.NET Framework: Developing Modern Web Apps with ASP.NET MVC – Workshop*PLUS*

Module 9: Security

Student Lab Manual

Instructor Edition (Book Title Hidden Style)

Version 1.0

**Conditions and Terms of Use**

**Microsoft Confidential**

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

© 2017 Microsoft Corporation. All rights reserved.

**Copyright and Trademarks**

© 2017 Microsoft Corporation. All rights reserved.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see Use of Microsoft Copyrighted Content at  
<http://www.microsoft.com/en-us/legal/intellectualproperty/Permissions/default.aspx>

Internet Explorer, Microsoft, Visual Studio, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

Contents

[Lab 9: Security 1](#_Toc480803687)

[Exercise 1: Implement Identity 3](#_Toc480803688)

[Task 1: Configure the app to use Identity 3](#_Toc480803689)

[Task 2: Tie Login and Register Methods to Identity 7](#_Toc480803690)

[Task 3: Modify the toolbar to display logged in user 10](#_Toc480803691)

[Task 4: Test the Identity Provider, Register, and Login functionality 11](#_Toc480803692)

[Task 5: Add more Password Requirement Rules 12](#_Toc480803693)

# Lab 9: Security

#### Introduction

In this lab, we are going to get back to adding functionality to MyShuttle, by implementing authentication. In order to do this, we will configure an identity provider, add the logic behind the Login and Register pages that we created in Module 7, and modify the main page to show which, if any, carrier is logged in.

#### Objectives

After completing this lab, you will be able to:

* Allow your web app to consume an identity provider for user authentication, and management.
* Complete the flow of user registration and login and query the authenticated user.
* Feedback model state errors to the view following server side validation failure.

#### Prerequisites (if applicable)

This lab will use the output from **Lab 8**. You can choose either to continue with the solution you ended up with following the previous lab, or start with the solution included in the folder **Labs\Module 09 - Security\Begin**. No additional SDKs or tools are required for this lab.

#### Scenario

So far, the website has not been able to login any user or authenticate them, and so we have not been able to secure any of our controller methods. In this lab, we will rectify that.

Carriers need to be able to register an account on the system, and then be able to login. Once logged in, their username will be displayed in the toolbar, and they will be given the option to logout.

#### System Requirements

To complete this lab, you need:

* Microsoft Visual Studio 2017
* Microsoft SQL Server (any edition)

#### Hosted Lab Credentials

If the lab is exercised in Microsoft cloud environment, use the following user credentials to sign in:

* Username: aspnetuser
* Password: @Cir9hvc6!w

#### Estimated time to complete this lab

40 minutes

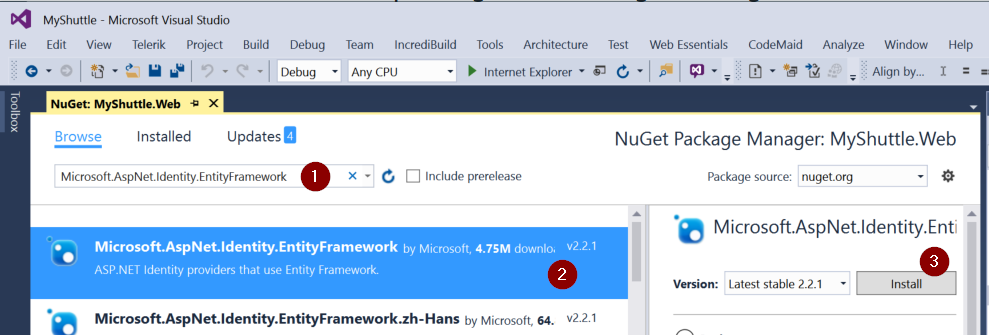
## Exercise 1: Implement Identity

#### Objectives

In this exercise, you will:

* Complete the lab objectives and implement the scenario mentioned earlier.

### Task 1: Configure the app to use Identity

1. Open nuget package manager and add the following packages:
   1. Microsoft.AspNet.Identity.EntityFramework
   2. Microsoft.AspNet.Identity.Owin
   3. Microsoft.Owin.Host.SystemWeb  
      
2. In previous lab we created **MyShuttleContext** class in **MyShuttle.Data** which we will use it as a DbContext for the ASP.NET Identity
3. In a previous lab we create **ApplicationUser** in **MyShuttle.Model** which represents the Identity User for the application
4. We need to create the user manager class for ASP.NET Identity, right click the **App\_Start** folder under **MyShuttle.Web** project and add new class with name **MyShuttleUserManager** and replace the contents with the following:  
   Reference file: Assets\MyShuttle\App\_Start\MyShuttleUserManager.cs

using Microsoft.AspNet.Identity;

using Microsoft.AspNet.Identity.EntityFramework;

using Microsoft.AspNet.Identity.Owin;

using Microsoft.Owin;

using MyShuttle.Data;

using MyShuttle.Model;

namespace MyShuttle.Web

{

public class MyShuttleUserManager : UserManager<ApplicationUser>

{

public MyShuttleUserManager(IUserStore<ApplicationUser> store) : base(store)

{

}

public static MyShuttleUserManager Create(IdentityFactoryOptions<MyShuttleUserManager> options, IOwinContext context)

{

var manager = new MyShuttleUserManager(new UserStore<ApplicationUser>(context.Get<MyShuttleContext>()));

return manager;

}

}

}

1. **MyShuttleUserManager** will be used to create users and set passwords policies later.
2. We need to create the sign in manager for ASP.NET Identity, right click the App\_Start folder under MyShuttle.Web project and add new class with name MyShuttleSignInManager and replace the contents with the following:  
   Reference file: Assets\MyShuttle\App\_Start\MyShuttleSignInManager.cs

using Microsoft.AspNet.Identity;

using Microsoft.AspNet.Identity.Owin;

using Microsoft.Owin;

using Microsoft.Owin.Security;

using MyShuttle.Model;

namespace MyShuttle.Web

{

public class MyShuttleSignInManager : SignInManager<ApplicationUser, string>

{

public MyShuttleSignInManager(UserManager<ApplicationUser, string> userManager, IAuthenticationManager authenticationManager) : base(userManager, authenticationManager)

{

}

public static MyShuttleSignInManager Create(IdentityFactoryOptions<MyShuttleSignInManager> options, IOwinContext context)

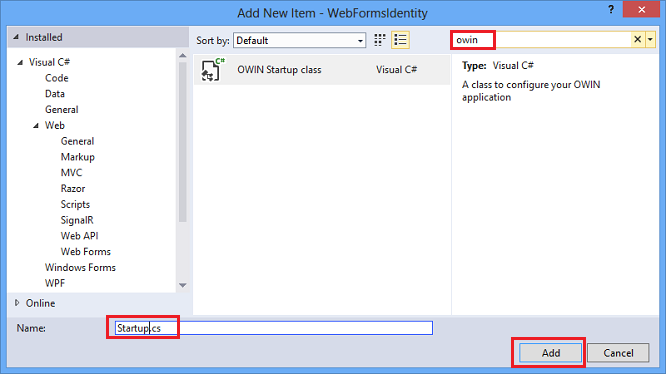
{

return new MyShuttleSignInManager(context.GetUserManager<MyShuttleUserManager>(), context.Authentication);

}

}

}

1. MyShuttleSignInManager will be used to sign in the user in the ASP.NET Identity.
2. Now, we are going to add authentication to login a user. ASP.NET Identity uses Microsoft OWIN Authentication middleware for forms authentication. The OWIN Cookie Authentication is a cookie and claims based authentication mechanism that can be used by any framework hosted on OWIN or IIS. It depends on two nuget package which we added in step 1:
   1. The **Microsoft.Aspnet.Identity.Owin** package contains a set of OWIN extension classes to manage and configure OWIN authentication middleware to be consumed by ASP.NET Identity Core packages.
   2. The **Microsoft.Owin.Host.SystemWeb** package contains an OWIN server that enables OWIN-based applications to run on IIS using the ASP.NET request pipeline.
3. To add OWIN Startup and Authentication Configuration Classes, In **Solution Explorer**, right-click your **MyShuttle.Web** project , click **Add**, and then **Add New Item**. In the search text box dialog, type "*owin*". Name the class "*Startup*" and click **Add**
4. Inside the Configuration method add the following code

app.CreatePerOwinContext(() => new Data.MyShuttleContext());

app.CreatePerOwinContext<MyShuttleUserManager>(MyShuttleUserManager.Create);

app.CreatePerOwinContext<MyShuttleSignInManager>(MyShuttleSignInManager.Create);

app.UseCookieAuthentication(new Microsoft.Owin.Security.Cookies.CookieAuthenticationOptions

{

AuthenticationType = Microsoft.AspNet.Identity.DefaultAuthenticationTypes.ApplicationCookie,

LoginPath = new PathString("/Carrier/Login"),

});

1. This class contains the **OwinStartup** attribute for specifying the OWIN startup class. Every OWIN application has a startup class where you specify components for the application pipeline.   
   The code we added is to configure OWIN to use our context and use of the cookies authentication for our application.  
   Notice in the **CookieAuthenticationOptions** we configure it to use the URL “/Carrier/Login” for the login URL.  
   Reference File: Assets\MyShuttle\Startup.cs

### Task 2: Tie Login and Register Methods to Identity

1. Open the **CarrierController** class in the same project, and locate the **Login** method that is called for **HttpPost** requests. Here we will use the sign-in manager to do all the hard work for us: validate the user's password, act on the **remember me** check box and let us know if the login was successful or not.
2. Add the following member **properties to the class**:

public MyShuttleSignInManager SignInManager { get; } = System.Web.HttpContext.Current.GetOwinContext().Get<MyShuttleSignInManager>();

public MyShuttleUserManager UserManager { get; } = System.Web.HttpContext.Current.GetOwinContext().Get<MyShuttleUserManager>();

1. Resolve the dependencies on ApplicationUser by adding the following using statement if not exists:

using Microsoft.AspNet.Identity;

using MyShuttle.Model;

using Microsoft.AspNet.Identity.Owin;

1. In the **Login** method with **HttpPost** replace the code inside the method with:

if (!ModelState.IsValid)

{

return View(model);

}

var signInStatus = await SignInManager.PasswordSignInAsync(model.UserName, model.Password, model.RememberMe, false);

if (signInStatus == SignInStatus.Success)

{

if (string.IsNullOrEmpty(returnUrl))

{

return RedirectToAction("Index", "Home");

}

return RedirectToLocal(returnUrl);

}

ModelState.AddModelError("", "Invalid username or password.");

return View(model);

1. Notice that the error message is added to ModelState above should the sign in fail; this will then appear in the ValidationSummary block of html we added in an earlier lab.
2. Ensure the Logoff method also calls the **SignInManager**. Replace the **LogOff** method code with the following.

HttpContext.GetOwinContext().Authentication.SignOut(DefaultAuthenticationTypes.ApplicationCookie);

return RedirectToAction("Index", "Home");

1. When a Register request is posted, use the identity provider to create a new user account, sign in with it and then redirect to the relevant action: Replace **Register** method with **HttpPost** code with the following

if (!ModelState.IsValid)

{

return View(model);

}

var user = new ApplicationUser { UserName = model.UserName, Email = model.UserName };

var result = await UserManager.CreateAsync(user, model.Password);

if (result.Succeeded)

{

await SignInManager.SignInAsync(user, false, false);

return RedirectToAction(nameof(HomeController.Index), "Home");

}

// Adds the erros which came from the user creation process to the model to show it in the form.

result.Errors.ToList().ForEach(e => ModelState.AddModelError(string.Empty, e));

return View(model);

1. Finally, for this controller, we need to configure it to only allow calls to the relevant functions like Logoff, if there is a user logged in. It is a good practice to ensure that your controller is protected by default. That is, all methods can only be called by an authorized user. This is achieved with the **[Authorize]** attribute, which can be applied to methods or classes.
2. Add **[Authorize]** to the class declaration. Note that the methods have already been marked up in this controller class appropriately with **[AllowAnonymous]** if users are allowed to call the method without being authenticated.

### Task 3: Modify the toolbar to display logged in user

1. Open the **\_Layout.cshtml** file in Views/Shared, and locate the markup for the action link to Login:

<li>

@Html.ActionLink("Login", "Login", "Carrier", null, new { @Class = "login-item" })

</li>

1. Replace this section with the following code:

@if (User.Identity.IsAuthenticated)

{

<li>

<a>@User.Identity.Name!</a>

</li>

<li>

@Html.ActionLink("Log off", "LogOff", "Carrier", null, new { @Class = "login-item" })

</li>

}

else

{

<li>

@Html.ActionLink("Login", "Login", "Carrier", null, new { @Class = "login-item" })

</li>

}

1. Review the code – it uses the current context's Identity to check whether the user is authenticated or not. If we are, then display a link with the user's name, and another link for the Logoff action. If we are not authenticated, just display the **Login** link.

### Task 4: Test the Identity Provider, Register, and Login functionality

1. Build and run the application.
2. Use a browser to navigate to http://localhost:[PORT]/carrier/register
3. Enter a username, for example, **JohnDoe**, a password – ensure the password you choose conforms to the rules below – there is no error handling or display on this page yet, so if something fails, you will be redirected to the same page. If there are no errors, you will be logged in and redirected to the home page.
   1. At least 6 characters long

### Task 5: Add more Password Requirement Rules

1. Edit the **Edit()** method in **MyShuttleUserManager.cs** file for the **MyShuttle.Web** project by replacing it with the following code.

var manager = new MyShuttleUserManager(new UserStore<ApplicationUser>(context.Get<MyShuttleContext>()));

// By default, ASP.NET Identity uses MinimumLengthValidator of 6 chars as a password validator

manager.PasswordValidator = new PasswordValidator

{

RequiredLength = 6,

RequireNonLetterOrDigit = true,

RequireDigit = true,

RequireLowercase = true,

RequireUppercase = true,

};

return manager;

1. Running the application again, register a different username, and note it will now work with relaxed password rules applied.