Module 11: **Controlling Access to ASP.NET MVC 5 Web Applications**

# ****Contents:****

|  |  |
| --- | --- |
|  | [Module Overview](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C11.html#P1) |
| **Lesson 1:** | [Implementing Authentication and Authorization](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C11.html#P2) |
| **Lesson 2:** | [Assigning Roles and Membership](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C11.html#P3) |
| **Lab:** | [Controlling Access to ASP.NET MVC 5 Web Applications](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C11.html#P4) |
|  | [Module Review and Takeaways](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C11.html#P5) |

# ****Module Overview****

Authentication is a vital requirement in most web-based applications. Developers usually display only restricted information to all users. Web applications require users to authenticate themselves to view exclusive information. Web applications also display specific information that is relevant to specific user roles. Microsoft ASP.NET includes various authentication models, including local authentication providers, claim-based authentication systems, and federated authentication systems. You need to know how to use these authentication models to implement authentication functionality in your web application. You should also know how to authorize users and roles in your application to restrict information access according to user and role membership.

### ****Objectives****

After completing this module, you will be able to:

|  |  |
| --- | --- |
| • | Implement authentication and authorization systems in your web application. |
| • | Manage users and roles in your web application. |
| • | Authorize users and roles in your web application. |

# Lesson 1: ****Implementing Authentication and Authorization****

Membership providers, which were introduced in ASP.NET 2.0, help you to create secure authentication and authorizations systems for web applications. You need to know how to use the membership provider model in ASP.NET, to change authentication methods with minimum changes to the code. You can use claim-based authentication and federated authentication to allow external users to authenticate themselves, and access and use the web application. You should know how to configure restrictions in the web application to ensure that users access only specific information that is permitted and relevant to them.

## ****Lesson Objectives****

After completing this lesson, you will be able to:

|  |  |
| --- | --- |
| • | List the local authentication options available for an ASP.NET web application. |
| • | Describe how client-based authentication works. |
| • | Describe how federated authentication works. |
| • | Describe how to restrict access to resources. |
| • | Authorize access to specific users to MVC 5 web application controller actions. |

## ****Local Authentication Providers****

Authentication providers include code that runs when ASP.NET needs to authorize a user. The code authenticates users by using the information stored in back-end databases, such as Active Directory or Microsoft SQL Server. The membership system in ASP.NET includes the following authentication providers: **ActiveDirectoryMembershipProvider**, **SqlMembershipProvider**, **SimpleMembershipProvider**, and **UniversalProviders**. The **SimpleMembershipProvider** and **UniversalProviders** are authentication providers that support the OAuth authentication mechanism.

The membership system in ASP.NET also includes the **ASP.NET Identity**. **The ASP.NET Identity** includes profile support and OAuth integration and works with OWIN.

The ASP.NET membership system allows application developers to switch amongst authentication providers, without modifying the code.

The following is a description of the authentication providers**—ActiveDirectoryMembershipProvider**and **SqlMembershipProvider**:

|  |  |
| --- | --- |
| • | ActiveDirectoryMembershipProvider. This provider class is defined in the **System.Web.Security** namespace, and the provider enables you to use Active Directory as the membership and role repository of your web application. |
| • | SqlMembershipProvider. This provider class is defined in the **System.Web.Security** namespace. The provider works with a specific table schema that you can generate by using the **aspnet\_regdb.exe** command. |

**ActiveDirectoryMembershipProvider** and **SqlMembershipProvider** are configured to work only with a specific table schema or directory. To overcome this restriction, Microsoft developed **SimpleMembershipProvider** and **UniversalProviders** to replace **ActiveDirectoryMembershipProvider** and **SqlMembershipProvider**.

|  |  |
| --- | --- |
| • | SimpleMembershipProvider. This is a membership provider that works with the SQL Server, SQL Server Compact Edition, Microsoft Azure SQL Database, and other versions of SQL Server. **SimpleMembershipProvider** requires only three key parameters—table name, user ID, and user name. You use this provider to implement authentication that works with any SQL Server database table schema. |
| • | UniversalProviders. This is a set of membership providers that works with any database that Entity Framework supports. However, these providers work only with the database schema designed by Microsoft. While initializing a universal provider, if the schema does not exist in the database, the provider generates the schema. |

You can use **ASP.NET Identity** with ASP.NET frameworks, such as ASP.NET MVC, Web Forms, Web Pages, Web API, and SignalR. You can use **ASP.NET Identity** when building web, phone, store, or hybrid applications.

**Question**: What is the benefit of using **SimpleMembershipProvider**?

## ****Claims-Based Authentication****

Claims-based authentication is a model that facilitates single sign-on. Single sign-on is a feature that allows you to receive a claim when you log on to a centralized authentication system. The claim is a ticket that authentication systems use to authenticate user logons. The claim contains user identity information that helps authentication systems identify users. With claims-based authentication systems, you can focus efforts on developing business functions, rather than worrying about authenticating users.

Claims-based authentication facilitates:

|  |  |
| --- | --- |
| • | Authenticating users to access applications. |
| • | Storing user account information and passwords. |
| • | Checking enterprise directories for user information. |
| • | Integrating the application with the identity systems of other platforms or companies. |

To implement claims-based authentication in your web application, you can use Windows Identity Foundation (WIF). WIF is a set of .NET Framework classes that helps read information from the claims in a web application.

The following steps describe the functioning of claims-based authentication systems:

|  |  |
| --- | --- |
| 1. | When an unauthenticated user requests a webpage, the request is redirected to the Identity Provider (IP) pages. |
| 2. | The IP requires you to enter credentials such as the user name and password. |
| 3. | After you enter the credentials, the IP issues a token. Then, the token is returned to the web browser. |
| 4. | The web browser is redirected to the webpage that you originally requested. WIF determines if the token satisfies the requirements to access the webpage. If the token satisfies all requirements, a cookie is issued to establish a session. This cookie ensures that the authentication process occurs only once. Then, the authentication control is passed on to the application. |

ASP.NET Identity supports claims-based authentication. In ASP.NET Identity a set of claims represents the user's identity.

**Question**: What are the benefits of using claims-based authentication?

## ****Federated Authentication****

Federations rely on claims-based authentication to allow external parties such as trusted companies to access their applications. Federations use the WS-Federations claim standard to enable the exchange of claims between two parties in a standardized manner.

WIF provides support for federations by using the **WSFederationAuthenticationModule** HTTP module. **WSFederationAuthenticationModule** enables you to implement support for federations in your ASP.NET application, without implementing individual logic.

Federated authentication enables Security Token Service (STS) to:

|  |  |
| --- | --- |
| • | Process the claims from business partners. |
| • | Extract user information from the claims. |

STS enables you to focus more on writing the business logic. It eliminates the need to manage the authentication information of business partners, in your web application.

Configuring the **WSFederationAuthenticationModule** helps specify the STS to which non-authenticated requests should be redirected. WIF provides two methods of federated authentication—FederatedPassiveSignIn and **Passive Redirect**.

**FederatedPassiveSignIn Control**

Consider cases when unauthenticated users try to access protected resources and you want to redirect these users to a logon page. For such cases, you can embed the FederatedPassiveSignIn control in the logon page of your web application, to redirect unauthenticated users to the logon page. The FederatedPassiveSignIn control is configured with issuer (STS) information.

You can use the FederatedPassiveSignIn control to:

|  |  |
| --- | --- |
| • | Exclude application-wide protection. |
| • | Include a logon page with clickable controls. |

**Passive Redirect**

Consider cases when unauthenticated users try to access a protected resource, and you want to redirect these users to an STS without using an application logon page. For such cases, you can use passive redirect.

Passive redirect enables STS to:

|  |  |
| --- | --- |
| • | Verify the identity of the unauthenticated users. |
| • | Issue security tokens that contain the appropriate claims for users. |

Passive redirect requires you to add **WSFederationAuthenticationModule** in the pipeline of the Hypertext Transfer Protocol (HTTP) modules, to identify unauthenticated user requests and redirect users to the STS you specify. Adding **WSFederationAuthenticationModule** in the HTTP pipeline processes the claim information before passing the claim to the ASP.NET engine.

You can instantiate **WSFederationAuthenticationModule**and use it to trigger the sign-on process. This functionality is similar to that of the FederatedPassiveSignIn control; however, passive redirect implements this sign-on functionality with minimum coding effort.

**Question**: What are the benefits of using federated authentication?

## ****Restricting Access to Resources****

You can restrict user access by implementing the **Authorize** attribute in a controller, instead of using the Web.config file as you would use in an ASP.NET WebForms application. The Web.config file requires physical files to exist, for access control to work. You cannot use the Web.config file to restrict user access because MVC applications route requests to actions, not pages.

**Authorizing Action Methods**

[Authorize()]

public ActionResult GetEmployee()

{

return View();

}

Observe the **Authorize** attribute in the code sample. If you specify this attribute, ASP.NET mandates that users should be authorized to access the view returned by the code sample.

If you add the **Authorize** attribute at the class level, the attribute requires users to log on before they can access anything in the controller class. To allow anonymous users to access a specific portion of your class, you can use the **AllowAnonymous** attribute.

**Using the AllowAnonymous Attribute**

[AllowAnonymous()]

public ActionResult Register()

{

return View();

}

**Question**: Why should you use the **Authorize** attribute, instead of the Web.config file to control authorization of pages in your ASP.NET MVC application?

## ****Demonstration: How to Authorize Access to Controller Actions****

In this demonstration, you will see how to:

|  |  |
| --- | --- |
| • | Generate authentication for access to a controller action. |
| • | Handle unauthenticated requests for actions that require authentication by using ASP.NET. |

### ****Demonstration Steps****

|  |  |
| --- | --- |
| • | You will find the steps in the “Lesson 1: Implementing Authentication and Authorization“ section on the following page: <https://github.com/MicrosoftLearning/20486-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486C/20486C_MOD11_DEMO.md>. |

# Lesson 2: ****Assigning Roles and Membership****

Roles and memberships complement authentication features, to help you control all modes of access in web applications. To define access levels for different types of users, you need to know how to implement roles and users. ASP.NET provides role providers and membership providers to help you assign roles and users. ASP.NET also enables you to create custom role providers and custom membership providers. These custom providers allow you to store role and user information in data sources not supported by ASP.NET, such as an Oracle database.

## ****Lesson Objectives****

After completing this lesson, you will be able to:

|  |  |
| --- | --- |
| • | Describe role providers in ASP.NET. |
| • | Describe how to add user accounts to roles. |
| • | Describe how to build custom role providers. |
| • | Describe membership services. |
| • | Describe how to build custom membership providers. |
| • | Add code to an MVC 5 web application to help users reset their password. |

## ****Role Providers in ASP.NET****

Role providers help develop applications by using a role-based security model. There are several role providers that are available in ASP.NET:

|  |  |
| --- | --- |
| • | ActiveDirectoryRoleProvider. This provider class is defined in the **System.Web.Security** namespace, and the provider enables you to use Active Directory as the management model for roles. |
| • | SqlRoleProvider. This provider class is defined in the **System.Web.Security** namespace. The provider works with only a specific table schema in Microsoft SQL Server. You can generate this schema by using the **aspnet\_regdb.exe** command. |
| • | WindowsTokenRoleProvider. This provider uses the Windows authentication token to determine the role of users. Then, the provider checks if users belong to any group stored in the Windows authentication token that was generated when they first logged on. The format of the group name is **Domain\Group**. |

**ActiveDirectoryRoleProvider** and **SqlRoleProvider** have restrictions such as lack of support for non-SQL databases or developer-defined schema. Therefore, Microsoft developed **SimpleRoleProvider** and **UniversalProviders** to replace **ActiveDirectoryRoleProvider** and **SqlRoleProvider**. The following list describes the **SimpleRoleProvider** and **UniversalProviders**:

|  |  |
| --- | --- |
| • | SimpleRoleProvider. This is a role provider that works with SQL Server, SQL Compact Editions, and others versions of SQL Server. **SimpleRoleProvider** enables you to implement authorization based on the table defined by application developers. |
| • | UniversalProviders. This is a database agnostic version that works with any database that Entity Framework supports. However, the database schema is determined by Microsoft. The provider usually generates tables during initialization. |

ASP.NET Identity contains a role provider that allows you to restrict access to the application using roles. For example, you can create a role named **Admin**. You can easily create users in ASP.NET Identity and add them to roles.

You can choose a role provider based on the membership provider that you select. You can also mix role providers; for example, you can combine **SimpleMembershipProvider** and **UniversalRoleProvider**. However, you should avoid mixing role providers because the manner in which each provider identifies users is different.

**Question**: What is the difference between **SimpleRoleProvider** and **SqlRoleProvider**?

## ****Adding User Accounts to Roles****

When you use a database-based role provider, such as **SimpleRoleProvider** or **UniversalRoleProvider**, you can load the initial data onto the database by directly editing the table. You can also edit the table by using the **AddUsersToRoles** function. To use the **AddUsersToRoles** function, you need to implement the custom management tool in your application. The custom management tool helps call the **AddUsersToRoles** function, to add roles to a database.

**Using the AddUsersToRoles Function**

AddUserToRoles("Peter", new string[] {"Admin","Staff"})

You can also use the **AddUsersToRoles** function in business applications to add users to a role. However, you must ensure that the role to which you want to add users exists, before using the **AddUsersToRoles** function.

## ****Building a Custom Roles Provider****

ASP.NET allows you to build custom role providers. Custom role providers enable you to implement role management that uses your own database schema and logic. To build a custom role provider, you need to create a class that inherits the **RoleProvider** class.

**Creating a Custom Role Provider**

public class CustomRoleProvider : RoleProvider

{

}

In the **RoleProvider** class, you need to implement the **GetRolesForUser** function. The **GetRolesForUser** function takes the user name as the input and returns a list of roles to which the user belongs. You can write your own code to obtain role information from the database or other back-end stores.

**Implementing the GetRolesForUser Function**

public override string[] GetRolesForUser(string username)

{

//code to return a list of roles for users

}

After implementing the **GetRolesForUser**function, you need to apply the custom role provider to the application by modifying the Web.config file.

**Configuring a Site to Use a Custom Role Provider**

<roleManager defaultProvider="CustomRoleProvider" enabled="true" cacheRolesInCookie="false">

<providers>

<clear />

<add name="CustomRoleProvider" type="CustomRoleProvider” />

</providers>

</roleManager>

**Question**: Why should you create a custom role provider?

## ****Providing Membership Services****

You need to add the **SimpleMembershipProvider** to the membership section of the Web.config file to use it for membership services.

**Configuring the Simple Membership Provider**

<membership defaultProvider="SimpleMembershipProvider">

<providers>

<clear/>

<add name="SimpleMembershipProvider" type="WebMatrix.WebData.SimpleMembershipProvider, WebMatrix.WebData" />

</providers>

</membership>

Next, you need to create a User table. If you do not have a User table created already, you can modify the AccountModel.cs file, to generate the table. To view the AccountModel.cs file, you should select the Internet Application template, when you create the project.

**Creating a User Table**

[Table("UserProfile")]

public class UserProfile

{

[Key]

[DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity)]

public int UserId { get; set; }

public string UserName { get; set; }

public string Country { get; set; }

}

After modifying the AccountModel.cs file, you need to call the **WebSecurity.InitializeDatabaseConnection** function in the App.Start.cs file. The **WebSecurity.InitializeDatabaseConnection** function ensures that **SimpleMembershipProvider** is configured with connection strings to the database.

**Connecting to the User Table**

WebSecurity.InitializeDatabaseConnection(“DBConn”,”User”,”id”,”Username”,false)

If a specific table does not exist in a database, the **autoCreateTables** parameter helps indicate to the membership provider that it may create the table. To initialize **SimpleMembershipProvider**, you need to add the **InitializeSimpleMembership** attribute to the **AccountController** class.

**Adding the InitializeSimpleMembership Attribute**

[Authorize]

[InitializeSimpleMembership]

public class AccountController : Controller

{

}

**Question**: Why should we use the **InitializeDatabaseConnection** method?

## ****Building a Custom Membership Provider****

If you need to add functionality to a membership provider, such as custom logic for authentication, you need to implement a custom membership provider.

**A Custom Membership Provider**

public class CustomMembershipProvider : SimpleMembershipProvider

{

}

**Overriding the ValidateUser Method**

public override bool ValidateUser(string username, string password)

{

}

In the preceding code sample, the **ValidateUser** method validates the user name and password against the membership store.

**Overriding the Provider Constructor**

public CustomAdminMembershipProvider(SimpleSecurityContext simpleSecurityContext)

{

}

**Adding a Custom Provider to Web.config**

<membership defaultProvider="CustomMemberProvider">

<providers>

<clear/>

<add name="CustomMemberProvider" type=" CustomAdminMembershipProvider " />

</providers>

</membership>

**Question**: Why should you implement a custom membership provider?

## ****Demonstration: How to Reset a Password****

In this demonstration, you will see how to write code that uses a Membership Services Provider to reset the user’s password. Using this code, you can enable users to control their own passwords and reset the password without involving a site administrator.

### ****Demonstration Steps****

|  |  |
| --- | --- |
| • | You will find the steps in the “Lesson 2: Assigning Roles and Membership“ section on the following page: <https://github.com/MicrosoftLearning/20486-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486C/20486C_MOD11_DEMO.md>. |

# ****Lab: Controlling Access to ASP.NET MVC 5 Web Applications****

### ****Scenario****

A large part of the functionality for your proposed Photo Sharing application is in place. However, stakeholders are concerned about security because there are no restrictions on the tasks that users can complete. The following restrictions are required:

|  |  |
| --- | --- |
| • | Only site members should be able to add or delete photos. |
| • | Only site members should be able to add or delete comments. |

You have been asked to resolve these concerns by creating a membership system for the Photo Sharing application. Visitors should be able to register as users of the web application and create user accounts for themselves. After registration, when the users sign in to the application, they will have access to actions such as adding and deleting photos and comments. Anonymous users will not have access to perform these actions. Additionally, registered users should also be able to reset their own password.

### ****Objectives****

After completing this lab, you will be able to:

|  |  |
| --- | --- |
| • | Configure a web application to use ASP.NET Form Authentication with accounts stored in Microsoft Azure SQL database. |
| • | Write models, controllers, and views to authenticate users in a web application. |
| • | Provide access to resources in a web application. |
| • | Enable users to reset their own password. |

##### ****Lab Setup****

Estimated Time: 90 minutes

You will find the high-level steps on the following page: <https://github.com/MicrosoftLearning/20486-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486C/20486C_MOD11_LAB_MANUAL.md>.

You will find the detailed steps on the following page: <https://github.com/MicrosoftLearning/20486-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486C/20486C_MOD11_LAK.md>.

### ****Exercise 1: Configuring Authentication and Membership Providers****

##### ****Scenario****

You want to use a Microsoft Azure SQL database to store user accounts and membership information.

In this exercise, you will:

|  |  |
| --- | --- |
| • | Create a Microsoft Azure SQL database. |
| • | Configure a provider to connect to the database. |

### ****Exercise 2: Building the Logon and Register Views****

##### ****Scenario****

You have configured the Photo Sharing application to connect to Microsoft Azure SQL database for authentication and membership services. However, to use forms authentication in an MVC application, you need to build model classes, controllers, and views that enable users to sign in, sign out, and register for an account.

In this exercise, you will:

|  |  |
| --- | --- |
| • | Add model classes. |
| • | Add controllers. |
| • | Import logon and register views. |
| • | Test the developed components. |

### ****Exercise 3: Authorizing Access to Resources****

##### ****Scenario****

Now that you have enabled and tested authentication, you can authorize access to resources for both anonymous and authenticated users.

You should ensure that:

|  |  |
| --- | --- |
| • | Only site members can add or delete photos. |
| • | Only site members can add or delete comments. |
| • | The account controller actions are authorized properly. |
| • | Only authenticated users see the**\_Create** view for comments in the **Display**view. |

### ****Exercise 4: Optional—Building a Password Reset View****

##### ****Scenario****

Site visitors can now register as users of the Photo Sharing application and sign in to the site so that they can add photos and comments. However, they do not have the facility to change their password. In this exercise, you will create a password reset page by using the membership services provider.

Complete this exercise if time permits.

### ****Review Question(s)****

**Check Your Knowledge**

**Discovery**

**In Exercise 3, when you tried to add a photo before logging on to the application, why did ASP.NET display the Login view?**

Show solution Reset

**Check Your Knowledge**

**Discovery**

**How can you ensure that only Adventure Works employees are granted access to the Delete action of the Photo controller?**

Show solution Reset

# ****Module Review and Takeaways****

In this module, you discussed the various membership and role providers in ASP.NET. You compared the benefits of using **SimpleProviders** and **UniversalProviders** with the benefits of using other providers, for database-centric applications. However, for applications that need to use Windows for authentication and authorization, you can use the existing role and membership providers. You also viewed how to implement custom providers, to add functionalities such as password encryption to your web application.

### ****Real-world Issues and Scenarios****

When you create web applications, you may need to create custom providers because you do not want to use the schema provided by Microsoft. However, you can use **SimpleProviders** to remove the need to develop custom providers and reduce the effort required for building applications.