Developing ASP.NET MVC Web Applications

Session: 10

Authentication and Authorization

Objectives

- Define and describe authentication
- Explain and describe how to implement authentication
- Define and describe ASP.NET Identity
- Define and describe the process of authorization

Authentication

- Authentication is the process of validating the identity of a user before granting access to a restricted resource in an application.
- When the application receives a request from users, it tries to authenticate the user.
- Typically, authentication allows identifying an individual based on the user name and password provided by the user.

Authentication Modes 1-8

The ASP.NET MVC Framework enables you to use the following three types of authentication modes:

- Forms authentication
- Windows authentication
- OpenID/OAuth

While using the Form authentication:

- The application itself is responsible for collecting user's credentials and then, authenticates the user.
- The application can authenticate the users by providing a login form on the Web page.
- When a user successfully logs into the application and gets authenticated to view a Web page, it generates a cookie that is served as an authentication token to the user.
- The cookie is then, used by the browser for the future requests made by the user, and thus allows the application to validate the requests.

Authentication Modes 2-8

- When you create an ASP.NET MVC application in Visual Studio 2013, the application does not automatically uses any type of authentication.
- To authenticate users by using Forms authentication you need to manually configure the <authentication> element under the <system.web> element present in the Web.config file.
- Following code snippet shows the default <authentication> element in the Web.config file:

Snippet

```
<system.web>
   <authentication mode="None" />
 </system.web>
```

- In this code, the <authentication> element contains the mode attribute that allows you to specify what type of authentication is to be used by the application.

Authentication Modes 3-8

- Now to use Forms authentication mode, you need to change the mode attribute of the <authentication> element to Forms.
- Following code snippet shows specifying the authentication mode as Forms:

Snippet

```
 <system.web>
    <authentication mode="Forms" />
  </system.web>
```

- In this code, the mode attribute of the <authentication> element is set to Forms, which indicates that application should use the Forms authentication.

Authentication Modes 4-8

- Windows authentication is best suited to an intranet environment where a set of known users are already logged on to a network.
- When you use the Windows authentication mode in your application, it is the firewall that takes care of the authentication process.
- In this mode, the application uses the users credential authenticated by the company's firewall for security checks.
- To configure your application to use the Windows authentication mode, you need to change the mode attribute of the <authentication> element to Windows.

Authentication Modes 5-8

- The OAuth and OpenID are authorization protocols that enable a user to logon to an application by using the credentials for third party authentication providers, such as Google, Twitter, Facebook, and Microsoft.
- When you create an ASP.NET MVC Web application using Visual Studio 2013, the third party authentication providers need to be configured.
- You can do this in the Startup.Auth.cs file present in the App_Start folder in the application directory.
- By default, the Startup.Auth.cs file contains the commented code that enables logging in with third party login providers.

Authentication Modes 6-8

- Following code snippet shows the content of the Startup.Auth.cs file:

Snippet

```
using Microsoft.AspNet.Identity;
using Microsoft.Owin;
using Microsoft.Owin.Security.Cookies;
using Owin;
namespace SecureApp
{
    public partial class Startup
    {
        // For more information on configuring authentication, please visit
http://go.microsoft.com/fwlink/?LinkId=301864
        public void ConfigureAuth(IAppBuilder app)
        {
        // Enable the application to use a cookie to store information for
the signed in user
        app.UseCookieAuthentication(new CookieAuthenticationOptions
          {
         AuthenticationType = DefaultAuthenticationTypes.ApplicationCookie,
                  LoginPath = new PathString("/Account/Login")
              });
```

Authentication Modes 7-8

- Following code snippet shows the content of the Startup.Auth.cs file:

Snippet

```
// Use a cookie to temporarily store information about a user logging in
with a third party login provider
app.UseExternalSignInCookie(DefaultAuthenticationTypes.ExternalCookie);
      // Uncomment the following lines to enable logging in with third
party login providers
      //app.UseMicrosoftAccountAuthentication(
      //    clientId: "",
      //    clientSecret: "");
      //app.UseTwitterAuthentication(
      //   consumerKey: "",
      //   consumerSecret: "");
      //app.UseFacebookAuthentication(
      //   appId: "",
      //   appSecret: "");
      //app.UseGoogleAuthentication();
        }
    }
}
```

Authentication Modes 8-8

- The preceding code shows content of the Startup.Auth.cs file where the third party login provider's code is commented.
- As you have seen that the third party login provider's code is commented, because before you can use the services provided by an external login providers, you have to register the application with them.
- To use the services of Google, you are not required to register with the ASP.NET MVC application if you are already registered with Google.
- To use the services of Google, you only need to uncomment the call to the OAuthWebSecurity.RegisterGoogleClient() method.

Using the Authorize Attribute

- When a proper security mechanism is implemented on an application, a user must be enforced to login to the application to access some specific resources.
- This type of mechanism is implemented in an application by using the Authorize action filter.
- You can use these attributes to add behavior that can be executed either before an action method is called or after an action method is executed.
- In an ASP.NET MVC application, you can use the Authorize attribute to secure an action method, a controller, or the entire application.

Securing a Controller Action 1-2

Consider a scenario of an online shopping store application,

- You want to display the list of available items in the home page from where a user can select an item and purchase it.
- You want that all users can see the list of item on the home page, but only the authenticated user can purchase an item using this application.

- In such situation, you need to create a BuyItem() action method in a controller class, named Product.
- While creating the BuyItem() action method, you should ensure that this method is only accessible for the authenticated users.
- For this, you need to add the Authorize attribute on the BuyItem() action method.

Securing a Controller Action 2-2

- Following code snippet shows adding the Authorize attribute the BuyItem() action method:

Snippet

```
[Authorize]
  public ActionResult BuyItem()
      {
           return View();
      }
```

- In this code, the Authorize attribute is added on the BuyItem() action method.

Securing a Controller 1-3

- You can also use the Authorize attribute to a controller to secure the entire controller.
- For this, you need to add the Authorize attribute on the controller.
- Following code snippet shows adding the Authorize attribute to a controller:

Snippet

```
[Authorize]
public class ProductController : Controller
{
          …
}
```

- The code will restricts all the action methods defined in the ProductController controller class.

Securing a Controller 2-3

- Sometime, you might not want to secure certain action methods in a controller.
- In such case, you can use the AllowAnonymous attribute on those action methods.
- Following code snippet shows using the AllowAnonymous attribute:

Snippet

```
[AllowAnonymous]
public ActionResult Index()
{
  return View();
}
```

- This code uses the AllowAnonymous attribute on the Index() action. As a result, access to this method does not require authentication.

Securing a Controller 3-3

- At times, you may want that all the users to be authenticated for an entire application.
- For this, you need to add the Authorize attribute as a global filter, by adding it to the global filters collection in the RegisterGlobalFilters() method in the FilterConfig.cs file.
- Following code snippet uses the RegisterGlobalFilters() method in the FilterConfig.cs file:

Snippet

```
public static void RegisterGlobalFilters(GlobalFilterCollection filters)
    {
        filters.Add(new System.Web.Mvc.AuthorizeAttribute());
        filters.Add(new HandleErrorAttribute());
    }
```

- This code applies the Authorize attribute to all action methods in the application. However, you can use the AllowAnonymous attribute to allow access to specific controllers or action methods.

ASP.NET Identity 1-7

- Prior to ASP.NET MVC 5, the MVC Framework used the Membership API for implementing authentication and authorization.
- ASP.NET MVC 5 uses ASP.NET Identity, which is a new membership system for ASP.NET MVC applications.
- You can use ASP.NET Identity to add login features to your application.
- ASP.NET Identity uses the Code First approach to persist user information in a database.
- When you create an ASP.NET MVC 5 application in Visual Studio 2013, the Identity and Account management classes are automatically added to the project.
- The IdentityModel.cs file created by Visual Studio 2013 manages the identity of an application.

ASP.NET Identity 2-7

- Following code snippet shows the IdentityModel.cs file:

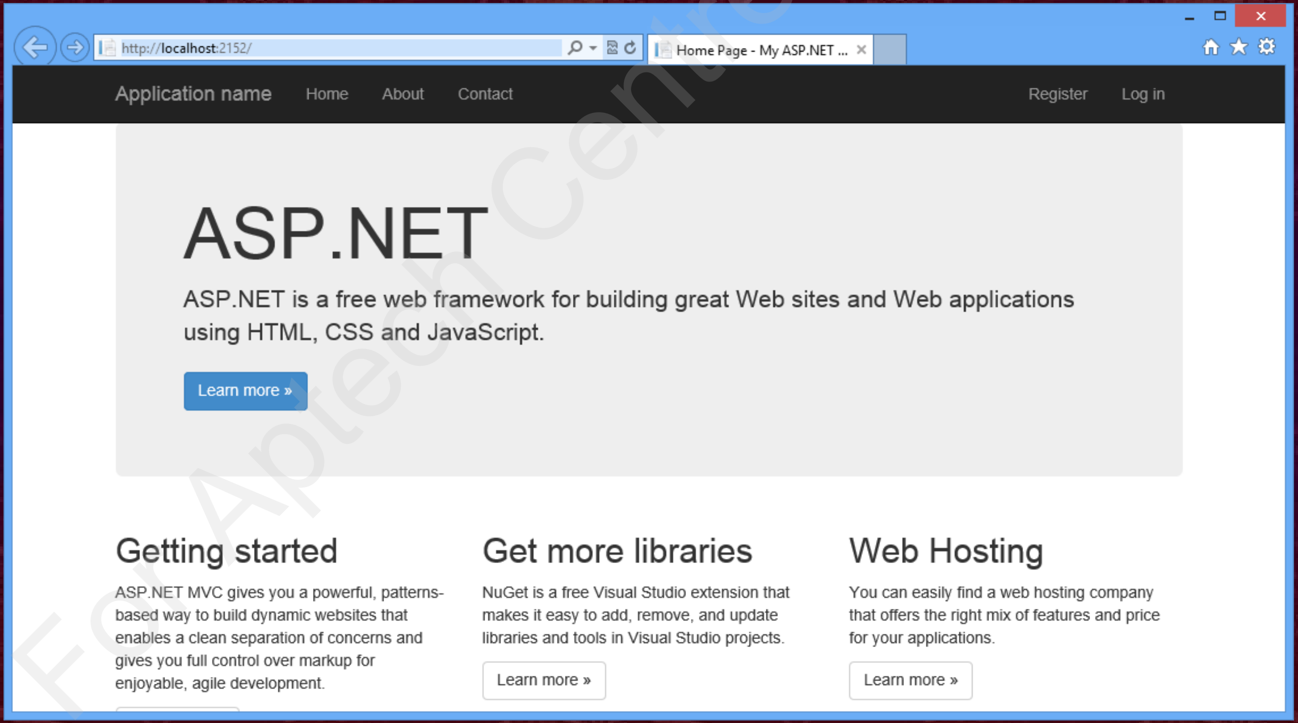
Snippet

```
namespace WebApplication1.Models {
    public class ApplicationUser : IdentityUser {
    }
    public class ApplicationDbContext :
     IdentityDbContext<ApplicationUser>
    {
        public ApplicationDbContext()
            : base("DefaultConnection")
        {
        }
    }
}
```

- In this code, the ApplicationUser class that extends from the IdentityUser class is responsible for managing user accounts in the application. The ApplicationDbContext class inherits from IdentityDbContext and allows the application to interact with the database where account data of users are stored.

ASP.NET Identity 3-7

- You will now learn how ASP.NET Identity is used in a default ASP.NET MVC application created using Visual Studio 2013.
- When you execute the default application, the Home page appears.
- Following figure shows the Home page:



ASP.NET

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

Learn more »

Getting started

ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and gives you full control over markup for enjoyable, agile development.

Get more libraries

NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.

Learn more »

Web Hosting

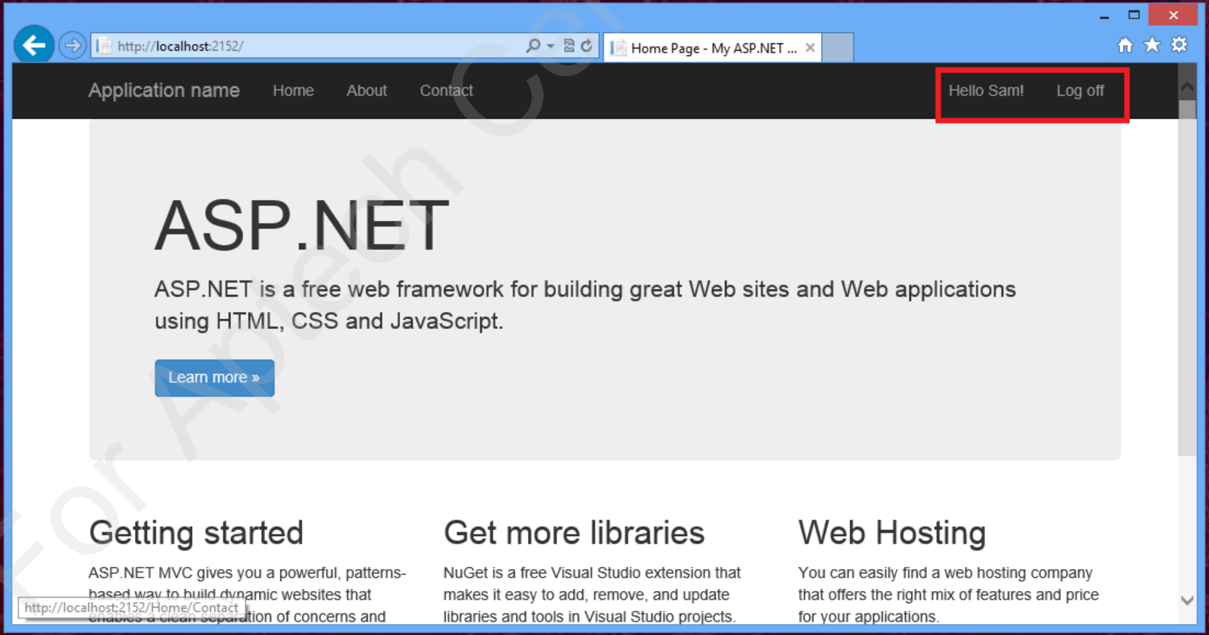
You can easily find a web hosting company that offers the right mix of features and price for your applications.

Learn more »

ASP.NET Identity 4-7

In the Home page, you need to perform the following tasks:

- Click the Register link. The Register page is displayed.
- Add the registration details.
- Click the Register button. The application logs you on and the home page is displayed with a welcome message based on the specified user name and a Log off option.
- Following figure shows the welcome message:



ASP.NET

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

Getting started

ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and

Get more libraries

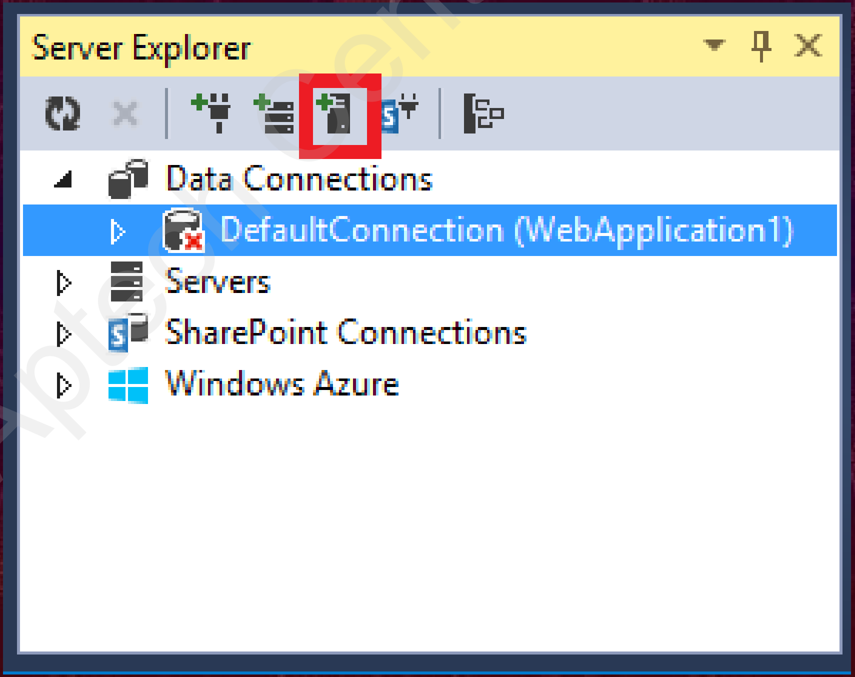
NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.

Web Hosting

You can easily find a web hosting company that offers the right mix of features and price for your applications.

ASP.NET Identity 5-7

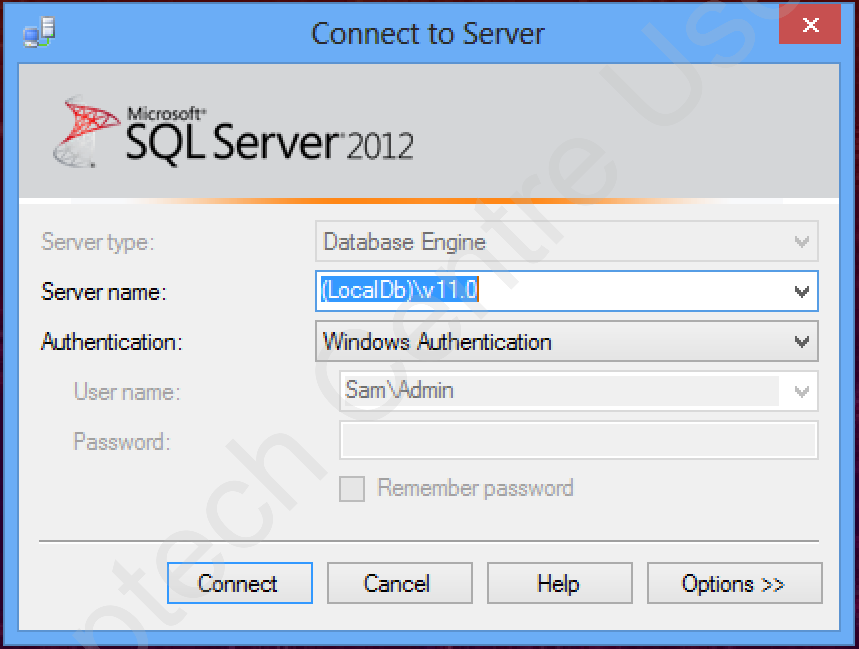
- Internally, the application uses ASP.NET Identity to create database tables for the application to hold account data.
- You can view the generated tables by performing the following tasks:
  - Select View → Server Explorer. The Server Explorer window is displayed.
  - Select DefaultConnection (WebApplication1) under the Data Connections node and click the Add SQL Server icon.
- Following figure shows the Server Explorer window:



ASP.NET Identity 6-7

- In the Connect to SQL Server dialog box that appears, type (LocalDb)\v11.0 in the Server name text field.

- Following figure shows the Connect to Server dialog box:

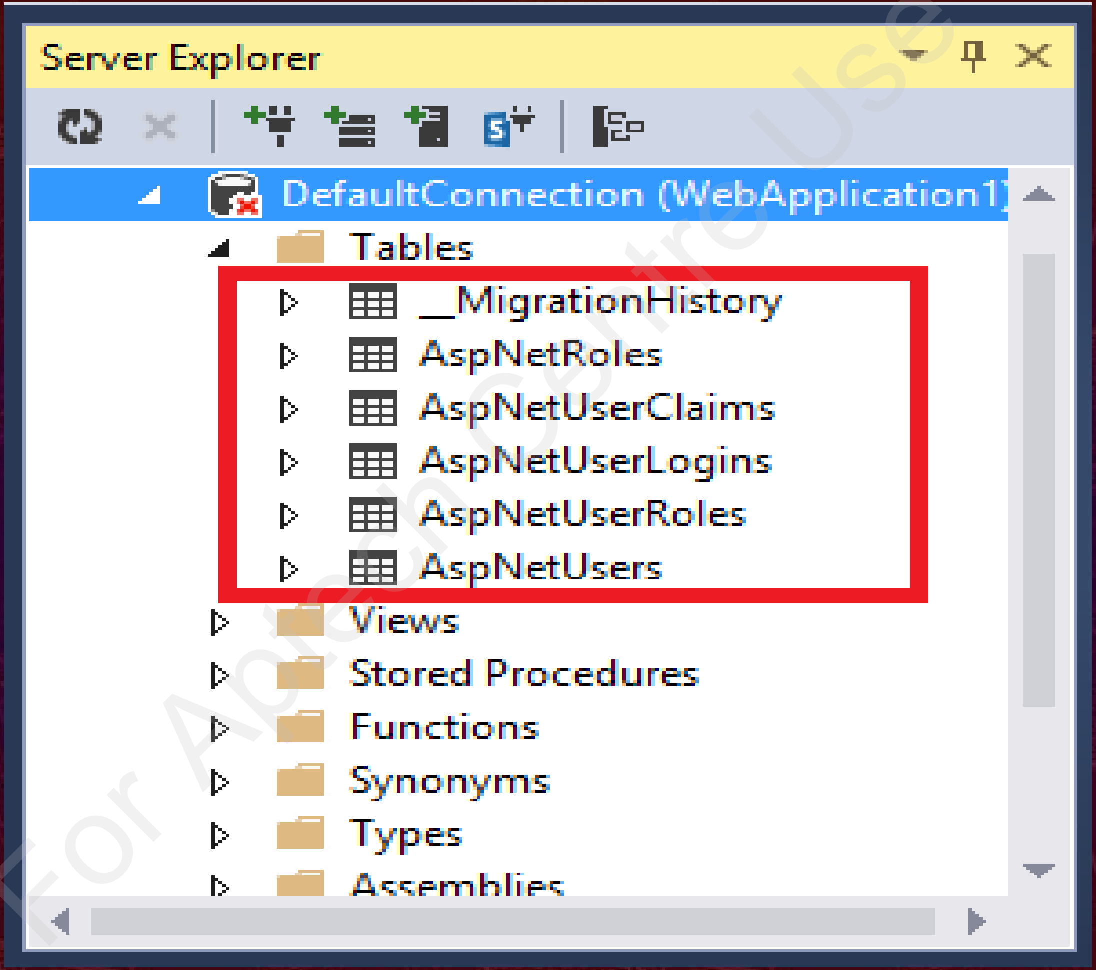


Server type: Database Engine
Server name: (LocalDb)\v11.0
Authentication: Windows Authentication
User name: Sam\Admin
Password:
Remember password

- Click Connect.
- Expand the DefaultConnection (WebApplication1) → Tables node to view the tables that are automatically generated by Visual Studio 2013 based on ASP.NET Identity.

ASP.NET Identity 7-7

- Following figure shows the tables that are automatically generated by Visual Studio 2013:



Server Explorer

DefaultConnection (WebApplication1)
- Tables
  - __MigrationHistory
  - AspNetRoles
  - AspNetUserClaims
  - AspNetUserLogins
  - AspNetUserRoles
  - AspNetUsers
- Views
- Stored Procedures
- Functions
- Synonyms
- Types
- Assemblies

Authorization

- Authorization is the process of verifying whether an authenticated user has the privilege to access a requested resource.
- After creating users that can access your application by using the membership management service, you need to specify authorization rules for the users.
- You need to grant different permissions to different users to provide accessibility to the pages on your application.

Users and Roles 1-3

- In order to maintain a different set of settings for each user is a complex task.
- So, you should assemble the users in a group user known as roles and then, define the permissions on these roles.
- In an ASP.NET MVC application, you can authorize some specific users and roles by using the Authorize attribute.
- Following code snippet uses the Authorize attribute to authorize specific users:

Snippet

```
[Authorize(Users="Mathews, Jones")]
public class ManagerProduct: Controller
{
 // Code to perform some operation
}
```

- In this code, only the users whose name is Mathews and Jones are the authorized users to access the ManagerProduct controller.

Users and Roles 2-3

- In an ASP.NET MVC application, you can also assign roles to different users.
- Thereafter, you can use the authorization at the role level.
- Following code snippet shows specifying role to users:

Snippet

```
 [Authorize(Roles="Administrator")]
public class ManagerProduct: Controller
   {
    // Code to perform some operation
   }
```

- This code allows accessing the ManagerProduct controller to users with the Administrator role.

Users and Roles 3-3

- In an ASP.NET MVC application, you can also use a combination of roles and users.
- Following code snippet shows how to use a combination of roles and users:

Snippet

```
[Authorize(Roles="Manager", Users="Steve, Mathews")]
public class ManagerProduct: Controller
   {
       // Code to perform some operation
   }
```

- This code allows the users whose name is Steve and Mathews, and users with the Manager role to access the ManageStore controller.

Role Providers 1-2

- You need to create roles so that you can assign specific rights to specific users.
- The ASP.NET MVC Framework provides different role providers that you can use to implement roles.
- Some of them are as follows:
  - ActiveDirectoryRoleProvider: This provider class enables you to manage role information for an application in the Active Directory.
  - SqlRoleProvider: This provider class enables you manage role information for an application in SQL database.
  - SimpleRoleProvider: This role provider works with different versions of SQL Server.
  - UniversalProviders: This provider works with any database that Entity Framework supports.

Role Providers 2-2

- You can use the Provider property provided by the Roles class to access the current role provider.
- Following code snippet uses the Provider property:

Snippet

```
var roles = (TestRoleProvider)Roles.Provider;
```

- In this code, the Provider property allows to access the role provider, named, TestRoleProvider. This piece of code will only work when the System.Web.Security namespace is included.

User Accounts and Roles 1-4

- When you create an MVC application using Visual Studio 2013, you need to create the membership database with a connection string that you need to specify in the Web.config file.
- You can add sample data in this database with the required roles by using the Seed() method.
- Following code snippet shows using the Seed() method:

Snippet

```
 var roles = (SimpleRoleProvider)Roles.Provider;
   if (!roles.RoleExists("Administrator"))
            {
                roles.CreateRole("Administrator");
            }
```

- In this code,
  - You first retrieve a reference of the simple role provider.
  - Then, you use this reference to check whether the Administrator exists in the database.
  - If it does not exist, the CreateRole() method creates the role named, Administrator.

User Accounts and Roles 2-4

- Once you have created an Administrator role in the database, you can add users to that role.
- Following code snippet shows assigning a user with the Administrator role:

Snippet

```
if  (!roles.GetRolesForUser("Mark_Jones").Contains("administrator"))
{
   roles.AddUsersToRoles(new[] { "Mark_Jones" }, new[] { "administrator"
});


 }
```

- In this code, if the user with the name Mark_Jones is already assigned with the Administrator role, then, the AddUserstoRoles() method adds the Administrator role, to the user with the name, Mark_Jones.

User Accounts and Roles 3-4

- Once you have created a role and users in the database, you can then, use the Authorize attribute on the action method that needs to be restricted.
- Following code snippet shows using the Authorize attribute so that the action method can be accessed by users with the Administrator role:

Snippet

```
[Authorize(Roles = "administrator")]
  public ActionResult ManageProduct()
         {
              return View();
         }
```

- This code will display a login page when a user tries to access the ManageProduct page. When the user provides the valid credentials for a user with the administrator role then, the ManageProduct page is displayed.

User Accounts and Roles 4-4

- You can also display only those links that a user has rights to access.
- For example, the ManageProduct link should be visible only to the users with administrator role.
- In such situation, you can use the ActionLink() method to hide the ManageProduct link from other users.
- Following code snippet shows using the ActionLink() method:

Snippet

```
@if(User.IsInRole("administrator"))
{
@Html.ActionLink("ManageProduct","ManageProduct")
}
```

- This code displays the ManageProduct link only when a user logs in as an administrator.

Summary

- Authentication is the process of validating the identity of a user before granting access to a restricted resource in an application.
- In an ASP.NET MVC application, you can use the Authorize attribute to secure an action method, a controller, or the entire application.
- ASP.NET Identity uses the Code First approach to persist user information in a database.
- Authorization is the process of verifying whether an authenticated user has the privilege to access a requested resource.
- In an ASP.NET MVC application, you can authorize some specific users and roles by using the Authorize attribute.
- The ASP.NET MVC Framework provides different role providers that you can use to implement roles.
- When you create an MVC application using Visual Studio 2013, you need to create the membership database with a connection string.