Contents

CAPITOLUL 6

ASP.NET MVC

1. Introducere

MVC este un model arhitectural utilizat în dezvoltarea aplicatiilor software. Acest izoleaza logica de business față de considerentele interfeței cu utilizatorul, astfel rezultand o aplicație unde partea vizuala si nivelele inferioare ale regulilor de business sunt mai ușor de modificat, fără a afecta alte nivele.

Arhitectura

- **Model**

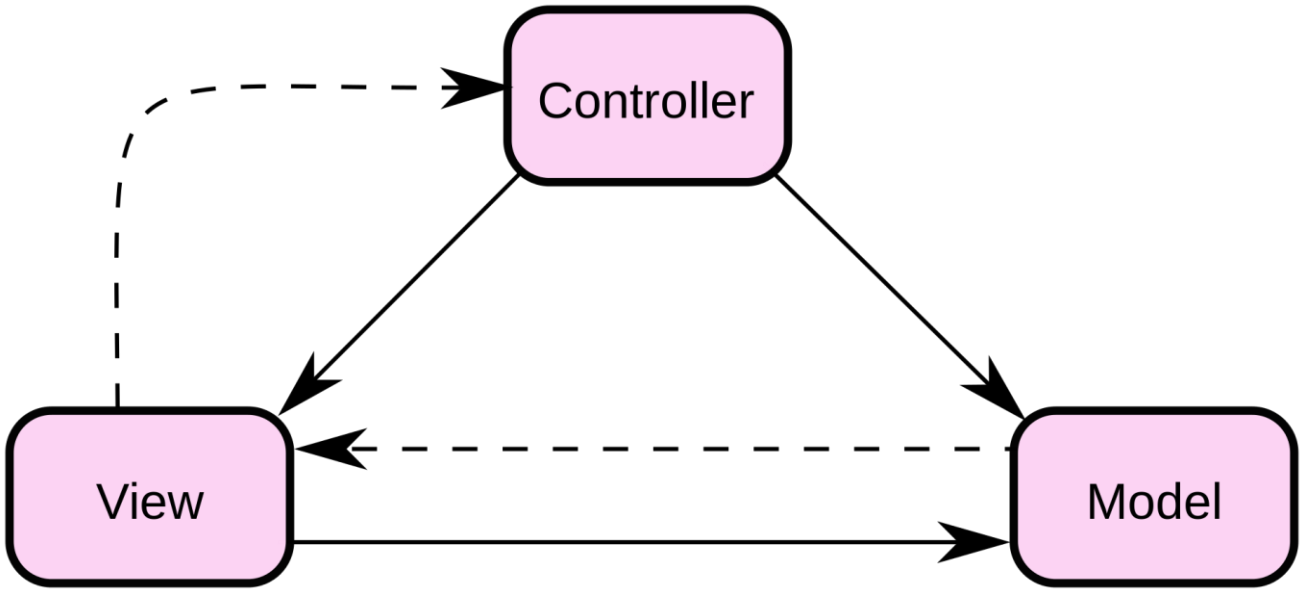
Această parte manipulează operațiunile logice și de utilizare de informație pentru a rezulta o formă ușor de înțeles.

- **Vizualizare**

Acestui membru îi corespunde reprezentarea grafică, sau mai bine zis, exprimarea ultimei forme a datelor: interfața grafică ce interacționează cu utilizatorul final. Rolul său este de a evidenția informația obținută până ce ea ajunge la controlor.

- **Controlor**

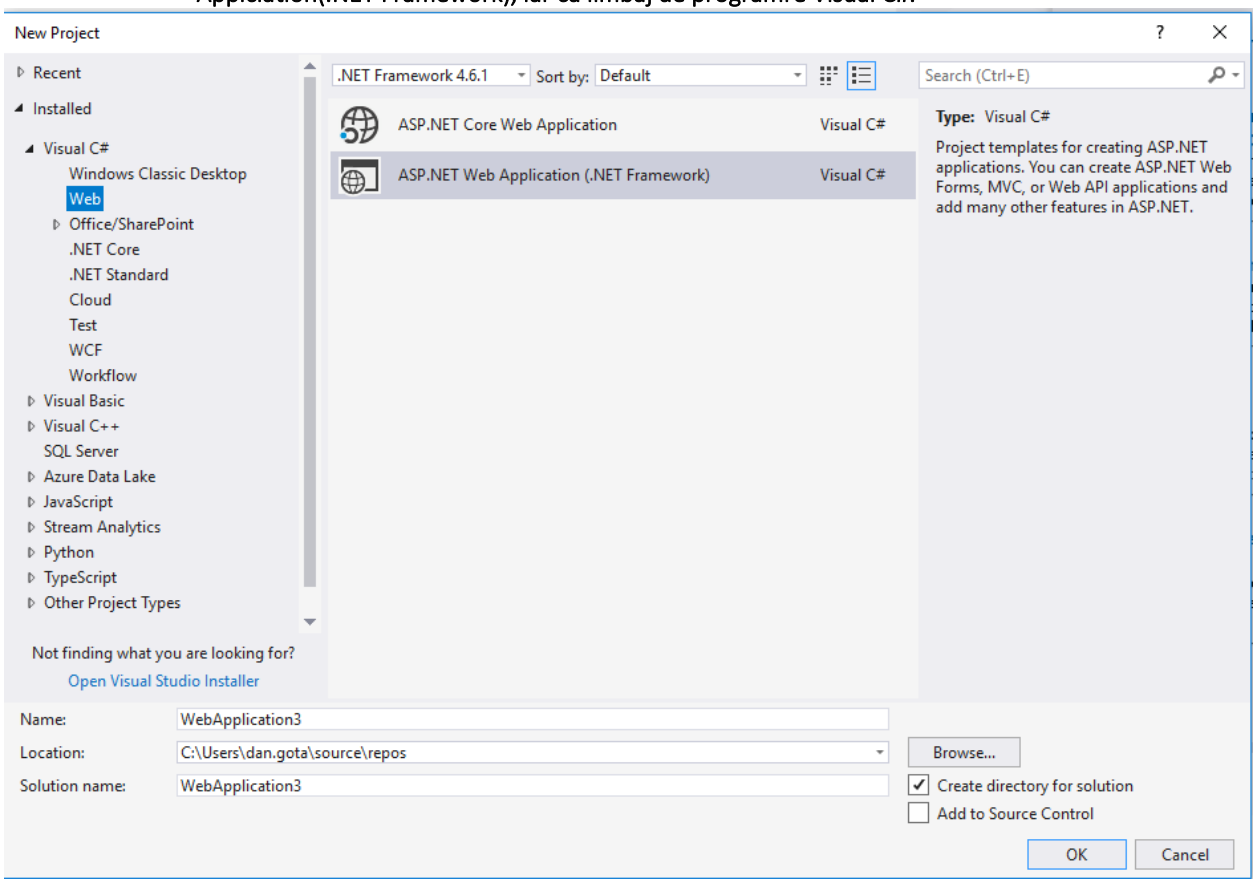
Cu acest element putem controla accesul la aplicația noastră. Pot fi fișiere, scripturi sau programe, in general orice tip de informație permisă de interfață.



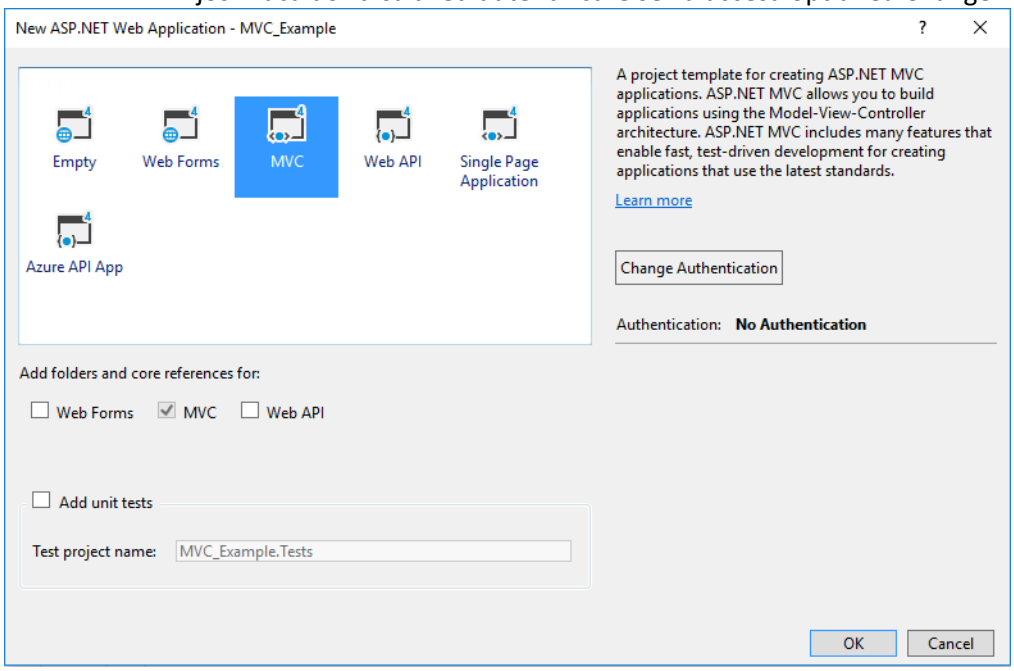
Sursa: https://ro.wikipedia.org/wiki/Model-view-controller

Pentru a cerea un nou proiect MVC trebuie urmati pasii urmatori:

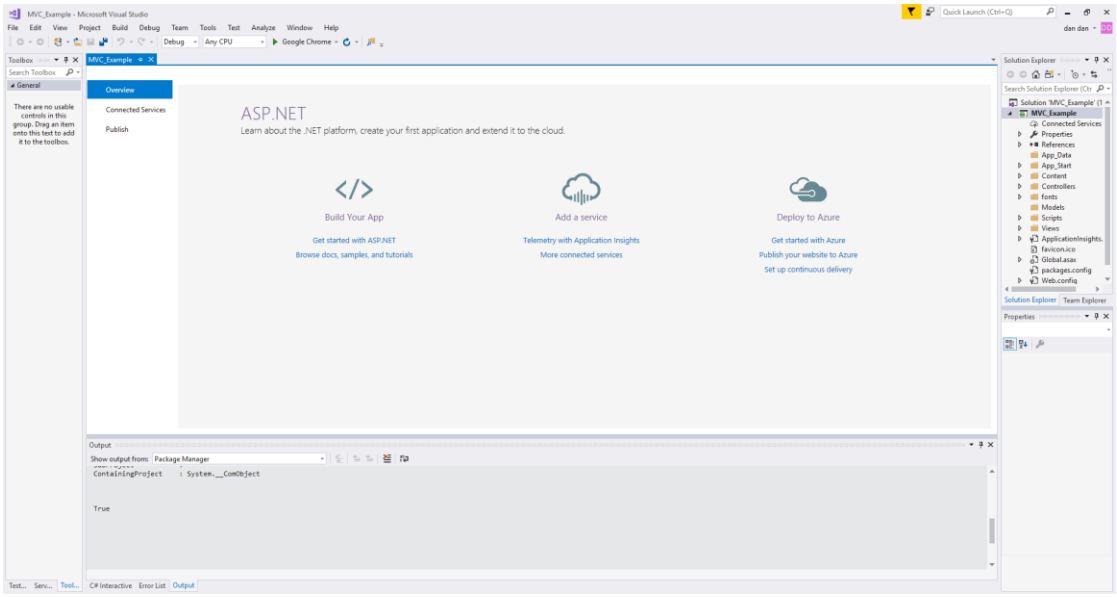
1. In Visual Studio->Create new project->Sub sectiunea Web aveti de selectat optiunea ASP.NET Web Applciation(.NET Framework), iar ca limbaj de programre Visual C#.



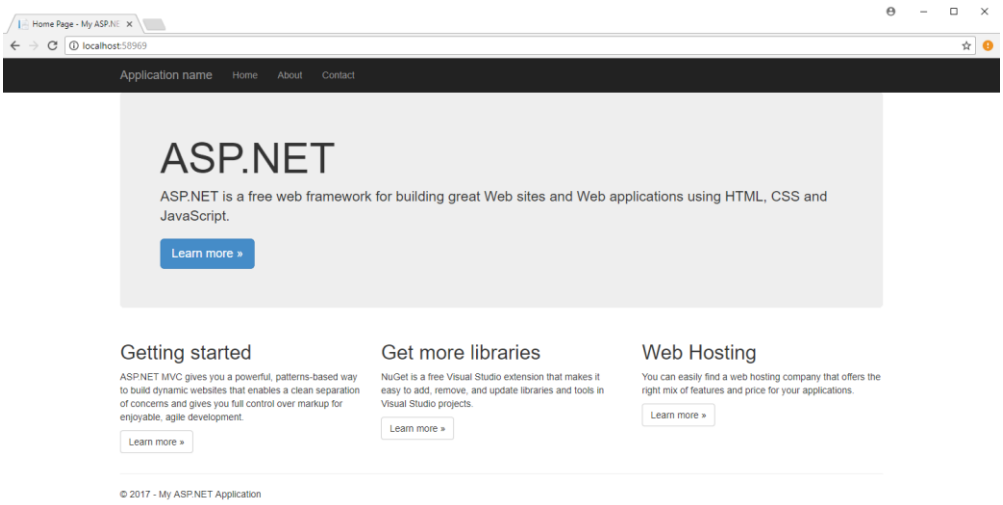
2. Selectarea unui template de proiect, si anume MVC din lista prezentata in imaginea de mai jos. Daca doriti sa aveti autentificare se va accesa optiunea Change Authentification.



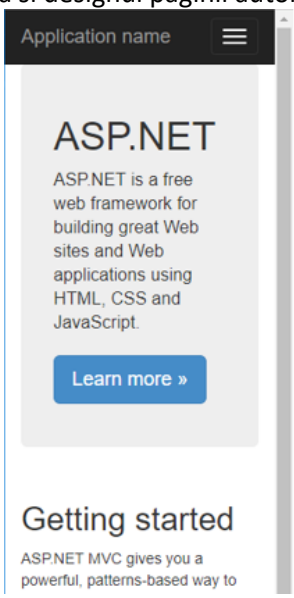
Actionand butonul/optiunea OK ca rezultat veti avea creat in Visual Studio un proiect care respecta regula MVC de structurare a proiectului.



Daca veti rula proiectul in acet moment, o sa obtineti rezultaul din imaginea de mai jos. Se va deschide intr-un Browser o pagina implicita care este gazduita temporar de calculatorul personal.Caracterul gazduirii temporale este dat de faptul ca la fiecare rulare a aplicatiei, portul de acces al paginii se schimba.



Asa cum se poate observa din imaginea de mai jos, designul paginii este responsive, si anume, odata cu modificarea rezolutiei ecranului se schimba si designul paginii automat.



O data cu crearea solutiei, proiectul MVC o sa contina urmatoarele:

- App_Data - > aici vom pune tot ce este legat de baza de date
- App_Start -> contine o serie de clase care sunt rulate cand aplicatie se starteaza
- Cotent -> imagini si css
- Controllers -> folderul va contine toate controller-ele din aplicatia noastra
- Fonts
- Models-> toate clasele domeniu vor fi continute aici
- Scripts-> folderul contine toate scripturile din aplicatie
- Views-> contine view-urile aplicatiei
- Favicon.ico -> icon-ul aplicatiei cand se va rula in browser
- Global.asax -> clasa care trateaza diverse evenimente in aplicatie
- Startup.cs -> logica de start a aplicatiei este stocata aici
- Web.config -> contine configurari utilizate in aplicatie ca si conexiuni catre baza de date si altele.

In cele ce urmeaza ne propunem sa cream o aplicatie cu ajutorul careia vom inchiriat filme. Pentru aceasta, ca prim pas vom avea nevoie sa cream clasa model Movie care va avea urmatoarele proprietati: Id (int), Nume(string) . Pentru aceasta urmati indicatiile oferite de imaginile de mai jos:

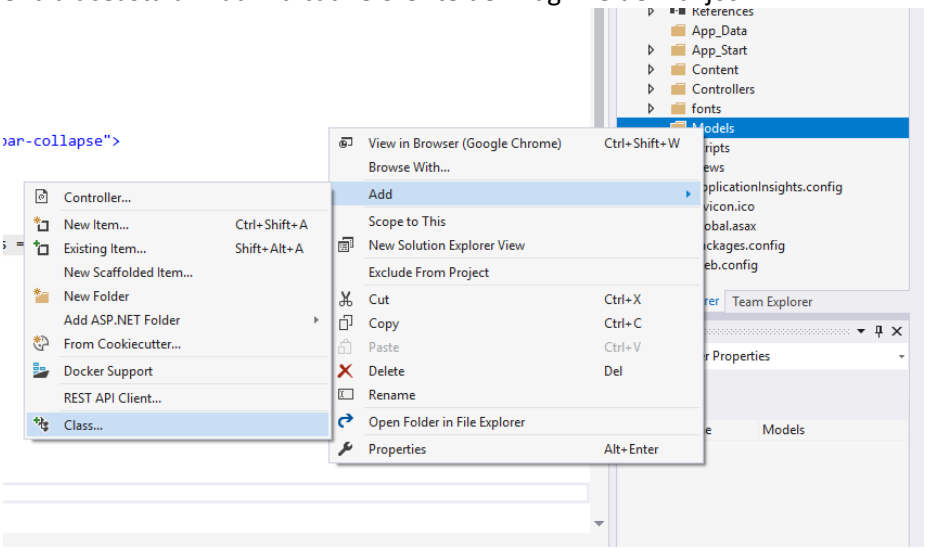


Fig xx: Adaugarea unei element nou in directorul Models

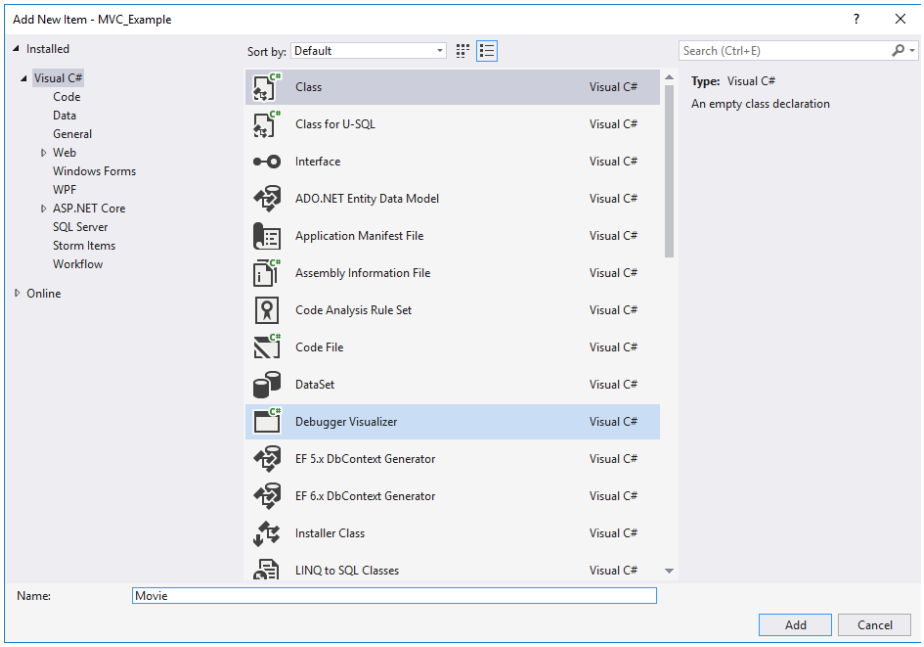
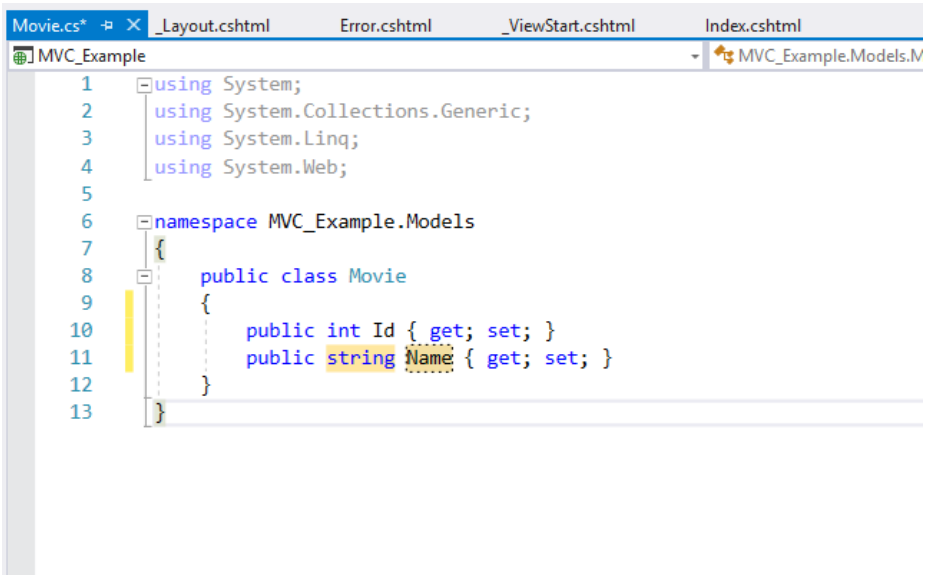


Fig xx: Adaugarea clasei denumita Movie in folderul Models



Fix xx: Adugarea celor 2 proprietari ale viitoarelor obiecte de tip Movie

Vom crea un controller care va utiliza aceasta clasa model. Figurile de mai jos va vor indruma pasii spre crearea acestui controller.

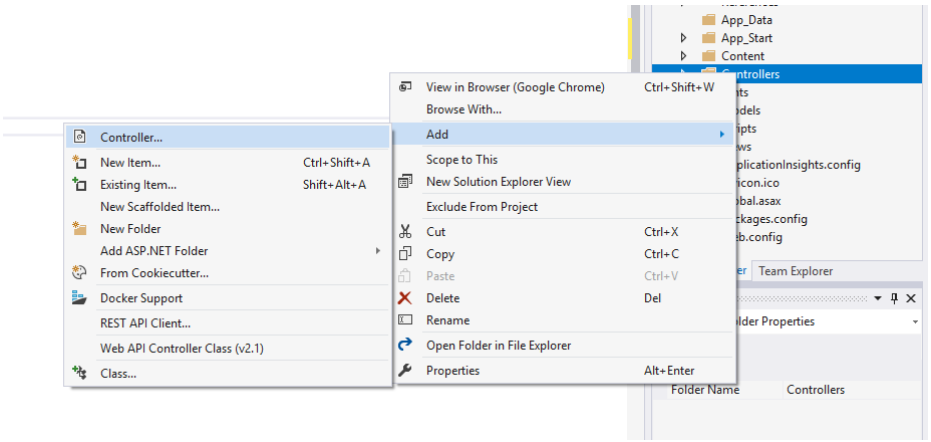


Fig xx: Adaugarea unui nou element Controller in directorul Controllers

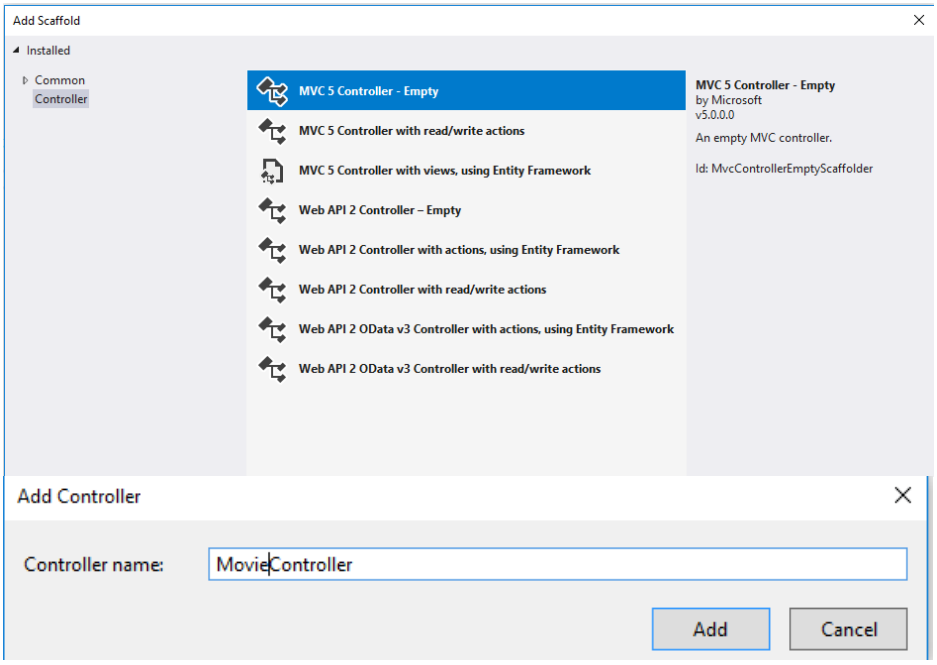


Fig xx: Selectarea unui controller gol si denumirea acestuia

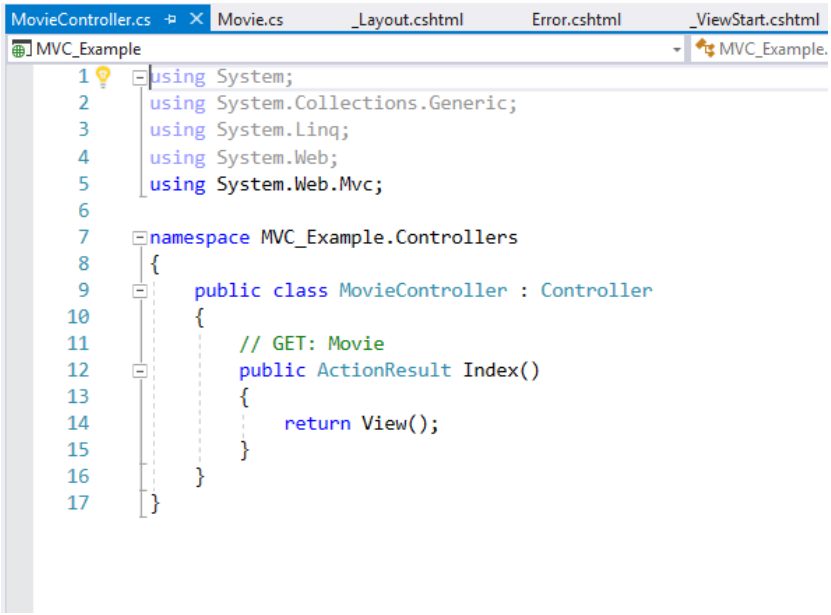
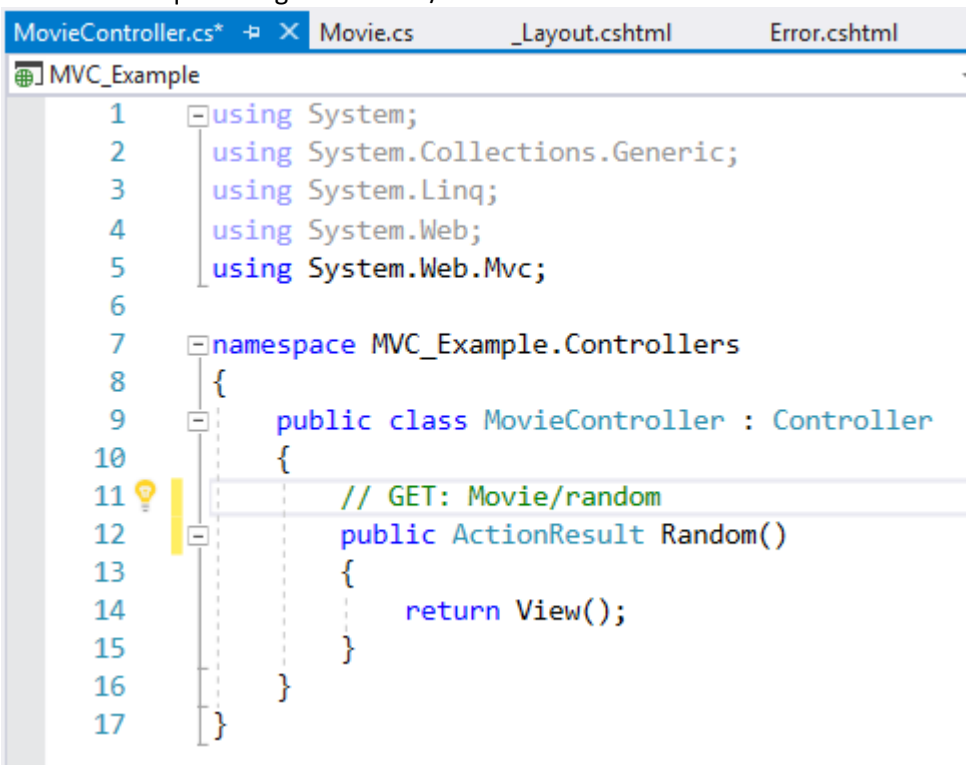


Fig xx: Rezultatul adaugarii controller-ului Movie in directorul de controller-e

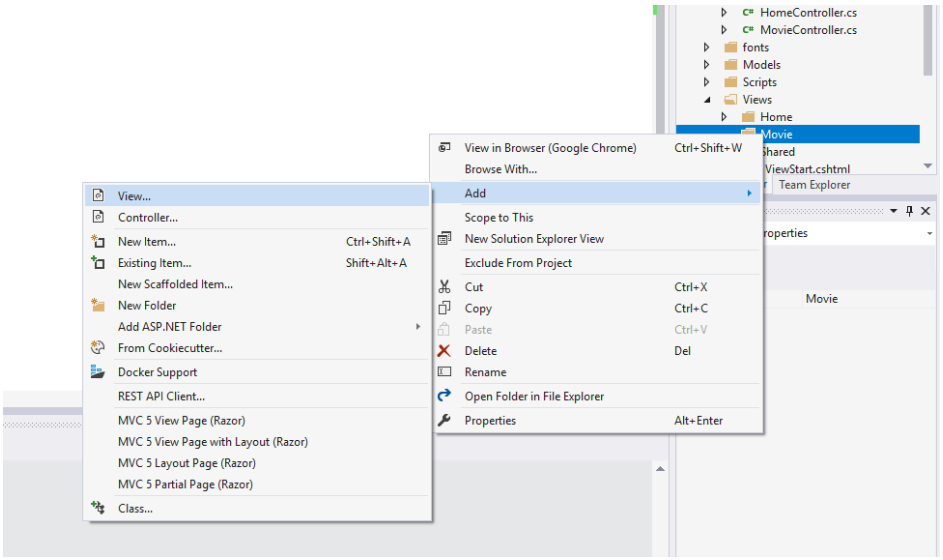
Schimbam numele Actiuni din Index in Random. Astfel aceasta actiune va fi apelata in momentul in care vom acea un url request de genul: Movie/random

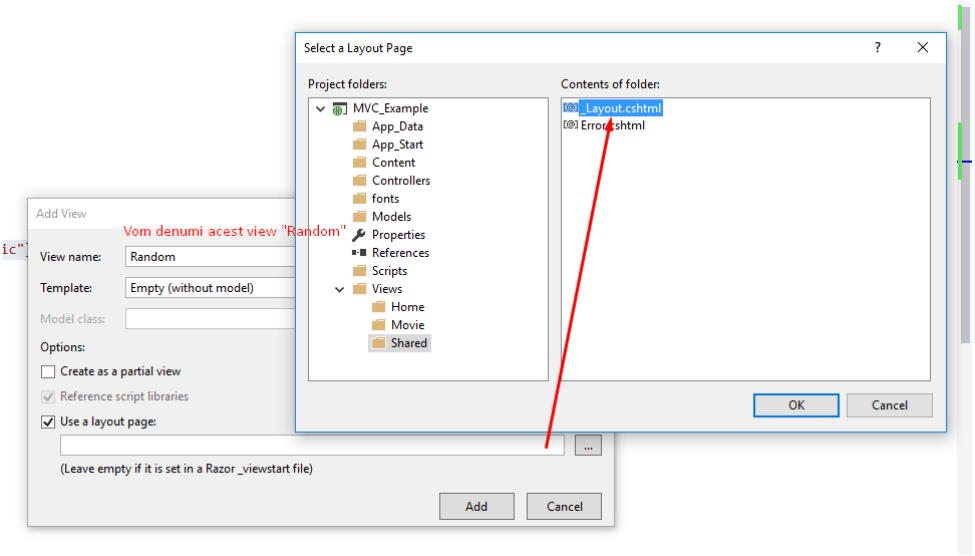


Von crea in cele ce urmeaza o instanta a modelului Movie.

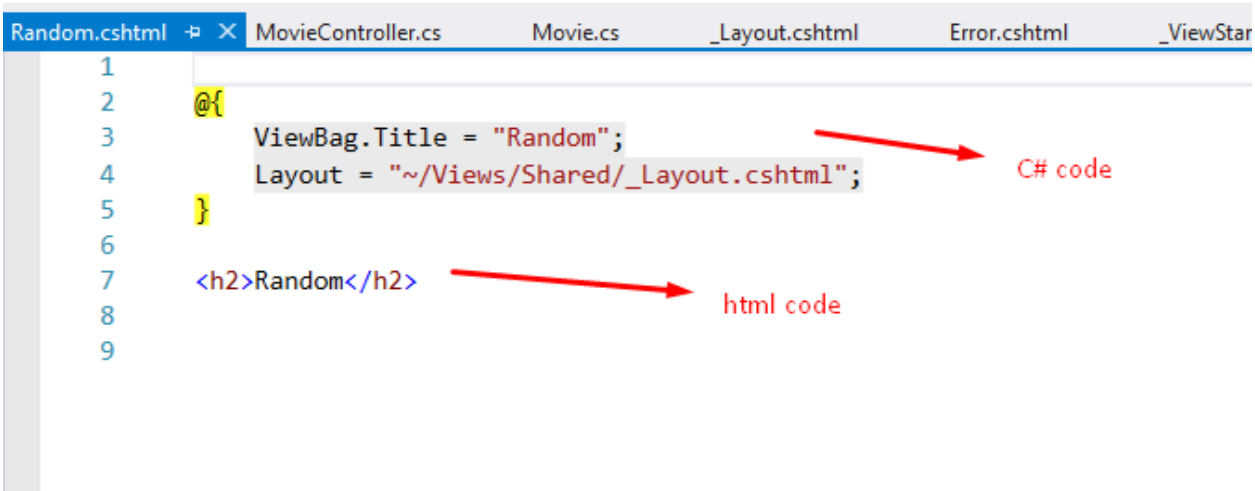
```
var movie = new Movie() { Name="Titanic"};
```

Vom crea in cele ce urmeaza view-ul pentru aceasta actiune:





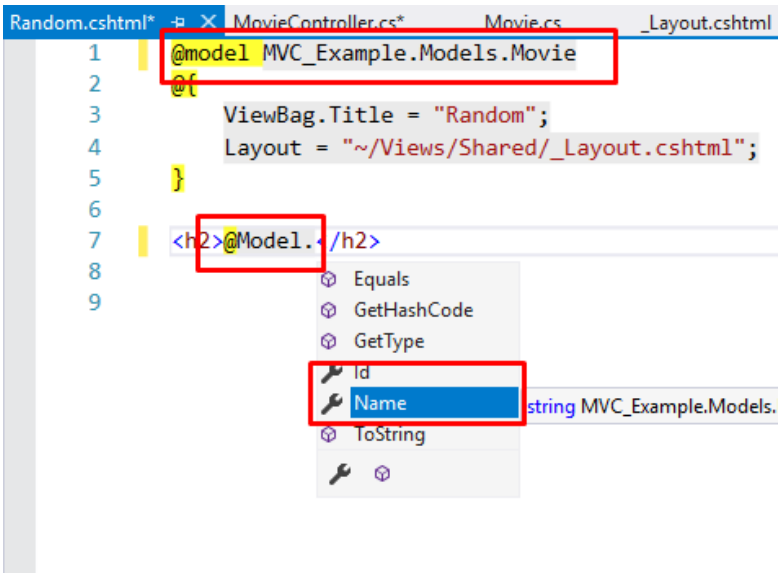
Numele acestui view va fi “Random” si vom selecta ca si layout general(master page) _Layout.cshtml din folderol Shared.



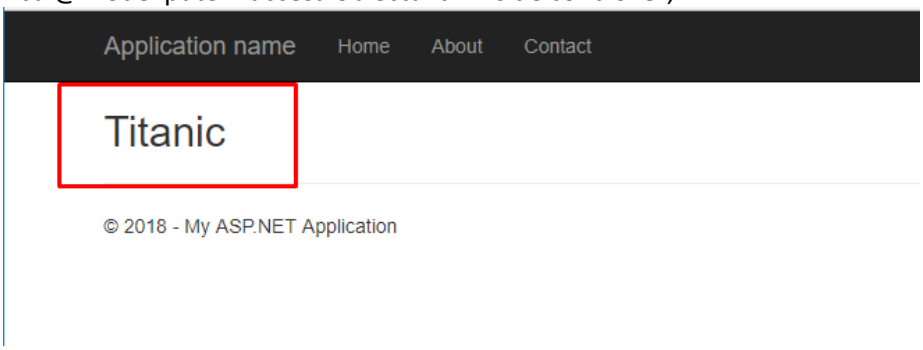
In momentul de fata avem legate controller-ul si view-ul de filme. Pentru a pasa date view-ului din controller vom schimba codul din controller dupa cum este evidentiat in secventa de cod de mai jos.

```
public ActionResult Random()
        {
            var movie = new Movie() { Name="Titanic"};

            return View(movie);
        }
```

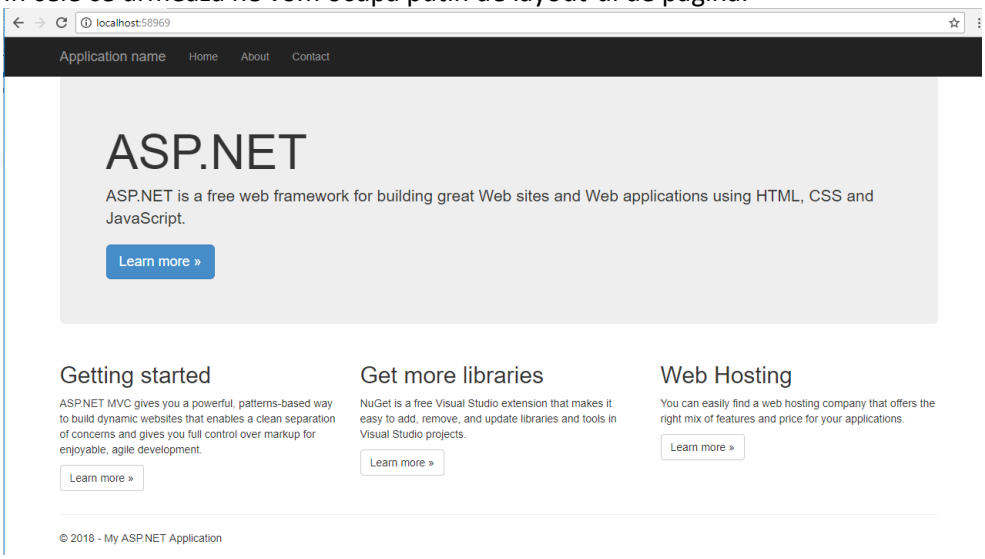
Pentru a afisa informatie din acest obiect “movie” trebuie sa schimbam in view dupa cum urmeaza:



- se adauga o directiva in care specificam ce tip de model folosim pt acest view;
- cu @Model putem accesa obiectul trimis de controller;

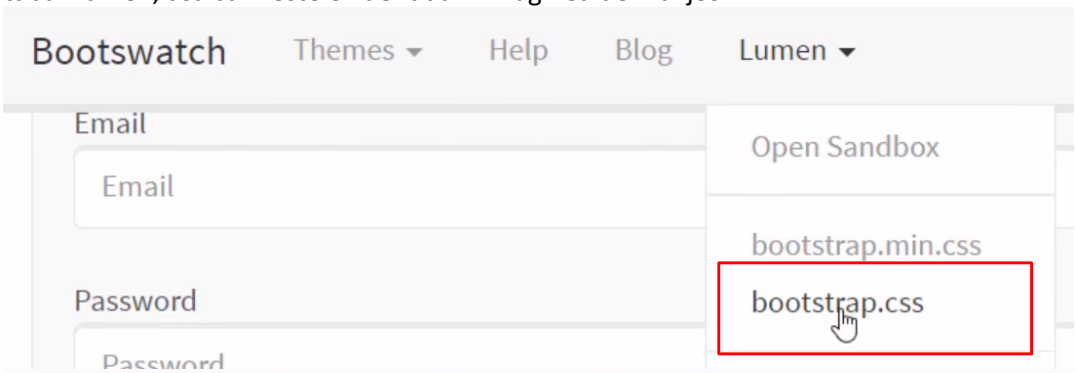


In cele ce urmeaza ne vom ocupa putin de layout-ul de pagina.



Vom schimba in cele ce urmeaza template-ul aplicatiei. ASP.NET MVC foloseste Bootstrap ca si framework CSS, astfel, pentru a schimba acest template va trebui sa gasim un template de tip Boostrap.
Navigati la bootswatch.com si sub Themes selectati Lumen. Pe pagina deschisa in Browser veti putea observa cum vor arata elementele din pagina daca vom importa acest template in proiect. Selectati boostrap.css din

tabul Lumen, asa cum este evidentiat in imaginea de mai jos.

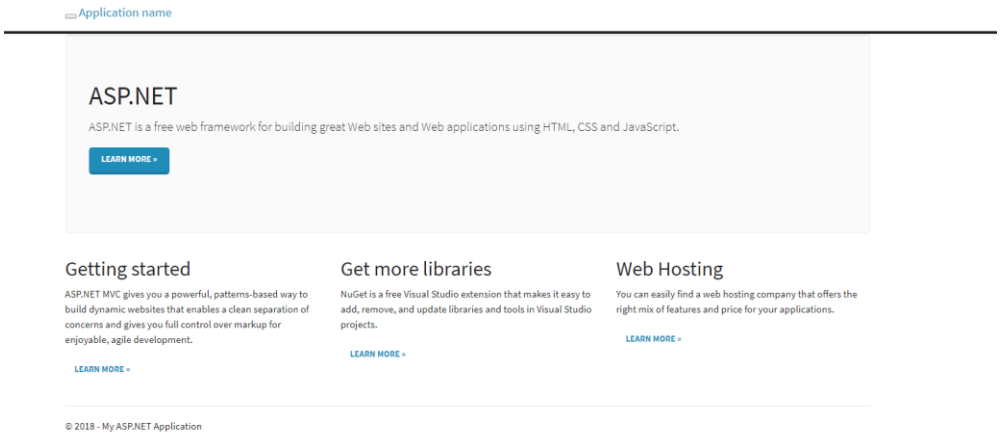



```
@import url("https://fonts.googleapis.com/css?family=Source+Sans+Pro:300,400,700,400italic");
/*!
 * bootswatch v3.3.6
 * Homepage: http://bootswatch.com
 * Copyright 2012-2016 Thomas Park
 * Licensed under MIT
 * Based on Bootstrap
*/
/*!
 * Bootstrap v3.3.6 (http://getbootstrap.com)
 * Copyright 2011-2015 Twitter, Inc.
 * Licensed under MIT (https://github.com/twbs/bootstrap/blob/master/LICENSE)
 */
/*! normalize.css v3.0.3 | MIT License | github.com/necolas/normalize.css */
html {
  font-family: sans-serif;
```

Continutul fisierului va trebui salvat pe disc cu denumirea **bootstrap-lumen.css** si mai apoi adaugati fisierul rezultat in folderul **Content** al proiectului. In acest moment va trebui sa modificam referinta catre bootstrap.css si sa cream o referinta la boostrap-lumen.css.
In App_Start deschideti BundleConfig.cs si modificati bundle-ul de css.

```
bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(
          "~/Scripts/bootstrap.js",
          "~/Scripts/respond.js"));

bundles.Add(new StyleBundle("~/Content/css").Include(
          "~/Content/bootstrap.css"
          "~/Content/site.css"));

          "~/Scripts/bootstrap.js",
          "~/Scripts/respond.js"));

bundles.Add(new StyleBundle("~/Content/css").Include(
          "~/Content/bootstrap-lumen.css",
          "~/Content/site.css"));
```

Vom rula aplicatia, si vom observa ca tema paginilor s-a schimbat.



ASP.NET fundamente

Action Results

In controllerul de filme am observant ca actiunea Random() returneaza un ActionResult. In finalul actiunii Random() apelam metoda View() care este o metoda helper mostenita din clasa de baza Controller. Aceasta metoda ne ajuta sa cream un ViewResult care este un subtip al ActionResult.

```
public class MoviesController : Controller
{
    // GET: Movies/Random
    public ActionResult Random()
    {
        var movie = new Movie() { Name = "Shrek!" };

        return View(movie);
    }
}
```

In imaginea de mai jos sunt afisate tipurile de Action results si metodele care trebuie apelate pentru a returna aceste tipuri.

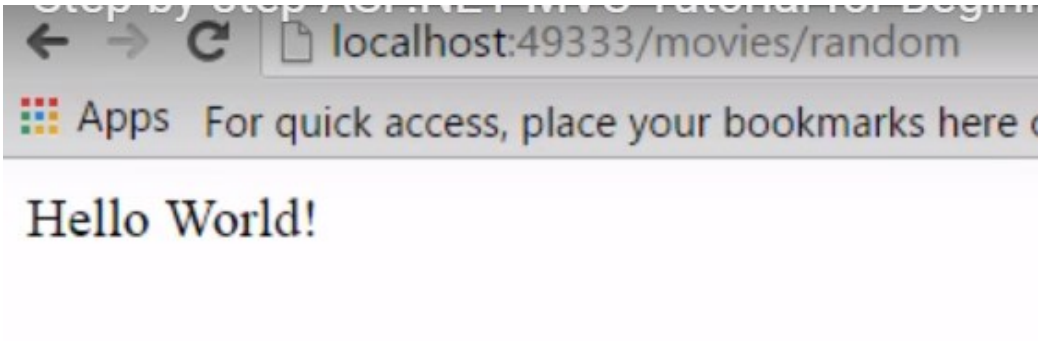
Action Results

Type	Helper Method	
ViewResult	View()	
PartialViewResult	PartialView()	returneaza un view partial
ContentResult	Content()	returneaza text simplu
RedirectResult	Redirect()	redireacteaza catre pagina
RedirectToRouteResult	RedirectToAction()	returneaza o actiune si nu URL
JsonResult	Json()	returneaza un fisier de tip JSON
FileResult	File()	returneaza fisier
HttpNotFoundResult	HttpNotFound()	pagini de eroare tip 404
EmptyResult		

Vom experimenta in cele ce urmeaza unele din aceste tipuri.
Comentati linia de cod cu return View(movie) si adaugati return Content("Hello World!");

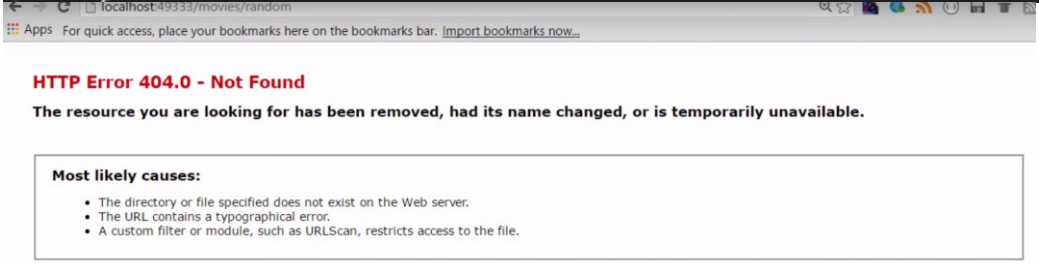
```
//              return View(movie);
                return Content("Hello World!");
        }
    }
```

Ca rezultat vom avea:



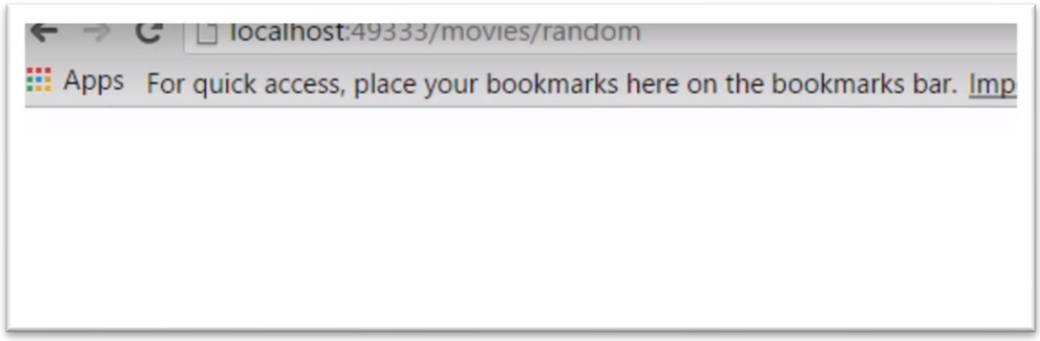
Return NotFound:

```
//              return View(movie);
//              return Content("Hello World!");
                return HttpNotFound();
        }
    }
```



Return Empty result:

```
//              return View(movie);
//              return Content("Hello World!");
//              return HttpNotFound();
            return new EmptyResult();
        }
```



Redirect to action:

```
//              return View(movie);
//              return Content("Hello World!");
//              return HttpNotFound();
//              return new EmptyResult();
            return RedirectToAction("Index", "Home");
        }
    }
```

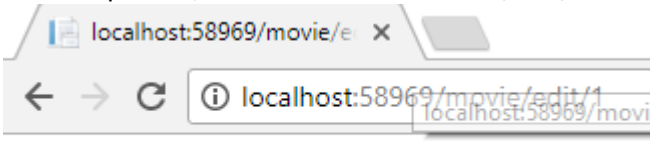
Actiunea ne va redirecta catre actiunea Index din controller-ul Home.

Action parameterms

Vom crea o noua actiunea de editare dupa cum se observa in imaginea de mai jos:

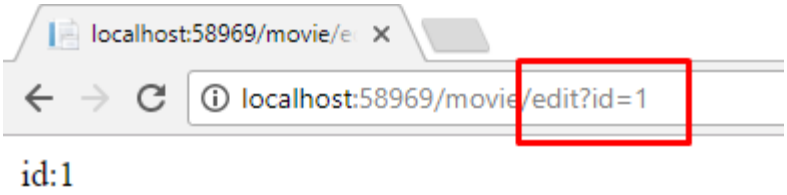
```
        public ActionResult Edit(int id)
        {
            return Content("id:" + id);
        }
    }
}
```

Ruland aplicatia, vom accesa url-ul: movie/edit/1 si vom obtine urmatorul rezultat:



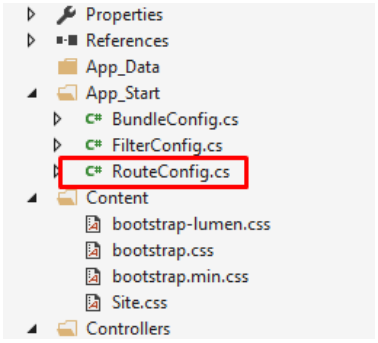
id:1

Acesta este un exemplu de **parametru continut in url**.
Exista posibilitatea de a trimite acest **parametru ca si query string:**



Rute in MVC

Pentru a crea rute custom vom modifica fisierul RouteConfig.cs



In momentul de fata avem definite doar o ruta, cea default, dupa cum se poate observa mai jos:

```
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 }
    );
}
```

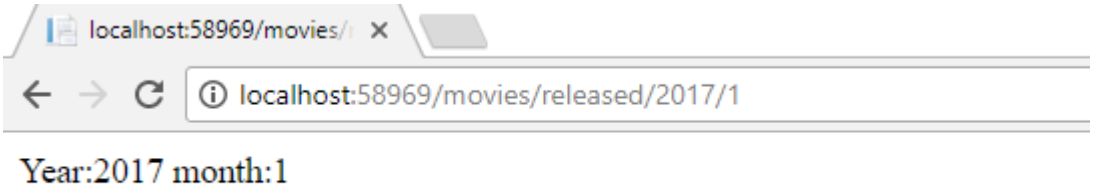
In cele ce urmeaza dorim sa cream o ruta custom de tipul:
/movies/released/2015/04
Toate rutele custom for trebui adaugate inainte de ruta default, deoarece ordinea rutelor este importanta. Se vor crea rutele de la cele mai specifice, primele, la cele mai generice, ultimele.
Vom adauga o moua ruta, inaintea rutei default, astfel:

```
routes.MapRoute(
                name: "MoviesByReleaseDate",
                url: "movies/released/{year}/{month}",
                defaults: new { controller = "Movie", action = "ByReleaseDate" }
            );
```

Va trebui in cele ce urmeaza sa cream actiunea MoviesByReleaseDate, astfel:

➔ In controller-ul de filme adaugati urmatorul bloc de cod:

```
public ActionResult ByReleaseDate(int year, int month)
        {
            return Content("Year:" + year + " month:" + month);
        }
```



Daca dorim sa impunem constrangeri asupra formatelor datelor din url atunci va trebui sa adaugam un nou

parametru in definirea rutei.

```
routes.MapRoute(
                name: "MoviesByReleaseDate",
                url: "movies/released/{year}/{month}",
                defaults: new { controller = "Movie", action = "ByReleaseDate" },
                constraints: new { year = @"\d{4}", month = @"\d{2}" }
            );
```

Prin adaugarea acestor constrangeri fortam ca anul sa fie de formatul a 4 digits si luna de format 2 digit.

Rute attribute

In MVC5 rutele se declara ca si attribute. Pentru aceasta trebuie sa activam acest tip de rute in fisierul RouteConfig.cs.

```
routes.MapMvcAttributeRoutes();
```

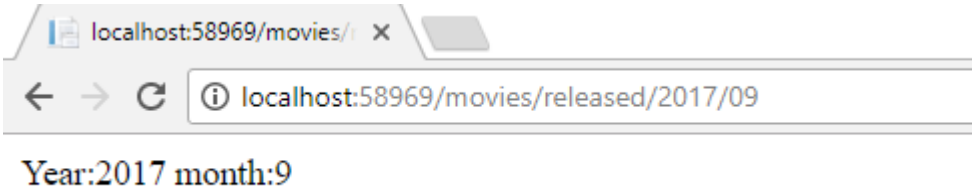
Vom sterge ruta creata anterior din fisierul RouteConfig.cs

Vom adauga in controller un atribut Route deasupra metodei ActionResult ByReleaseDate(int year, int month).

Codul va arata ca cel de mai jos:

```
[Route("movies/released/{year:regex(\\d{4})}/{month:regex(\\d{2}):range(1,12)}")]
        public ActionResult ByReleaseDate(int year, int month)
        {
            return Content("Year:" + year + " month:" + month);
        }
```

Iar rezultatul va fi cel asteptat, cel de mai jos:



Pentru mai multe detalii legate de contrangeri ale rutei atribut cautati pe google: MVC Attributes Route Constrains – si veti descoperii toate contrangerile posibile.

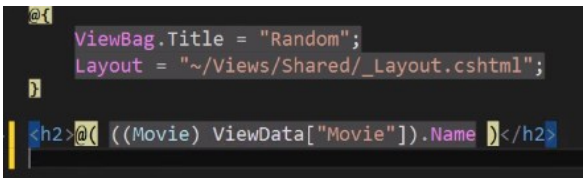
Trimiterea datelor catre View

In introducere in actiunea Random am trimis modelul catre View ca si argument al metodei View. `return View(movie);`

Exista inca doua metode de trimitere a datelor:

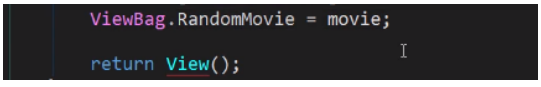
1) **Utilizarea dictionarului ViewData**

```
public ActionResult Random()
{
    var movie = new Movie() { Name = "Shrek!" };

    ViewData["Movie"] = movie;

    return View();
}
```

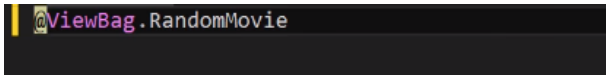


```
@{
    ViewBag.Title = "Random";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>@( ((Movie) ViewData["Movie"]).Name )</h2>
```

2) **Utilizarea ViewBag**



```
ViewBag.RandomMovie = movie;

return View();
```



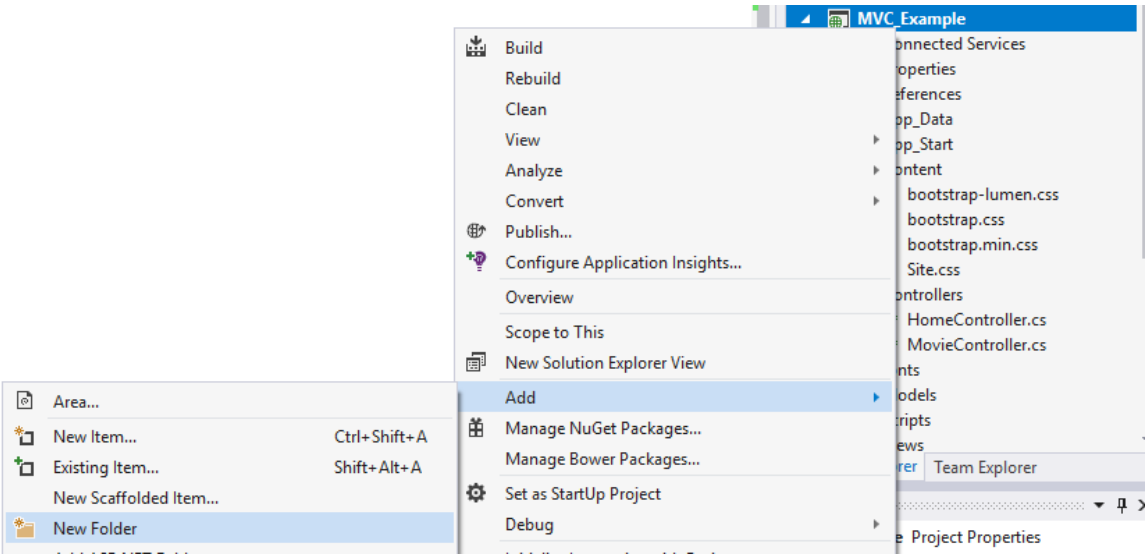
```
@ViewBag.RandomMovie
```

3) **Utilizarea ViewModel**

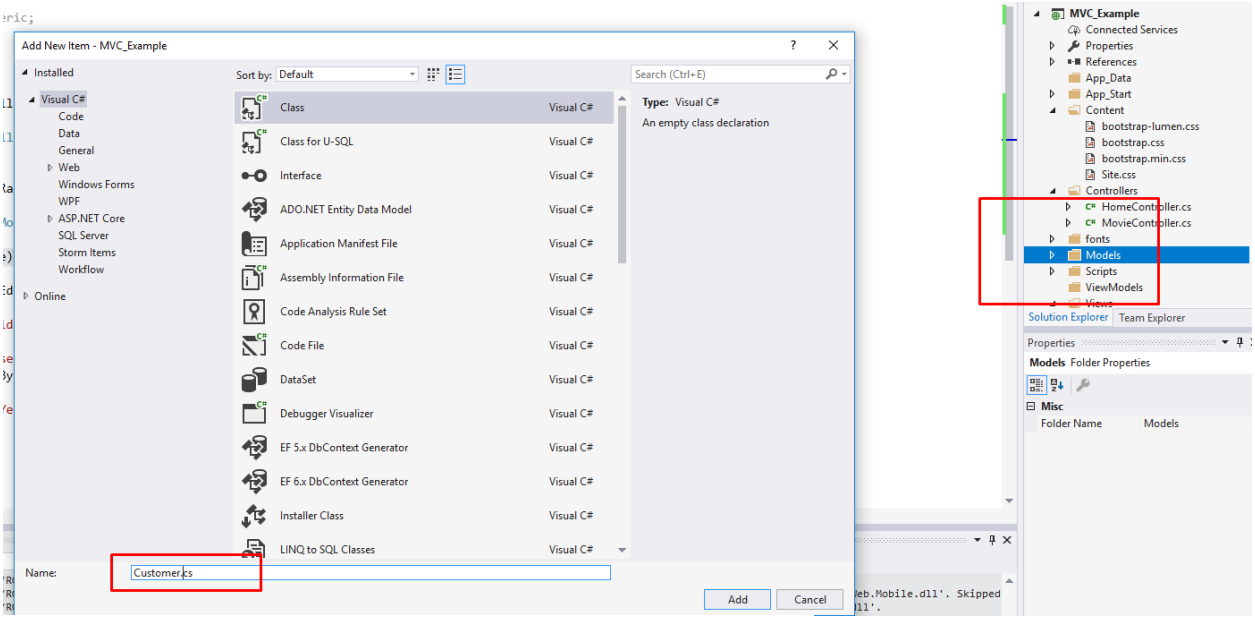
- Este metoda prezentata in introducere, cea mai utilizata si recomandata.

Pana in acest moment am trimis ca si model un obiect simplu. Daca dorim sa cream un obiect de tipul client care au inchiriat un anumit tip de film, va trebui sa cream obiecte ViewModel.

Vom crea pentru acesta un folder nou, numit ViewModels

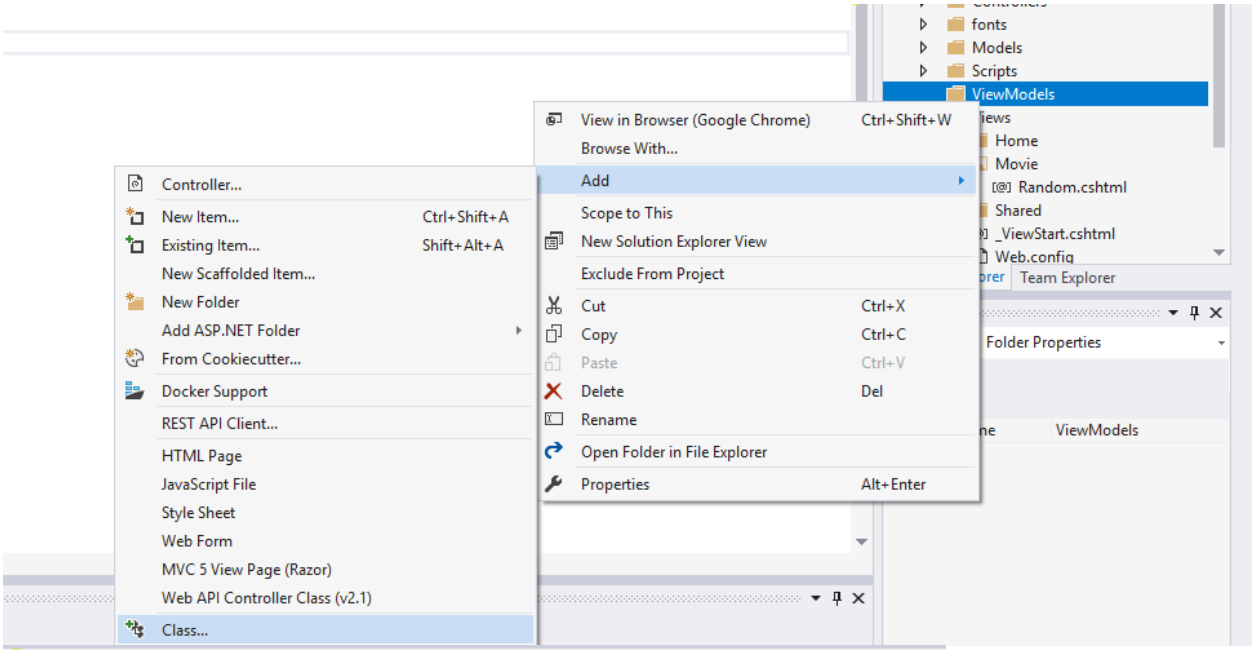


Vom crea un mou model in folderol Models, denumit Customers.




```
namespace MVC_Example.Models
{
    public class Customer
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }
}
```

Vom adauga o noua clasa model in folderul ViewModels numita RandomMovie ViewModel.




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

namespace MVC_Example.ViewModels
{
    public class RandomMovieViewModel
    {
        public Movie Movie { get; set; }
        public List<Customer> Customers { get; set; }
    }
}
```

Vom schimba actiunea Radom() din controller dupa cum este afisat in imaginea de mai jos:

```
// GET: Movie/random
public ActionResult Random()
{
    var movie = new Movie() { Name="Titanic"};
    var customers = new List<Customer>
    {
        new Customer{Name ="Customer 1"},
        new Customer{Name ="Customer 2"}
    };

    var viewModel = new RandomMovieViewModel {
        Movie = movie,
        Customers= customers
    };

    return View(viewModel);
}
```

Iar in Random view schimbam dupa cum urmeaza:

```
@model MVC_Example.ViewModels.RandomMovieViewModel
@{
    ViewBag.Title = "Random";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

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

Pentru a randa lista de client va trebui sa ne folosim de **sintaxa Razor:**

Customer.cs | RouteConfig.cs | BundleConfig.cs | Random.cshtml*

```
@model MVC_Example.ViewModels.RandomMovieViewModel
@{
    ViewBag.Title = "Random";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

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

@if (Model.Customers.Count==0)
{
    <p> Nu a inchiriat nimeni acest film </p>
}
else
{
    <ul>
        @foreach (var customer in Model.Customers)
        {
            <li> @customer.Name</li>
        }
    </ul>
}
```

La rulare vom obtine rezultatul de mai jos:

Application name

Titanic

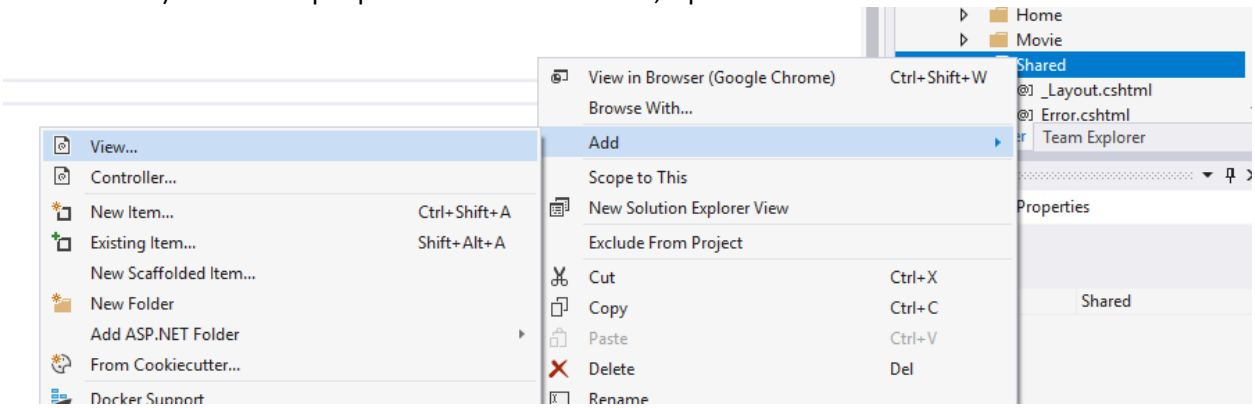
- Customer 1
- Customer 2

View-uri partiale

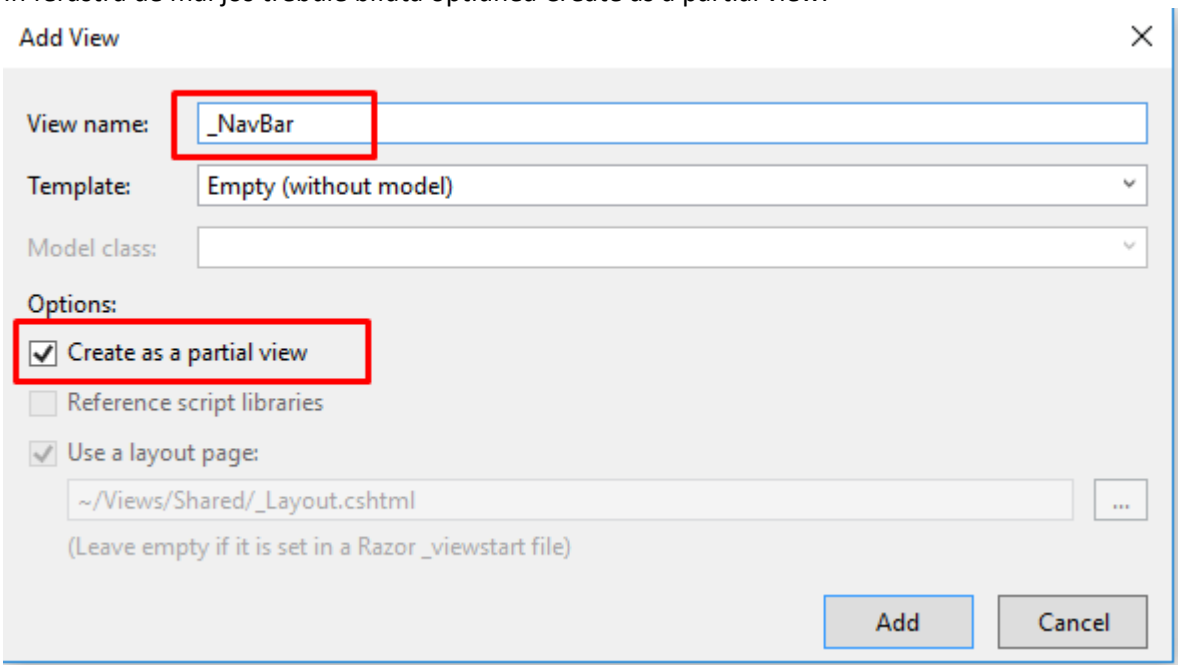
Dupa cum le spune si numele view-urile partiale, sunt partiale ☺, acestea nu au layout, se pot reutiliza oriunde este nevoie sau pot fi create pentru a structura mai bine view-uri care sunt foarte mari. Vom incerca sa introducem view-uri partiale in fisierul _Layout.cshtml din folderul Views, Shared.

In el avem mai multe sectiuni: head, body. In body avem un div pentru navigatie si un container pentru body. Haideti sa introducem elementul de navigatie intr-un partial view. Pentru crearea unui view partial vom efectua urmatorii pasi:

a) Click dreapta pe folderul Views Shared, Optiunea Add si View



In ferastra de mai jos trebuie bifata optiunea Create as a partial view.



Copiem to div-ul de navigare in vederea partiala _NavBar.

```
<div class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            @Html.ActionLink("Application name", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })
        </div>
        <div class="navbar-collapse collapse">
            <ul class="nav navbar-nav">
                <li>@Html.ActionLink("Home", "Index", "Home")</li>
                <li>@Html.ActionLink("About", "About", "Home")</li>
                <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
            </ul>
        </div>
    </div>
</div>
```

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - My ASP.NET Application</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
</head>
<body>

    @Html.Partial("_NavBar")
    <div class="container body-content">...</div>

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

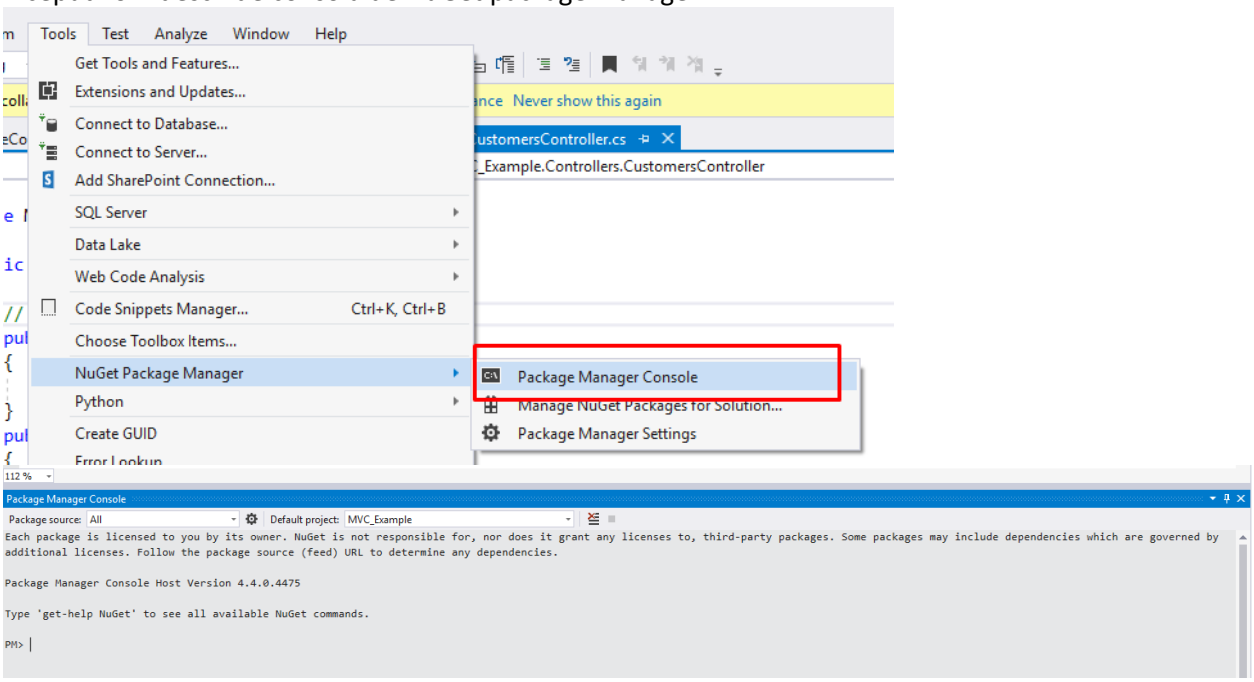
In cazul in care vrem sa pasam un model catre acest partial view avem ca optiune sa adaugam la metoda Partial un al doilea argument.

```
@model MVC_Example.ViewModels.RandomMovieViewModel
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - My ASP.NET Application</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
</head>
<body>

    @Html.Partial("_NavBar", Model.Customers)
    <
        ▲ 2 of 4 ▼ (extension) MvcHtmlString HtmlHelper.Partial(string partialViewName, object model)
        Renders the specified partial view as an HTML-encoded string.
        model: The model for the partial view.
        <footer>
            <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
        </footer>
    </div>

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

Lucru cu date

In aceasta sectiune vom folosi EntityFramework pentru a obtine date dintr-o baza de date si astfel sa evitam hardcodarea datelor.

Putem lucra cu Entity Framework in doua moduri: Code first sau DBFirst. In continuare vom folosi code first ca mod de lucru pentru a avea o productivitate sporita. In abordarea code first vom incepe prin scriere de cod. Ori de cate ori vom modifica modelele vom crea o micrare si mai apoi o vom rula in baza de date. Pentru inceput vom deschide consola de NuGet package manager.



Pentru a rula migrari trebuie mai intai sa le activam ruland comanda **enable-migrations.** Inainte de acest lucru va trebui sa cream o clasa in proiect cu numele MyDBContext care va mostenii DbContext. Clasa va contine urmatorul cod:

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

namespace MVC_Example
{
    public class MyDBContext:DbContext
    {
        public MyDBContext()
        {

        }
    }
}
```

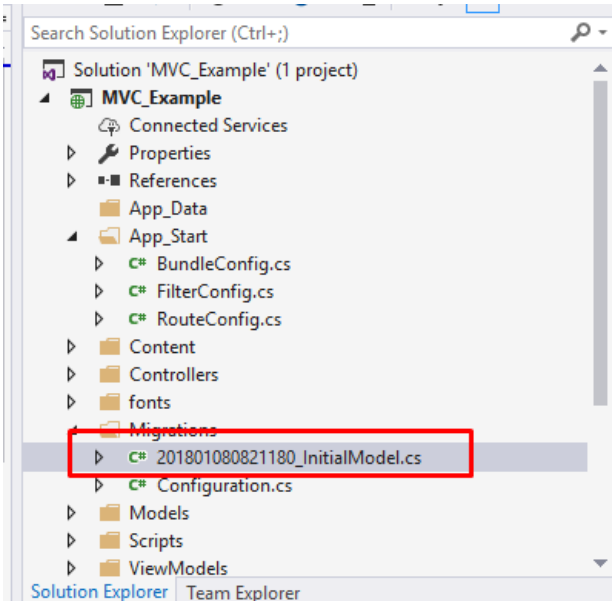
In package manager console vom rula urmatoarea comanda:
Install-Package EntityFramework -IncludePrerelease iar mai apoi **Enable-Migrations** .

```
PM> Enable-Migrations
Checking if the context targets an existing database...
Code First Migrations enabled for project MVC_Example.
PM>
```

Vom crea in cele ce urmeaza prima noastra migrare, ruland comanda **add-migration InitialModel** .

```
PM> add-migration InitialModel
Scaffolding migration 'InitialModel'.
The Designer Code for this migration file includes a snapshot of your current Code First model. This snapshot is used to calculate the changes to your model when you scaffold the next migration. If you make additional changes to your model that you want to include in this migration, then you can re-scaffold it by running 'Add-Migration InitialModel' again.
PM>
```

Ca rezultat, in folderol Migrations vom avea o noua clasa.



In initial model nu avem nimic pentru acum deoarece nu avem nimica referentiat de clasa DBContext. Vom naviga in clasa creata anterior si vom adauga o proprietate noua

```
public class MyDBContext:DbContext
    {
        public DbSet<Customer> Customers { get; set; }
        public MyDBContext()
        {

        }
    }
```

Vom rula din nou comanda **add-migration InitialModel –force**, vom folosi atributul force deoarece migratia deja exista.

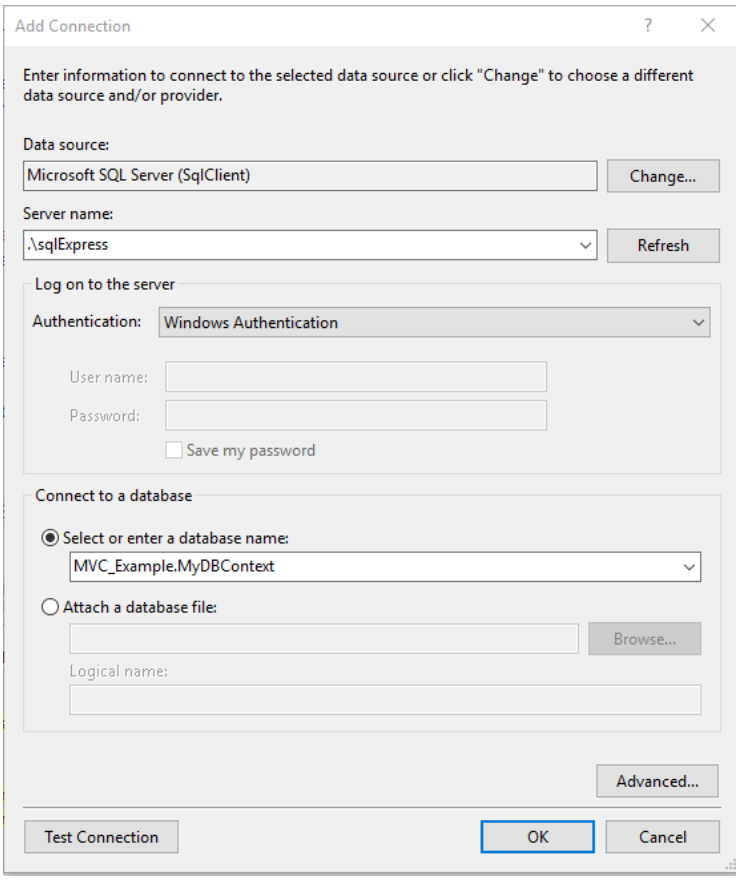
```
PM> add-migration InitialModel -force
Re-scaffolding migration 'InitialModel'.
PM>
```

Ca si rezultat avem urmatoarele:

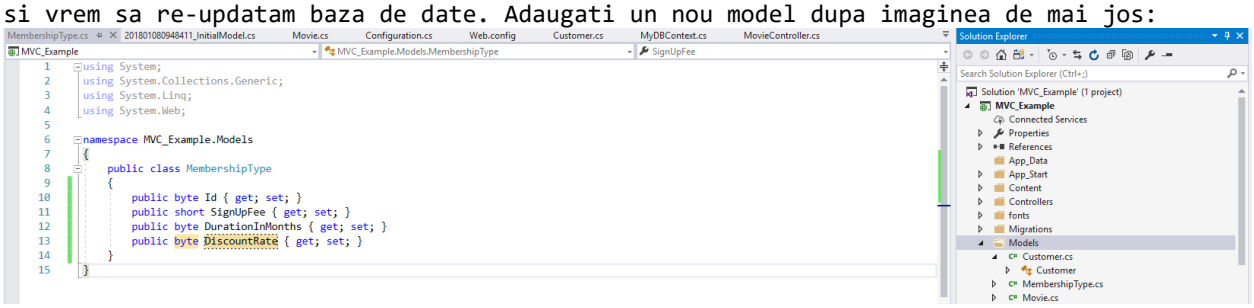
```
2   {
3       using System;
4       using System.Data.Entity.Migrations;
5
6       public partial class InitialModel : DbMigration
7       {
8           public override void Up()
9           {
10              CreateTable(
11                  "dbo.Customers",
12                  c => new
13                      {
14                          Id = c.Int(nullable: false, identity: true),
15                          Name = c.String(),
16                          IsSubscribedToNewsletter = c.Boolean(nullable: false),
17                          Birthdate = c.DateTime(),
18                      })
19                  .PrimaryKey(t => t.Id);
20
21          }
22
```

Pentru a rula si crea aceasta tabela in baza de date va trebui sa rulam o comanda in consola manager de pachete: **update-database**

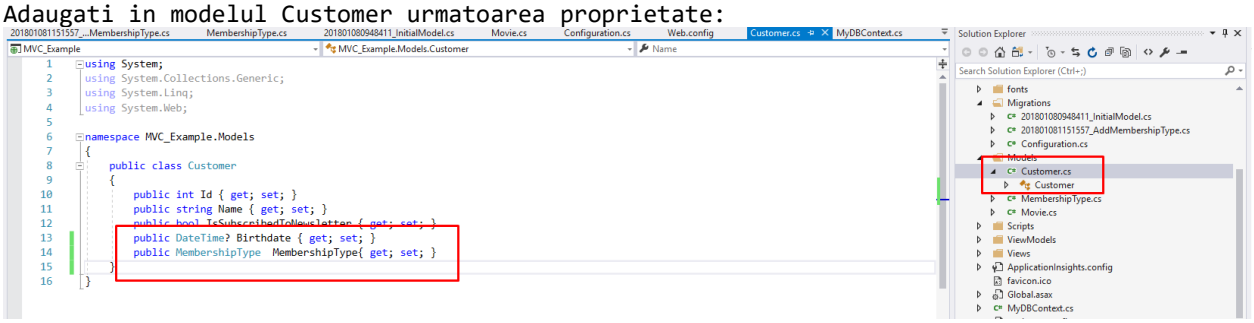
```
PM> update-database -force
Specify the '-Verbose' flag to view the SQL statements being applied to the target database.
Applying explicit migrations: [201801080948411_InitialModel].
Applying explicit migration: 201801080948411_InitialModel.
Caution: Changing any part of an object name could break scripts and stored procedures.
Caution: Changing any part of an object name could break scripts and stored procedures.
Running Seed method.
```



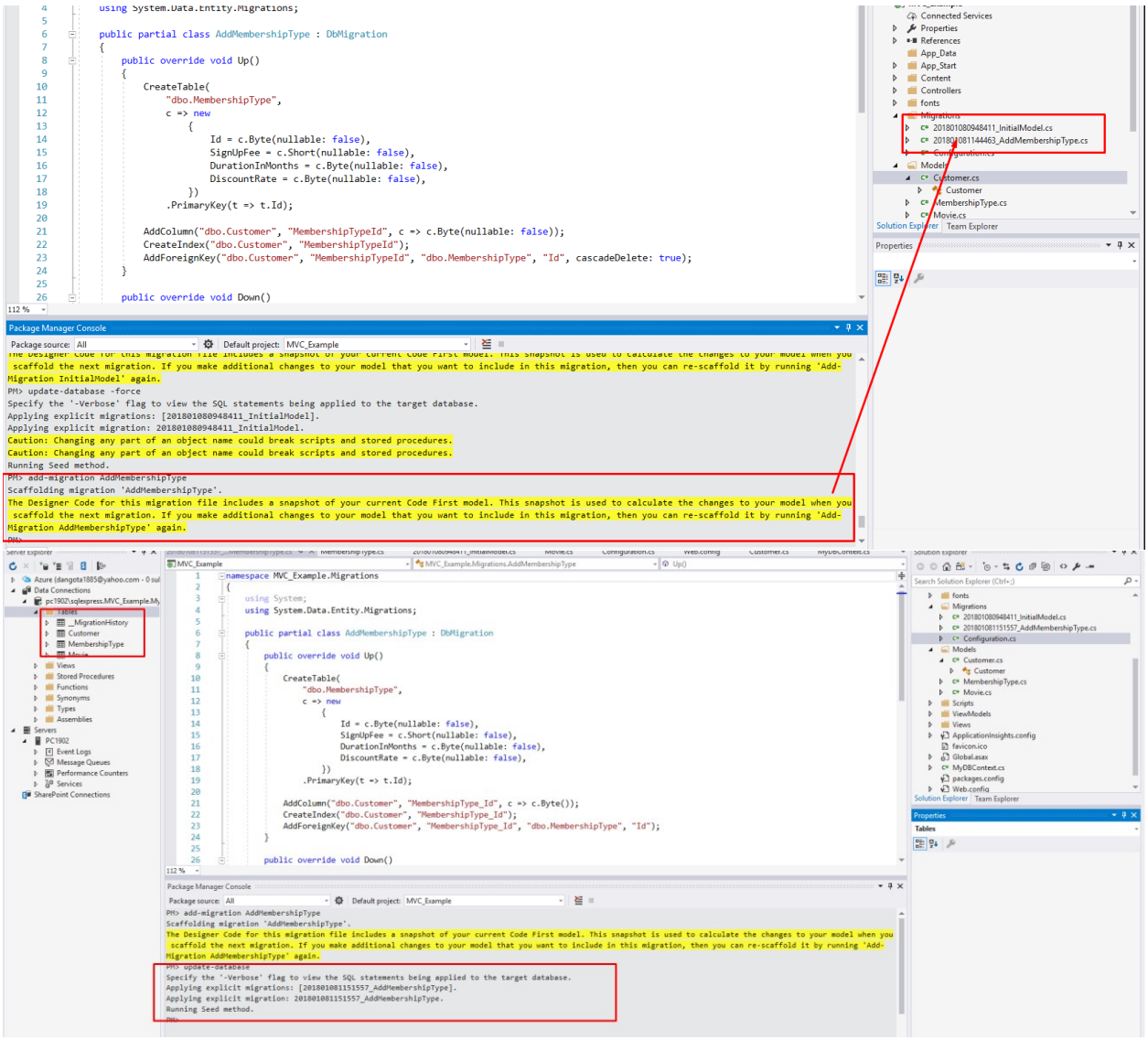
Haideti acum sa modificam clasa Customer sa tratam si cazul in care ceva se modifica in model si vrem sa re-updatam baza de date. Adaugati un nou model dupa imaginea de mai jos:



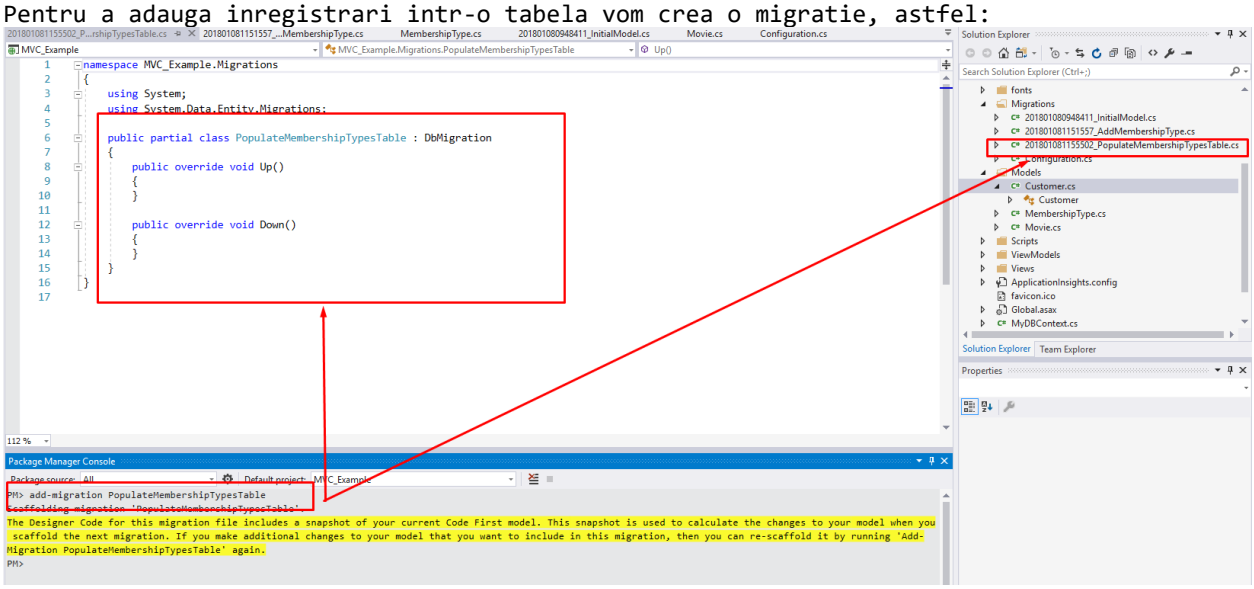
Adaugati in modelul Customer urmatoarea proprietate:



Vom crea o noua migratie numita “AddMembershipType”.



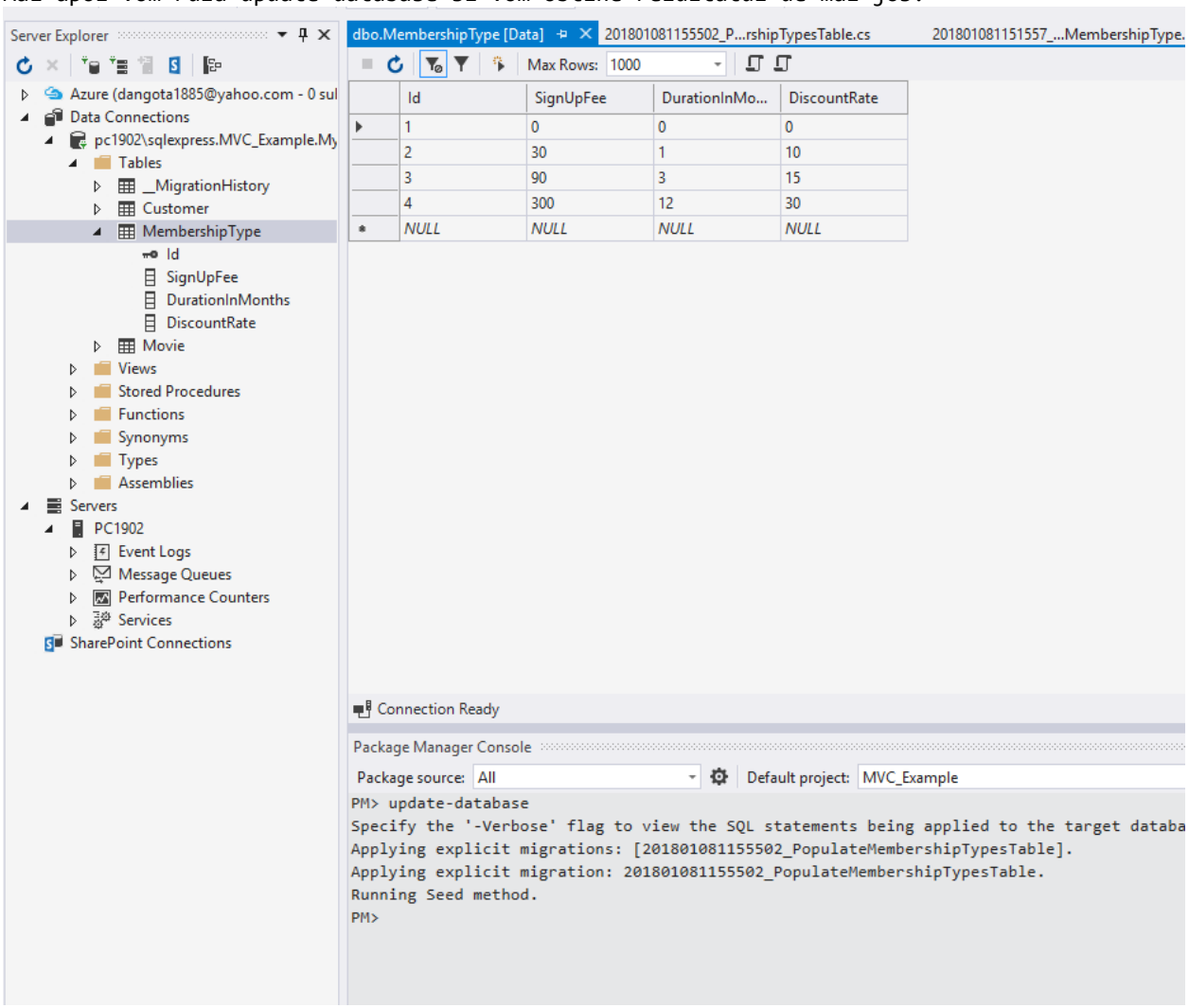
Pentru a adauga inregistrari intr-o tabela vom crea o migratie, astfel:



In metoda up() vom introduce urmatorul cod:

```
  public override void Up()
        {
            Sql("INSERT INTO MembershipType (Id,SignUpFee,DurationInMonths,DiscountRate)
VALUES (1,0,0,0)");
            Sql("INSERT INTO MembershipType (Id,SignUpFee,DurationInMonths,DiscountRate)
VALUES (2,30,1,10)");
            Sql("INSERT INTO MembershipType (Id,SignUpFee,DurationInMonths,DiscountRate)
VALUES (3,90,3,15)");
            Sql("INSERT INTO MembershipType (Id,SignUpFee,DurationInMonths,DiscountRate)
VALUES (4,300,12,30)");
        }
```

Mai apoi vom rula update-database si vom obtine rezultatul de mai jos:



Cod pentru CustomerController:

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

namespace MVC_Example.Controllers
{


    public class CustomersController : Controller
    {
        private MyDBContext _context;

        public CustomersController()
```

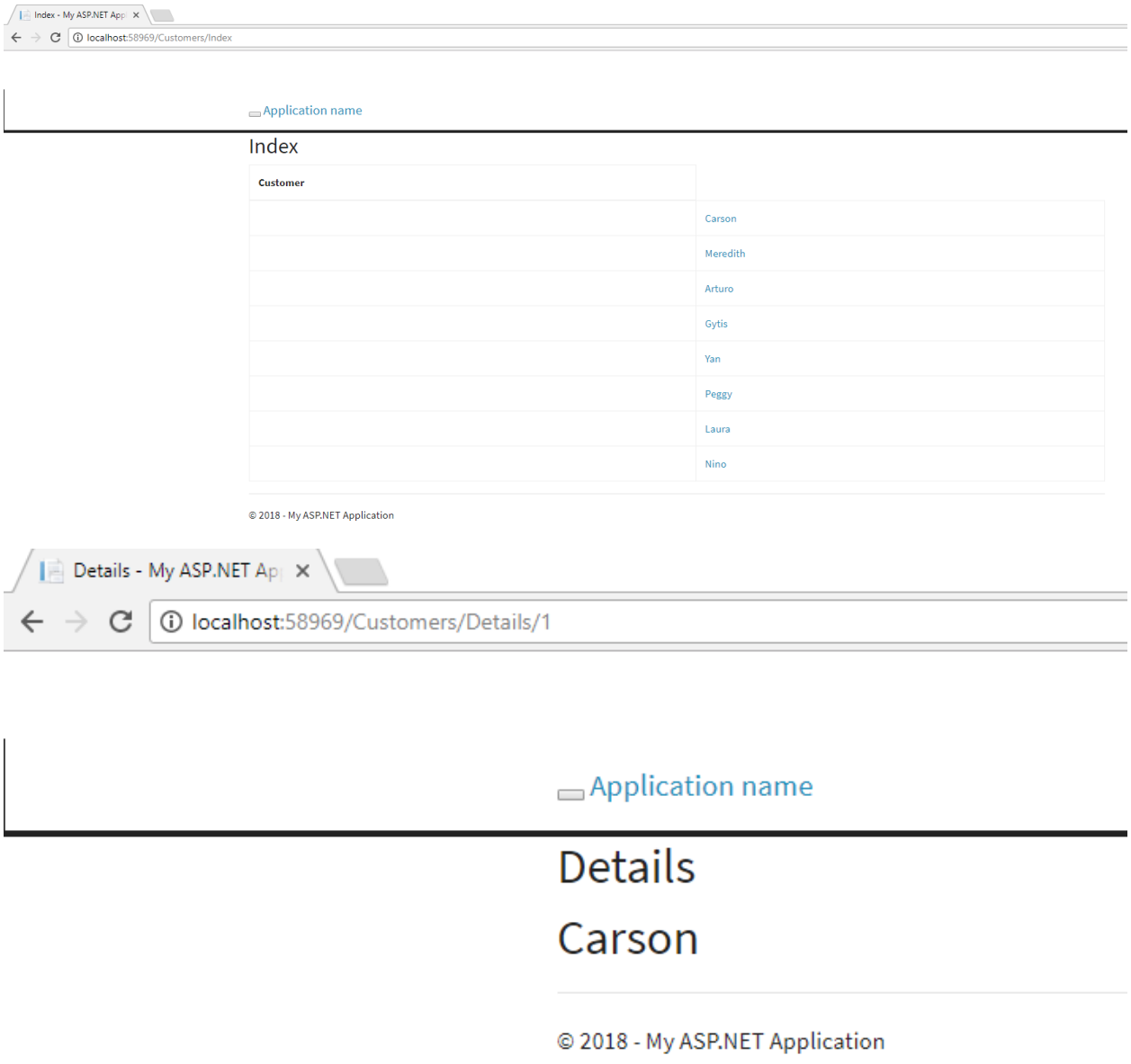
```
        {
            _context = new MyDBContext();
        }
        protected override void Dispose(bool disposing)
        {
            _context.Dispose();
            base.Dispose(disposing);
        }
        // GET: Customers
        public ActionResult Index()
        {
            var customers = _context.Customers.ToList();
            return View(customers);
        }
        public ActionResult New()
        {
            return View();
        }

        [HttpPost]
        public ActionResult Create(Customer newCustomer )
        {
            return RedirectToAction("Index","Customers");
        }
        public ActionResult Details(int id)

            {
                var customer = _context.Customers.SingleOrDefault(c => c.Id == id);

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

                        return View(customer);
                    }


        }
}
Cod pentru view de index customers:
@model IEnumerable<MVC_Example.Models.Customer>
@{
    ViewBag.Title = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Index</h2>
<table class="table table-bordered table-hover">

    <thead>

        <tr>

            <th>Customer</th>

        </tr>

    </thead>
    <tbody>
        @foreach (var customer in Model)
        {
            <tr>
                <td>
                <td>@Html.ActionLink(customer.Name, "Details", "Customers", new { id =
customer.Id }, null)</td>

            </tr>
```

```
        }
    </tbody>
    </table>
```

Cod pentru view de details customer:

```
@model MVC_Example.Models.Customer
@{
    ViewBag.Title = "Details";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

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

Rezultatele sunt urmatoarele:



In pagina de Customer avem display doar pe nume. Daca dorim sa afisam si reducerea va trebui sa efectuam in view urmatoarele modificari:

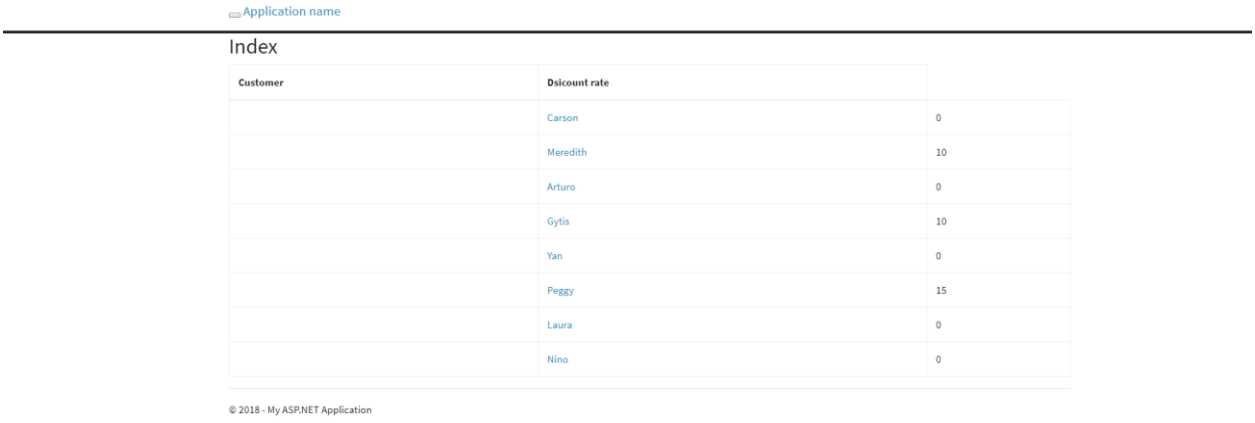
```
</thead>
<tbody>
    @foreach (var customer in Model)
    {
        <tr>
            <td>
            <td>@Html.ActionLink(customer.Name, "Details", "Customers", new { id = customer.Id }, null)</td>
            <td>@customer.MembershipType.DiscountRate</td>
        </tr>
    }
```

Cand vom rula vom obtine urmatorul ecran:

Este o eroare datorata faptului ca Entity Framework nu incarca toate obiectele, si initial incarca doar obtiectele de tip customer, nu si MembershipType. Pentru aceasta trebuie sa executam o schimbare prin care sa incarcam si obiectele din relatie. Acest lucru se numeste **Eager Loading.**

```
    // GET: Customers
    public ActionResult Index()
    {
        var customers = _context.Customers.Include(c => c.MembershipType).ToList();
        return View(customers);
    }
    public ActionResult New()
    {
        return View();
    }
```

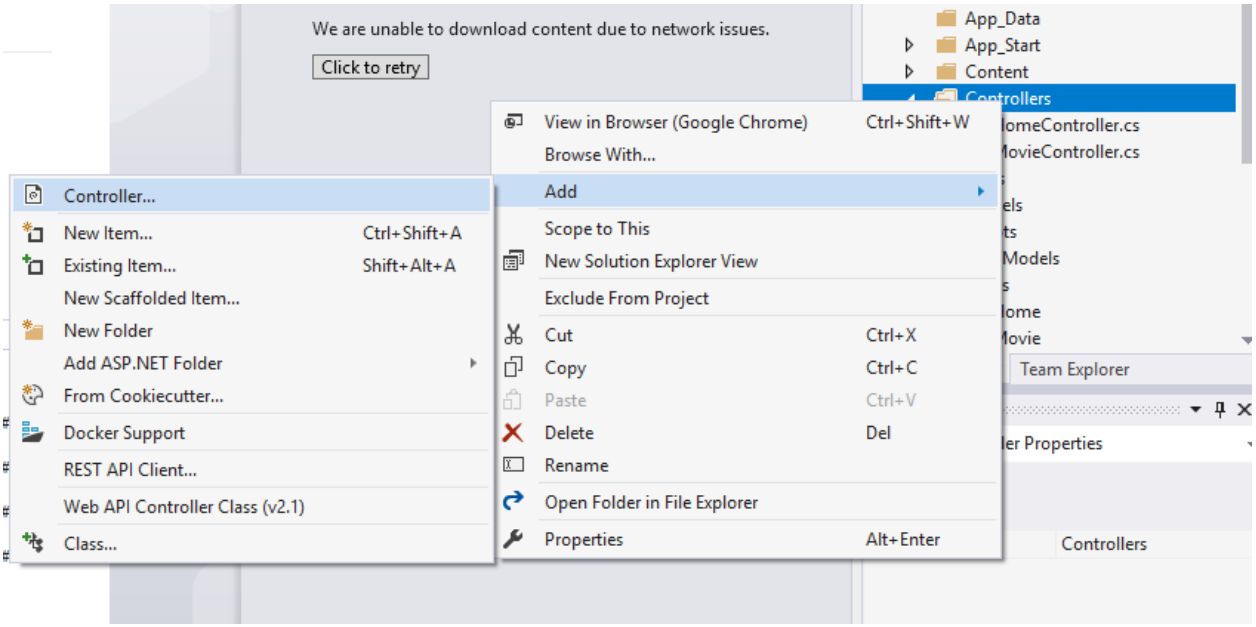
```
using System.Data.Entity;
public ActionResult Index()
        {
            var customers = _context.Customers.Include(c => c.MembershipType).ToList();
            return View(customers);
        }
```



MVC Forms

Pentru a crea un formular intr-o pagina si a face legatura cu actiunea unui controller trebuie effectuate urmatoarele:

➔ Adaugam un nou controller gol denumit Customers



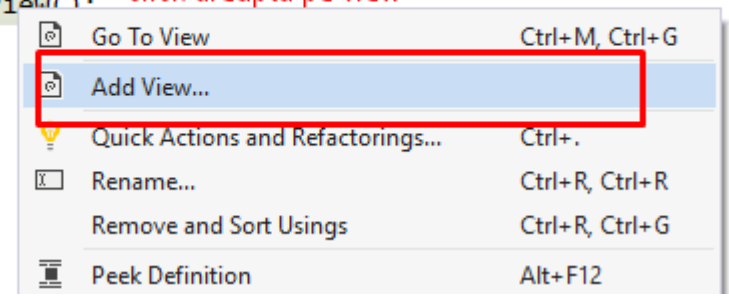
➔ Vom adauga o noua actiune denumita New

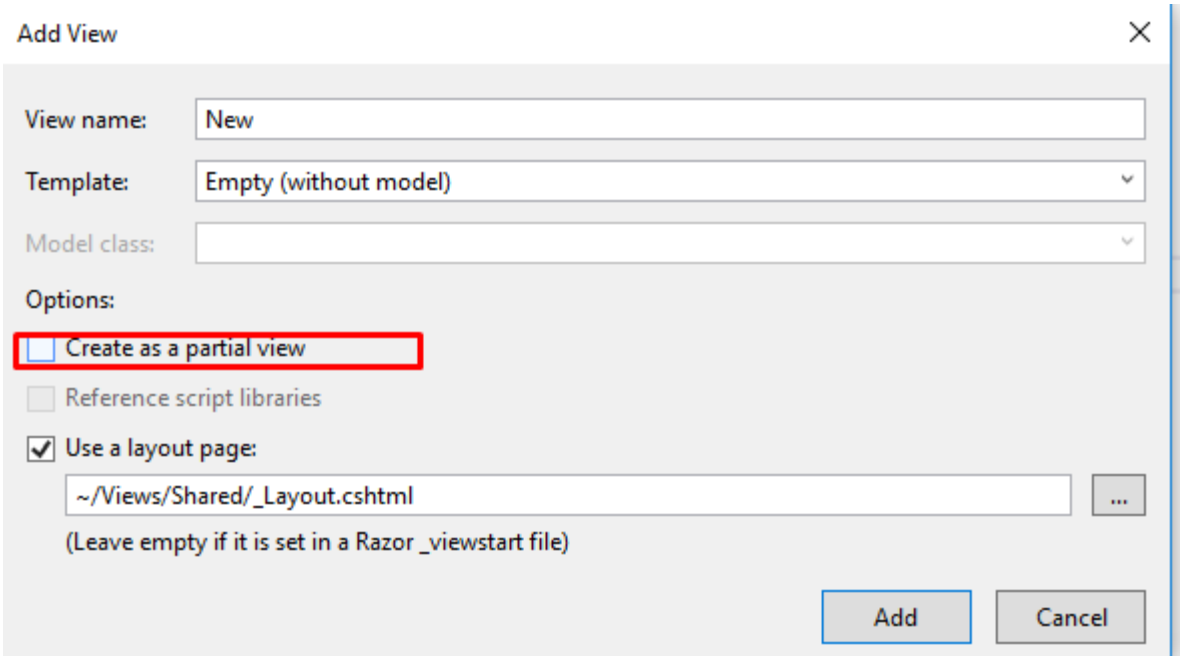
```
7   namespace MVC_Example.Controllers
8   {
9       public class CustomersController : Controller
10      {
11          // GET: Customers
12          public ActionResult Index()
13          {
14              return View();
15          }
16          public ActionResult New()
17          {
18              return View();
19          }
20      }
21  }
```

➔ Vom crea view-ul pentru aceasta actiune. Click dreapta pe cuvantul View din actiunea noastra si selectati optiunea Add view

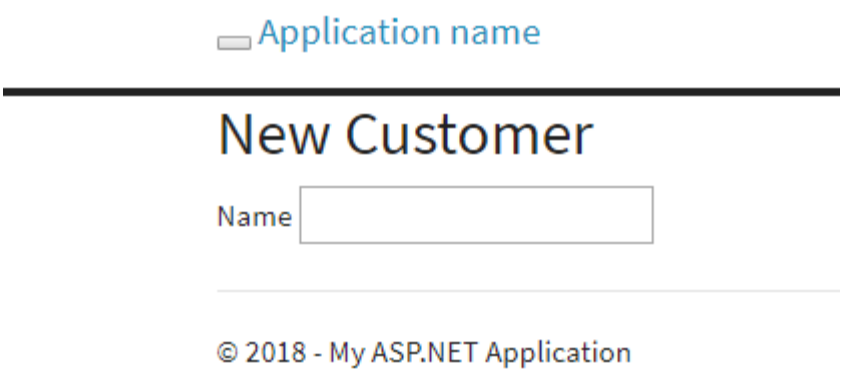
```
    }
    public ActionResult New()
    {
        return View();
    }
}
```

click dreapta pe view



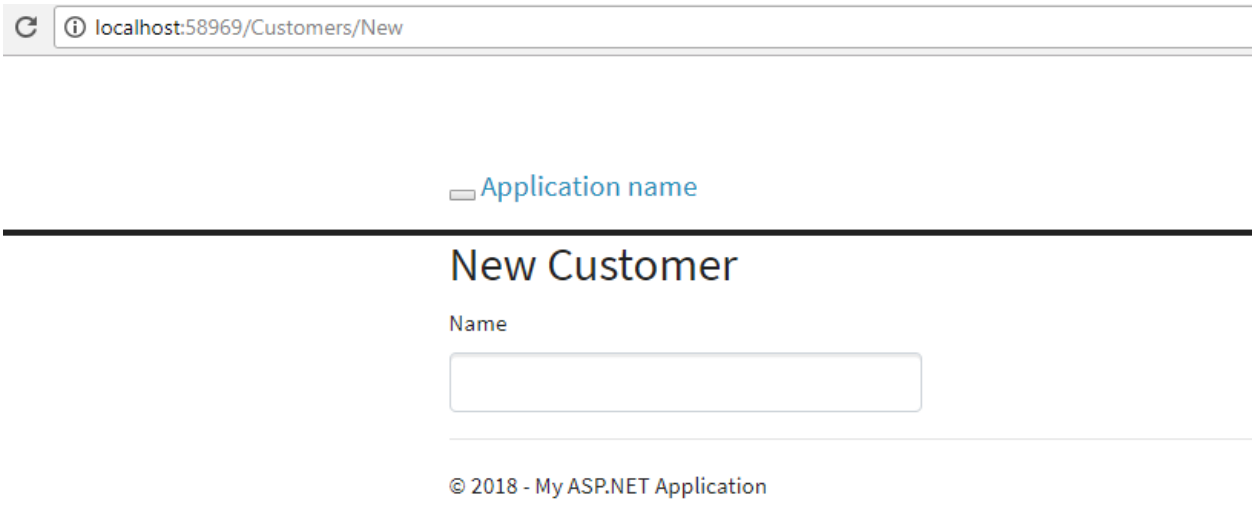



```
@model MVC_Example.Models.Customer
@{
    ViewBag.Title = "New";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>New Customer</h2>

@using (@Html.BeginForm("Create", "Customers"))
{
    <div class="form-group">
        @Html.LabelFor(m=> m.Name)
        @Html.TextBoxFor(m=>m.Name)
    </div>
}
```



Daca vrem sa schimbam class sau sa adaugam orice atribut unul element, efectuam urmatoarele:

Iar ca rezultat avem imaginea de mai jos:



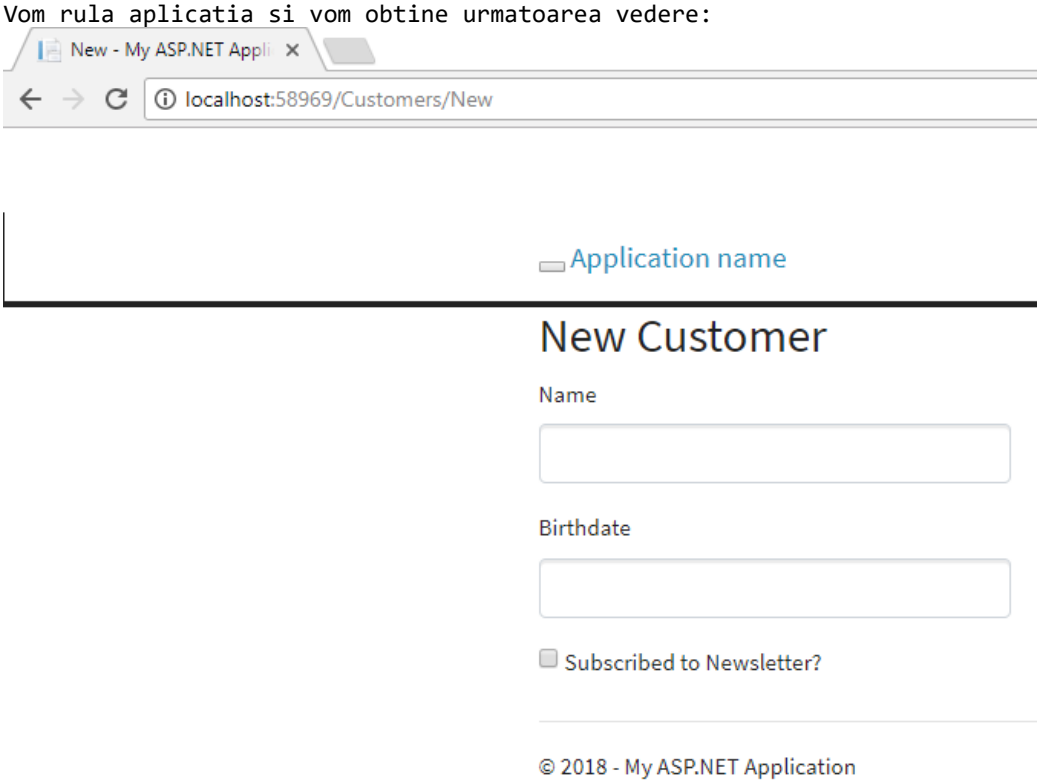
In cele ce urmeaza modificati modelul Customer astfel:

```
public class Customer
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public bool IsSubscribedToNewsletter { get; set; }
        public DateTime? Birthdate { get; set; }
          }
```

Adaugam urmatorul cod in view-ul de new customer”

```
@model MVC_Example.Models.Customer
@{
    ViewBag.Title = "New";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>New Customer</h2>

@using (@Html.BeginForm("Create", "Customers"))
{
    <div class="form-group">
        @Html.LabelFor(m=> m.Name)
        @Html.TextBoxFor(m=>m.Name, new { @class="form-control" })
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Birthdate)
        @Html.TextBoxFor(m => m.Birthdate, new { @class = "form-control" })
    </div>

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

    </div>
      }
```

Vom rula aplicatia si vom obtine urmatoarea vedere:



Daca vrem sa schimbam un label sa aiba alta denumire va trebui sa imcludem in model, la proprietatea dorita

sa fi schimbara, un atribut ca cel de jos:

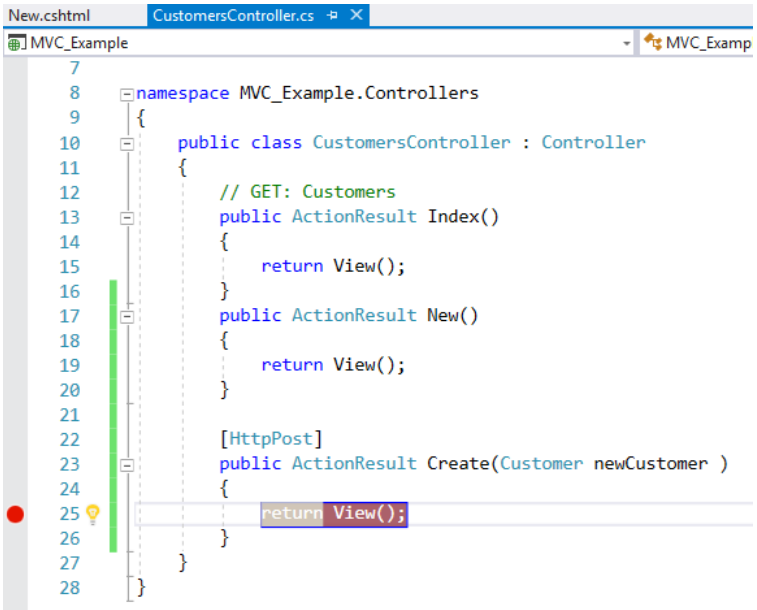
[Display (Name= “Date of Birth”)]

Model binding

Am finalizat formularul nostru, iar ca actiuni urmatoare va trebui sa adaugam un buton sis a salvam un client nou. In view va trebui sa introducem urmatol cod inainte de ultima “}”

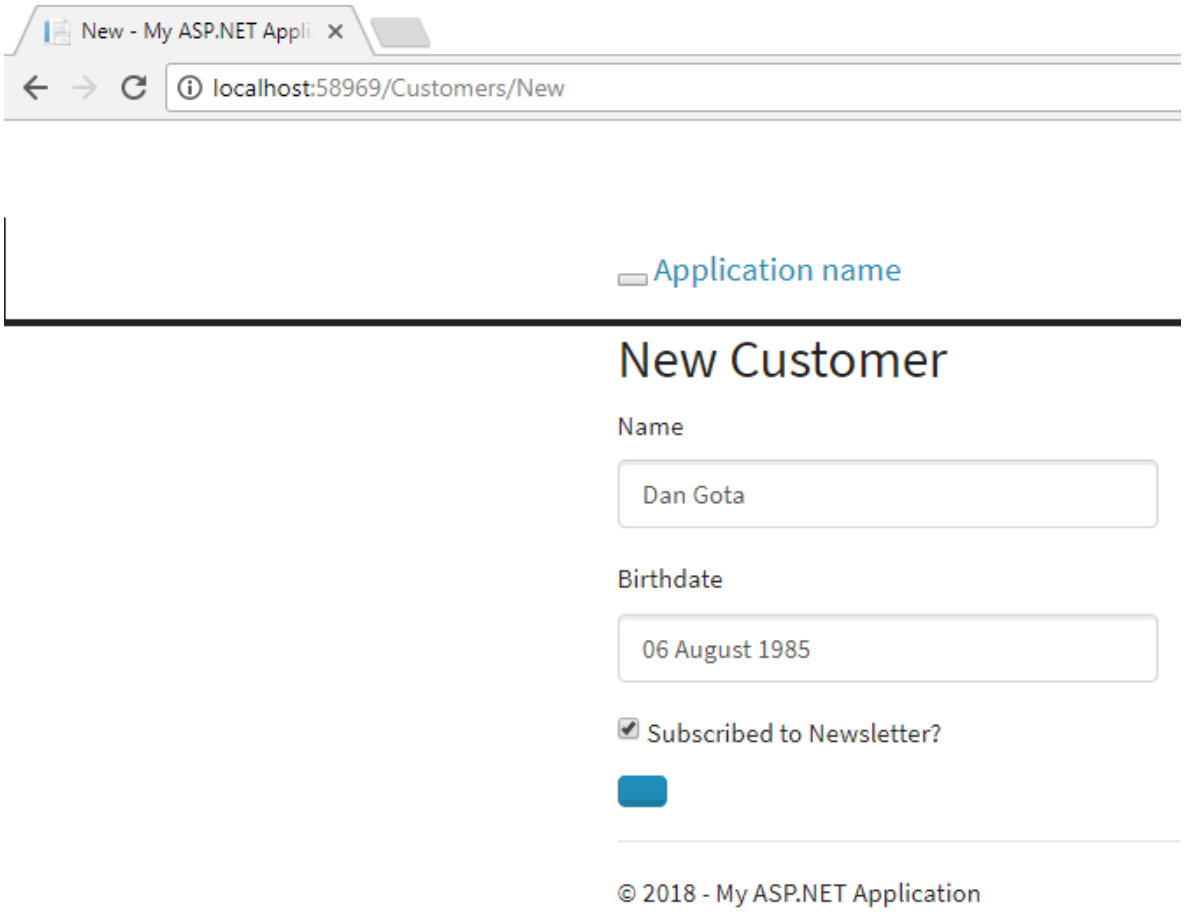
```
<button type="submit" class="btn btn-primary">
                SAVE
            </button>
```

Va trebui sa cream actiunea de creare client nou in controllerul Customer.

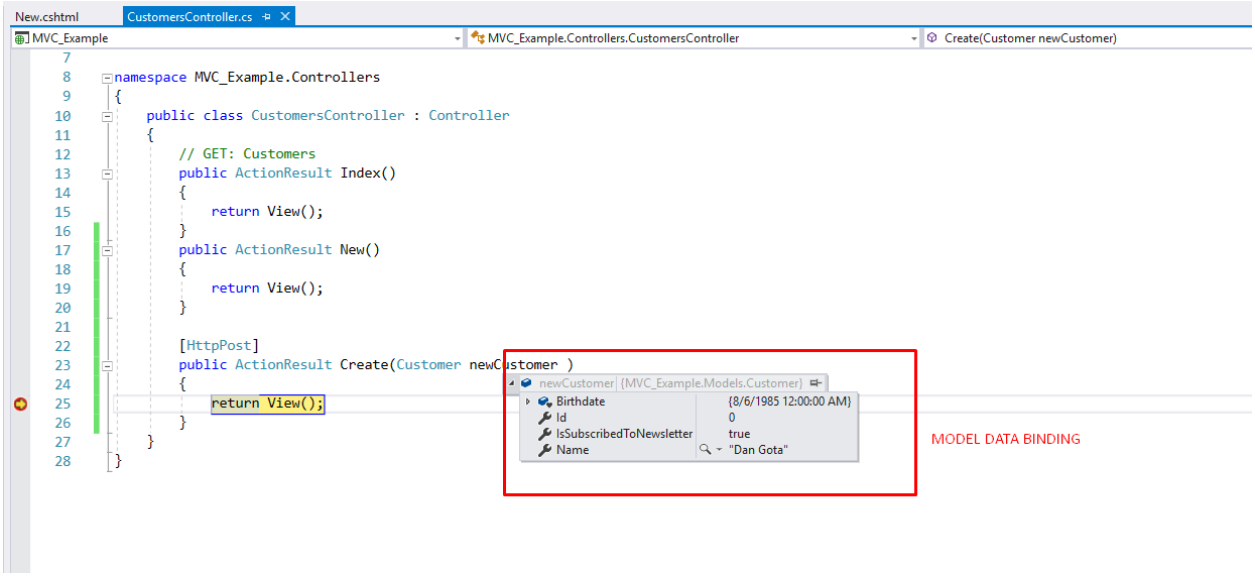


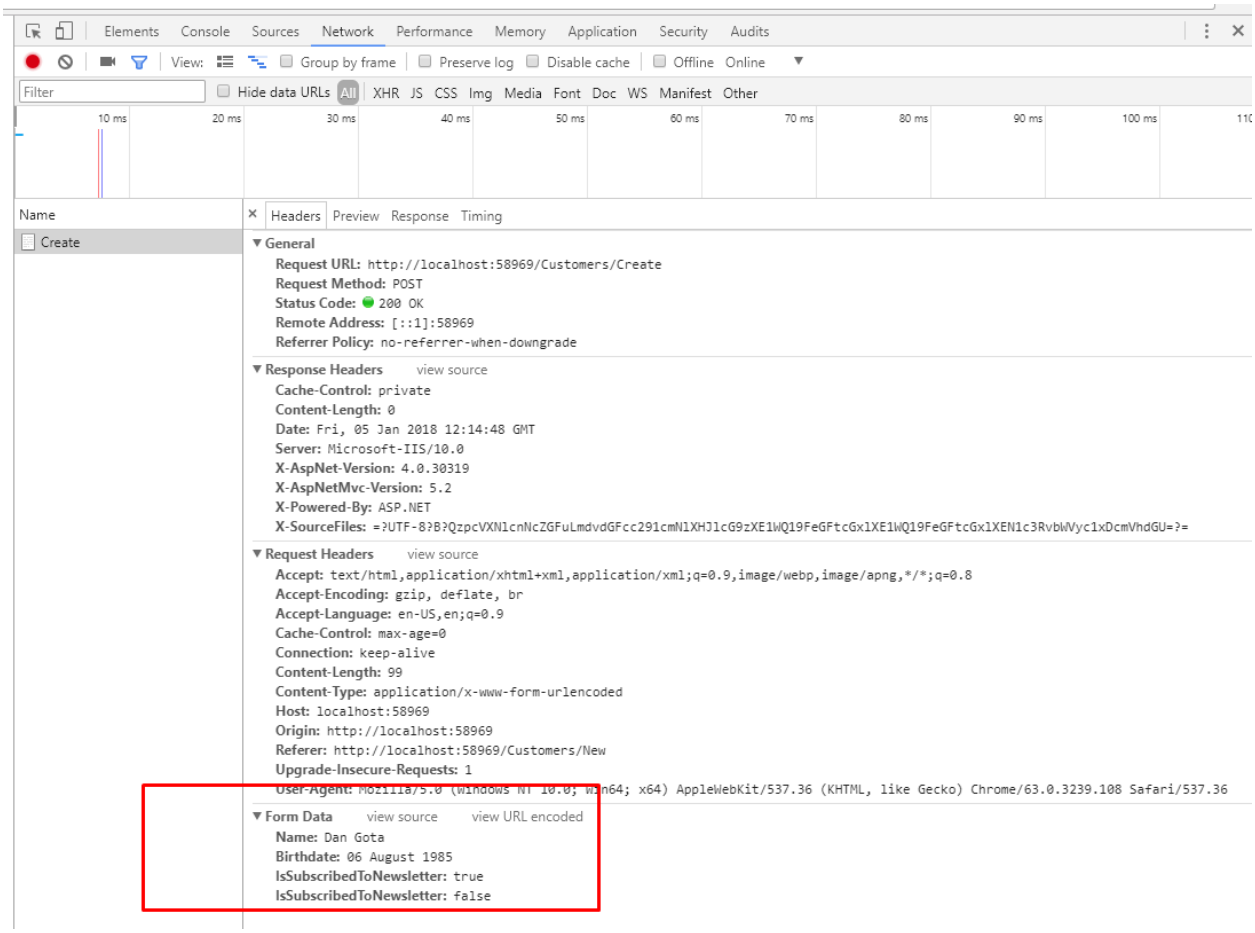

```
7
8   namespace MVC_Example.Controllers
9   {
10      public class CustomersController : Controller
11      {
12          // GET: Customers
13          public ActionResult Index()
14          {
15              return View();
16          }
17          public ActionResult New()
18          {
19              return View();
20          }
21
22          [HttpPost]
23          public ActionResult Create(Customer newCustomer )
24          {
25              return View();
26          }
27      }
28  }
```

Rulam aplicatia



Dup ace am pus un breakpoint in action, vedem ca dupa efectuarea de click pe butonul de save, aplicatia va face un model binding si in controller va fi apelata actiunea cu parametrul newCustomer, care va contine datele introduse in formular.





VALIDARI IN MVC

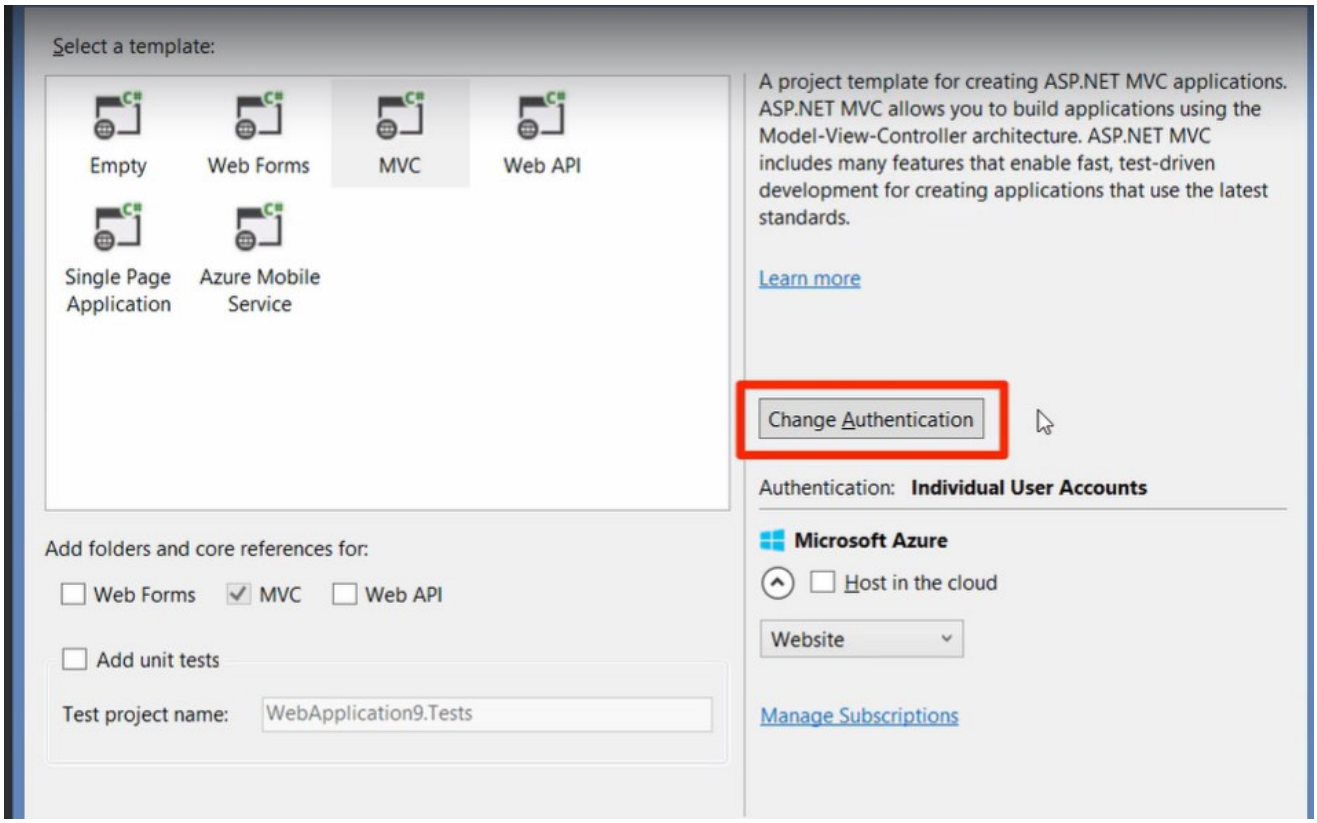
Pentru a adauga anumite restrictii pe model putem sa ne folosim de anotatii(annotations). Avem de exemplu in imaginea de mai jos, pentru proprietatea Name a modelului de Customer, un atribut prin care specificam faptul ca este o proprietate pe care modelul trebuie neaparat sa o aiba completata in momentul in care se creaza un nou obiect de tip Customer. Acest lucru se realizeaza prin utilizarea atributului [Required]. Se poate observa in modelul de mai jos, faptul ca avem mai multe validari pentru proprietatiile obiectului. Avem Required, avem Range – ne limiteaza valorile posibile pentru proprietatea respectiva, StringLength – restrictionarea marimii unei proprietati. In tabelul de mai jos sunt prezentate atribtele posibile pentru date.

Atribut	Descriere
Required	Indica faptul ca proprietatea este obligatorie
StringLength	Defineste o lungime maxima pentru un field de tip string
Range	Defineste un minim si maxim pentru o valoare numerica
RegularExpression	Specifica faptul ca proprietatea se conformeaza unei expresii predefinite
CreditCard	Specifica faptul ca proprietatea este te tip numar de card de credit
CustomValidation	Specifica validari customizate pentru proprietate
EmailAddress	Valideaza proprietatea sub format email
FileExtension	Valideaza cu extensia fisierului
MaxLength	Specifica valoarea maxima a unui field de tip string
MinLength	Specifica valoarea minima a unui field de tip string
Phone	Specifica faptul ca data introdusa repezinta un numar de telefon si compara cu masca pentru aceasta

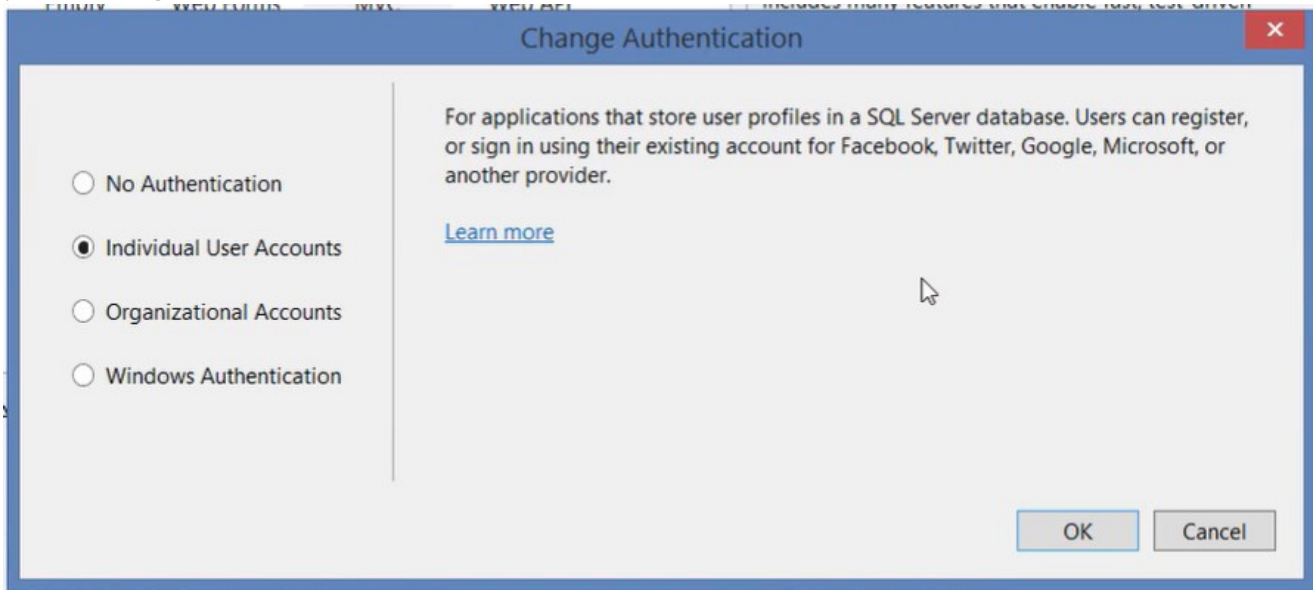
```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Web;

namespace MVC_MSOA_PA_II.Models
{
    public class Movie
    {
        public int Id { get; set; }

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

        public Genre Genre { get; set; }

        [Display(Name = "Genre")]
        [Required]
        public byte GenreId { get; set; }

        public DateTime DateAdded { get; set; }

        [Display(Name = "Release Date")]
        public DateTime ReleaseDate { get; set; }

        [Display(Name = "Number in Stock")]
        [Range(1, 20)]
        public byte NumberInStock { get; set; }

        public byte NumberAvailable { get; set; }
    }
}
```

Autentificare si Autorizare in MVC

In momentul crearii proiectului am ales/am avut optiunea sa alegem modalitatile de autentificare utilizate pentru proiectul nostru. Daca nu va mai aduceti aminte de fereastra, aceasta este prezentata in imaginea de mai jos:



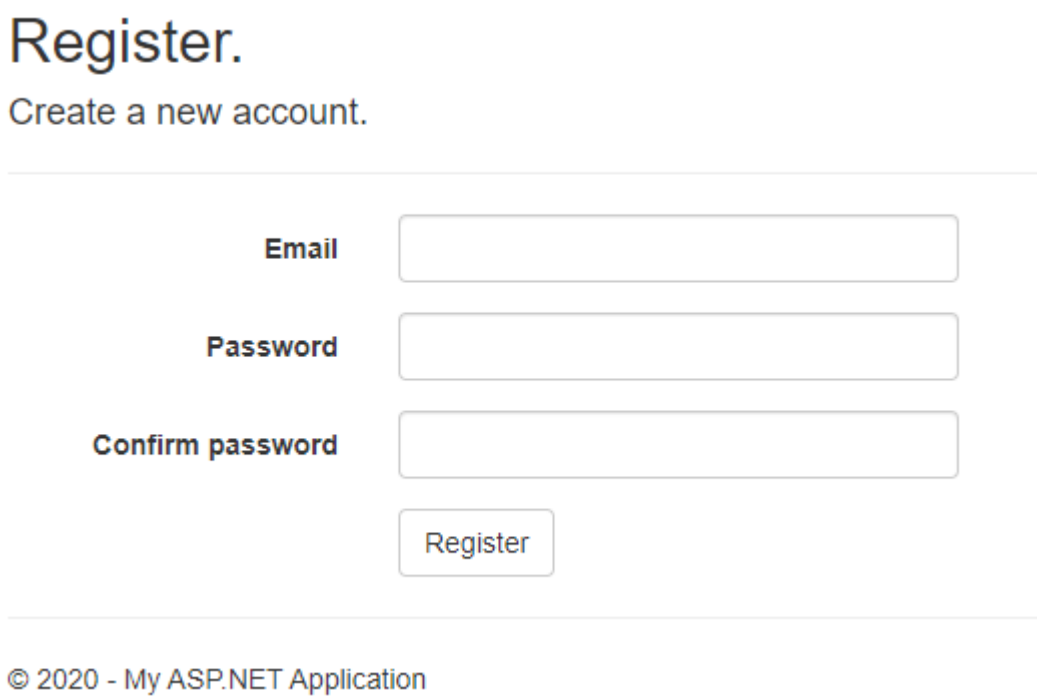
Accesand optiunea „Change Authentication” vom avansa in fereastra de mai jos, de unde putem alege din cele 4 metode de autentificare diferite.



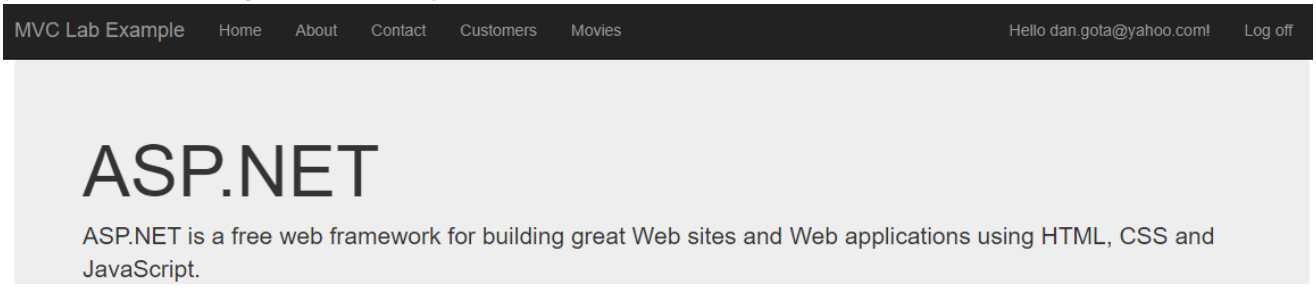
No Authentication - aplicatia este disponibila utilizatorilor anonimi
Individual User Accounts – este optiunea implicita si care se preteaza pentru web site-uri accesibile via internet unde utilizatorii se vor loga in aplicatie
Organizational Accounts – este folosita in special in companii, unde se doreste sa se efectueze autentificare cu Active Directory, Microsoft Azure, Office 365.
Windows Authentication – Utilizata pentru aplicatii intranet. Se vor autentifica in aplicatie automat cu userul logat pe statia de lucru.

In cele ce urmeaza vom discuta optiunea „Individual User Accounts”.

In figura de mai sus puteti observa optiunile „Register” si „Log in” ale aplicatiei dezvoltate pana in prezent. Prima optiunea ne da posibilitatea de a introduce un utilizator nou pentru aplicatia dezvoltata.



A doua optiunea, cea de log in, ofera posibilitatea utilizatorilor inregistrati sa se logheze in aplicatia dezvoltata. Dupa Log in sau Register o sa fim introdusi direct in site, asa cum este prezentat in imaginea de mai jos.



Intregul proces de inregistrare si logare in aplicatie este asigurat de framework-ul Microsoft Asp.Net Identity. Acest framework dispune de clase domeniu(Domain classes) cum ar fi: IdentityUser si Role si asigura un API simplu care lucreaza cu aceste clase. Clasele din API sunt UserManager, SignInManager si RoleManager, etc. Datele sunt stocate in baze de date SQL iar mecanismul de accesare a datelor este Entity Framework.

In cele ce urmeaza efectuati urmatorii pasi:

1) Inregistrati un nou utilizator
2) Introduceti valori pentru email si parola
3) Efectuati click pe register.

Se observa in cele ce urmeaza faptul ca sunteti direct redirectionati in pagina site-ului iar un mesaj de intampinare va fi afisat in coltul din dreapta sus al ecranului. Pentru a studia intreg mecanismul de inregistrare, imaginea de mai jos prezinta controllerul `AccountController` impreuna cu actiunea de register.

Prima actiunea de Register este cea care va returna vederea de pagina Register. A doua actiune de register va efectua urmatoarele:

```
// POST: /Account/Register
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Register(RegisterViewModel model)
{
    if (ModelState.IsValid)
    {
        var user = new ApplicationUser { UserName = model.Email, Email = model.Email };
        var result = await UserManager.CreateAsync(user, model.Password);
        if (result.Succeeded)
        {
            await SignInManager.SignInAsync(user, isPersistent:false, rememberBrowser:false);

            // For more information on how to enable account confirmation and password reset please visit https://go.microsoft.com/fwlink/?LinkID=320771
            // Send an email with this link
            // string code = await UserManager.GenerateEmailConfirmationTokenAsync(user.Id);
            // var callbackUrl = Url.Action("ConfirmEmail", "Account", new { userId = user.Id, code = code }, protocol: Request.Url.Scheme);
            // await UserManager.SendEmailAsync(user.Id, "Confirm your account", "Please confirm your account by clicking <a href=\"" + callbackUrl + "\">here</a>"

            return RedirectToAction("Index", "Home");
        }
        AddErrors(result);
    }

    // If we got this far, something failed, redisplay form
    return View(model);
```

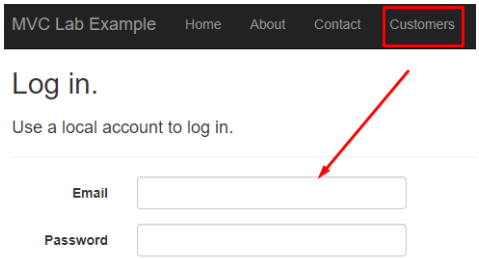
Data ModelState este valid se va apela asincron o metoda de creare utilizator (CreateAsync). Daca aceasta actiune este finalizata cu succes, atunci va loga userul prin apelarea metodei SignInAsync si va efectua o redirectare care actiunea Index din controllerul Home. Codul comentat din imaginea de mai jos este util in momentul in care nu doriti sa fiti logat automat in pagina, ci doriti sa implementati mecanismul de trimitere email de confirmare adresa de email si abia apoi sa puteti sa va logati in aplicatie.

Restrictionarea accesului in aplicatie se face cu ajutorul unui atribut numit [Authorize]. Acesta va decora orice actiune si va fi apelat de fiecare data inainte de executia acelei actiuni. Vom restrictiona in cele ce urmeaza actiunea Index din controller-ul Customer.

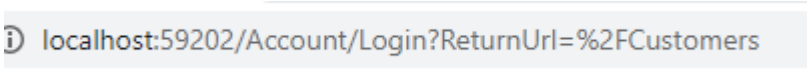
```
[Authorize]
public ActionResult Index()
{
    //var customers = _context.Customers.Include(c => c.MembershipType).ToList();
    return View();
}
```

Ca preconditii, trebuie sa fim delogati, iar ca rezultat vom avea urmatoarele imagini prezentate mai jos:

Efectuand click pe Customers vom fi redirectionati catre pagina de log in



Ca si url vom observa faptul ca este putin mai complex. Acesta va mai contine pe langa Account/Login si calea unde o sa fie redirectionat utilizatorul daca se va loga cu succes, in acest caz controllerul Customers, actiune default de Index.



In cazul de fata, atributul [Authorize] a fost aplicat unei singure actiuni. Putem de asemenea sa aplicam atributul unei intregi clase, sau chiar global.

```
[Authorize]
public class CustomersController : Controller
{
    private MyDBContext _context;
```

Aplicarea atributului intregii clase. In acest caz, toate actiunile din clasa sunt protejate, iar userii trebuie sa fie logati pentru a le accesa.

```
//filter config este localizat in App_Start
public class FilterConfig
{
    public static void RegisterGlobalFilters(GlobalFilterCollection filters)
    {
        filters.Add(new HandleErrorAttribute());
        filters.Add(new AuthorizeAttribute());
    }
```

In cazult de mai sus aplicatia este protejata global, iar orice user trebuie sa se logheze pentru a accesa orice actiune disponibila.

Cazul in care aplicam filtrul global este putin ineficient, deoarece nimeni nu va putea accesa pagina doar fiind logat. Astfel, exista posibilitatea ca avand atributul/filtrul aplicat global, totusi sa permitem unor actiuni sa fie accesate anonimizat. Pentru aceasta o sa decoramt actiunea de Index din controller-ul Home cu atributul [AllowAnonymuous] ca in imaginea de mai jos. In acest moment, desi aplicatia necesita autentificare, accesarea view-ului din actiunea Index din controller-ul Home este posibila si fara a fi logat.

```
[AllowAnonymous]
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }
```

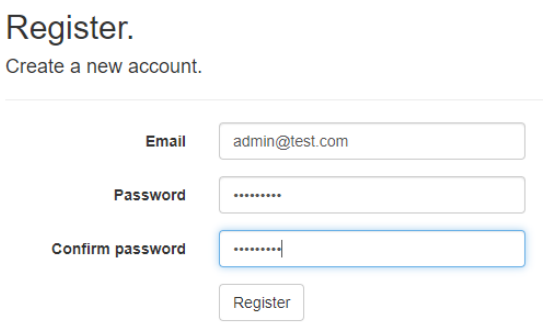
Pana acum am benificiat de functionalitati standard in aplicatie. Urmeaza sa complicam putin lucrurile si sa adaugam roluri in aplicatia noastra. Pentru sarcina urmatoare, ne propunem sa cream un rol de manager in site-ul creat. Pentru aceasta vom efectua urmatorii pasi:

1) Accesam actiunea Register din controllerul AccountController

2) Adaugam temporat codul din imaginea de mai jos. Acesta va forta ca toti userii creati sa aiba rolul de ManagerFilme in aplicatia noastra.

```
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Register(RegisterViewModel model)
{
    if (ModelState.IsValid)
    {
        var user = new ApplicationUser { UserName = model.Email, Email = model.Email };
        var result = await UserManager.CreateAsync(user, model.Password);
        if (result.Succeeded)
        {
            //temp code
            var roleStore = new RoleStore<IdentityRole>(new ApplicationDbContext()); //creare magazin
            var roleManager = new RoleManager<IdentityRole>(roleStore);              //creare rol manager
            await roleManager.CreateAsync(new IdentityRole("ManagerFilme"));              //creare rol cu denumirea ManagerFilme
            await UserManager.AddToRoleAsync(user.Id, "ManagerFilme");

            await SignInManager.SignInAsync(user, isPersistent:false, rememberBrowser:false);

            // For more information on how to enable account confirmation and password reset please visit https://go.microsoft.com/f
            // Send an email with this link
```

3) Rulam aplicatia si cream un user cu denumirea Admin ca in imaginea de mai jos:

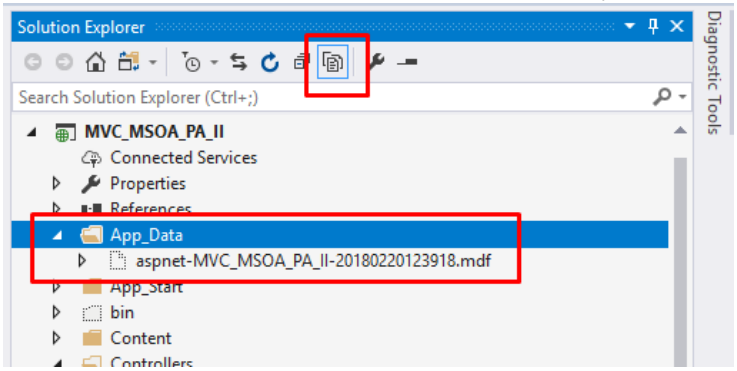


4) Inchidem aplicatia si comentam codul introdus in pasii 1+2 la fel ca in imaginea de mai jos:

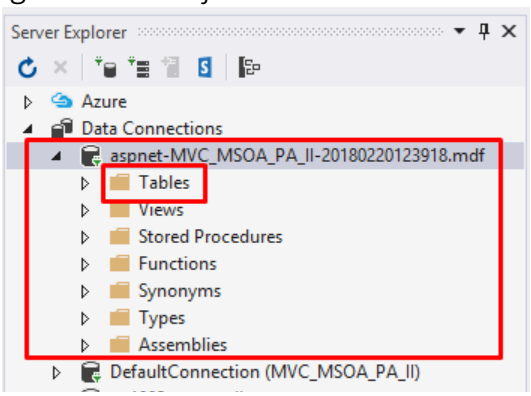
```
var user = new ApplicationUser { UserName = model.Email, Email = model.Email };
var result = await UserManager.CreateAsync(user, model.Password);
if (result.Succeeded)
{
    ////temp code
    //var roleStore = new RoleStore<IdentityRole>(new ApplicationDbContext()); //creare magazin
    //var roleManager = new RoleManager<IdentityRole>(roleStore);              //creare rol manager
    //await roleManager.CreateAsync(new IdentityRole("ManagerFilme"));              //creare rol cu denumirea ManagerFilme
    //await UserManager.AddToRoleAsync(user.Id, "ManagerFilme");

    await SignInManager.SignInAsync(user, isPersistent:false, rememberBrowser:false);
```

In momentul acesta o sa avem userii normali creati initial si un user admin care are rolul de ManagerFilme in aplicatia dezvoltata.

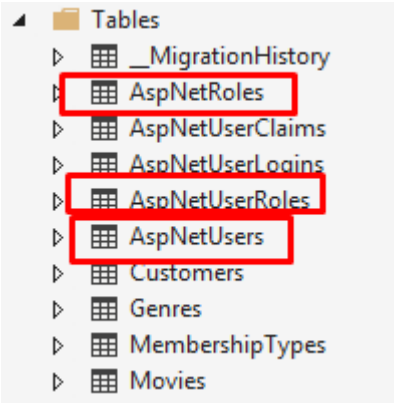
Vizualizarea rezultatelor acestor 5 pasi se poate efectua astfel. In App_Data cu optiunea „Show all data” vom deschide baza de date locala creata de aplicatia dezvoltata.

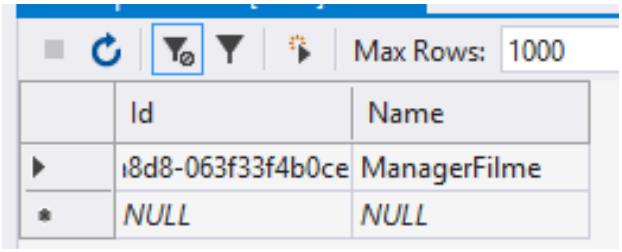


Fereastra prezentata in imaginea de mai jos o sa se deschida:



Deschizand folder-ul Tables o sa avem o privire de ansamblu asupra tabelelor aplicatiei.

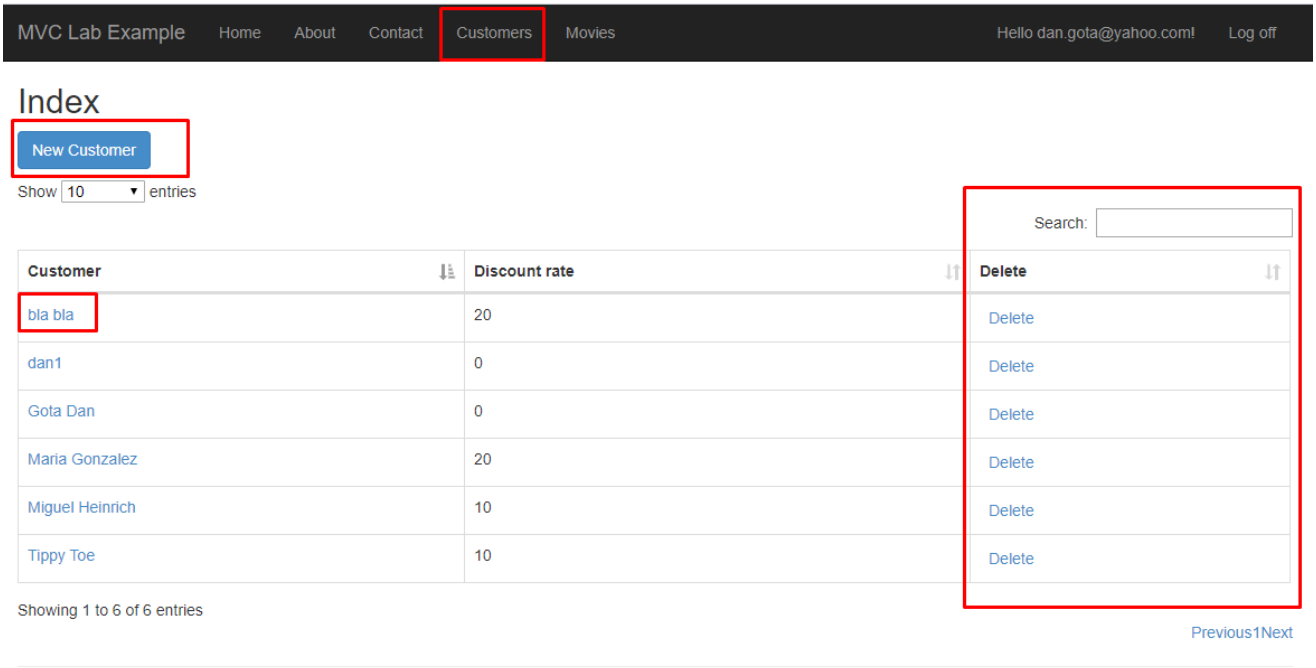




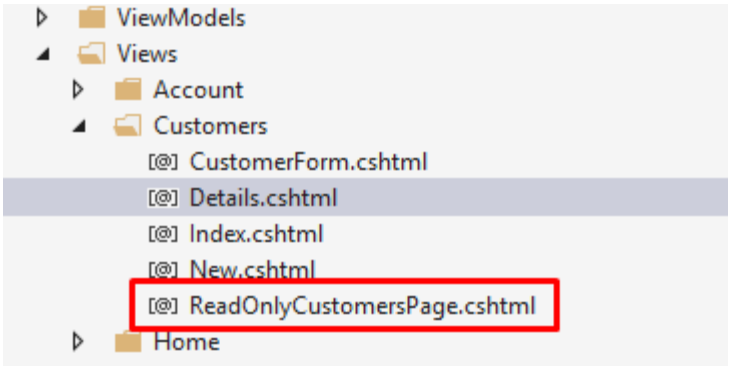
Continutul tabelei AspNetRoles unde vom avea rolul creat custom cu codul din pasii efectuati anterior.

In tabela AspNetUsers o sa fie prezenti userii creati pana in acest moment in aplicatie. In tabela AspNetRoles o sa avem rolurile create, iar in tabela AspNetuserRoles o sa avem tabela de asociere rol la user.

Introducand rolurile, un pas firesc este sa afisam continut diferit in functie de rolul setat pentru fiecare user. Astfel, vom dori sa modificam pagina de „Customers” astfel incat pentru utilizatorii normali sa nu fie afisate optiunile de „New Customer”, sa nu fie afisata coloana Delete din tabel, iar continutul coloanei Customer din tabel sa nu fie un url catre pagina de editare date client.



Cel mai simplu mod de a face acest lucru este de a ascunde elementele din pagina pe care le dorim afisate numai pentru un anumit rog. Acest lucru, desi cel mai simplu, este ineficient. Rolurile se pot schimba, noi roluri de pot adauga, iar codul scris nu va mai respecta normele de editare. Asadar, modul prin care dorim sa prezentam continut pe baza rolurilor detinute va fi prin creara de noi vederi(view-uri) specifice per rol.
Vom duplica view-ul de Index din folderul Views->Customers si il vom redenumi ca in figura de mai jos:



Vom modifica in cele ce urmeaza view-ul creat precedent, astfel incat sa nu mai afiseze elementele care duc spre editarea/inserarea de noi clienti. Vom sterge urmatoarele linii de cod:

```
@Html.ActionLink("New Customer", "New", "Customers", null, new { @class = "btn btn-primary" })
```

Vom sterge headerul de Delete din tabel:

```
<th>Delete</th>
```

Pentru a elimina din tabel posibilitatea de a efectua click pe numele clientului si a fi redirectionat catre pagina de editare client o sa stergem urmatoarele linii de cod:

```
<script>
    $(document).ready(function () {

        var table = $("#customers").DataTable({
            ajax: {
                url: "/api/customers",
                dataSrc: ""
            },
            columns: [
                {
                    data: "name",
                    render: function (data, type, customer) {
                        return "<a href='/customers/edit/" + customer.id + "'>" + customer.name + "</a>";
                    }
                },
```

Pentru a sterge coloana de delete din tabel vom sterge din script urmatoarele linii de cod (atentie la stergere, se va sterge numai de la VIRGULA. DOAR TEXTUL SELECTAT in imaginea de mai jos se va sterge):

```
                },
                {
                    data: "membershipType.discountRate"
                },
                {
                    data: "id",
                    render: function (data) {
                        return "<button class='btn-link js-delete' data-customer-id=" + data + ">Delete</button>";
                    }
                }
            ]
        });
```

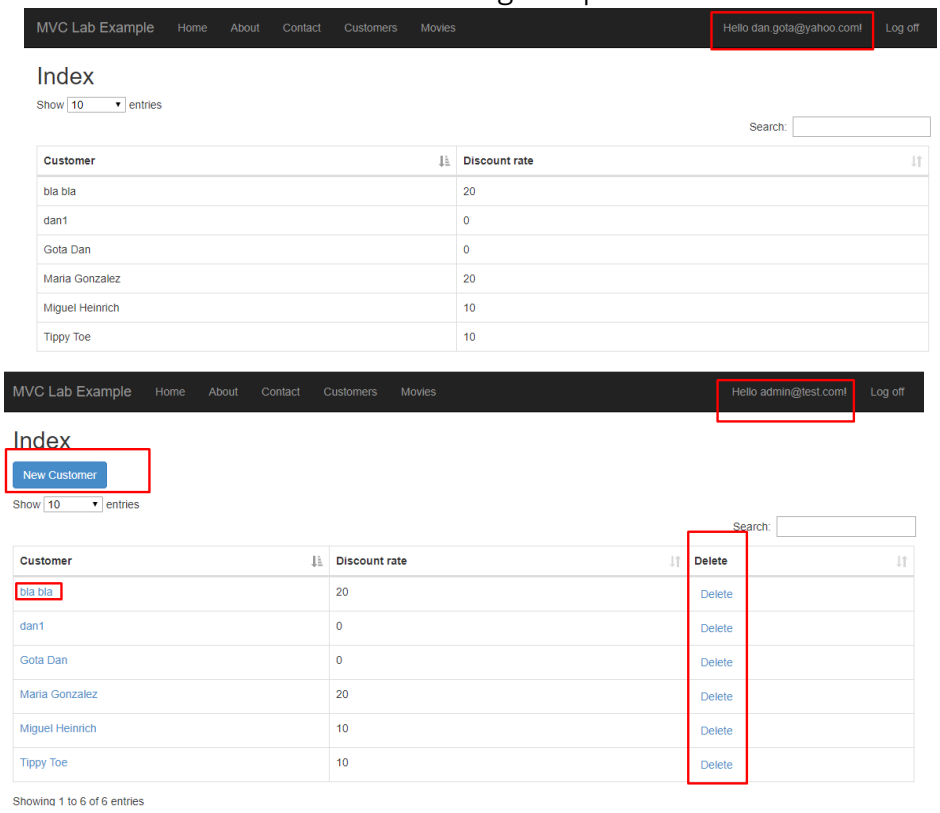
De asemenea nu mai avem nevoie de functionalitatea de on click din script:

```
$("#customers").on("click", ".js-delete", function () {
    var button = $(this);
    bootbox.confirm("Are you sure you want to delete this customer?", function (result) {
        if (result) {
            $.ajax({
                url: "/api/customers/" + button.attr("data-customer-id"),
                method: "DELETE",
                success: function () {
                    table.row(button.parents("tr")).remove().draw();
                    location.reload(true);

                },
                error: function () {
                    alert("there was an error at delete!");
                }
            });
        }
    });

});
```

Dupa efectuarea acestor pasi, vom inchide view-ul creat si vom deschide controller-ul CustomersController pentru a efectua modificarile necesare completarii acestui exercitiu. Actiune de Index va fi modificata astfel incat sa contina urmatoarele linii de cod:

```
public ActionResult Index()
        {
            //var customers = _context.Customers.Include(c => c.MembershipType).ToList();
            if (User.IsInRole("ManagerFilme"))
                return View("Index");
            else
                return View("ReadOnlyCustomersPage");
        }
```

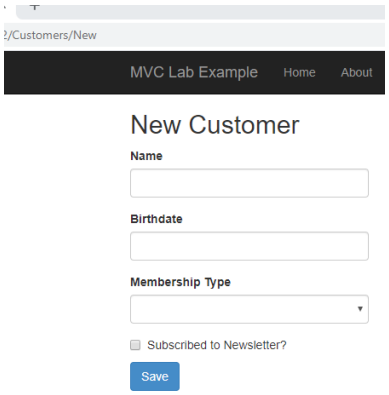
Aceste linii de cod testeaza daca userul care acceseaza aceasta pagina are sau nu rolul de „ManagerFilme” creat anterior. Functie de acest lucru se va returna vederea(view-ul) corespunzator. Rezultatul este cel afisat in imaginile prezentate in cele ce urmeaza:



In acest moment view-ul randat in momentul accesarii linkului este cel in functie de rolul utilizatorului. Fara alte modificari efectuate, in acest moment, daca cineva stie linkul catre adaugare client/customer nou, o poate face fara probleme. Pentru exemplificare, introduceti in bara de navigare urmatorul url:

http://localhost:59202/Customers/New

aveti in vedere faptul ca aplicatia dezvoltata pe calculatorul propriu, mai mult ca sigur nu va avea acelasi port(valoarea de port 59202 este cu siguranta diferita in cazut dumneavoastra). Dupa introducere o sa fiti directionati catre pagina de adaugare Client now.



Acest lucru este de dorit sa poata fi accesat numai daca utilizatorul are rolul de ManagerFilme. Pentru aceasta vom efectua urmatoarele modificari.

```
}
[Authorize(Roles ="ManagerFilme")]
public ActionResult New()
{
    var membershipTypes = _context.MembershipType.ToList();
    var viewModel = new CustomerFormViewModel
    {
        Customer = new Customer(),
        MembershipTypes = membershipTypes
    };

    return View("CustomerForm", viewModel);
}
```

Rezultatul este faptul ca utilizatorul daca nu are rolul precizat o sa fie redirectionat catre pagina de „Log in”. Pentru o mai buna implementare vom crea o clasa statica in care vom crea aceste roluri ca constante. Astfel vom elimina sansele de a insera gresit rolurile, iar in cazul in care un rol isi schimba numele schimbarea in cod va trebui facuta intr-un singur loc.

Asadar, efectuati ca si exercitiu urmatorii pasi:

1) In Folderul Model creati o clasa statica noua cu denumirea de RoleName
2) Adaugati o constanta
3) Schimbati unde ati decorat actiunile cu atribute/filtre de autorizare denumirea lor.

(dispuneti mai jos de cele doua imagini rezultate in urma implementarii(clasa si controller-ul))

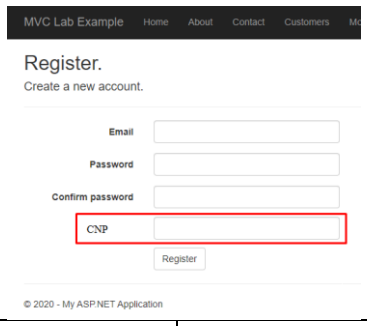
```
namespace MVC_MSOA_PA_II.Models
{
    public static class RoleName
    {
        public const string ManagerFilme = "ManagerFilme";
    }
}
```

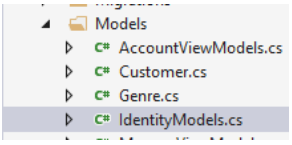
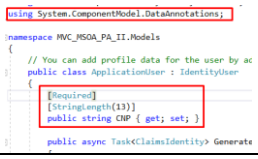
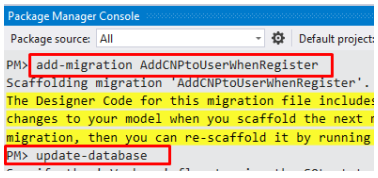
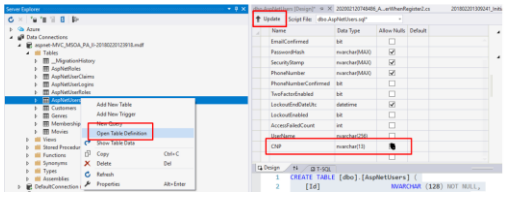
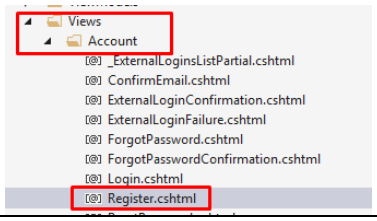
```
public ActionResult Index()
{
    //var customers = _context.Customers.Include(
    if (User.IsInRole(RoleName.ManagerFilme))
        return View("Index");
    else
        return View("ReadOnlyCustomersPage");
}
[Authorize(Roles =RoleName.ManagerFilme)]
public ActionResult New()
{
    var membershipTypes = _context.MembershipType
    var viewModel = new CustomerFormViewModel
    {
        Customer = new Customer(),
        MembershipTypes = membershipTypes
```

Dupa ce ati efectuat aceste modificari, rulati aplicatia inregistrat ca si guest, si fortati accesarea pagii /Customers/New. Ce se intampla diferit?

Customizarea paginii de register

Daca dorim sa mai adaugam campuri, de exemplu in pagina de inregistrare user, vom urma pasii descrisi mai jos. Dorim sa adaugam un camp astfel incat la fiecare inregistrare de utilizator nou sa introducem si CNP-ul.



1. Deschidem Models->IdentityModel.cs	
2. Adaugam o noua proprietate denumita CNP	
3. Vom crea o noua migratie si vom updata baza de date folosind in consola comenzile din imaginea alaturata	
4. Daca in tabela nu s-a creat campul CNP vom efectua pasii din imaginea alaturata. Vom finaliza acest pas prin efectuarea unui Update in baza de date.	
5. Vom accesa view-ul Register din folderul Account pentru a adauga campul CNP si in pagina care se va randa la rulare	

6. Inserati codul din coloana alaturata in view-ul deschid, inainte de butonul de Submit

```
<div class="form-group">
        @Html.LabelFor(m => m.CNP, new {
@class = "col-md-2 control-label" })
        <div class="col-md-10">
            @Html.TextBoxFor(m => m.CNP, new {
@class = "form-control" })
        </div>
                                </div>
```

7. In acest moment CNP nu este o proprietate cunoscuta. Astfel, trebuie sa adaugam aceasta proprietate in clasa model RegisterViewModel

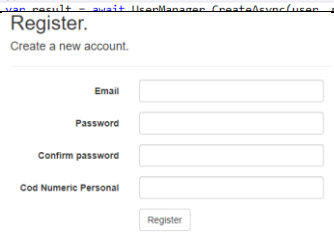
```
public class RegisterViewModel
{
    [Required]
    [StringLength(13)]
    [Display(Name ="Cod Numeric Personal")]
    public string CNP { get; set; }
```

8. In controller-ul Account trebuie sa adaugam in actiunea Register care are ca parametru un ViewModel aceasta noua proprietate.

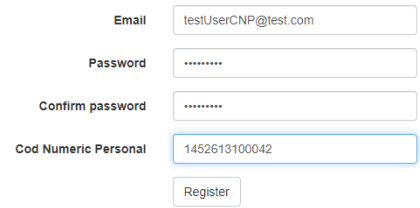
9. Adaugati in variabila user codul afisat in coloana 2 din tabel

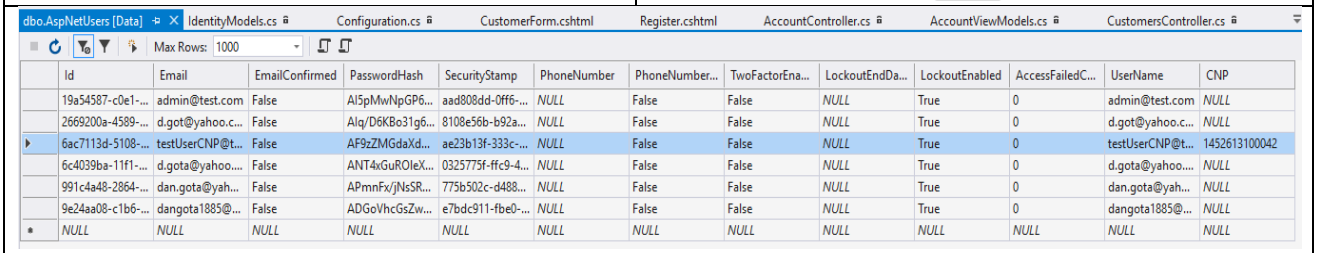
```
[ValidateAntiForgeryToken]
public async Task<ActionResult> Register(RegisterViewModel model)
{
    if (ModelState.IsValid)
    {
        var user = new ApplicationUser
        {
            UserName = model.Email,
            Email = model.Email,
            CNP=model.CNP
        };
        var result = await UserManager.CreateAsync(user, model.Pass
```

10. Rulati aplicatia si apasati butonul Register



11. Introduceti date de test si apasati butonul Register. Mai apoi accesati baza de date si afisati datele din tabela





Id	Email	EmailConfirmed	PasswordHash	SecurityStamp	PhoneNumber	PhoneNumber...	TwoFactorEna...	LockoutEndDa...	LockoutEnabled	AccessFailedC...	UserName	CNP
19a54587-c0e1-...	admin@test.com	False	AI5pMwNpGP6...	aad808dd-0ff6-...	NULL	False	False	NULL	True	0	admin@test.com	NULL
2669200a-4589-...	d.got@yahoo.c...	False	AIq/D6KBo31g6...	8108e56b-b92a...	NULL	False	False	NULL	True	0	d.got@yahoo.c...	NULL
6ac7113d-5108-...	testUserCNP@t...	False	AF9zZMGdaXd...	ae23b13f-333c-...	NULL	False	False	NULL	True	0	testUserCNP@t...	1452613100042
6c4039ba-11f1-...	d.gota@yahoo....	False	ANT4xGuROIeX...	0325775f-ffc9-4...	NULL	False	False	NULL	True	0	d.gota@yahoo....	NULL
991c4a48-2864-...	dan.gota@yah...	False	APmnFx/jNsSR...	775b502c-d488...	NULL	False	False	NULL	True	0	dan.gota@yah...	NULL
9e24aa08-c1b6-...	dangota1885@...	False	ADGoVhcGsZw...	e7bdc911-fbe0-...	NULL	False	False	NULL	True	0	dangota1885@...	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Logarea in aplicatie utilizand retelele sociale

Facebook si alte aplicatii de socializare folosesc ca protocol de autentificare OAuth(Open Authorisation). Procesul de autentificare folosing userul de Facebook este descris in imaginea de mai jos:

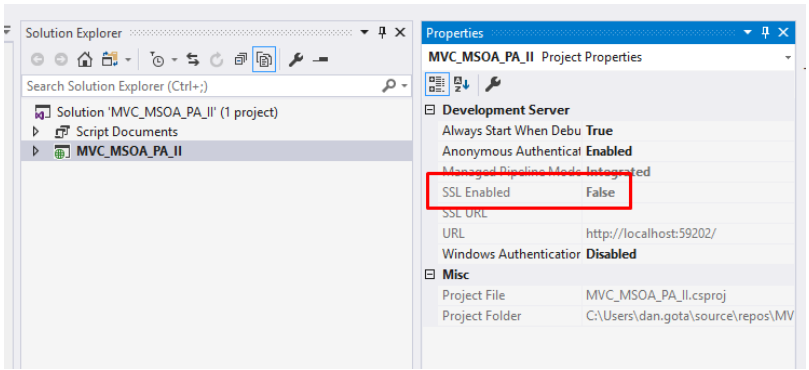
Primul lucru pentru a implementa acest mecanism este de a inregistra aplicatia noastra in Facebook. Astfel se va crea un parteneriat intre aplicatie si platforma de socializare. Aceast lucru va face disponibil accesul printr-un API la datele userului care doreste sa se logheze in aplicatia noastra.

Cand un user va incerca sa se logheze in Aplicatie cu datele de Facebook, se va efectua un call in care aplicatia va trimite catre Facebook cheia si codul secretul pus la dispozitie in momentul inregistrarii aplicatiei. Facebook la randul lui va permite userului sa se legheze cu userul de Facebook in platforma. Daca logarea este cu succes, va trimite catre aplicatia dezvoltata de noi un cod de autorizare. Pentru a ne asigura de faptul ca nu s-a produs o injectare de cod de autorizare de catre un hacker, pasul 3 este de a trimite catre Facebook cheia, codul secret si codul de autorizare. Pasul final este ca Facebook sa retrimita catre aplicatie un access token prin care noi putem accesa datele permise despre utilizatorul logat in Facebook.

Asadar, pentru a putea implementa logarea cu conturile de social media, avem nevoie sa permitem comunicarea securizata (SSL) si sa inregistram aplicatia dezvoltata in Facebook.

Implementarea SSL

1. Click pe solutie si apasati mai apoi tasta F4



Setati pe TRUE proprietatea SSL Enabled. Daca aceasta optiune nu este permisa,

inseamna ca nu aveti instalat pe statia de lucru un certificat. Pentru a genera si intala certificatul urmati pasii descrisi mai jos.

Crearea unui certificat SSL temporar

Pentru crearea unui certificat temporar, deschideti command prompt-ul (Start-> cmd(rulat ca administrator)), navigati in partitia C:\\ si rulati urmatoarea comanda

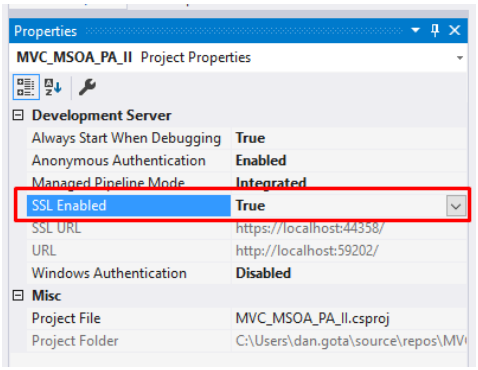
makecert.exe -n "CN=Development CA" -r -sv TempCA.pvk TempCA.cer

Odata creat, certificatul o sa fie disponibil pe calculator la locatia specificata in consola.Urmatorul pas este sa instalam acest certificat.

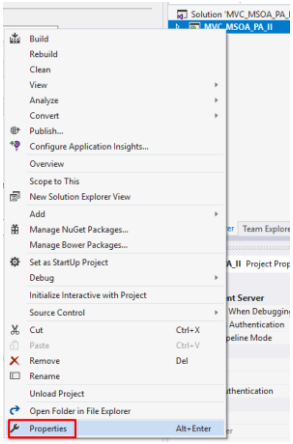
Instalarea certificatului SSL

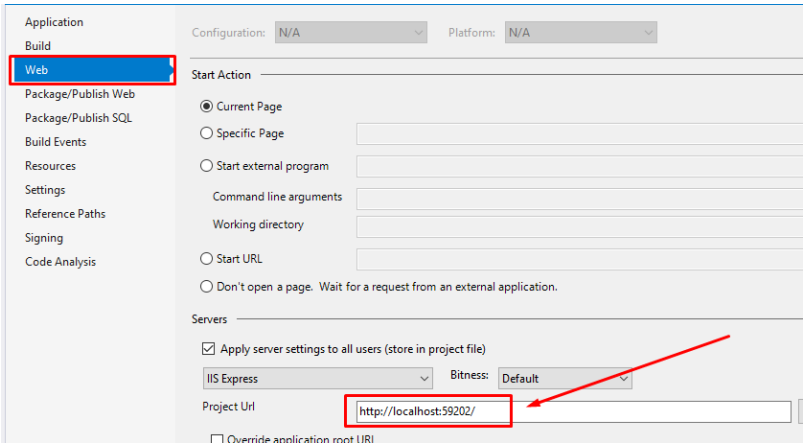
La Start scrieti MMC si deschideti consola (Management console)
Accesati optiunile File - > Add or Remove Snap Ins
Selectati Certificates din optiunile disponibile
Click pe butonul ADD
Selectati Computer account din fereastra care se deschide
Selectati Local Computer Account
Click pe optiunea Next si apoi OK
In acest moment certificatul este adaugat in consola de management de certificate(MMC).
In acest moment trebuie sa efectuam pasii de instalare a certificatului

Selectati Certificates si deschideti-l prin dublu click.
Selectati “Trusted root certification Authorities”
Selectati Action - > All Tasks - > Imports
Selectati certificatul din calea specificata in pasul de creare certificat.
In acest moment avem un certificat SSL instalat pe statia de lucru si putem seta in aplicatia Visual Studio Enable SSL.



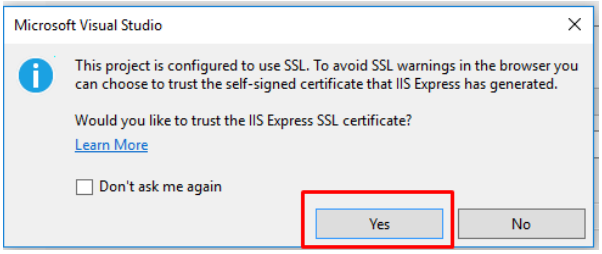
In aceeasi pagina de proprietati avem la dispozitie noul SSL URL pentru aplicati. Copiati aceasta valoare iar mai apoi accesati proiectul nostru, efectuati click dreapta, selectati Properties.

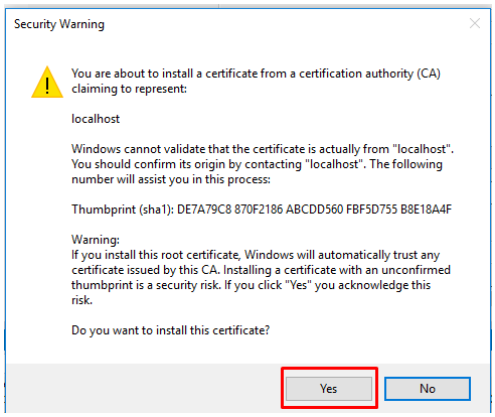




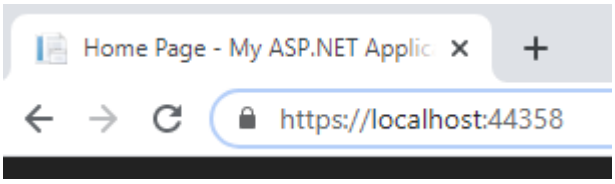
Copiati valoare SSL URL in campul Project Url.

Salvati fisierul si rulati aplicatia. La aparitia ferestrei de mai jos selectati optiunea Yes



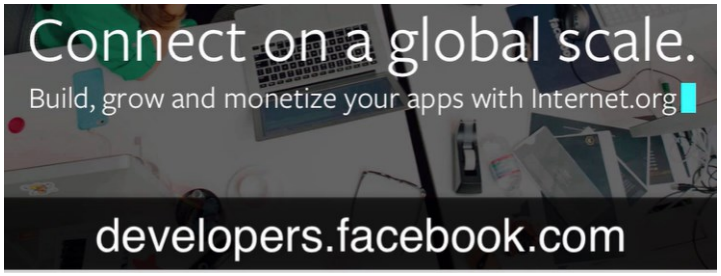


Se va observa faptul ca noul Url al aplicatiei este securizat:

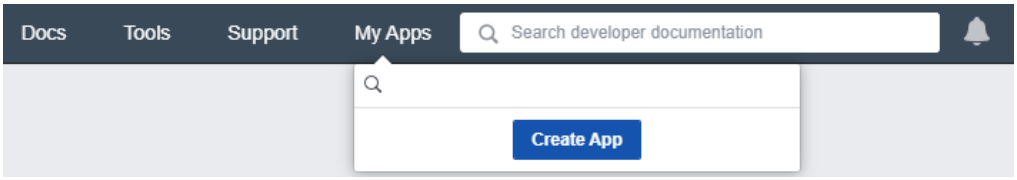


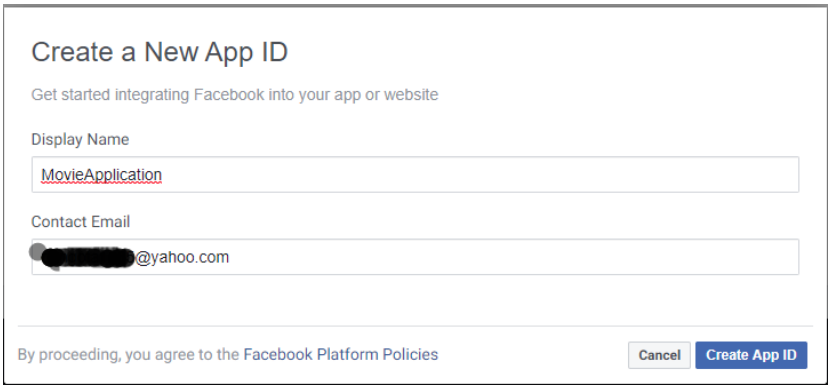
Inregistrarea aplicatiei pe platforma Facebook

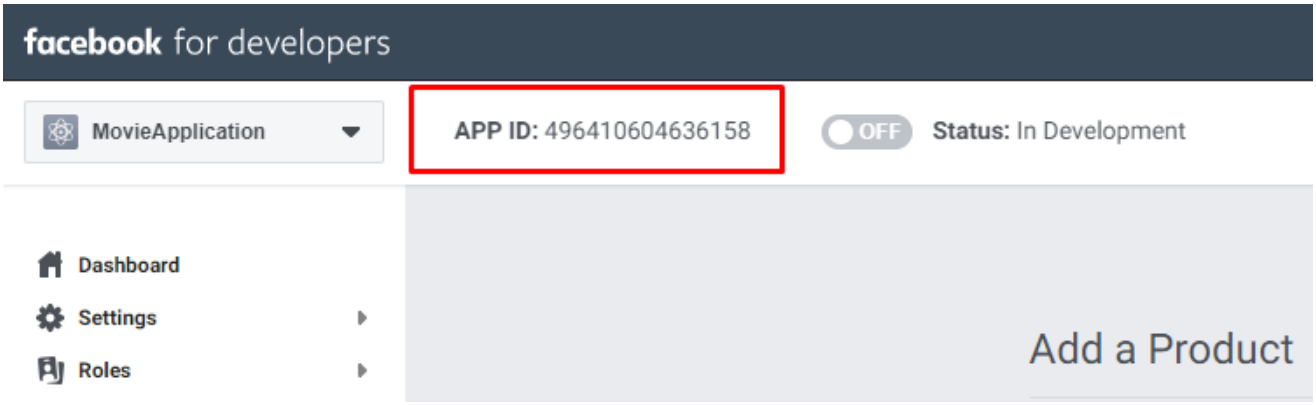
Pentru a inregistra aplicatia in platforma facebook trebuie sa accesam developers.facebook.com, sa ne cream un cont dupa care sa adaugam aplicatia.



Pentru a adauga aplicatia in platforma vom crea o noua aplicatie pe facebook developers prin accesarea optiunii din imaginea de mai jos:

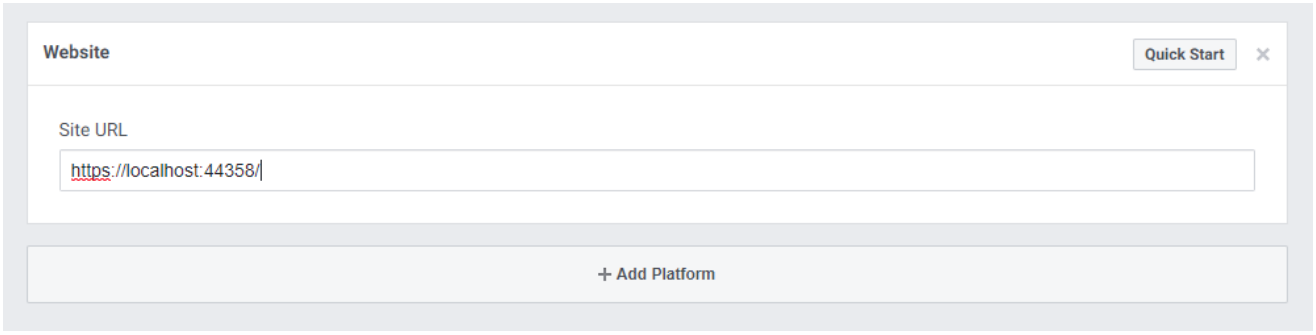




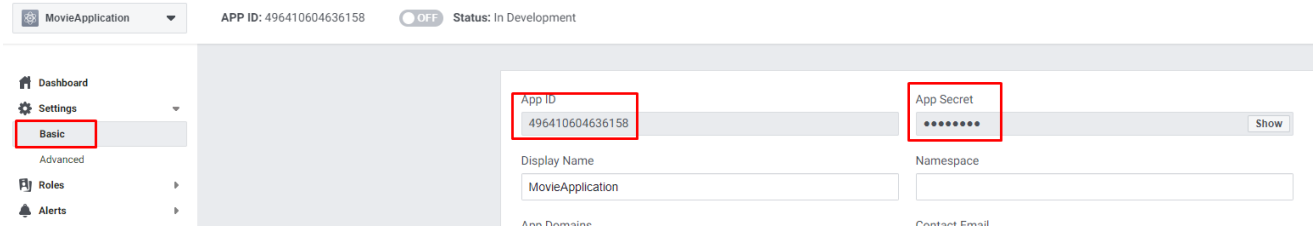


In imeaginea de mai sus se poate observa APP ID-ul asignat pentru aplicatia noastra.

In setarile de baza suntem nevoiti sa adaugam o noua platforma(Website) unde specificam URL-ul site-ului la fel ca in imaginea urmatoare:

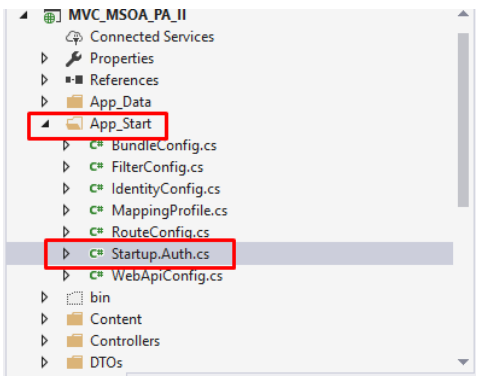


In imaginea de mai jos avem disponibil si App Secret(codul secret). Fereastra se poate accesa prin navigarea in Setarile de baza din meniul de navigare din stanga ecranului.



Cu aceste doua coduri vom merge in aplicatia MVC creata, in sectiunea App_Start si vom accesa clasa StartUp.Auth.cs care va contine linii de cod care vor integra aceste doua

coduri secrete.

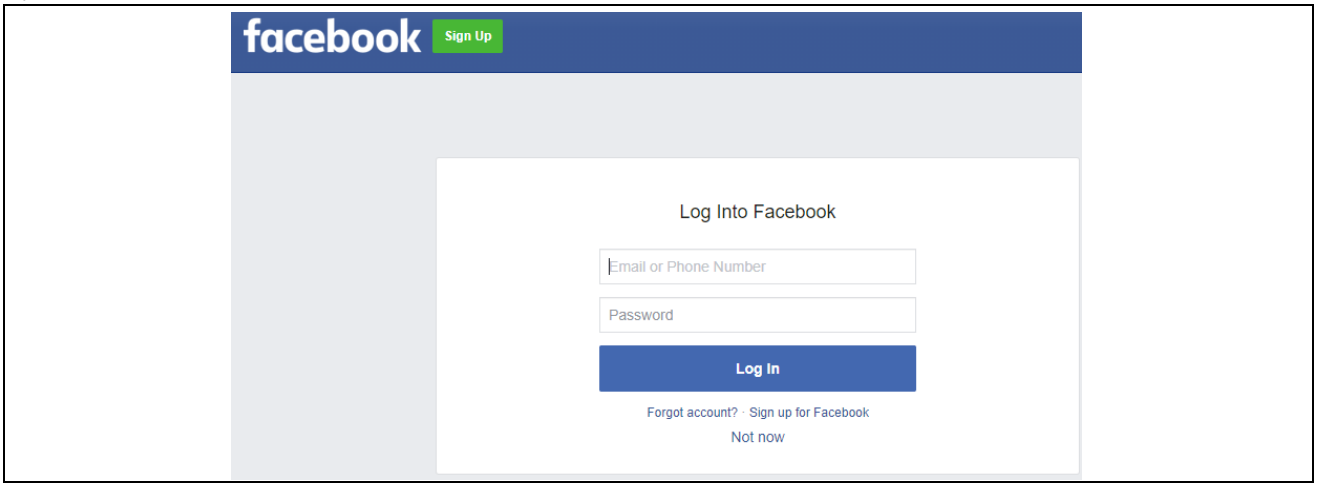


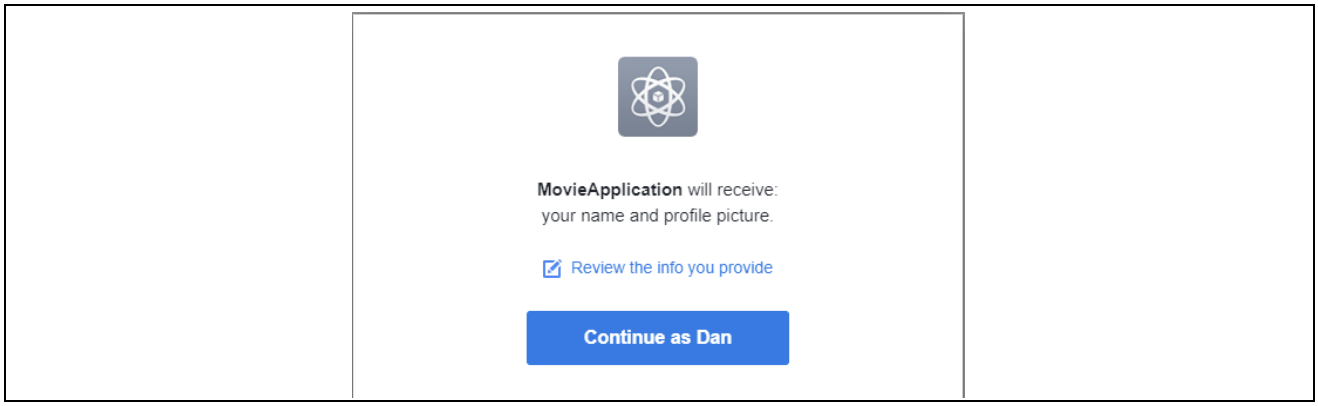

```
56
57          //app.UseFacebookAuthentication(
58          //   appId: "",
59          //   appSecret: "");
```

Dupa completarea datelor, daca vom rula aplicatia vom avea urmatoarele rezultate:

```
57          app.UseFacebookAuthentication(
58             appId: "496410604636158",
59             appSecret: "[redacted]f4ccba4fca73e5e02da73b206");
60
```

Efectuand click pe butonul Facebook o sa fim redirectionati catre ecranul de logare in aplicatia Facebook.





In acest moment o sa fiti redirectat catre site-ul creat unde o sa fie necesar sa introduceti adresa de email, moment in care logarea cu Facebook este cu succes.