

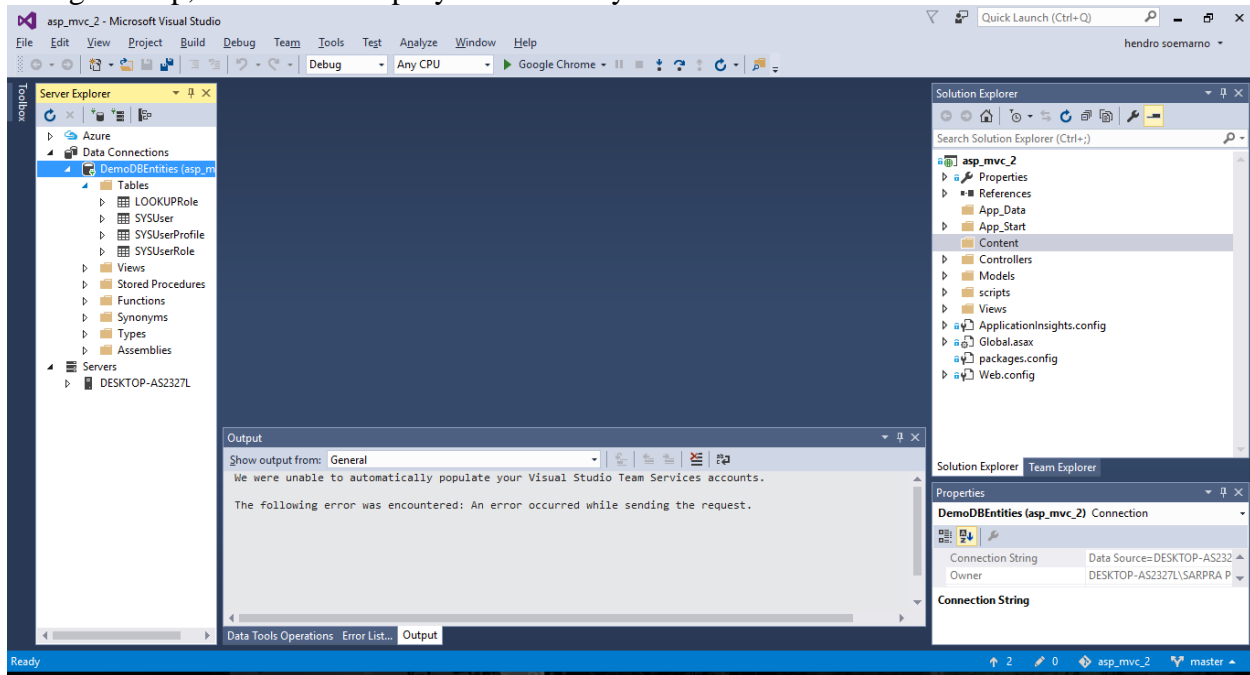
Membangun Aplikasi Web Menggunakan Entity Framework dan MVC 5: Bagian 2

Ringkasan Form Authentication

Keamanan merupakan bagian yang menyatu dari setiap aplikasi berbasis Web. Sebagian besar situs web saat ini sangat bergantung pada otentikasi dan otorisasi untuk mengamankan aplikasi mereka. Anda dapat menganggap situs web sebagai analogi ke kantor perusahaan di mana kantor terbuka untuk orang-orang seperti pelamar atau utusan untuk datang, tetapi ada bagian-bagian tertentu dari fasilitas, seperti workstation dan ruang konferensi, yang hanya bisa diakses oleh orang dengan mandat tertentu, seperti karyawan. Contohnya adalah ketika Anda membangun aplikasi OL Shop yang menerima informasi kartu kredit pengguna untuk tujuan pembayaran dan menyimpannya ke database Anda; ASP.NET membantu melindungi database Anda dari akses publik dengan menyediakan mekanisme otentikasi dan otorisasi.

Bentuk otentikasi memungkinkan Anda mengotentikasi pengguna dengan menggunakan kode Anda sendiri dan kemudian mempertahankan token otentikasi dalam cookie atau di halaman URL. Untuk menggunakan bentuk otentikasi, Anda membuat halaman login yang mengumpulkan kredensial dari pengguna dan itu termasuk kode untuk mengotentikasi kredensial. Biasanya Anda mengkonfigurasi aplikasi untuk mengarahkan permintaan ke halaman login ketika pengguna mencoba untuk mengakses sumber daya yang dilindungi, seperti halaman yang memerlukan otentikasi. Jika kredensial pengguna yang valid, Anda dapat memanggil metode dari kelas FormsAuthentication untuk mengarahkan permintaan kembali ke source awalnya diminta dengan tiket otentikasi yang sesuai (cookie).

Sebagai rekap, inilah struktur proyek sebelumnya di bawah ini:



Implementasi Login Page

LANGKAH 1: Enabling Forms Authentication

Untuk memungkinkan bentuk otentikasi dalam aplikasi Anda, hal pertama yang harus dilakukan adalah untuk mengkonfigurasi FormsAuthentication yang mengelola layanan bentuk otentikasi untuk aplikasi web Anda. Modus otentikasi default untuk ASP.NET adalah "windows". Untuk mengaktifkan otentikasi bentuk, tambahkan otentikasi dan bentuk elemen di bawah ini elemen system.web di web.config Anda:

```
<system.web>
  <authentication mode="Forms">
    <forms loginUrl="/Account/Login"
           defaultUrl="/Home/Welcome">
    </forms>
  </authentication>
</system.web>
```

Mengatur loginUrl memungkinkan aplikasi untuk menentukan di mana untuk mengarahkan pengguna un-dikonfirmasi yang mencoba untuk mengakses halaman dijamin. The defaultUrl mengarahkan pengguna ke halaman tertentu setelah berhasil log-in.

LANGKAH 2: Menambah Model UserLoginView

Mari kita pergi ke depan dan menciptakan model tampilan kelas untuk halaman Login, dengan menambahkan kode berikut di bawah ini dalam kelas "UserModel":

```
public class UserLoginView
{
    [Key]
    public int SYSUserID { get; set; }
    [Required(ErrorMessage = "*")]
    [Display(Name = "Login ID")]
    public string LoginName { get; set; }
    [Required(ErrorMessage = "*")]
    [DataType(DataType.Password)]
    [Display(Name = "Password")]
    public string Password { get; set; }
}
```

Anda juga dapat melihat bahwa field diberi atribut Required, Display and DataType. atribut ini disebut Data Annotations.

LANGKAH 3: Adding the GetUserPassword() Method

Tambahkan kode berikut di bawah ini di bawah "UserManager.cs" class:

```
public string GetUserPassword(string loginName) {
    using (DemoDBEntities db = new DemoDBEntities()) {
        var user = db.SYSUsers.Where(o =>
o.LoginName.ToLower().Equals(loginName));
        if (user.Any())
            return user.FirstOrDefault().PasswordEncryptedText;
        else
            return string.Empty;
    }
}
```

Password yang sesuai dari database untuk nama login tertentu menggunakan query LINQ.

LANGKAH 4: Adding the Login Action Methods

Tambahkan kode berikut di bawah ini di bawah kelas "AccountController":

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

[HttpPost]
public ActionResult LogIn(UserLoginView ULV, string returnUrl) {
    if (ModelState.IsValid) {
        UserManager UM = new UserManager();
        string password = UM.GetUserPassword(ULV.LoginName);

        if (string.IsNullOrEmpty(password))
            ModelState.AddModelError("", "The user login or password provided
is incorrect.");
        else {
            if (ULV.Password.Equals(password)) {
                FormsAuthentication.SetAuthCookie(ULV.LoginName, false);
                return RedirectToAction("Welcome", "Home");
            }
            else {
                ModelState.AddModelError("", "The password provided is
incorrect.");
            }
        }
    }

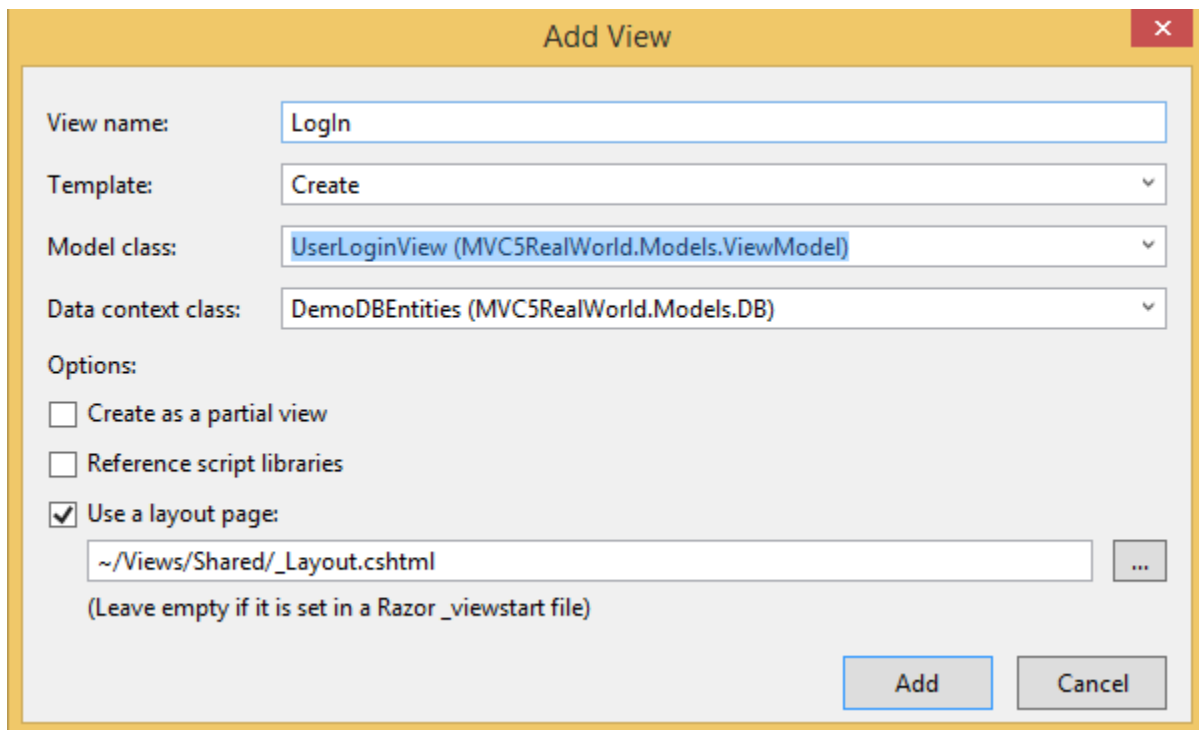
    // If we got this far, something failed, redisplay form
    return View(ULV);
}
```

Seperti yang Anda lihat ada dua metode di atas dengan nama yang sama. Yang pertama adalah metode "Login" yang hanya mengembalikan tampilan LogIn.cshtml. Tampilan ini ada pada langkah berikutnya. Yang kedua juga disebut sebagai "Login" tapi itu diberi atribut "[HttpPost]". Atribut ini menentukan kelebihan dari metode "Login" yang dapat dipanggil hanya untuk permintaan POST.

Metode kedua akan dipicu setelah tombol "Login" ditekan. Apa itu, pertama akan memeriksa apakah field yang diperlukan disediakan sehingga memeriksa kondisi ModelState.IsValid. Ini akan membuat sebuah instance dari kelas UserManager dan memanggil metode GetUserPassword () dengan memberikan value yang telah diberikan oleh pengguna. Jika password mengembalikan sebuah string kosong maka akan menampilkan kesalahan ke tampilan. Jika password yang diberikan adalah sama dengan password diambil dari database maka akan mengarahkan pengguna ke halaman Selamat datang, jika tidak menampilkan kesalahan yang menyatakan bahwa nama login dan password yang diberikan tidak valid.

LANGKAH 5: Menambah Login View

Sebelum menambahkan view, pastikan untuk membangun aplikasi pertama bebas dari kesalahan. Setelah sukses build, arahkan ke class "AccountController" dan klik kanan pada metode Login action dan kemudian pilih "Add View". Ini akan memunculkan dialog berikut di bawah ini:



Sekarang klik pada Add. Berikut modifikasi HTML:

```
@model Asp_mvc_2.Models.ViewModel.UserLoginView

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

```
<h2>LogIn</h2>
```

```
@using (Html.BeginForm())
```

```
{
```

```
    @Html.AntiForgeryToken()
```

```
    <div class="form-horizontal">
```

```
        <hr />
```

```
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
```

```
        <div class="form-group">
```

```
            @Html.LabelFor(model => model.LoginName, htmlAttributes: new { @class =  
"control-label col-md-2" })
```

```
            <div class="col-md-10">
```

```
                @Html.EditorFor(model => model.LoginName, new { htmlAttributes = new  
{ @class = "form-control" } })
```

```
                @Html.ValidationMessageFor(model => model.LoginName, "", new { @class  
= "text-danger" })
```

```
            </div>
```

```
        </div>
```

```
        <div class="form-group">
```

```
            @Html.LabelFor(model => model.Password, htmlAttributes: new { @class =  
"control-label col-md-2" })
```

```
            <div class="col-md-10">
```

```
                @Html.EditorFor(model => model.Password, new { htmlAttributes = new {  
@class = "form-control" } })
```

```
                @Html.ValidationMessageFor(model => model.Password, "", new { @class  
= "text-danger" })
```

```
            </div>
```

```
        </div>
```

```
        <div class="form-group">
```

```
            <div class="col-md-offset-2 col-md-10">
```

```
                <input type="submit" value="Login" class="btn btn-default" />
```

```
            </div>
```

```
        </div>
```

```
</div>
```

```
}

<div>
    @Html.ActionLink("Back to Main", "Index", "Home")
</div>
```

LANGKAH 6: Implementing the Logout Functionality

Kode logout cukup mudah. Hanya menambahkan metode berikut di bawah ini di kelas AccountController.

```
[Authorize]
public ActionResult SignOut() {
    FormsAuthentication.SignOut();
    return RedirectToAction("Index", "Home");
}
```

Metode FormsAuthentication.SignOut menghilangkan tiket otentikasi dari browser. Lalu mengarahkan pengguna ke halaman Index setelah sign out.

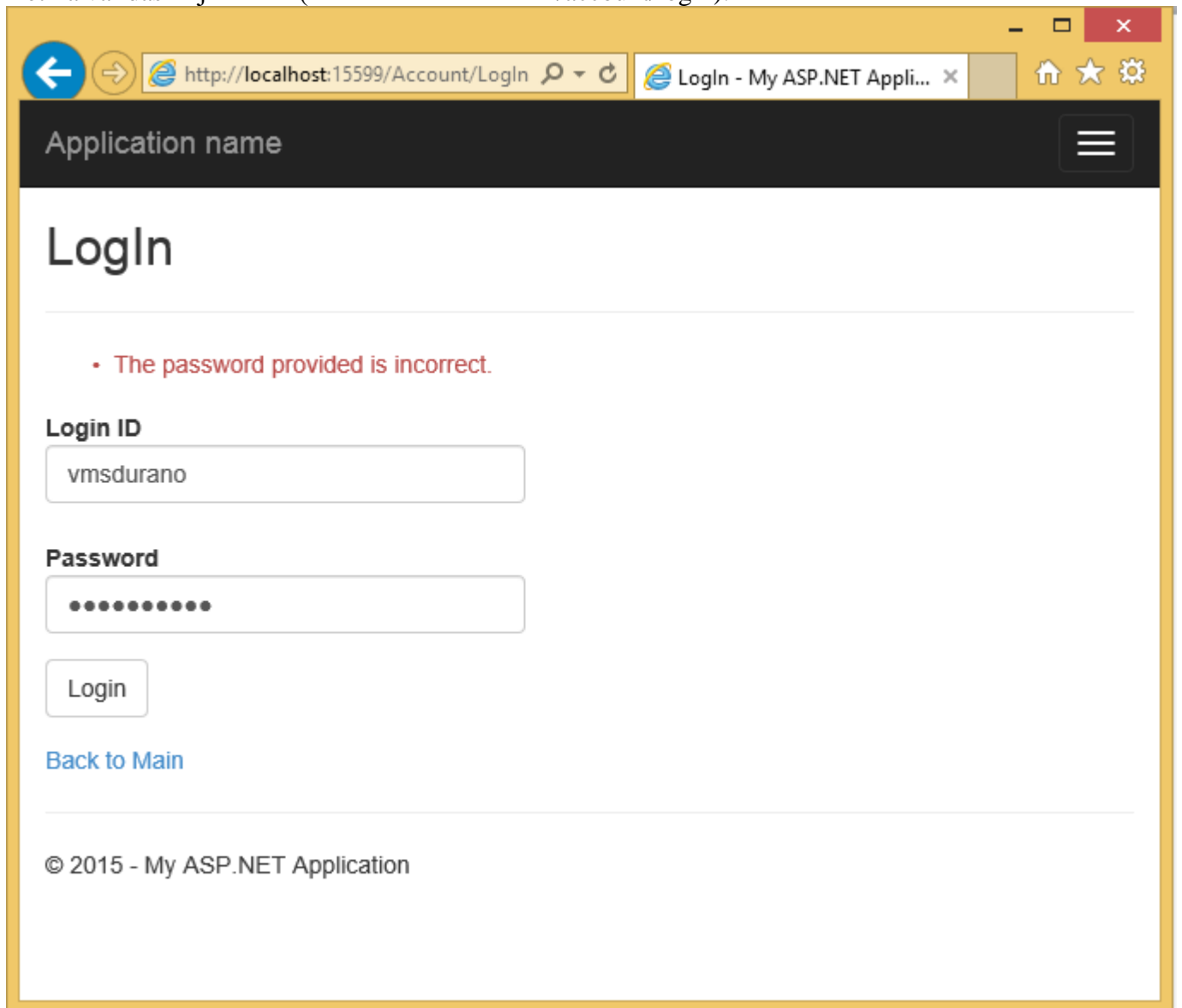
Berikut tindakan tautan yang sesuai untuk Keluar:

```
@Html.ActionLink("Signout", "SignOut", "Account")
```

LANGKAH 7: Running the Application

Menjalankan aplikasi Anda harus menampilkan sesuatu seperti ini:

Ketika validasi dijalankan (alamat: localhost:...../account/login):



The screenshot shows a web browser window with a yellow border. The address bar displays 'http://localhost:15599/Account/Login'. The page title is 'Login - My ASP.NET Appli...'. The page content includes a dark header with 'Application name' and a hamburger menu icon. The main heading is 'Login'. A red error message states: '• The password provided is incorrect.' Below this, there are two input fields: 'Login ID' containing 'vmsdurano' and 'Password' containing ten dots. A 'Login' button is positioned below the password field. A blue link 'Back to Main' is located below the login button. At the bottom, the footer text reads '© 2015 - My ASP.NET Application'.

Application name

Login

- The password provided is incorrect.

Login ID

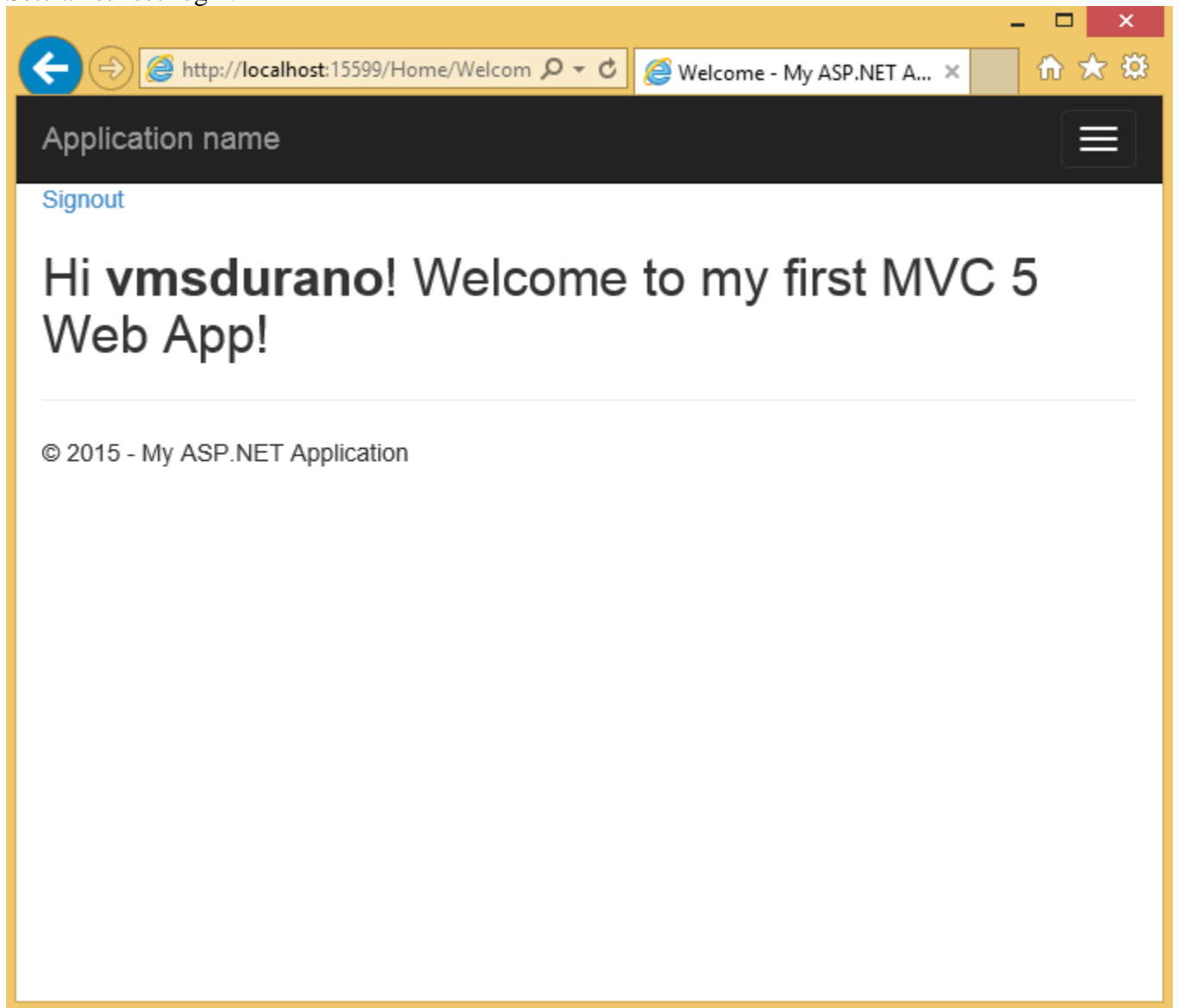
Password

Login

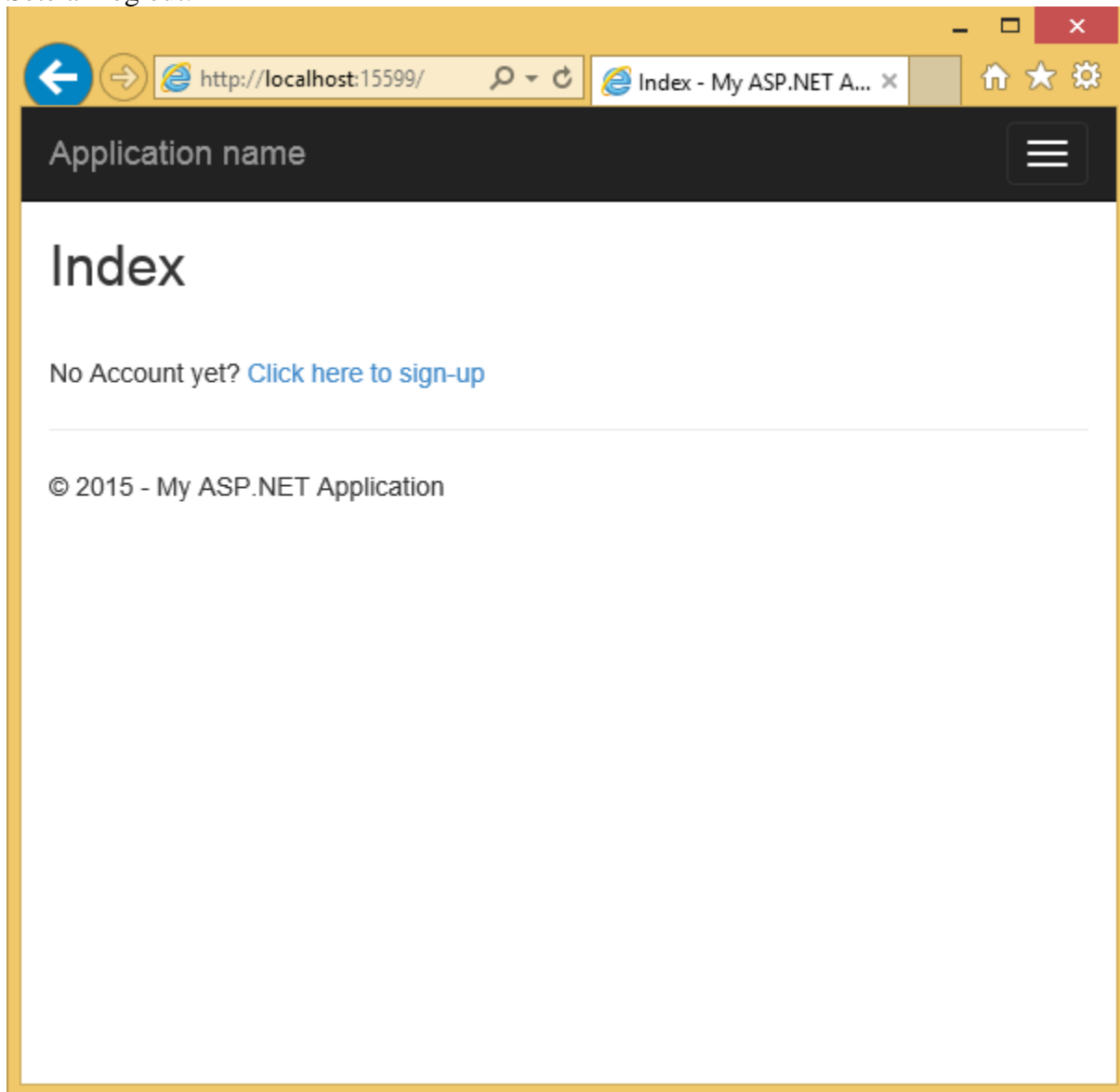
[Back to Main](#)

© 2015 - My ASP.NET Application

Setelah sukses login:



Setelah log out:



Sesederhana itu! Sekarang mari kita lihat bagaimana menerapkan role-based page authorization sederhana.

Implementasi Role-Based Page Authorization Sederhana

Authorization adalah fungsi yang menentukan hak akses ke halaman. Salah satu contoh praktis adalah memiliki halaman yang hanya peran pengguna tertentu dapat memiliki akses ke sana. Misalnya, hanya memungkinkan administrator untuk mengakses halaman maintenance untuk aplikasi Anda. Pada bagian ini kita akan membuat implementasi sederhana tentang cara untuk mencapai itu.

LANGKAH 1: Membuat Method `UserRoleInRole`

Tambahkan kode berikut di bawah ini di kelas "userManager":

```
public bool IsUserInRole(string loginName, string roleName) {
    using (DemoDBEntities db = new DemoDBEntities()) {
        SYSUser SU = db.SYSUsers.Where(o =>
o.LoginName.ToLower().Equals(loginName)).FirstOrDefault();
        if (SU != null) {
            var roles = from q in db.SYSUserRoles
                        join r in db.LOOKUPRoles on q.LOOKUPRoleID equals
r.LOOKUPRoleID
                        where r.RoleName.Equals(roleName) &&
q.SYSUserID.Equals(SU.SYSUserID)
                        select r.RoleName;

            if (roles != null) {
                return roles.Any();
            }
        }

        return false;
    }
}
```

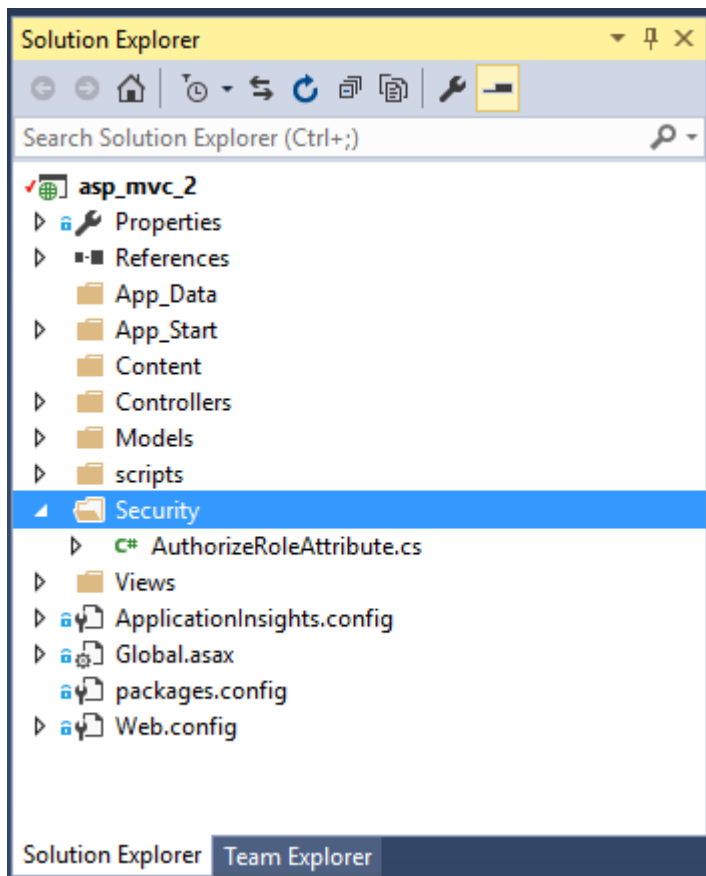
metode di atas mengambil loginname dan RoleName sebagai parameter. Dilakukannya untuk memeriksa catatan yang ada dalam tabel SYSUser dan kemudian memvalidasi jika pengguna yang sesuai dengan peran yang ditugaskan untuk itu.

LANGKAH 2: Membuat Filter Atribut Authorization yang dapat diubah

Jika Anda ingat kita menggunakan [Authorize] atribut untuk membatasi pengguna anonim dari mengakses metode tindakan tertentu. [Authorize] atribut menyediakan filter untuk pengguna dan peran dan itu cukup mudah untuk menerapkannya jika Anda menggunakan membership provider. Karena menggunakan database sendiri untuk menyimpan pengguna dan peran, kita perlu menerapkan filter otorisasi kita sendiri dengan memperluas kelas AuthorizeAttribute.

AuthorizeAttribute menetapkan bahwa akses ke controller atau tindakan metode dibatasi untuk pengguna yang memenuhi persyaratan otorisasi. Tujuan kami di sini untuk memungkinkan halaman otorisasi berdasarkan peran pengguna dan tidak ada yang lain. Jika Anda ingin menerapkan filter kustom untuk melakukan tugas tertentu dan menggunakan pemisahan konsentrasi, maka lihat IAuthenticationFilter.

Untuk memulai, tambahkan folder baru dan beri nama sebagai "Security". Kemudian tambahkan class "AuthorizeRoleAttribute". Berikut ini adalah screen shot dari struktur:



Dan inilah blok kode untuk filter kustom Anda:

```
using System.Web;
using System.Web.Mvc;
using Asp_mvc_2.Models.DB;
using Asp_mvc_2.Models.EntityManager;

namespace Asp_mvc_2.Security
{
    public class AuthorizeRolesAttribute : AuthorizeAttribute
    {
        private readonly string[] userAssignedRoles;
        public AuthorizeRolesAttribute(params string[] roles) {
```

```

        this.userAssignedRoles = roles;
    }
    protected override bool AuthorizeCore(HttpContextBase httpContext) {
        bool authorize = false;
        using (DemoDBEntities db = new DemoDBEntities()) {
            UserManager UM = new UserManager();
            foreach (var roles in userAssignedRoles) {
                authorize = UM.IsUserInRole(httpContext.User.Identity.Name,
roles);

                if (authorize)
                    return authorize;
            }
        }
        return authorize;
    }
    protected override void HandleUnauthorizedRequest(AuthorizationContext
filterContext) {
        filterContext.Result = new RedirectResult("~/Home/Unauthorized");
    }
}
}

```

Ada dua metode utama dalam kelas diatas yang telah di override. Method AuthorizeCore () adalah titik masuk untuk memeriksa otentikasi. Di sinilah kita memeriksa peran yang ditugaskan untuk pengguna tertentu dan mengembalikan hasilnya jika pengguna diijinkan untuk mengakses halaman atau tidak. The HandleUnauthorizedRequest () adalah metode di mana kita mengarahkan pengguna un-resmi ke halaman tertentu.

LANGKAH 3: Menambahkan halaman AdminOnly dan Unauthorized

Sekarang beralih kembali ke kelas "HomeController" dan tambahkan kode berikut:

```

[AuthorizeRoles("Admin")]
public ActionResult AdminOnly() {
    return View();
}

public ActionResult Unauthorized() {

```

```
        return View();  
    }
```

Jika Anda perhatikan, AdminOnly dengan otorisasi filter khusus dengan menggunakan value "Admin" sebagai nama method. Ini berarti bahwa hanya memungkinkan pengguna admin untuk mengakses halaman AdminOnly. Untuk mendukung akses beberapa peran, hanya menambahkan nama peran lain dengan memisahkannya dengan koma, misalnya [AuthorizeRoles ("Admin", "Manager")]. Perhatikan bahwa nilai "Admin" dan "manager" harus sesuai dengan nama-nama peran dari database Anda. Dan akhirnya, pastikan untuk referensi namespace di bawah ini sebelum menggunakan atribut AuthorizeRoles:

```
using Asp_mvc_2.Security;
```

Berikut tampilan AdminOnly.cshtml:

```
@{  
    ViewBag.Title = "AdminOnly";  
    Layout = "~/Views/Shared/_Layout.cshtml";  
}  
  
<h2>For Admin users only!</h2>
```

Dan inilah tampilan Unauthorized.cshtml:

```
@{  
    ViewBag.Title = "Unauthorized";  
    Layout = "~/Views/Shared/_Layout.cshtml";  
}  
  
<h2>Unauthorized Access!</h2>  
<p>Oops! You don't have permission to access this page.</p>  
  
<div>  
    @Html.ActionLink("Back to Main", "Welcome", "Home")  
</div>
```

LANGKAH 4: Testing the functionality

Sebelum kita menguji fungsionalitas memungkinkan menambahkan user admin ke database pertama. Untuk demo masukkan data berikut ke database:

```
INSERT INTO SYSUser (LoginName,PasswordEncryptedText, RowCreatedSYSUserID,
RowModifiedSYSUserID)
VALUES ('Admin','Admin',1,1)
GO

INSERT INTO SYSUserProfile (SYSUserID,FirstName,LastName,Gender,RowCreatedSYSUserID,
RowModifiedSYSUserID)
VALUES (2,'Vinz','Durano','M',1,1)
GO

INSERT INTO SYSUserRole (SYSUserID,LOOKUPRoleID,IsActive,RowCreatedSYSUserID,
RowModifiedSYSUserID)
VALUES (2,1,1,1,1)
```

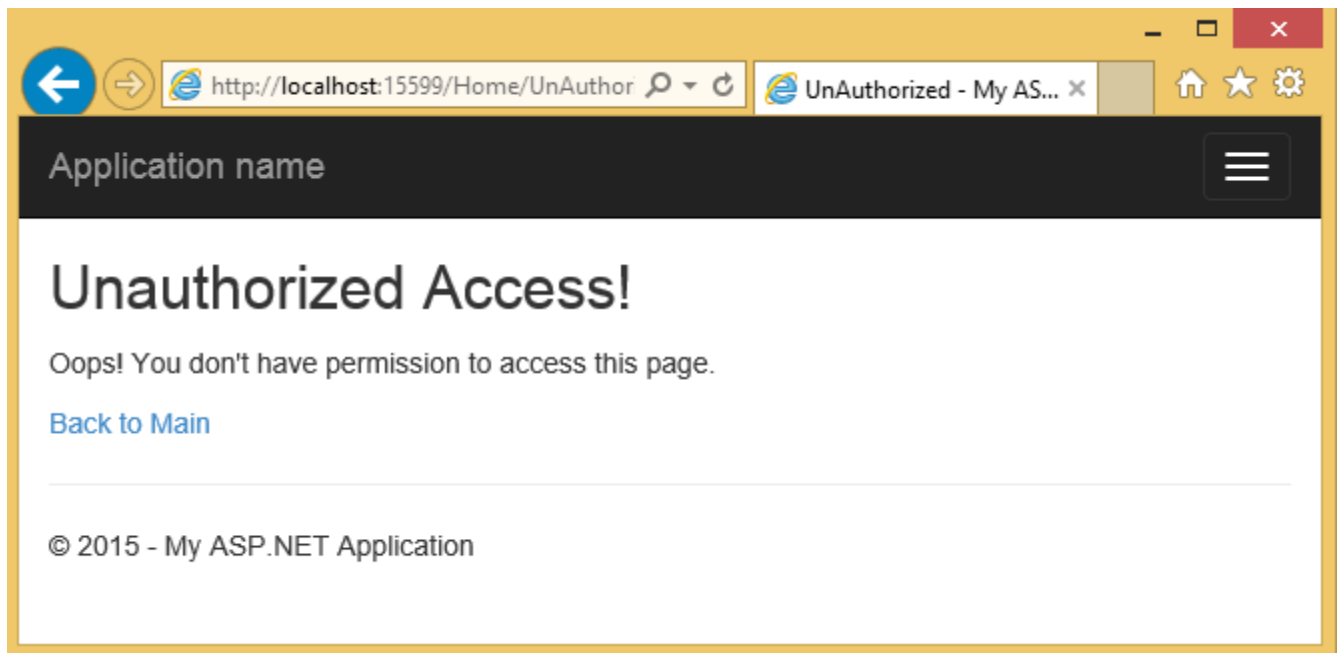
Data untuk menguji siap dan aplikasi siap dijalankan.

LANGKAH 5: Menjalankan Aplikasi

Berikut adalah beberapa screen shot diambil selama tes.

Ketika login sebagai user biasa dan mengakses berikut

URL: <http://localhost:15599/Home/AdminOnly>



Saat masuk sebagai pengguna Admin dan mengakses berikut
URL: <http://localhost:15599/Home/AdminOnly>

