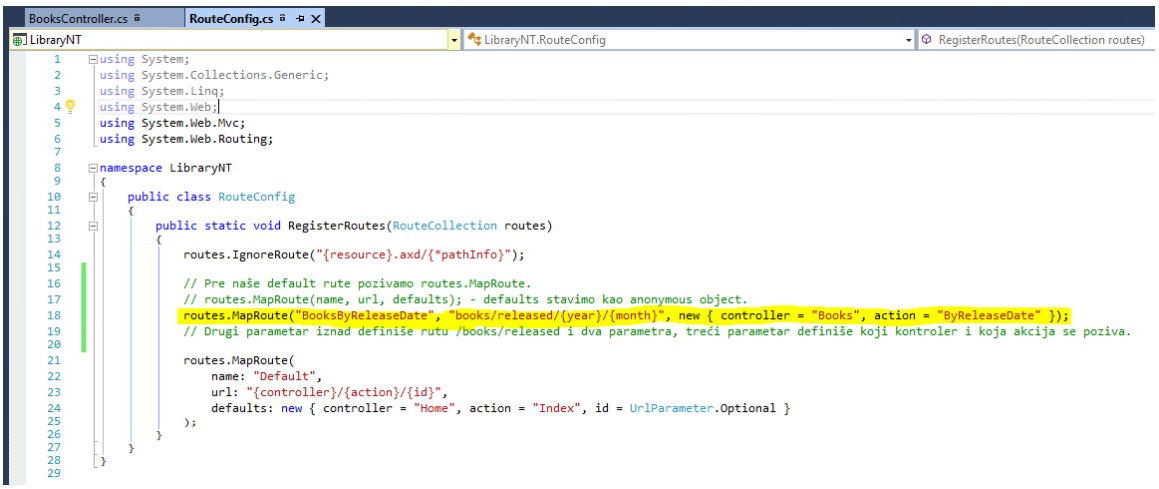
Opcionalni parametri

```
// Opcionalni parametri. Index se otvara kada odemo na /movies/.
// Ako pageIndex nije specifiran = 1, sortBy nije specifiran = Name.
// int? = Parametar je nullable. string je sam po sebi nullable.
public ActionResult Index(int? pageIndex, string sortBy)
{
    if (!pageIndex.HasValue)
        pageIndex = 1;
    if (String.IsNullOrWhiteSpace(sortBy))
        sortBy = "Name";

    return Content(String.Format("pageIndex={0}&sortBy={1}", pageIndex, sortBy));
}
```



12 - Convention-based routing

RouteConfig.cs - Naziv, Url, Defaults (kontroler i akcija).




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Routing;

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

            // Pre naše default rute pozivamo routes.MapRoute.
            // routes.MapRoute(name, url, defaults); - defaults stavimo kao anonymous object.
            routes.MapRoute("BooksByReleaseDate", "books/released/{year}/{month}", new { controller = "Books", action = "ByReleaseDate" });
            // Drugi parametar iznad definiše rutu /books/released i dva parametra, treći parametar definiše koji kontroler i koja akcija se poziva.

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

BooksController.cs

```
BooksController.cs   RouteConfig.cs
LibraryNT                                   LibraryNT.Controllers.BooksController
4   using System.Web;
5   using System.Web.Mvc;
6   using LibraryNT.Models;
7
8   namespace LibraryNT.Controllers
9   {
10      public class BooksController : Controller
11      {
12          // GET: Books/Random
13          public ActionResult Random()...
23
24          // Metoda Edit sa parametrom id.
25          public ActionResult Edit(int id)...
29
30          // Opcionalni parametri. Index se otvara kada odemo na /movies/. ...
33          public ActionResult Index(int? pageIndex, string sortBy)...
42
43          public ActionResult ByReleaseDate(int year, int month)
44          {
45              return Content(year + "/" + month);
46          }
47      }
48  }
```

```
localhost:55344/books/re
localhost:55344/books/released/2014/12
2014/12
```

Ograničavanje parametara u url-u - dodavanje četvrtog parametra koji je anonymous object.

```
BooksController.cs   RouteConfig.cs
LibraryNT                                   LibraryNT.RouteConfig                    RegisterRoutes(RouteCollection ro
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Web;
5   using System.Web.Mvc;
6   using System.Web.Routing;
7
8   namespace LibraryNT
9   {
10      public class RouteConfig
11      {
12          public static void RegisterRoutes(RouteCollection routes)
13          {
14              routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
15
16              // Pre naše default rute pozivamo routes.MapRoute.
17              // routes.MapRoute(name, url, defaults); - defaults stavimo kao anonymous object.
18              routes.MapRoute(
19                  "BooksByReleaseDate",
20                  "books/released/{year}/{month}",
21                  new { controller = "Books", action = "ByReleaseDate" },
22                  new { year = @"\d{4}", month = "\\d{2}" });
23              // Drugi parametar iznad definiše rutu /books/released i dva parametra, treći parametar definiše koji kontroler i koja akcija se poziva.
24              // Četvrti (opcioni) parametar definiše ograničenja za parametre u url-u. Može @ ili \\.
25
26              routes.MapRoute(
27                  name: "Default",
28                  url: "{controller}/{action}/{id}",
29                  defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
30              );
31          }
32      }
33  }
```

new { year = @"2015|2016" } - ograničava year parametar na vrednosti 2015 ili 2016.

13 - Attribute Routing

```
BooksController.cs   RouteConfig.cs
LibraryNT                                   LibraryNT.RouteConfig
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Web;
5   using System.Web.Mvc;
6   using System.Web.Routing;
7
8   namespace LibraryNT
9   {
10      public class RouteConfig
11      {
12          public static void RegisterRoutes(RouteCollection routes)
13          {
14              routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
15
16              // Novi način pravljenja ruta [Route("url")]. Constrainte uradimo tako što kod atributa stavimo dve tačke.
17              // Prvo stavimo routes.MapMvcAttributeRoutes();
18              routes.MapMvcAttributeRoutes();
19
20              routes.MapRoute(
21                  name: "Default",
22                  url: "{controller}/{action}/{id}",
23                  defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
24              );
25          }
26      }
27  }
28
```

```
BooksController.cs •   RouteConfig.cs
LibraryNT                                   LibraryNT.Controllers.Book
4   using System.Web;
5   using System.Web.Mvc;
6   using LibraryNT.Models;
7
8   namespace LibraryNT.Controllers
9   {
10      public class BooksController : Controller
11      {
12          // GET: Books/Random
13          public ActionResult Random()...
23
24          // Metoda Edit sa parametrom id.
25          public ActionResult Edit(int id)...
29
30          // Opcionalni parametri. Index se otvara kada odemo na /movies/. ...
33          public ActionResult Index(int? pageIndex, string sortBy)...
42
43          // Stavimo rutu iznad akcije koja treba da se izvrši.
44          [Route("movies/released/{year}/{month:regex(\\d{2}):range(1, 12)}")]
45          public ActionResult ByReleaseDate(int year, int month)
46          {
47              return Content(year + "/" + month);
48          }
49      }
50  }
```

14 - Passing data to views

tl;dr - Prosleđuj kao do sad.

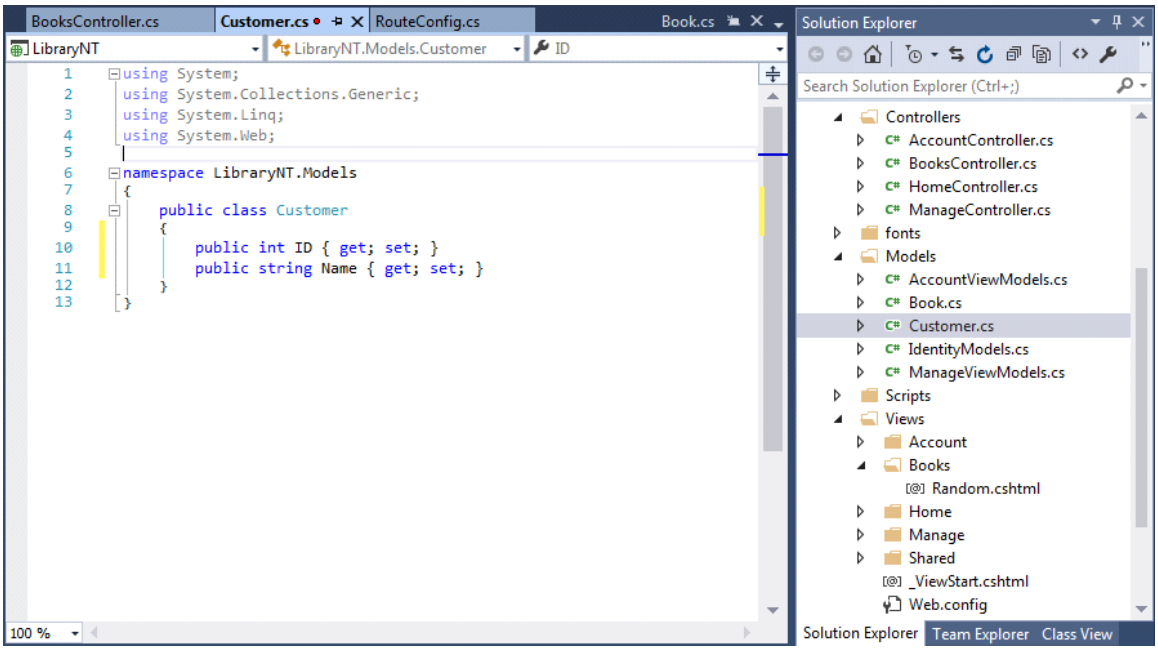
15 - View model

Trenutno u *Random* view samo prikazujemo naziv knjige. Šta ako bismo hteli da takođe izlistamo spisak korisnika koji su iznajmili tu knjigu?

```
@model LibraryNT.Models.Book
@{
    ViewBag.Title = "Random";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>@Model.Name</h2>
```

U našem domain modelu, moguće je da ne postoji relacija između Book i Customer klase, tako da je potrebno da prolsedimo dva različita modela ovom view-u. Problem je što imamo samo jedan model atribut (@model). Kako rešiti ovaj problem? Korišćenjem **view modela**.

View model je model specifično napravljen za neki view. Obuhvata sve podatke i pravila vezana za taj pogled.



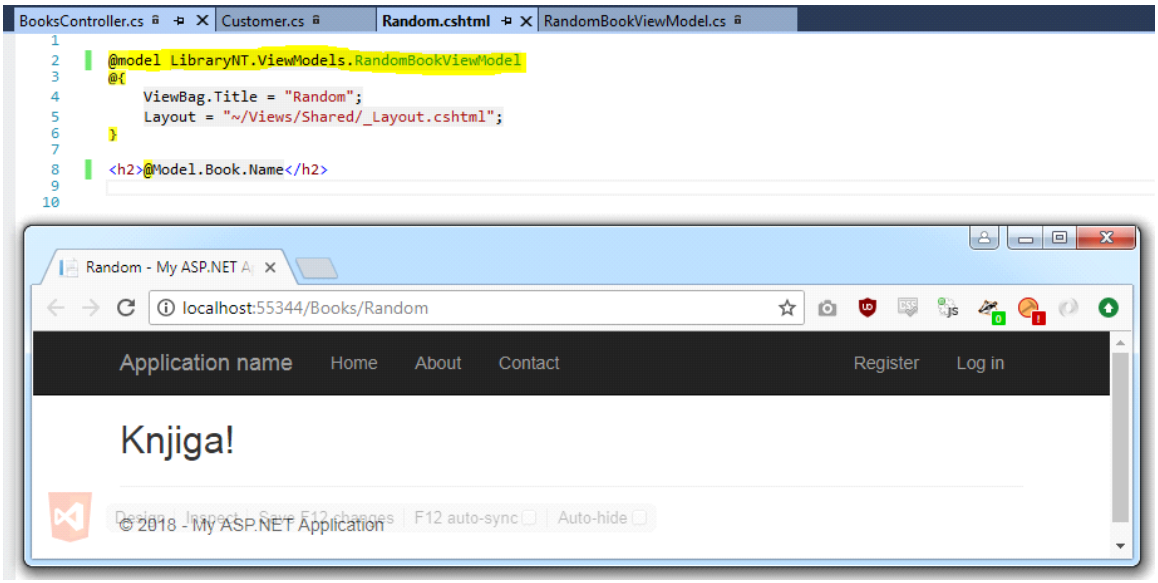
View modele ubacujemo u poseban folder u projektu. Add > New folder > ViewModels. Ovde dodajemo klase.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using LibraryNT.Models;

namespace LibraryNT.ViewModels
{
    public class RandomBookViewModel
    {
        public Book Book { get; set; }
        public List<Customer> Customers {get; set;}
    }
}
```

U kontroleru pravimo **viewModel** objekat u koji ubacujemo sve i prosleđujemo View-u.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using LibraryNT.Models;
using LibraryNT.ViewModels;

namespace LibraryNT.Controllers
{
    public class BooksController : Controller
    {
        // GET: Books/Random
        public ActionResult Random()
        {
            var Book = new Book() { Name = "Knjiga!", ID = 1 };
            // Kreiramo listu Customers.
            var Customers = new List<Customer>
            {
                new Models.Customer { Name = "Customer 1" },
                new Models.Customer { Name = "Customer 2" }
            };

            // Kreiramo ViewModel objekat.
            var viewModel = new RandomBookViewModel
            {
                Book = Book,
                Customers = Customers
            };

            // Prosleđujemo objekat view-u.
            return View(viewModel);
        }
    }
}
```

View

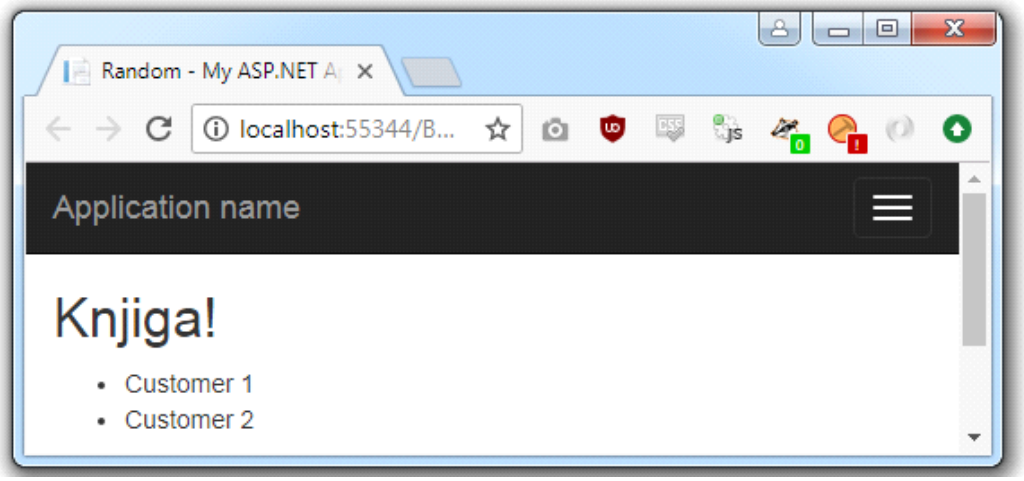


16 - Razor syntax

Za prikaz liste mušteterija, možemo jednostavno koristiti **foreach blok**.

BooksController.cs | Customer.cs | Random.cshtml | RandomBookViewModel.cs

```
1
2   @model LibraryNT.ViewModels.RandomBookViewModel
3   @{
4       ViewBag.Title = "Random";
5       Layout = "~/Views/Shared/_Layout.cshtml";
6   }
7
8   <!-- Ako ima više od jednog korisnika dodaće se klasa popular na h2. -->
9   <!-- Ovo je dinamičko dodavanje atributa. Komentarisanje može i sa @* *@-->
10  @{
11      var className = Model.Customers.Count > 1 ? "popular" : null;
12  }
13
14  <h2 class="@className">@Model.Book.Name</h2>
15
16  <!-- Lista korisnika. -->
17  @if (Model.Customers.Count == 0)
18  {
19      <p>No one has rented this movie before.</p>
20  }
21  <ul>
22      @foreach (var Customer in Model.Customers)
23      {
24          <li>@Customer.Name</li>
25      }
26  </ul>
27
```



17 - Partial views

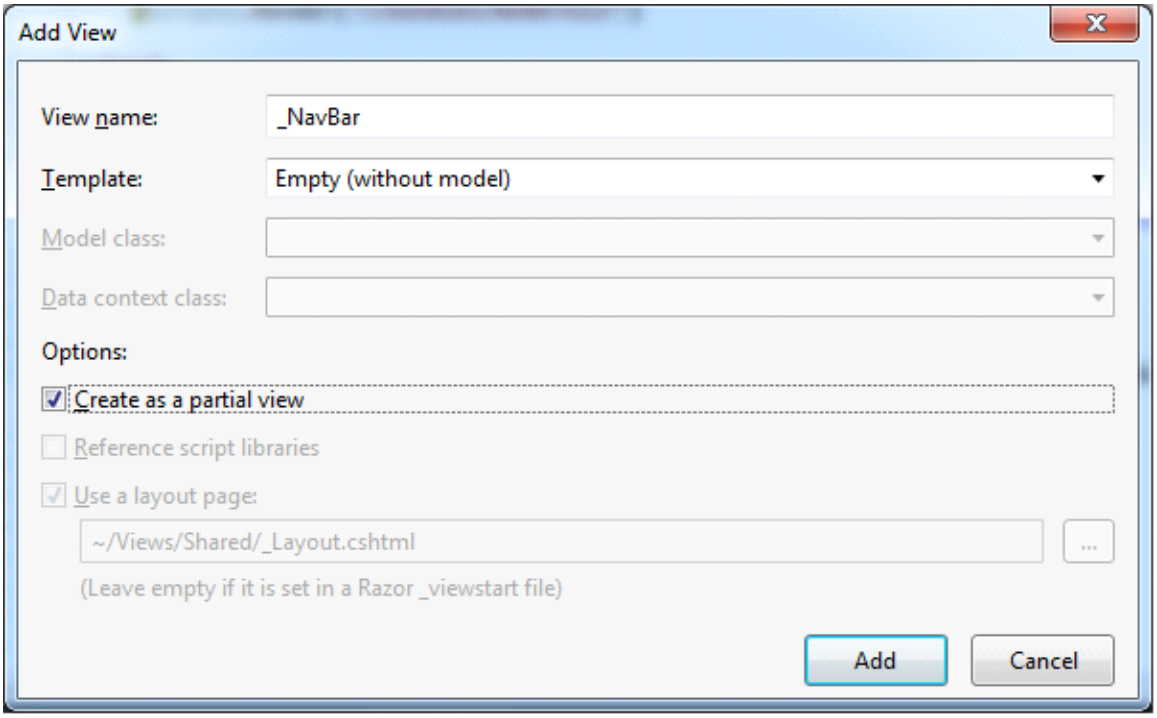
Views > Shared > Layout.cshtml

```
BooksController.cs | Customer.cs | Random.cshtml | RandomBookViewModel.cs | _Layout.cshtml
1   <!DOCTYPE html>
2   <html>
3   <head>
4       <meta charset="utf-8" />
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <title>@ViewBag.Title - My ASP.NET Application</title>
7       @Styles.Render("~/Content/css")
8       @Scripts.Render("~/bundles/modernizr")
9
10  </head>
11  <body>
12      <div class="navbar navbar-inverse navbar-fixed-top">...</div>
32      <div class="container body-content">
33          @RenderBody()
34          <hr />
35          <footer>
36              <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
37          </footer>
38      </div>
39
40      @Scripts.Render("~/bundles/jquery")
41      @Scripts.Render("~/bundles/bootstrap")
42      @RenderSection("scripts", required: false)
43  </body>
44  </html>
45
100 %
```

Ono što je ubačeno u view-ovima će biti na mestu RenderBody() metode. Partial view je kao mali view koji možemo iskoristiti na više view-ova, ali možemo ih i koristiti kako bi rasklopili jedan složen view u više jednostavnijih.

Izvlačenje navbar-a.

Desni klik na Shared > Add > View. Po konvenciji parcijalni view-ovi počinju sa _. _NavBar.



Uzmemo i cut-ujemo ceo NavBar div iz Layout u _NavBar.

```
BooksController.cs | Customer.cs | Random.cshtml | RandomBookViewModel.cs | _Layout.cshtml | _NavBar.cshtml
1  <div class="navbar navbar-inverse navbar-fixed-top">
2      <div class="container">
3          <div class="navbar-header">
4              <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
5                  <span class="icon-bar"></span>
6                  <span class="icon-bar"></span>
7                  <span class="icon-bar"></span>
8              </button>
9              @Html.ActionLink("Application name", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })
10         </div>
11         <div class="navbar-collapse collapse">
12             <ul class="nav navbar-nav">
13                 <li>@Html.ActionLink("Home", "Index", "Home")</li>
14                 <li>@Html.ActionLink("About", "About", "Home")</li>
15                 <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
16             </ul>
17             @Html.Partial("_LoginPartial")
18         </div>
19     </div>
20 </div>
21
100 %
```

Partial view include-ujemo uz **@Html.Partial("<ime_viewa>", [Model])**

```
BooksController.cs | Customer.cs | Random.cshtml | RandomBookViewModel.cs | _Layout.cshtml | _NavBar.cshtml
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8" />
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>@ViewBag.Title - My ASP.NET Application</title>
7      @Styles.Render("~/Content/css")
8      @Scripts.Render("~/bundles/modernizr")
9
10 </head>
11 <body>
12     @Html.Partial("_NavBar")
13     <div class="container body-content">
14         @RenderBody()
15         <hr />
16         <footer>
17             <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
18         </footer>
19     </div>
20
21     @Scripts.Render("~/bundles/jquery")
22     @Scripts.Render("~/bundles/bootstrap")
23     @RenderSection("scripts", required: false)
24 </body>
25 </html>
26
100 %
```

ZADATAK :

Izmeniti NavBar da ima Customers i Books. Customers ima 2 Customer-a u listi (ne dolaze iz baze, sami ih pravimo). Klikom na ime Customer-a idemo na **Customers/Details/<id>**. Ako nema customer-a sa tim id-em, baca 404 grešku. Ako nema Customer-a, treba da ispiše da ih trenutno nema. Druga strana Books je veoma slična osim što nema linkova na njima.

Navbar

```
<li>@Html.ActionLink("Customers", "Customers", "Home")</li>
<li>@Html.ActionLir
.>
ıl.Partial("_LoginPa
```

(extension) MvcHtmlString HtmlHelper.ActionLink(string linkText, string actionName, string controllerName) (+ 9 overloads)
Returns an anchor element (a element) for the specified link text, action, and controller.

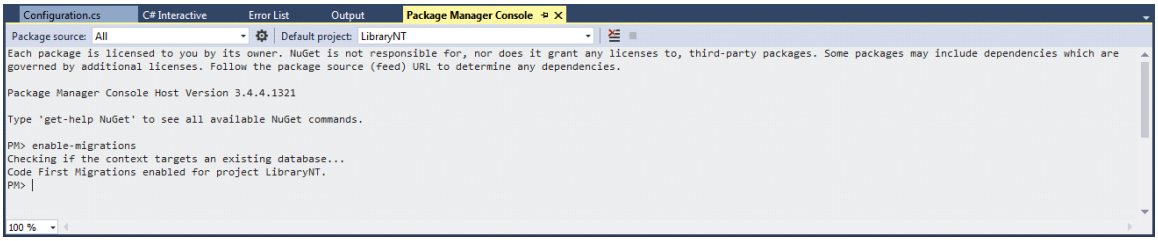
Exceptions:
ArgumentException

tl;dr - Kopiraj Hamedanija sa Github-a, veoma loše objašnjeno.

25 - Code-first Migrations

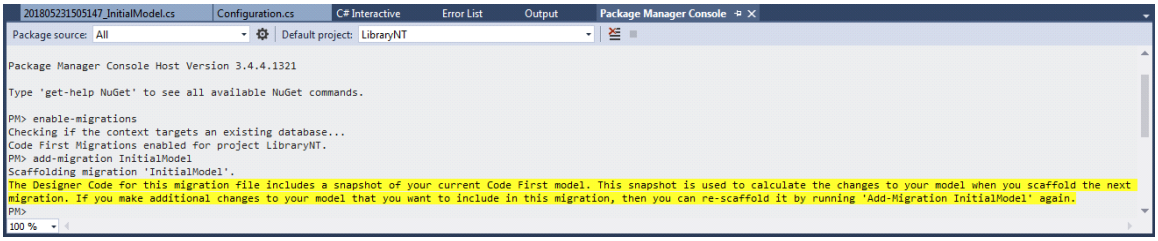
Svaki put kada modifikujemo naš domain model dodavanjem nove klase ili izmenom postojećih mi pravimo migraciju i onda je zapisujemo u bazu podataka.

Tools > NuGet package manager > Package manager console. Kucamo **enable-migrations**.

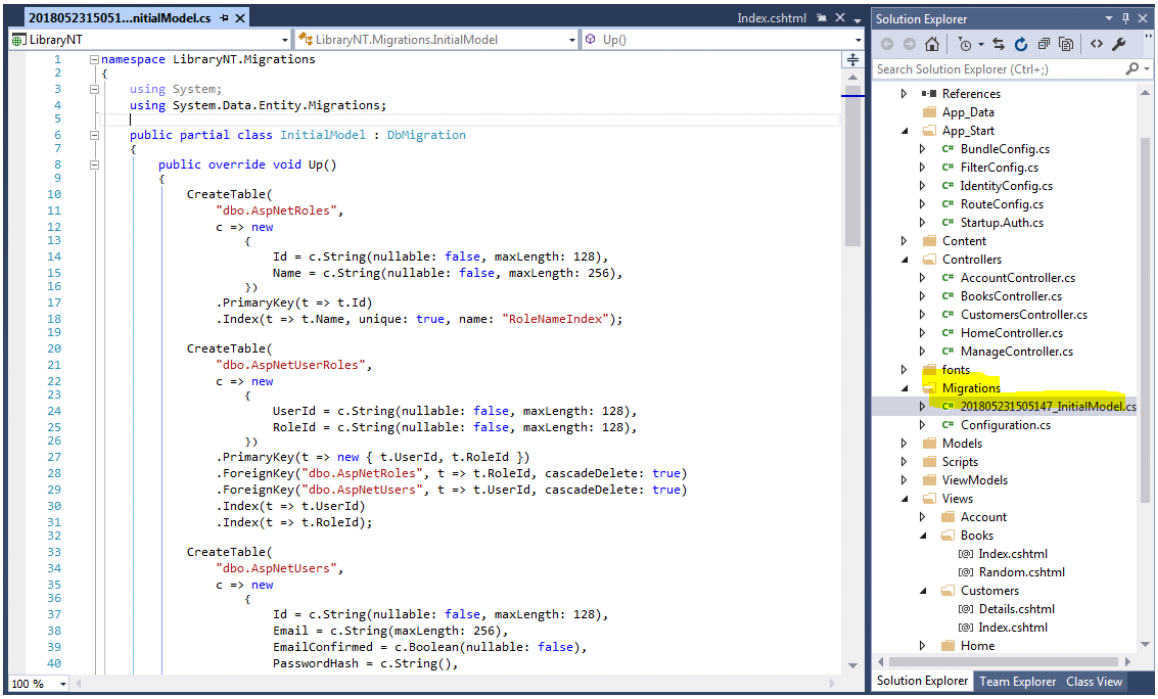


U projektu se napravio novi folder **Migrations** u kojem će biti sve migracije.

Kucamo u konzoli **add-migration <Ime>**



...



Kako bismo referencirali naše domain klase, moramo ih referencirati preko **DbContext-a** naše migracije.

Models > IdentityModels.cs > ApplicationDbContext() > Dodajemo DbSet.
public DbSet<NazivKlase> Naziv { get; set; }

```
using System.Data.Entity;
using System.Security.Claims;
using System.Threading.Tasks;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.EntityFramework;

namespace LibraryNT.Models
{
    // You can add profile data for the user by adding more properties to your ApplicationUser class, please visit http://go.microsoft.com/fwlink/?
    LinkID=317594 to learn more.
    public class ApplicationUser : IdentityUser
    {
        public async Task<ClaimsIdentity> GenerateUserIdentityAsync(UserManager<ApplicationUser> manager)
        {
            // Note the authenticationType must match the one defined in CookieAuthenticationOptions.AuthenticationType
            var userIdentity = await manager.CreateIdentityAsync(this, DefaultAuthenticationTypes.ApplicationCookie);
            // Add custom user claims here
            return userIdentity;
        }
    }

    public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
    {
        public DbSet<Customer> Customers { get; set; }
        public ApplicationDbContext()
            : base("DefaultConnection", throwIfV1Schema: false)
        {
        }

        public static ApplicationDbContext Create()
        {
            return new ApplicationDbContext();
        }
    }
}
```

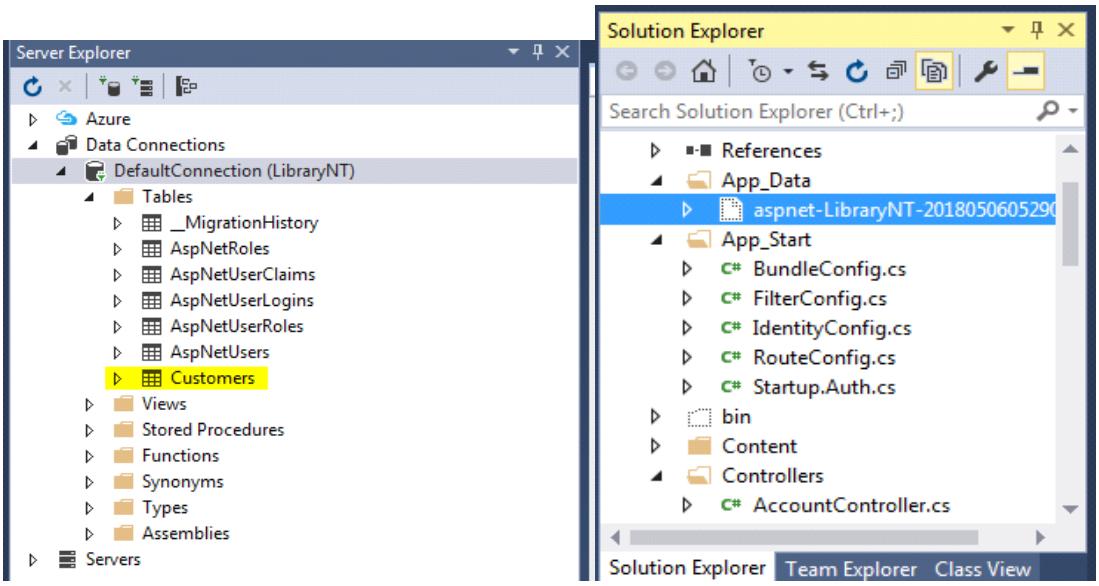
Sada ponovo idemo u NuGet kako pismo izvršili migraciju.
add-migration InitialModel -force *(jer već postoji pa overwrite)*

Sada vidimo Customer u Migraciji.

```
namespace LibraryNT.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class InitialModel : DbMigration
    {
        public override void Up()
        {
            CreateTable(
                "dbo.Customers",
                c => new
                    {
                        ID = c.Int(nullable: false, identity: true),
                        Name = c.String(),
                    })
                .PrimaryKey(t => t.ID);

            CreateTable(
                "dbo.AspNetRoles",
                c => new
                    {
                        Id = c.String(nullable: false, maxLength: 128),
                        Name = c.String(nullable: false, maxLength: 256),
                    })
                .PrimaryKey(t => t.Id)
                .Index(t => t.Name, unique: true, name: "RoleNameIndex");

            CreateTable(
                "dbo.AspNetUserRoles",
                c => new
                    {
                        UserId = c.String(nullable: false, maxLength: 128),
                        RoleId = c.String(nullable: false, maxLength: 128),
                    })
```

U NuGet konzli kucam *update-database*

U solution exploreru kliknemo *Show All Files* i pod *App_Data* vidimo našu bazu.

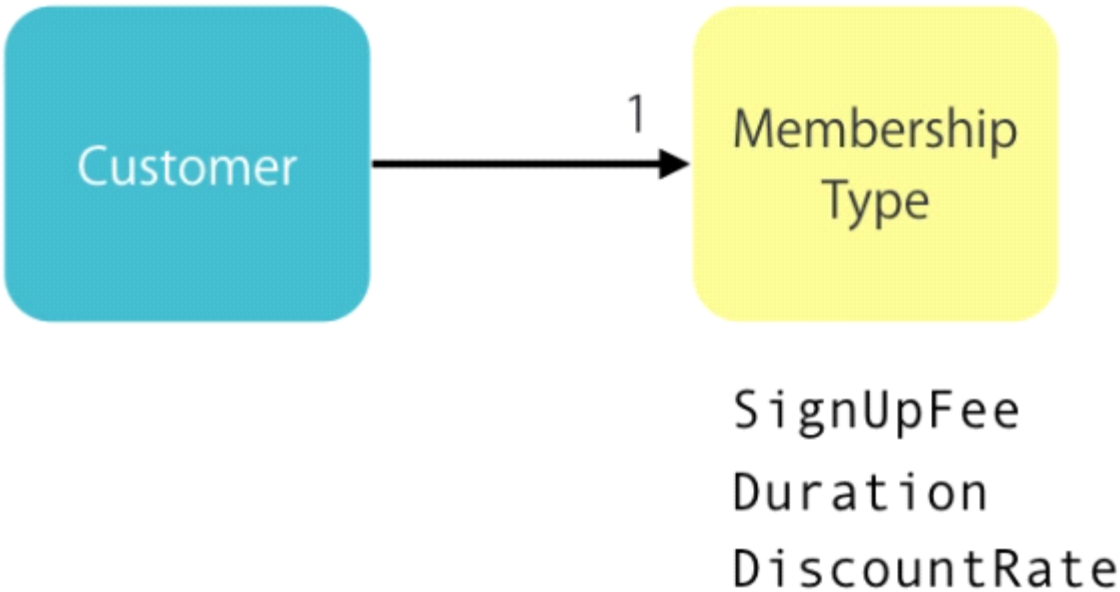


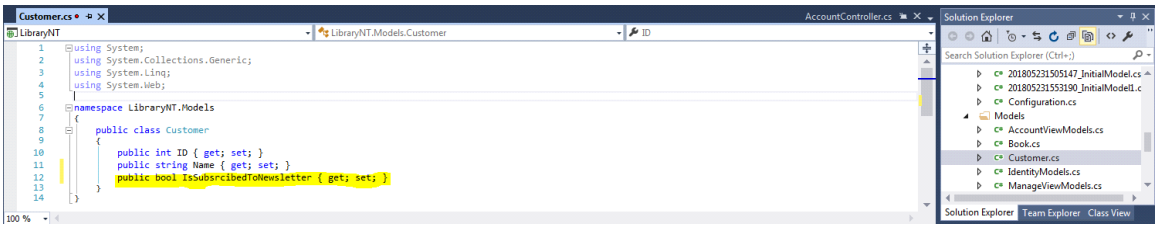
AspNet tabele su deo entity framework-a i služe za autentifikaciju i autorizaciju.

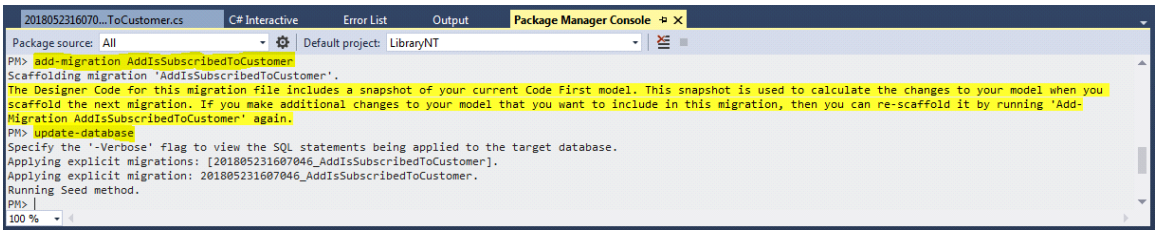
26 - Changing the Model

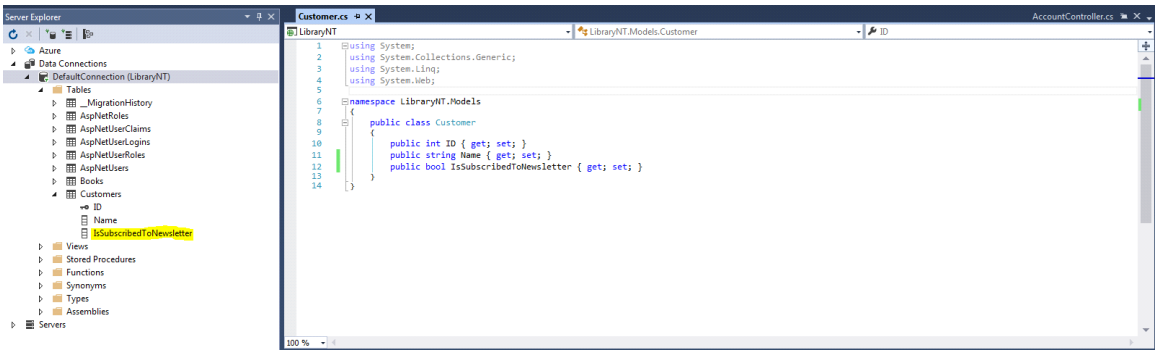
Membership	Sign Up Fee	Discount
Pay as You Go	Free	0%
Monthly	$30	10%
Quarterly	$90	15%
Annual	$300	20%

Pravimo klasu Membership Type. Svaki customer ima jedan i samo jedan membership type.

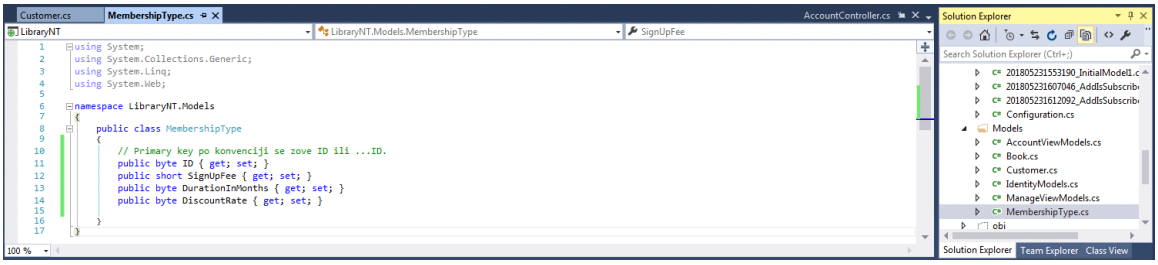




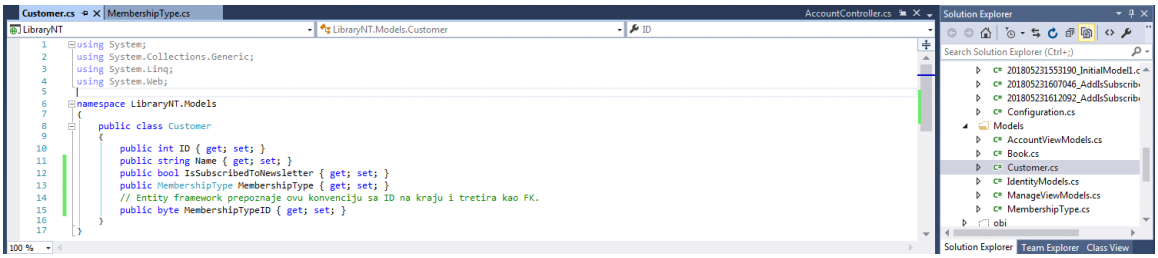




Desni klik Models > Add new > Class > MembershipType.cs



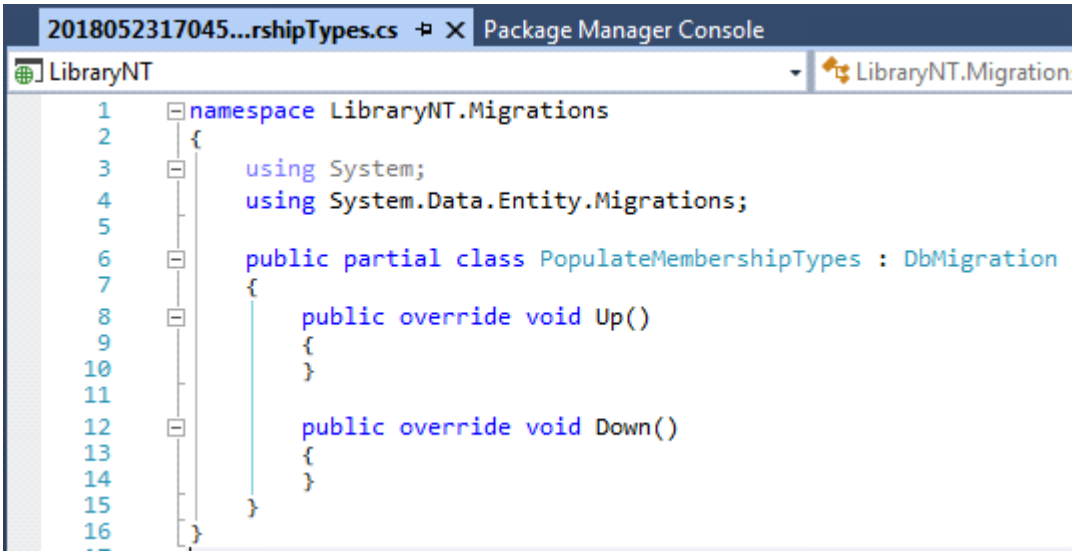
U Customer.cs dodajemo property tipa MembershipType



27 - Seeding the Database

MembershipTypes tabela u bazi je prazna, a trebamo da imamo neke fiksne vrednosti za nju. Ove vrednosti ćemo osigurati da svudu budu iste tako što ćemo ih obuhvatiti migracijom. Svaka izmena na bazi podataka bi trebalo da prolazi kroz migracione skripte.

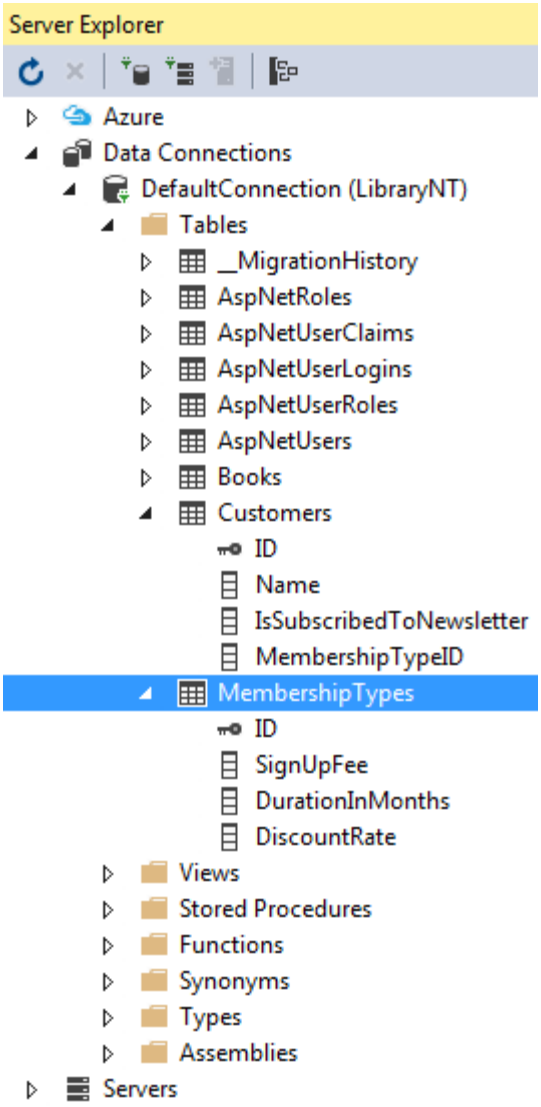
Package manager > add-migration PopulateMembershipTypes . Otvorimo i vidimo praznu migraciju u **Migrations** folderu.



U **Up()** metodi možemo pozvati **SQL()** metodu i unutra proslediti SQL upit.

```
namespace LibraryNT.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class PopulateMembershipTypes : DbMigration
    {
        public override void Up()
        {
            Sql("INSERT INTO MembershipTypes (ID, SignUpFee, DurationInMonths, DiscountRate) VALUES (1, 0, 0, 0)");
            Sql("INSERT INTO MembershipTypes (ID, SignUpFee, DurationInMonths, DiscountRate) VALUES (2, 30, 1, 10)");
            Sql("INSERT INTO MembershipTypes (ID, SignUpFee, DurationInMonths, DiscountRate) VALUES (3, 90, 3, 15)");
            Sql("INSERT INTO MembershipTypes (ID, SignUpFee, DurationInMonths, DiscountRate) VALUES (4, 300, 12, 20)");
        }

        public override void Down()
        {
        }
    }
}
```

U package manager-u **update-database**.



Desni klik na MembershipTypes > New Query

20180523170450...ershipTypes.cs | SQLQuery1.sql

aspnet-LibraryNT-20180506052901

```
SELECT * FROM MembershipTypes
```

100 %

T-SQL | Results | Message

	ID	SignUpFee	DurationInMonths	DiscountRate
1	1	0	0	0
2	2	30	1	10
3	3	90	3	15
4	4	300	12	20

28 - Overriding Conventions

U modelu dodamo **using System.ComponentModel.DataAnnotations;**

Customer.cs | MembershipType.cs

LibraryNT | LibraryNT.Models.Customer

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;

namespace LibraryNT.Models
{
    public class Customer
    {
        public int ID { get; set; }
        [Required]
        [StringLength(255)]
        public string Name { get; set; }
        public bool IsSubscribedToNewsletter { get; set; }
        public MembershipType MembershipType { get; set; }
        // Entity framework prepoznaje ovu konvenciju sa ID na kraju i tretira kao FK.
        public byte MembershipTypeID { get; set; }
    }
}
```

U NuGet **add-migration** pa **update-database**.

29 - Querying Objects

Demonstriraćemo pregled naših korisnika - Customers. Više ne želimo da imamo hard-kodirane kupce već želimo da ih povlačimo iz baze podataka.

```
private IEnumerable<Customer> GetCustomers()
{
    return new List<Customer>
    {
        new Customer { ID = 1, Name = "John Smith" },
        new Customer { ID = 2, Name = "Mary Williams" }
    };
}
```

Potreban nam je **DbContext** kako bismo pristupili bazi podataka.

```
Customer.cs | CustomersController.cs | MembershipType.cs
LibraryNT | LibraryNT.Cont
4   using System.Web;
5   using System.Web.Mvc;
6   using LibraryNT.Models;
7
8   namespace LibraryNT.Controllers
9   {
10      public class CustomersController : Controller
11      {
12          // Deklarisanje DbContext promenjive.
13          private ApplicationDbContext _context;
14
15          // Pravljenje konstruktora.
16          public CustomersController()
17          {
18              _context = new ApplicationDbContext();
19          }
20
21          // DbContext je disposable objekat, pa ga uklanjamo.
22          protected override void Dispose(bool disposing)
23          {
24              _context.Dispose();
25          }
```

Umesto pozivanja **GetCustomers()** metode, incijalizujemo customers kao **_context.Customers**. Customers svojstvo je DbSet definisan u našem DbContext-u *(vidi 25)*. Ovime dobijamo sve Customere iz base.

```
27      // GET: Customers
28      public ActionResult Index()
29      {
30          var customers = _context.Customers;
31
32          return View(customers);
33      }
```

Upit se ne izvršava ovim, već kada vršimo iteraciju kroz ovaj objekat. Vidi view.

```
15      <table class="table table-bordered table-hover">
16          <thead>
17              <tr>
18                  <th>Customer</th>
19              </tr>
20          </thead>
21          <tbody>
22              @foreach (var customer in Model)
23              {
24                  <tr>
25                      <td>@Html.ActionLink(customer.Name, "Details", "Customers", new { ID = customer.ID }, null)</td>
26                  </tr>
27              }
28          </tbody>
29      </table>
30  }
```

Alternativno, možemo query-ovati dodavanjem .ToList().

```
// GET: Customers
public ActionResult Index()
{
    var customers = _context.Customers.ToList();

    return View(customers);
}
```

Više nam ne treba GetCustomers() metoda.

Customer.cs | CustomersController.cs | MembershipType.cs

LibraryNT | LibraryNT.Controllers.CustomersControl

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Web;
5   using System.Web.Mvc;
6   using LibraryNT.Models;
7
8   namespace LibraryNT.Controllers
9   {
10      public class CustomersController : Controller
11      {
12          // Deklarisanje DbContext promenjive.
13          private ApplicationDbContext _context;
14
15          // Pravljenje konstruktora.
16          public CustomersController()
17          {
18              _context = new ApplicationDbContext();
19          }
20
21          // DbContext je disposable objekat, pa ga uklanjamo.
22          protected override void Dispose(bool disposing)
23          {
24              _context.Dispose();
25          }
26
27          // GET: Customers
28          public ActionResult Index()
29          {
30              var customers = _context.Customers.ToList();
31
32              return View(customers);
33          }
34
35          public ActionResult Details(int id)
36          {
37              // Naš query će se odmah izvršiti zbog SingleOrDefault metode.
38              var customer = _context.Customers.SingleOrDefault(c => c.ID == id);
39
40              if (customer == null)
41                  return HttpNotFound();
42
43              return View(customer);
44          }
45      }
46  }
```

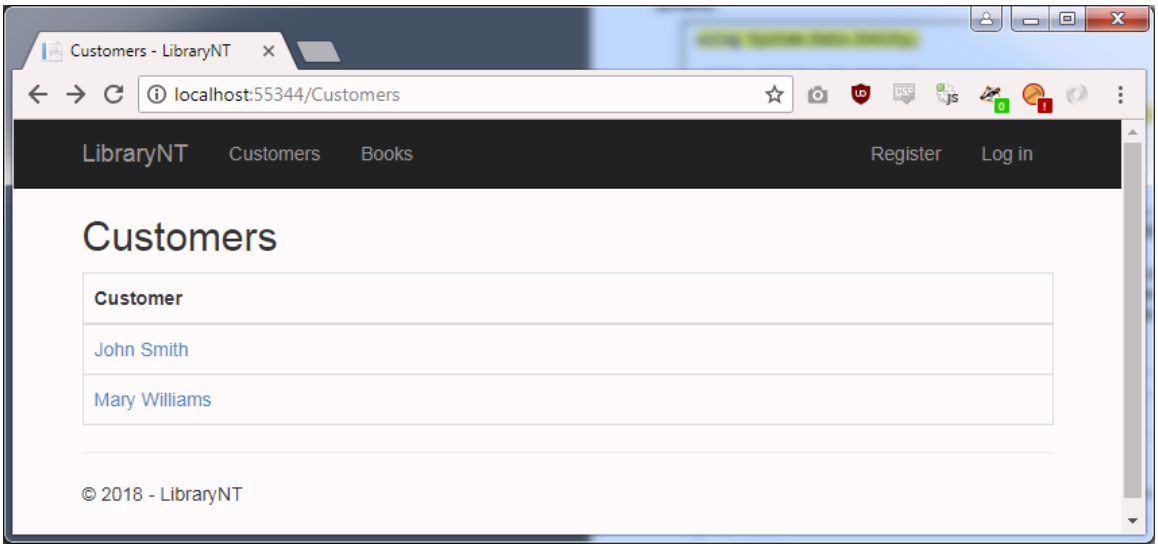
Popunimo bazu sa nekim podacima. Ovo možemo ručno, MembershipType je bio migracija jer je uvek isti.

Customer.cs | CustomersController.cs | MembershipType.cs | dbo.Customers [Data]

Max Rows: 1000

ID	Name	IsSubscribedT...	MembershipT...
1	John Smith	False	1
2	Mary Williams	True	2
NULL	NULL	NULL	NULL

Radi.



30 - Eager Loading

Dodajemo discount rate pored customer-a u view-u. Sa eager loading customer-e i membership type-ove učitavamo zajedno.

CustomersController.cs - menjamo index stranicu.

CustomersController.cs | Customer.cs | Index.cshtml | MembershipType.cs | dbo.Customers [Data]

LibraryNT | LibraryNT.Controllers.CustomersController

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Web;
5   using System.Web.Mvc;
6   using LibraryNT.Models;
7   using System.Data.Entity;
8
9   namespace LibraryNT.Controllers
10  {
11      public class CustomersController : Controller
12      {
13          // Deklarisanje DbContext promenjive.
14          private ApplicationDbContext _context;
15
16          // Pravljenje konstruktora.
17          public CustomersController()
18          {
19              _context = new ApplicationDbContext();
20          }
21
22          // DbContext je disposable objekat, pa ga uklanjamo.
23          protected override void Dispose(bool disposing)
24          {
25              _context.Dispose();
26          }
27
28          // GET: Customers
29          public ActionResult Index()
30          {
31              var customers = _context.Customers.Include(c => c.MembershipType).ToList();
32
33              return View(customers);
34          }
35
36          public ActionResult Details(int id)
37          {
38              // Naš query će se odmah izvršiti zbog SingleOrDefault metode.
39              var customer = _context.Customers.SingleOrDefault(c => c.ID == id);
40
41              if (customer == null)
42                  return HttpNotFound();
43
44              return View(customer);
45          }
46      }
47  }
```

Views > Customers > Index.cshtml

Customer.cs | CustomersController.cs | Index.cshtml | MembershipType.cs | dbo.Customers [Data]

```
1   @model IEnumerable<LibraryNT.Models.Customer>
2
3   @{
4       ViewBag.Title = "Customers";
5       Layout = "~/Views/Shared/_Layout.cshtml";
6   }
7
8   <h2>Customers</h2>
9   @if (!Model.Any())
10  {
11      <p>We don't have any customers yet.</p>
12  }
13  else
14  {
15      <table class="table table-bordered table-hover">
16          <thead>
17              <tr>
18                  <th>Customer</th>
19                  <th>Discount Rate</th>
20              </tr>
21          </thead>
22          <tbody>
23              @foreach (var customer in Model)
24              {
25                  <tr>
26                      <td>@Html.ActionLink(customer.Name, "Details", "Customers", new { ID = customer.ID }, null)</td>
27                      <td>@customer.MembershipType.DiscountRate</td>
28                  </tr>
29              }
30          </tbody>
31      </table>
32  }
```

Vežba 1

Dodati Name atribut MembershipType-u, updateovati bazu i podatke preko migracija i prikazati u view-u.

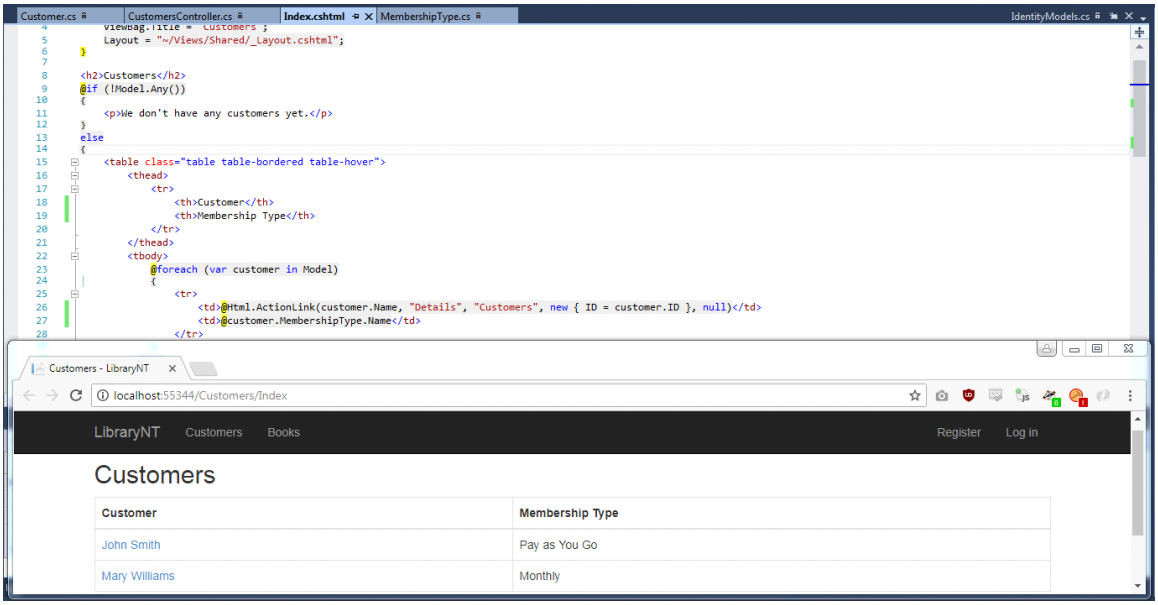
1. Models > MembershipType.cs dodamo public string Name { get; set; }
2. add-migration AddNameMembershipType; update-database.
3. add-migration UpdateNameMembershipType
4. Otvorimo tu migraciju, u Up() metodi pišemo **SQL("UPDATE MembershipTypes SET ...")**;
Sql("UPDATE MembershipTypes SET Name = 'Quarterly' WHERE ID = 3");
5. update-database

20180524122739...ershipTypes.cs | dbo.MembershipTypes [Data] | Output

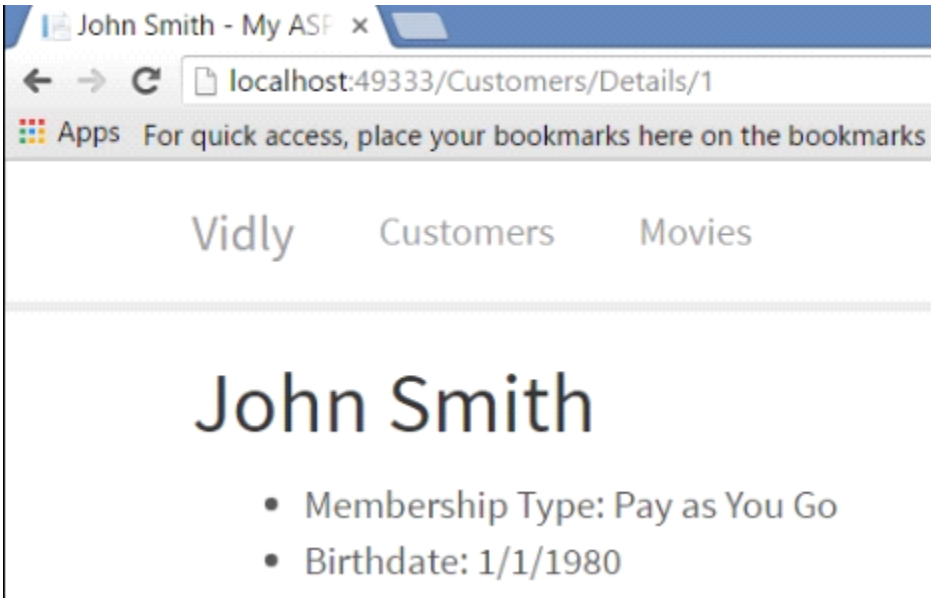
Max Rows: 1000

ID	SignUpFee	DurationInMo...	DiscountRate	Name
1	0	0	0	Pay as You Go
2	30	1	10	Monthly
3	90	3	15	Quarterly
4	300	12	20	Annual
NULL	NULL	NULL	NULL	NULL

6. Izmenimo view - **@customer.MembershipType.Name**



Vežba 2



U customers/details strani prikazati Membership Type i Birthdate ako je on definisan. Birthdate je nullable datetime.

1.

dbo.Customers [Data]

LibraryNT | LibraryNT.Models.Customer

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Web;
5   using System.ComponentModel.DataAnnotations;
6
7   namespace LibraryNT.Models
8   {
9       public class Customer
10      {
11          public int ID { get; set; }
12          [Required]
13          [StringLength(255)]
14          public string Name { get; set; }
15          public bool IsSubscribedToNewsletter { get; set; }
16          public MembershipType MembershipType { get; set; }
17          // Entity framework prepoznaje ovu konvenciju sa ID na kraju i tretira kao FK.
18          public byte MembershipTypeID { get; set; }
19          public DateTime? Birthdate { get; set; }
20      }
21  }
```

2. add-migration

3. update-database

4. Ručno unesemo datum u jedno polje.

5. CustomersController.cs , eager loadujemo Details metodu.

CustomersController.cs | Details.cshtml | dbo.Customers [Data]

LibraryNT | LibraryNT.Controllers.CustomersController | Dispose(bool disposing)

```
36      public ActionResult Details(int id)
37      {
38          // Naš query će se odmah izvršiti zbog SingleOrDefault metode.
39          var customer = _context.Customers.Include(c => c.MembershipType).SingleOrDefault(c => c.ID == id);
40
41          if (customer == null)
42              return HttpNotFound();
43
44          return View(customer);
45      }
46  }
47  }
```

6. Views > Customers > Details.cshtml

```
CustomersController.cs | Details.cshtml | dbo.Customers [Data]
1   @model LibraryNT.Models.Customer
2
3   @{
4       ViewBag.Title = Model.Name;
5       Layout = "~/Views/Shared/_Layout.cshtml";
6   }
7
8   <h2>@Model.Name</h2>
9
10
11  <ul>
12      <li>Membership Type: @Model.MembershipType.Name</li>
13      @if (Model.Birthdate.HasValue)
14      {
15      <li>Birthdate: @Model.Birthdate.Value.ToShortDateString()</li>
16      }
17  </ul>
18
19
100 %
```

Vežba 3

1. Desni klik na models > add > new class > genre.cs . Using DataAnnotations, strlen255.

```
Book.cs | CustomersController.cs | Details.cshtml | Genre.cs
LibraryNT | LibraryNT.Models.Ge
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Web;
5   using System.ComponentModel.DataAnnotations;
6
7   namespace LibraryNT.Models
8   {
9       public class Genre
10      {
11          public int ID { get; set; }
12          [StringLength(255)]
13          public string Name { get; set; }
14      }
15  }
```

- add-migration AddBookGenre; update-database.

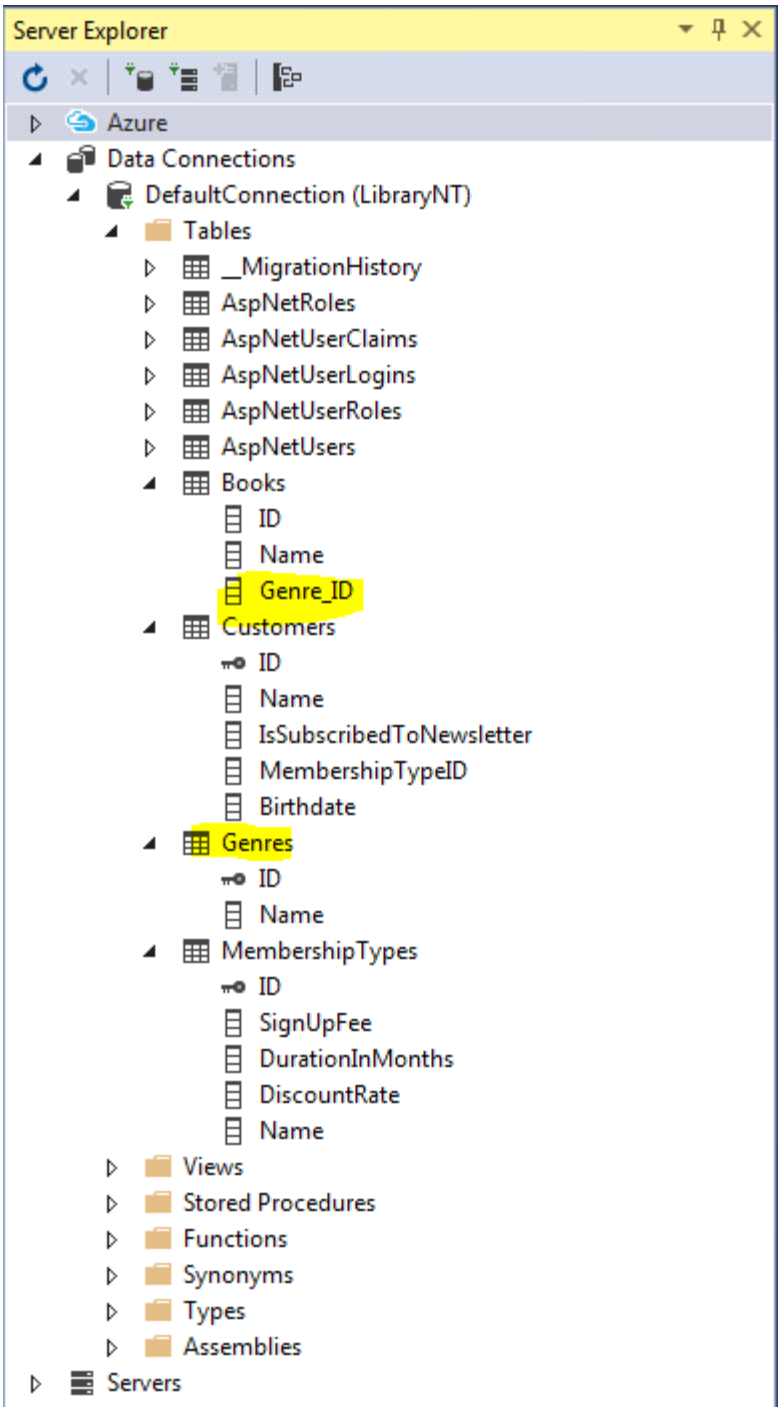
2. Models > Book.cs, dodamo Genre objekat i par novih atributa.

Book.cs | BooksController.cs | CustomersController.cs

LibraryNT

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Web;
5   using System.ComponentModel.DataAnnotations;
6
7   namespace LibraryNT.Models
8   {
9       public class Book
10      {
11          public int ID { get; set; }
12          public string Name { get; set; }
13          public Genre Genre { get; set; }
14          public DateTime DateAdded { get; set; }
15          public byte NumberInStock { get; set; }
16      }
17  }
18
```

3. add-migration ConnectBookGenre ; **update-database**.
Sam je napravio Genre_ID



4. add-migration PopulateGenre, pa PopulateBooks, u **Up()** metodi pišemo SQL naredbe.

```
2018052417332...ulateBooks.cs | 2018052417272...ddBookGenre.cs | 2018052417314...okAndGenre.cs | 2018052417461...dudantBooks.cs | Details.cshtml
LibraryNT | LibraryNT.Migrations.PopulateBooks
1   namespace LibraryNT.Migrations
2   {
3       using System;
4       using System.Data.Entity.Migrations;
5
6       public partial class PopulateBooks : DbMigration
7       {
8           public override void Up()
9           {
10              Sql("INSERT INTO Books(Name, DateAdded, NumberInStock, Genre_ID) VALUES ('Romeo and Juliet', '2018/05/24', 15, 1)");
11              Sql("INSERT INTO Books(Name, DateAdded, NumberInStock, Genre_ID) VALUES ('Macbeth', '2018/05/18', 10, 1)");
12              Sql("INSERT INTO Books(Name, DateAdded, NumberInStock, Genre_ID) VALUES ('The Shining', '2018/05/24', 5, 4)");
13          }
14
15          public override void Down()
16          {
17          }
18      }
19  }
20
```

5. Controllers > BooksController.cs - Dodajemo ono _context i Include(b => b.Genre) + details.

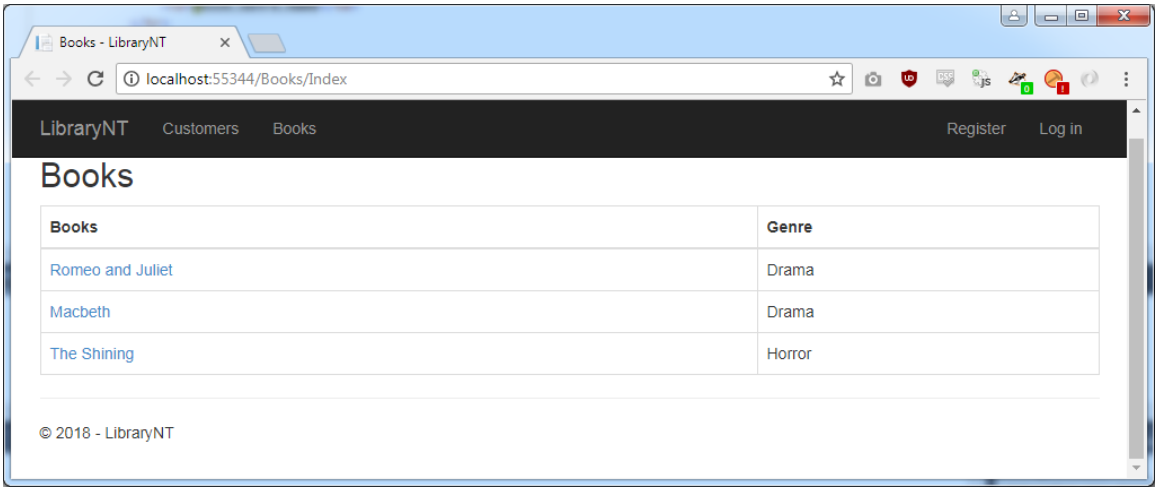
```
Book.cs | BooksController.cs | CustomersController.cs | Details.cshtml | Genre.cs
LibraryNT | LibraryNT.Controllers.BooksController
4   using System.Web;
5   using System.Web.Mvc;
6   using LibraryNT.Models;
7   using LibraryNT.ViewModels;
8   using System.Data.Entity;
9
10  namespace LibraryNT.Controllers
11  {
12      public class BooksController : Controller
13      {
14          // Deklarisanje DbContext promenjive.
15          private ApplicationDbContext _context;
16
17          // Pravljenje konstruktora.
18          public BooksController()
19          {
20              _context = new ApplicationDbContext();
21          }
22
23          // DbContext je disposable objekat, pa ga uklanjamo.
24          protected override void Dispose(bool disposing)
25          {
26              _context.Dispose();
27          }
28
29          // Ne zaboravi using System.Data.Entity gore.
30          public ViewResult Index()
31          {
32              var books = _context.Books.Include(b => b.Genre).ToList();
33
34              return View(books);
35          }
36
37          public ActionResult Details(int id)
38          {
39              // Naš query će se odmah izvršiti zbog SingleOrDefault metode.
40              var books = _context.Books.Include(b => b.Genre).SingleOrDefault(c => c.ID == id);
41
42              if (books == null)
43                  return HttpNotFound();
44
45              return View(books);
46          }
47      }
48  }
```

6. Views > Books > Index.cshtml

Dodajemo ActionLink(tekst koji piše, metoda, kontroler, parametar, html atributi)

Book.cs | BooksController.cs | CustomersController.cs | Details.cshtml | Genre.cs | Index.cshtml

```
1   @model IEnumerable<LibraryNT.Models.Book>
2
3   @{
4       ViewBag.Title = "Books";
5       Layout = "~/Views/Shared/_Layout.cshtml";
6   }
7
8   <h2>Books</h2>
9   <table class="table table-bordered table-hover">
10      <thead>
11          <tr>
12              <th>Books</th>
13              <th>Genre</th>
14          </tr>
15      </thead>
16      <tbody>
17          @foreach (var book in Model)
18          {
19              <tr>
20                  <td>@Html.ActionLink(book.Name, "Details", "Books", new { id = book.ID }, null)</td>
21                  <td>@book.Genre.Name</td>
22              </tr>
23          }
24      </tbody>
25  </table>
26
```



Views > Books > Detail.cshtml

Book.cs | BooksController.cs | CustomersController.cs | Details.

```
1   @model LibraryNT.Models.Book
2
3   @{
4       ViewBag.Title = Model.Name;
5       Layout = "~/Views/Shared/_Layout.cshtml";
6   }
7
8   <h2>@Model.Name</h2>
9   <ul>
10      <li>Genre: @Model.Genre.Name</li>
11      <li>Date Added: @Model.DateAdded.ToLongDateString()</li>
12      <li>Number in Stock: @Model.NumberInStock</li>
13  </ul>
```

Romeo and Juliet - Librar

localhost:553...

LibraryNT

Romeo and Juliet

- Genre: Drama
- Date Added: 24. maj 2018
- Number in Stock: 15

© 2018 - LibraryNT

38 - Building forms - markup

Za pravljenje forme za dodavanje Customera, prvo nam je potrebna akcija koja vraća view koji sadrži formu.

Controllers > CustomersController.cs

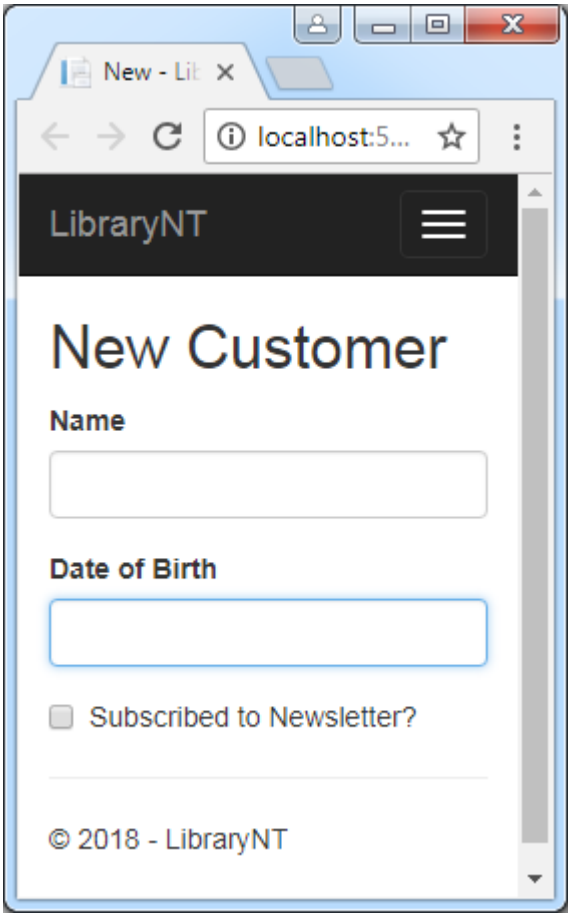
mvcaction4 <tab> <tab> ; nazovemo metodu New. Desni klik na View unutra > Add view.

```
public ActionResult New()
{
    return View();
}
```

Views > Customers > New.cshtml

New.cshtml | Book.cs | BooksController.cs | CustomersController.cs | Details.cshtml | Details.cshtml | Genre.cs

```
@model LibraryNT.Models.Customer

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

<h2>New Customer</h2>

@* Html.BeginForm("CiljanaAkcijaNakonPostovanjaForme", "Kontroler")
   Ovo je disposable objekat tako da ga koristimo sa using.        *@
@using (@Html.BeginForm("Create", "Customers"))
{
    <div class="form-group">
        @* m as a model (model is Customer) goes to m.name. LabelFor will use m.Name's value for text.*@
        @Html.LabelFor(m => m.Name)
        @* Drugi argument je anoniman objekat u kojem možemo staviti HTML atribute.*@
        @Html.TextBoxFor(m => m.Name, new { @class = "form-control" })
    </div>

    <div class="form-group">
        <label for="BirthDate">Date of Birth</label>
        @Html.TextBoxFor(m => m.Birthdate, new { @class = "form-control" })
    </div>

    <div class="checkbox">
        <label>
            @Html.CheckBoxFor(m => m.IsSubscribedToNewsletter) Subscribed to Newsletter?
        </label>
    </div>
}
```



40 - Drop-down Lists

```
public ActionResult New()
{
    // Nabavljamo listu MembershipTypes iz baze podataka.
    // Pre ovoga idemo u models > IdentityModels.cs
    // public DbSet<MembershipType> MembershipTypes { get; set; }
    var membershipTypes = _context.MembershipTypes.ToList();
    // Sad trebamo da prosledimo i membershipTypes i Customer view-u
    // Zato pravimo viewModel.
    return View();
}
```

Desni klik na ViewModels > Add > Class > NewCustomerViewModel.cs

```
NewCustomerViewModel.cs   CustomersController.cs
LibraryNT                 LibraryNT.ViewMoc
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Web;
5   using LibraryNT.Models;
6
7   namespace LibraryNT.ViewModels
8   {
9       public class NewCustomerViewModel
10      {
11          // Lista/IEnumerable membershipTypes.
12          public IEnumerable<MembershipType> MembershipTypes { get; set; }
13          public Customer Customer { get; set; }
14      }
15  }
```

Controllers > CustomersController.cs

```
public ActionResult New()
{
    // Nabavljamo listu MembershipTypes iz baze podataka.
    // Pre ovoga idemo u models > IdentityModels.cs
    // public DbSet<MembershipType> MembershipTypes { get; set; }
    var membershipTypes = _context.MembershipTypes.ToList();
    // Sad trebamo da prosledimo i membershipTypes i Customer view-u
    // Zato pravimo viewModel.
    // using LibraryNT.ViewModels na vrhu.
    var viewModel = new NewCustomerViewModel
    {
        MembershipTypes = membershipTypes
    };

    return View(viewModel);
}
```

Views > New.cshtml

Promenimo @model u @model LibraryNT.**ViewModels.NewCustomerViewModel**

```
Customer.cs   New.cshtml
1   @model LibraryNT.ViewModels.NewCustomerViewModel
2
3   @{
4       ViewBag.Title = "New";
5       Layout = "~/Views/Shared/_Layout.cshtml";
6   }
7
8   <h2>New Customer</h2>
9
10  @* Html.BeginForm("CiljanaAkcijaNakonPostovanjaForme", "Kontroler")
11     Ovo je disposable objekat tako da ga koristimo sa using.        *@
12  @using (@Html.BeginForm("Create", "Customers"))
13  {
14      <div class="form-group">
15          @* m as a model (model is Customer) goes to m.name. LabelFor will use m.Name's value for text.*@
16          @Html.LabelFor(m => m.Customer.Name)
17          @* Drugi argument je anoniman objekat u kojem možemo staviti HTML atribute.*@
18          @Html.TextBoxFor(m => m.Customer.Name, new { @class = "form-control" })
19      </div>
20
21      <div class="form-group">
22          <label for="BirthDate">Date of Birth</label>
23          @Html.TextBoxFor(m => m.Customer.Birthdate, new { @class = "form-control" })
24      </div>
25
26      <div class="checkbox">
27          <label>
28              @Html.CheckBoxFor(m => m.Customer.IsSubscribedToNewsletter) Subscribed to Newsletter?
29          </label>
30      </div>
31
32      <div class="form-group">
33          @* Umesto ručne HTML labele možemo i u modelu iznad prop da stavimo*@
34          @* [Display(Name = "Membership Type")]*@
35          <label for="MembershipTypeID">Membership Type</label>
36          @* Drugi argument je lista predmeta koji će biti u listi.*@
37          @* SelectList(<ListaPredmeta>, <PropertyUDatojKlasiKojiSadrziValue>, <PropKojiSadrziTekst>)*@
38          @Html.DropDownListFor(m => m.Customer.MembershipTypeID, new SelectList(Model.MembershipTypes, "ID", "Name"), "Select Membership Type", new { @class = "form-control" })
39      </div>
40  }
```

41 - Model Binding

Dodajemo submit dugme, na vrhu u **BeginForm** vidimo da se cilja na akciju **Create** te u **CustomersController.cs** pravimo tu metodu.

```
@using (@Html.BeginForm("Create", "Customers"))
{
    <div class="form-group">
        @* m as a model (model is Customer) goes to m.name. LabelFor will use m.Name's value for text.*@
        @Html.LabelFor(m => m.Customer.Name)
        @* Drugi argument je anoniman objekat u kojem možemo staviti HTML atribute.*@
        @Html.TextBoxFor(m => m.Customer.Name, new { @class = "form-control" })
    </div>

    <div class="form-group">
        <label for="BirthDate">Date of Birth</label>
        @Html.TextBoxFor(m => m.Customer.Birthdate, new { @class = "form-control" })
    </div>

    <div class="checkbox">
        <label>
            @Html.CheckBoxFor(m => m.Customer.IsSubscribedToNewsletter) Subscribed to Newsletter?
        </label>
    </div>

    <div class="form-group">
        @* Umesto ručne HTML labele možemo i u modelu iznad prop da stavimo*@
        @* [Display(Name = "Membership Type")]*@
        <label for="MembershipTypeID">Membership Type</label>
        @* Drugi argument je lista predmeta koji će biti u listi.*@
        @* SelectList(<ListaPredmeta>, <PropertyUDatojKlasiKojiSadrziValue>, <PropKojiSadrziTekst>)*@
        @Html.DropDownListFor(m => m.Customer.MembershipTypeID, new SelectList(Model.MembershipTypes, "ID", "Name"), "Select Membership Type", new { @class = "form-control" })
    </div>
    <button type="submit" class="btn btn-primary">Save</button>
}
```

CustomersController.cs

```
// [HttpPost] - Akcija može biti pozvana samo POST zahtevom.
// Kao argument ovoj metodi prosleđujemo CustomerViewModel koji je u view-u.
// MVC framework automatski mapira request data za viewModel objekat. - Model binding.
[HttpPost]
public ActionResult Create(NewCustomerViewModel viewModel)
{
    return View();
}
```

42 - Saving Data

```
// [HttpPost] - Akcija može biti pozvana samo POST zahtevom.
// Kao argument ovoj metodi prosleđujemo Customer iz view-a.
[HttpPost]
public ActionResult Create(Customer customer)
{
    // Pre dodavanja customera u bazu, mora prvo da ide u DbContext.
    _context.Customers.Add(customer);
    // Još nije u bazi, već je samo u memoriji.
    // Sada će DbContext proći kroz sve dodate, izmenjene i uklonjene stavke.
    // I ispaliti niz SQL naredbi nad bazom za vreme 1 transakcije.
    _context.SaveChanges();
    // Na kraju korisnika vraćamo na spisak korisnika.
    // (akcija Index u kontroleru CustomersController)
    return RedirectToAction("Index", "Customers");
}
```

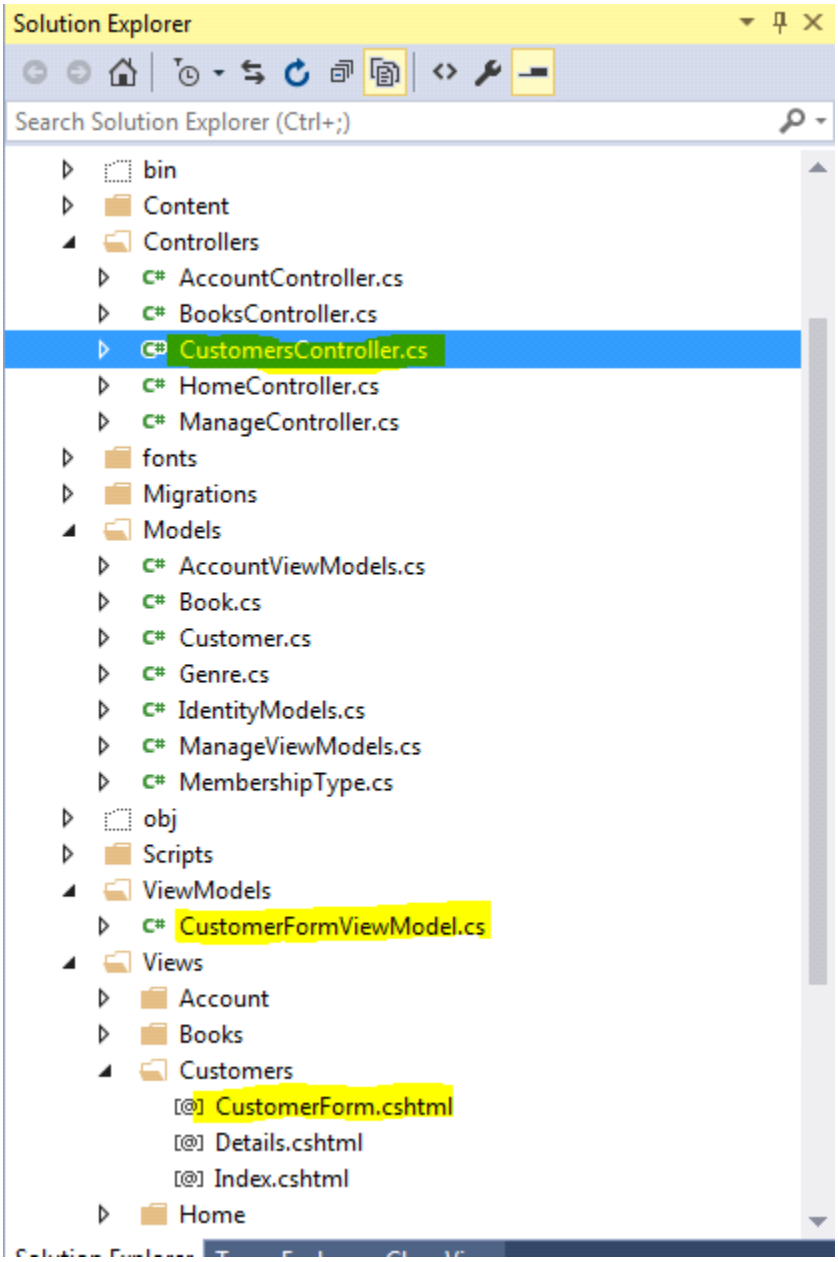
43 - Edit form

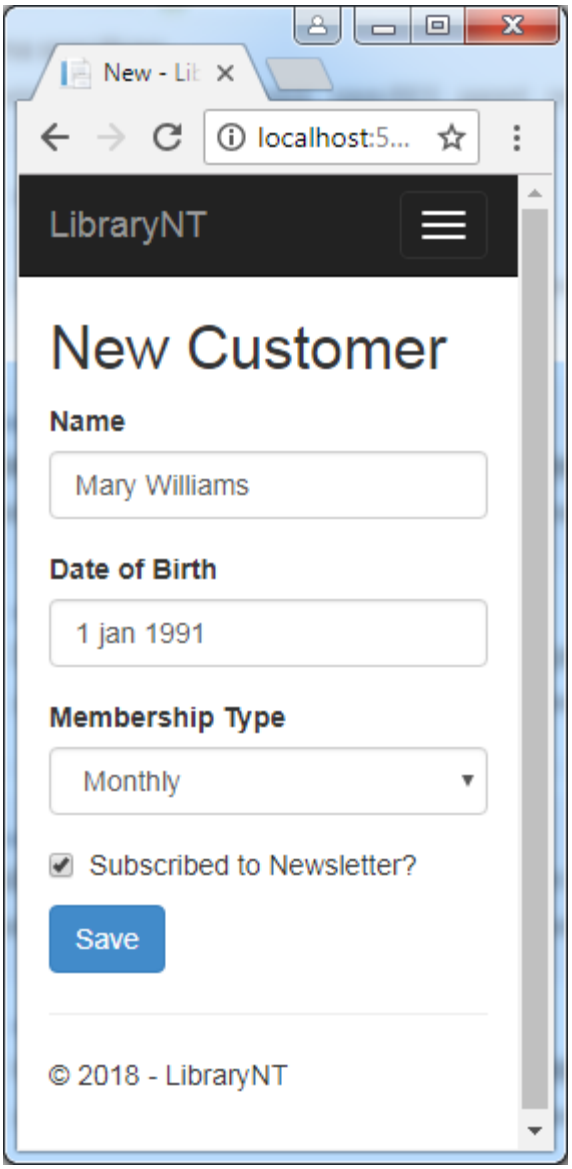
U **Index.cshtml**, klikom na korisnika hoćemo da ode na Edit formu. Menjamo **Html.ActionLink** argument iz **Details** u **Edit** i napravićemo **Edit** metodu u **CustomersController.cs**

```
<tbody>
    @foreach (var customer in Model)
    {
        <tr>
            <td>@Html.ActionLink(customer.Name, "Edit", "Customers", new { ID = customer.ID }, null)</td>
            <td>@customer.MembershipType.Name</td>
        </tr>
    }
</tbody>
```

CustomersController.cs

```
public ActionResult Edit(int ID)
{
    // Prvo kupimo korisnika iz baze koji ima prosleđen id iz Index view-a.
    var customer = _context.Customers.SingleOrDefault(c => c.ID == ID);

    if (customer == null)
        return HttpNotFound();

    //3. Model u New view-u je NewCustomerViewModel, tako da ga pravimo.
    //4. Pošto istu formu koristimo i za New i Edit, radimo rename
    // u CustomerFormViewModel.
    var viewModel = new CustomerFormViewModel
    {
        Customer = customer,
        MembershipTypes = _context.MembershipTypes.ToList()
    };

    // 1. Ako postoji, vraćamo customera i renderujemo formu.
    // 2. Unutra ubacujemo naziv view-a, u suprotnom MVC traži view zvan Edit.
    // 5. Isto tako preimenujemo naziv New u CustomerForm.
    return View("CustomerForm", viewModel);
}
```


Solution Explorer
Search Solution Explorer (Ctrl+;)
bin
Content
Controllers
AccountController.cs
BooksController.cs
CustomersController.cs
HomeController.cs
ManageController.cs
fonts
Migrations
Models
AccountViewModels.cs
Book.cs
Customer.cs
Genre.cs
IdentityModels.cs
ManageViewModels.cs
MembershipType.cs
obj
Scripts
ViewModels
CustomerFormViewModel.cs
Views
Account
Books
Customers
CustomerForm.cshtml
Details.cshtml
Index.cshtml
Home



44 - Updating data

Trenutno u **CustomerForm** view-u imamo formu koja cilja na akciju/metodu **Create** u Customers kontroleru. Možemo ili da napravimo novu akciju **Save** koja će se koristiti i za dodavanje i izmenu customer-a, ili da dinamički menjamo **Html.BeginForm** u zavisnosti da li akcija koja šalje na ovaj view prosleđuje ID ili ne (napravimo novu akciju **Update**). Uradićemo prvu metodu.

Preimenovali smo New u Save.

CustomersController.cs

```
48        // [HttpPost] - Akcija može biti pozvana samo POST zahtevom.
49        // Kao argument ovoj metodi prosleđujemo Customer iz view-a.
50        [HttpPost]
51        public ActionResult Save(Customer customer)
52        {
53            // Pre dodavanja customera u bazu, mora prvo da ide u DbContext.
54            _context.Customers.Add(customer);
55            // Još nije u bazi, već je samo u memoriji.
56            // Sada će DbContext proći kroz sve dodate, izmenjene i uklonjene stavke.
57            // I ispaliti niz SQL naredbi nad bazom za vreme 1 transakcije.
58            _context.SaveChanges();
59            // Na kraju korisnika vraćamo na spisak korisnika.
60            // (akcija Index u kontroleru CustomersController)
61            return RedirectToAction("Index", "Customers");
62        }
```

CustomerForm.cshtml

```
@using (@Html.BeginForm("Save", "Customers"))
{
```

CustomersController.cs

```
// [HttpPost] - Akcija može biti pozvana samo POST zahtevom.
// Kao argument ovoj metodi prosleđujemo Customer iz view-a.
[HttpPost]
public ActionResult Save(Customer customer)
{
    // Proveravamo da li customer ima ID. Ako nema ID to znači da je nov.
    // Ako je nov, dodajemo u bazu. Ako nije, update-ujemo.
    if (customer.ID == 0)
        // Pre dodavanja customera u bazu, mora prvo da ide u DbContext.
        _context.Customers.Add(customer);
    else
    {
        // Da bismo update-ovali entitet, prvo ga moramo dohvatiti iz baze.
        // Zatim DbContext prati izmene unutar tog entiteta.
        // Na kraju pozivamo SaveChanges();
        var customerInDb = _context.Customers.Single(c => c.ID == customer.ID);

        // TryUpdateModel(customerInDb); Uzima sve podatke iz Request data.
        // A možemo i ručno da podesimo property-e customer objekta.
        customerInDb.Name = customer.Name;
        customerInDb.Birthdate = customer.Birthdate;
        customerInDb.MembershipTypeID = customer.MembershipTypeID;
        customer.IsSubscribedToNewsletter = customer.IsSubscribedToNewsletter;
    }
    // Još nije u bazi, već je samo u memoriji.
    // Sada će DbContext proći kroz sve dodate, izmenjene i uklonjene stavke.
    // I ispaliti niz SQL naredbi nad bazom za vreme 1 transakcije.
    _context.SaveChanges();
    // Na kraju korisnika vraćamo na spisak korisnika.
    // (akcija Index u kontroleru CustomersController)
    return RedirectToAction("Index", "Customers");
}
```

U **Views > Customers > CustomerForm.cshtml** dodajemo hidden CustomerID.

```
@using (@Html.BeginForm("Save", "Customers"))
{
    <div class="form-group">
        @* m as a model (model is Customer) goes to m.name. LabelFor will use m.Name's value for text.*@
        @Html.LabelFor(m => m.Customer.Name)
        @* Drugi argument je anoniman objekat u kojem možemo staviti HTML atribute.*@
        @Html.TextBoxFor(m => m.Customer.Name, new { @class = "form-control" })
    </div>

    <div class="form-group">
        <label for="BirthDate">Date of Birth</label>
        @Html.TextBoxFor(m => m.Customer.Birthdate, "{0:d MMM yyyy}", new { @class = "form-control" })
    </div>

    <div class="form-group">
        @* Umesto ručne HTML labele možemo i u modelu iznad prop da stavimo*@
        @* [Display(Name = "Membership Type")]*@
        <label for="MembershipTypeID">Membership Type</label>
        @* Drugi argument je lista predmeta koji će biti u listi.*@
        @* SelectList(<ListaPredmeta>, <PropertyUDatojKlasiKojiSadrziValue>, <PropKojiSadrziTekst>)*@
        @Html.DropDownListFor(m => m.Customer.MembershipTypeID, new SelectList(Model.MembershipTypes, "ID", "Name"), "Select Membership Type", new { @class = "form-control" })
    </div>

    <div class="checkbox">
        <label>
            @Html.CheckBoxFor(m => m.Customer.IsSubscribedToNewsletter) Subscribed to Newsletter?
        </label>
    </div>
    @Html.HiddenFor(m => m.Customer.ID)
    <button type="submit" class="btn btn-primary">Save</button>
}
```

50 - Adding Validation

ASP.NET koristi DataAnnotations iz modela za validaciju parametara.

Za validaciju pratimo 3 koraka :

1. Dodavanje DataAnnotations nad entitetima / u modelu.

```
using System;
using System.ComponentModel.DataAnnotations;

namespace MovieNT.Models
{
    public class Customer
    {
        public int Id { get; set; }

        [Required]
        [StringLength(255)]
        public string Name { get; set; }

        public bool IsSubscribedToNewsletter { get; set; }

        public MembershipType MembershipType { get; set; }

        // Entity framework prepoznaje ovu konvenciju sa ID na kraju i tretira to kao FK.
        // [Display] koristimo ovde kako bi u View-u labele imale taj tekst.
        // Alternativno, u view-u ručno pravimo labele.
        [Display(Name = "Membership Type")]
        public byte MembershipTypeId { get; set; }

        [Display(Name = "Date of Birth")]
        public DateTime? Birthdate { get; set; }
    }
}
```

2. Korišćenje ModelState.IsValid za izmenu toka programa.
Controllers > CustomersController.cs

```
// [HttpPost] - Akcija može biti pozvana samo POST zahtevom.
// Kao argument ovoj metodi prosleđujemo Customer iz view-a.
[HttpPost]
public ActionResult Save(Customer customer)
{
    // Koristimo ModelState property za pristup podacima za validaciju.
    if (!ModelState.IsValid)
    {   // Ako ModelState nije validan, onda želimo da vratimo korisnika na formu.
        // Za taj view nam je potreban CustomerFormViewModel, i u njega ubacujemo primljeni customer objekat.
        var viewModel = new CustomerFormViewModel
        {
            Customer = customer,
            MembershipTypes = _context.MembershipTypes.ToList()
        };
        return View("CustomerForm", viewModel);
    }

    // U suprotnom, radi create ili update i vraća korisnika na spisak Customera.

    // Proveravamo da li customer ima ID. Ako nema ID to znači da je nov.
    // Ako je nov, dodajemo u bazu. Ako nije, update-ujemo.
    if (customer.Id == 0)
        // Pre dodavanja customera u bazu, mora prvo da ide u DbContext.
        _context.Customers.Add(customer);
```

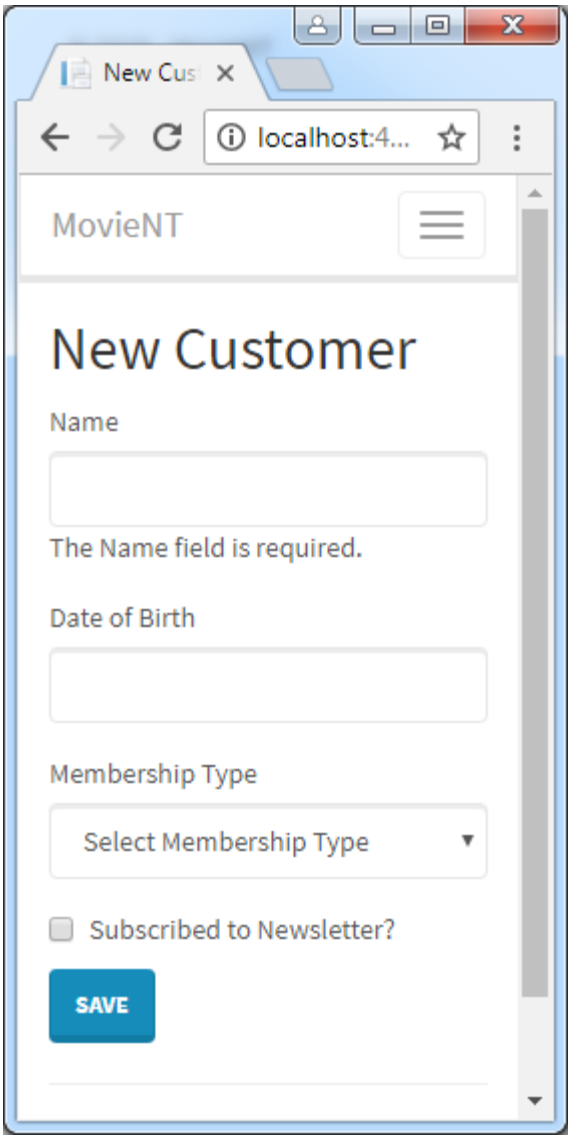
3. Dodavanje validacionih poruka nad formom.

U **Views > Customers > CustomerForm.cshtml** moramo da stavimo placeholder za validacione poruke pored svakog polja kojem je potrebna validacija.

Na primer, Customer.Name je obavezno i maksimalna dužina mu je 255 karaktera. *(vidi iznad)*

```
CustomerForm.cshtml   CustomersController.cs
1   @model MovieNT.ViewModels.CustomerFormViewModel
2   @{
3       ViewBag.Title = Model.Title;
4       Layout = "~/Views/Shared/_Layout.cshtml";
5   }
6   <h2>@Model.Title</h2>
7   @* Html.BeginForm("CiljanaAkcijaNakonPostovanjaForme", "Kontroler")
8       Ovo je disposable objekat tako da ga koristimo sa using.        *@
9   @using (Html.BeginForm("Save", "Customers"))
10  {
11      <div class="form-group">
12          @* m as a model (model is Customer) goes to m.name. LabelFor will use m.Name's value for text.*@
13          @Html.LabelFor(m => m.Customer.Name)
14          @* Drugi argument je anoniman objekat u kojem možemo staviti HTML atribute.*@
15          @Html.TextBoxFor(m => m.Customer.Name, new { @class = "form-control" })
16          @* Placeholder za validacione poruke. *@
17          @Html.ValidationMessageFor(m => m.Customer.Name)
18      </div>
```

Rezultat



51 - Styling Validation Errors

Content > Site.css

Samo dodamo CSS klasama koje se pojavljuju prilikom neuspele validacije nad inputima.

CustomerForm.cshtml | CustomersController.cs | Site.css

```
1  body {
2      padding-top: 50px;
3      padding-bottom: 20px;
4  }
5
6  /* Set padding to keep content from hitting the edges */
7  .body-content {
8      padding-left: 15px;
9      padding-right: 15px;
10 }
11
12 /* Override the default bootstrap behavior where horizontal description lists
13    will truncate terms that are too long to fit in the left column
14 */
15 .dl-horizontal dt {
16     white-space: normal;
17 }
18
19 /* Set width on the form input elements since they're 100% wide by default */
20 input,
21 select,
22 textarea {
23     max-width: 280px;
24 }
25
26 .field-validation-error {
27     color: red;
28 }
29
30 .input-validation-error {
31     border: 2px solid red;
32 }
```

52 - Data Annotations

Data Annotations

- [Required]
- [StringLength(255)]
- [Range(1, 10)]
- [Compare("OtherProperty")]
- [Phone]
- [EmailAddress]
- [Url]
- [RegularExpression("···")]

Custom validation message

```
[Required(ErrorMessage = "Please enter customer's name.")]
[StringLength(255)]
public string Name { get; set; }
```

Custom validation

Ako customer hoće da ide na Monthly, Quarterly ili Annual membership mora da ima iznad 18 godina. Dakle, svaki Membership Type osim Pay as You Go mora da bude stariji od 18.

Desni klik models > Add > New class > Min18YearsIfAMember

Ova klasa mora biti izvedena od **ValidationAtribute**, koji je definisan System.ComponentModel.DataAnnotations. Unutra overridujemo IsValid metodu.

Customer.cs | Models\Min18...sIfAMember.cs

MovieNT | MovieNT.Models.Min18YearsIfAMember

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.ComponentModel.DataAnnotations;
6
7  namespace MovieNT.Models
8  {
9      public class Min18YearsIfAMember : ValidationAttribute
10     {
11         // Ovde override-ujemo metodu IsValid().
12         protected override ValidationResult IsValid(object value, ValidationContext validationContext)
13         {
14             return base.IsValid(value, validationContext);
15         }
16     }
17 }
```

Models > Customer.cs

```
Customer.cs  Models\Min18Y...sIfAMember.cs
MovieNT                                  MovieNT.Models.Customer
1   using System;
2   using System.ComponentModel.DataAnnotations;
3   using MovieNT.Models;
4
5   namespace MovieNT.Models
6   {
7       public class Customer
8       {
9           public int Id { get; set; }
10
11          [StringLength(255)]
12          public string Name { get; set; }
13
14          public bool IsSubscribedToNewsletter { get; set; }
15
16          public MembershipType MembershipType { get; set; }
17
18          // Entity framework prepoznaje ovu konvenciju sa ID na kraju i tretira to kao FK.
19          // [Display] koristimo ovde kako bi u View-u labele imale taj tekst.
20          // Alternativno, u view-u ručno pravimo labele.
21          [Display(Name = "Membership Type")]
22          public byte MembershipTypeId { get; set; }
23
24          [Display(Name = "Date of Birth")]
25          [Min18YearsIfAMember]
26          public DateTime? Birthdate { get; set; }
27      }
28  }
```

Custom validacija u Models > Min18YearsIfAMember.cs

```
Customer.cs  Models\Min18...sIfAMember.cs
MovieNT                                  MovieNT.Models.Min18YearsIfAMember   IsValid(object value, ValidationContext validationC
5   using System.ComponentModel.DataAnnotations;
6
7   namespace MovieNT.Models
8   {
9       public class Min18YearsIfAMember : ValidationAttribute
10      {
11          // Ovde override-ujemo metodu IsValid().
12          // Dakle, svi osim 'Pay as You Go' MembershipTypes moraju imati korisnika > 18 godina.
13          protected override ValidationResult IsValid(object value, ValidationContext validationContext)
14          {
15              // validationContext.ObjectInstance nam daje pristup objektu koji se obrađuje.
16              // Castujemo ga u Customer, jer radimo sa Customer-om.
17              var customer = (Customer)validationContext.ObjectInstance;
18
19              // Ako želimo da vratimo uspeh onda vraćamo ValidationResult.Success.
20              if (customer.MembershipTypeId == 1)
21                  return ValidationResult.Success;
22
23              // U suprotnom, unutar ValidationResult objekta pišemo tekst greške.
24              if (customer.Birthdate == null)
25                  return new ValidationResult("Birthdate is required.");
26
27              var age = DateTime.Today.Year - customer.Birthdate.Value.Year;
28
29              return (age >= 18)
30                  ? ValidationResult.Success
31                  : new ValidationResult("Customer should be at least 18 years old to go on a membership");
32          }
33      }
34  }
100 %

Solution Explorer
Search Solution Explorer (Ctrl+;)
RouteConfig.cs
Startup.Auth.cs
Content
Controllers
AccountController.cs
CustomersController.cs
HomeController.cs
ManageController.cs
MoviesController.cs
fonts
Migrations
Models
AccountViewModels.cs
Customer.cs
Genre.cs
IdentityModels.cs
ManageViewModels.cs
MembershipType.cs
Min18YearsIfAMember.cs
Movie.cs
Scripts
ViewModels
Views
favicon.ico
Solution Explorer  Team Explorer  Class View
```

Sada trebamo da dodamo placeholder za ovu validaciju u **Views > CustomerForm.cshtml**

```
<div class="form-group">
    @Html.LabelFor(m => m.Customer.Birthdate)
    @Html.TextBoxFor(m => m.Customer.Birthdate, "{0:d MMM yyyy}", new { @class = "form-control" })
    @Html.ValidationMessageFor(m => m.Customer.Birthdate)
</div>
```

54 - Refactoring Magic Numbers

Umesto da pitamo customer.MembershipTypeId == 1, u samom customer modelu postavimo novi read

only prop sa nazivom PayAsYouGo = 1 i onda pitamo customer.MembershipType = Customer.PayAsYouGo;

Models > MembershipType.cs

```
public class MembershipType
{
    public byte Id { get; set; }
    [Required]
    public string Name { get; set; }
    public short SignUpFee { get; set; }
    public byte DurationInMonths { get; set; }
    public byte DiscountRate { get; set; }

    // Za validaciju :
    // Za nepoznat MembershipType.
    public static readonly byte Unknown = 0;
    public static readonly byte PayAsYouGo = 1;
}
```

Models > Min18YearsIfAMember.cs

```
MovieNT                                   MovieNT.Models.Min18YearsIfAMember
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Web;
5   using System.ComponentModel.DataAnnotations;
6
7   namespace MovieNT.Models
8   {
9       public class Min18YearsIfAMember : ValidationAttribute
10      {
11          // Ovde override-ujemo metodu IsValid().
12          // Dakle, svi osim 'Pay as You Go' MembershipTypes moraju imati korisnika > 18 godina.
13          protected override ValidationResult IsValid(object value, ValidationContext validationContext)
14          {
15              // validationContext.ObjectInstance nam daje pristup objektu koji se obrađuje.
16              // Castujemo ga u Customer, jer radimo sa Customer-om.
17              var customer = (Customer)validationContext.ObjectInstance;
18
19              // Ako želimo da vratimo uspeh onda vraćamo ValidationResult.Success.
20              if (customer.MembershipTypeId == MembershipType.Unknown || customer.MembershipTypeId == MembershipType.PayAsYouGo)
21                  return ValidationResult.Success;
22
23              // U suprotnom, unutar ValidationResult objekta pišemo tekst greške.
24              if (customer.Birthdate == null)
25                  return new ValidationResult("Birthdate is required.");
26
27              var age = DateTime.Today.Year - customer.Birthdate.Value.Year;
28
29              return (age >= 18)
30                  ? ValidationResult.Success
31                  : new ValidationResult("Customer should be at least 18 years old to go on a membership");
32          }
33      }
34  }
```

56 - Client-side Validation

Benefits

- Immediate feedback
- No waste of server-side resources

Views > Shared > Layout.cshtml

@RenderSection nam omogućava da dodamo script sekcije u našim view-ovima. Ovo će nam omogućiti

da referenciramo na jQuery validation bundle (koji radi client-side) iz **App_Start > BundleConfig.cs**

```
BundleConfig.cs | CustomerForm.cshtml | CustomersController.cs | _Layout.cshtml
MovieNT | MovieNT.BundleConfig
1   using System.Web;
2   using System.Web.Optimization;
3
4   namespace MovieNT
5   {
6       public class BundleConfig
7       {
8           // For more information on bundling, visit http://go.microsoft.com/fwlink/?LinkId=301862
9           public static void RegisterBundles(BundleCollection bundles)
10          {
11              bundles.Add(new ScriptBundle("~/bundles/jquery").Include(
12                          "~/Scripts/jquery-{version}.js"));
13
14              bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(
15                          "~/Scripts/jquery.validate*"));
16
17              // Use the development version of Modernizr to develop with and learn from. Then, when you're
18              // ready for production, use the build tool at http://modernizr.com to pick only the tests you need.
19              bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(
20                          "~/Scripts/modernizr-*"));
21
22              bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(
23                        "~/Scripts/bootstrap.js",
24                        "~/Scripts/respond.js"));
25
26              bundles.Add(new StyleBundle("~/Content/css").Include(
27                        "~/Content/bootstrap-lumen.css",
28                        "~/Content/site.css"));
29          }
30      }
31  }
```

Views > Shared > _Layout.cshtml

```
@Scripts.Render("~/bundles/jquery")
@Scripts.Render("~/bundles/bootstrap")
@RenderSection("scripts", required: false)
```

Views > Customers > CustomerForm.cshtml
Na dnu.

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

Ova client-side validacija radi samo za originalne Data Annotations, ne i custom.

Data Annotations

- [Required]
- [StringLength(255)]
- [Range(1, 10)]
- [Compare("OtherProperty")]
- [Phone]
- [EmailAddress]
- [Url]
- [RegularExpression("…")]

57 - Anti-forgery Tokens

Sprečava krađu sesije korisnika. Anti-forgery token garantuje da request isključivo dolazi iz te forme, a ne nekog drugog sajta npr.

Kod skrivenog ID-a u formi dodajemo **@Html.AntiForgeryToken()**.

Views > Customers > CustomerForm.cshtml

```
@* Anti-forgery token - Sprečava krađu sesije i garantuje da request dolazi isključivo sa ove forme. *@
@Html.AntiForgeryToken()
```

U kontroleru kod metode ka kojoj kontroler šalje stavljamo **[ValidateAntiForgeryToken]**.

```
// [HttpPost] - Akcija može biti pozvana samo POST zahtevom.
// Kao argument ovoj metodi prosleđujemo Customer iz view-a.
// Stavljamo [ValidateAntiForgeryToken] za validaciju.
[ValidateAntiForgeryToken]
[HttpPost]
public ActionResult Save(Customer customer)
{
    // Koristimo ModelState property za pristup podacima za validaciju.
    if (!ModelState.IsValid)
    {   // Ako ModelState nije validan, onda želimo da vratimo korisnika na formu.
        // Za taj view nam je potreban CustomerFormViewModel, i u njega ubacujemo primljeni customer objekat.
        var viewModel = new CustomerFormViewModel
        {
            Customer = customer,
            MembershipTypes = _context.MembershipTypes.ToList()
        };
        return View("CustomerForm", viewModel);
    }
```

Vežba - Napraviti validaciju za MovieForm, ograničiti Number in stock između 1 i 20.

Models > Movie.cs

```
1
2   using System;
3   using System.ComponentModel.DataAnnotations;
4
5   namespace MovieNT.Models
6   {
7       public class Movie
8       {
9           public int Id { get; set; }
10
11          [Required]
12          [StringLength(255)]
13          public string Name { get; set; }
14
15          public Genre Genre { get; set; }
16
17          [Display(Name = "Genre")]
18          [Required]
19          public byte GenreId { get; set; }
20
21          public DateTime DateAdded { get; set; }
22
23          [Display(Name = "Release Date")]
24          public DateTime ReleaseDate { get; set; }
25
26          [Range(1,20)]
27          [Display(Name = "Number in Stock")]
28          public byte NumberInStock { get; set; }
29      }
30  }
```

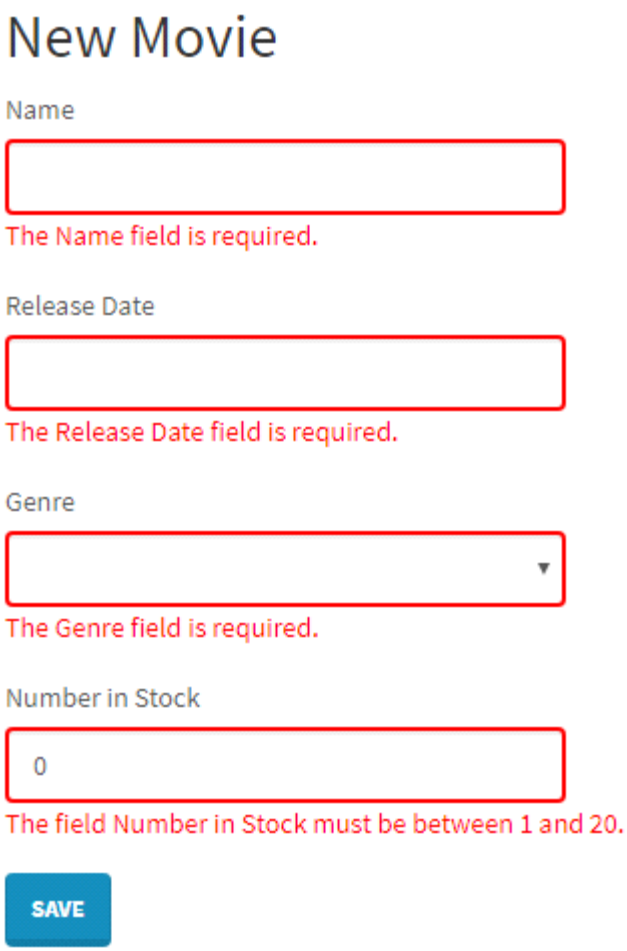
Views > MovieForm.cshtml

```
MovieForm.cshtml | BundleConfig.cs | CustomerForm.cshtml | Movie.cs | MoviesController.cs | _Layout.cshtml
1   @model MovieNT.ViewModels.MovieFormViewModel
2   @* ISTO KAO CustomerForm.cshtml, samo se igraj sa form-group i parametrima. *@
3   @{
4       ViewBag.Title = Model.Title;
5   }
6
7   <h2>@Model.Title</h2>
8
9   @using (Html.BeginForm("Save", "Movies"))
10  {
11      @Html.ValidationSummary(true, "Please fix the following errors.")
12      <div class="form-group">
13          @Html.LabelFor(m => m.Movie.Name)
14          @Html.TextBoxFor(m => m.Movie.Name, new { @class = "form-control" })
15          @Html.ValidationMessageFor(m => m.Movie.Name)
16      </div>
17      <div class="form-group">
18          @Html.LabelFor(m => m.Movie.ReleaseDate)
19          @Html.TextBoxFor(m => m.Movie.ReleaseDate, "{0:d MMM yyyy}", new { @class = "form-control" })
20          @Html.ValidationMessageFor(m => m.Movie.ReleaseDate)
21      </div>
22      <div class="form-group">
23          @Html.LabelFor(m => m.Movie.GenreId)
24          @Html.DropDownListFor(m => m.Movie.GenreId, new SelectList(Model.Genres, "Id", "Name"), "", new {@class = "form-control"})
25          @Html.ValidationMessageFor(m => m.Movie.GenreId)
26      </div>
27      <div class="form-group">
28          @Html.LabelFor(m => m.Movie.NumberInStock)
29          @Html.TextBoxFor(m => m.Movie.NumberInStock, new { @class = "form-control" })
30          @Html.ValidationMessageFor(m => m.Movie.NumberInStock)
31      </div>
32      @Html.HiddenFor(m => m.Movie.Id)
33      @Html.AntiForgeryToken()
34      <button type="submit" class="btn btn-primary">Save</button>
35  }
36
37  @section scripts
38  {
39      @Scripts.Render("~/bundles/jqueryval")
40  }
```

Controller > MoviesController.cs

```
[ValidateAntiForgeryToken]
[HttpPost]
public ActionResult Save(Movie movie)
{
    if (movie.Id == 0)
    {
        movie.DateAdded = DateTime.Now;
        _context.Movies.Add(movie);
    }
    else
    {
        var movieInDb = _context.Movies.Single(m => m.Id == movie.Id);
        movieInDb.Name = movie.Name;
        movieInDb.GenreId = movie.GenreId;
        movieInDb.NumberInStock = movie.NumberInStock;
        movieInDb.ReleaseDate = movie.ReleaseDate;
    }

    _context.SaveChanges();

    return RedirectToAction("Index", "Movies");
}
```

Rezultat



64 - RESTful Convention

65 - Building an API

Desni Klik na Controllers > New folder > Api

Desni klik na folder > Add > Controller Web API 2 Controller - Empty

Sada moramo da podesimo API.

```
Visual Studio has added the full set of dependencies for ASP.NET Web API 2 to project 'MovieNT'.

The Global.asax.cs file in the project may require additional changes to enable ASP.NET Web API.

1. Add the following namespace references:

    using System.Web.Http;
    using System.Web.Routing;

2. If the code does not already define an Application_Start method, add the following method:

    protected void Application_Start()
    {
    }

3. Add the following lines to the beginning of the Application_Start method:

    GlobalConfiguration.Configure(WebApiConfig.Register);
```

Global.asax.cs

```
MovieNT                                                    MovieNT.MvcA
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Optimization;
using System.Web.Routing;
using System.Web.Http;
using System.Web.Routing;

namespace MovieNT
{
    public class MvcApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            GlobalConfiguration.Configure(WebApiConfig.Register);
            AreaRegistration.RegisterAllAreas();
            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);
        }
    }
}
```

Controllers > Api > CustomersController.cs

CustomersController.cs

MovieNT MovieNT.Controllers.Api.Cust

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Net;
5   using System.Net.Http;
6   using System.Web.Http;
7   using MovieNT.Models;
8
9   namespace MovieNT.Controllers.Api
10  {
11      public class CustomersController : ApiController
12      {
13          // Deklaracija context-a.
14          private ApplicationDbContext _context;
15
16          public CustomersController()
17          {
18              _context = new ApplicationDbContext();
19          }
20
21          // Vraćanje liste customera.
22          // Pošto vraćamo listu objekata, ova akcija/metoda
23          // će po konvenciji biti na GET /api/customers
24          public IEnumerable<Customer> GetCustomers()
25          {
26              // Koristimo context da izvučemo Customere iz baze.
27              return _context.Customers.ToList();
28          }
29
30          // Akcija za dobijanje jednog customera.
31          // GET /api/customers/1
32          public Customer GetCustomer(int id)
33          {
34              var customer = _context.Customers.SingleOrDefault(c => c.Id == id);
35
36              if (customer == null)
37                  throw new HttpResponseException(HttpStatusCode.NotFound);
38
39              return customer;
40          }
42          // POST-ujemo customera u customer kolekciju.
43          // Ovde kreiramo resurs te koristimo [HttpPost]
44          // Alternativno, možemo da mu damo naziv PostCustomer.
45          // POST /api/customers
46          [HttpPost]
47          public Customer CreateCustomer(Customer customer)
48          {
49              // Provera ispravnosti unetih podataka.
50              if (!ModelState.IsValid)
51                  throw new HttpResponseException(HttpStatusCode.BadRequest);
52
53              // Ako je sve okej, dodajemo POST-ovanog customer-a u context.
54              _context.Customers.Add(customer);
55              // Sačuvamo izmene.
56              _context.SaveChanges();
57
58              // Vraćamo objekat.
59              return customer;
60          }
```

```
62          // Ažuriranje customer-a.
63          // Tip može biti ili void ili customer.
64          // Jedan parametar je id koji se čita iz URL-a.
65          // Drugi je customer koji dolazi iz request tela.
66          // PUT /api/customers/1
67          [HttpPut]
68          public void UpdateCustomer(int id, Customer customer)
69          {
70              if (!ModelState.IsValid)
71                  throw new HttpResponseException(HttpStatusCode.BadRequest);
72
73              // Kupimo korisnika iz baze na osnovu prosleđenog id-a.
74              var customerInDb = _context.Customers.SingleOrDefault(c => c.Id == id);
75
76              // Proveravamo da korisnik zaista postoji u bazi.
77              if (customerInDb == null)
78                  throw new HttpResponseException(HttpStatusCode.NotFound);
79
80              // Ako je sve okej, update-ujemo customera.
81              customerInDb.Name = customer.Name;
82              customerInDb.Birthdate = customer.Birthdate;
83              customerInDb.IsSubscribedToNewsletter = customer.IsSubscribedToNewsletter;
84              customerInDb.MembershipTypeId = customer.MembershipTypeId;
85
86              // Na kraju čuvamo izmene.
87              _context.SaveChanges();
88          }
90          // Uklanjanje customer-a.
91          public void DeleteCustomer(int id)
92          {
93              // Kupimo korisnika iz baze na osnovu prosleđenog id-a.
94              var customerInDb = _context.Customers.SingleOrDefault(c => c.Id == id);
95
96              // Proveravamo da korisnik zaista postoji u bazi.
97              if (customerInDb == null)
98                  throw new HttpResponseException(HttpStatusCode.NotFound);
99
100             // Uklanjamo customer-a iz memorije.
101             _context.Customers.Remove(customerInDb);
102
103             // Čuvamo izmene (koje se primenjuju u bazi).
104             _context.SaveChanges();
105         }
106     }
107 }
```

<u>Ovde sam prestao da radim, pošto je do poglavlja 65 dovoljno za prolaz (CRUD + API).</u>