

COMP 3111: Software Engineering

Web Site Security in ASP.NET Tutorial and Lab Notes

Because of the easy access provided by the Web, security is an important issue for web applications. For some web applications, it is necessary to authenticate the user (e.g., if a user is allowed to access and change only his personal information). ASP.NET web applications provide the following authentication types.

- *Individual User Account* – ASP.NET Identity is used, which enables a user to register an account, by creating a username and password on the website or by signing in with social providers such as Facebook, Google, Microsoft Account or Twitter.
- *Organizational Accounts* – ASP.NET relies on Windows Server Active Directory or Azure Active Directory to authenticate users using Windows Identity Foundation (WIF).
- *Windows Authentication* – ASP.NET relies on IIS to authenticate users.

After the user has been authenticated, each request needs to be authorized, which means deciding whether the user is permitted to access the requested resource (e.g., a web page). Authorization in ASP.NET can be

- *File authorization* – File authorization checks an access control list (ACL) to determine whether a user should have access to the file. This is useful when Windows authentication is used. Setting appropriate permissions for files or directories in Windows Explorer authorizes access to resources.
- *URL authorization* – URL authorization explicitly allows or denies access to a particular directory by user name or role. It is enabled by specifying a list of users or roles in the `allow` and `deny` elements of the authorization section of a directory's `Web.config` file. The permissions established for a directory also apply to its subdirectories, unless overridden by `Web.config` files in a subdirectory.

OWIN AUTHENTICATION

OWIN (the Open Web Interface for .NET) is an open-source specification describing an abstraction layer between web servers and application components. The goal of the OWIN specification is to facilitate a simple, pluggable architecture for .NET-based Web applications and the servers upon which they rely, encouraging development of small, focused application components (known as "middleware") which can be assembled into a processing pipeline through which the server can then route incoming HTTP requests.

ASP.NET IDENTITY

ASP.NET Identity is the membership system used to build an ASP.NET web application and is used in the Visual Studio Web Forms project template. It provides a built-in way to validate and store user credentials thereby helping manage user authentication in web, phone, store or hybrid applications. ASP.NET Identity uses OWIN Authentication for log-in/log-out of users in the web site.

ASP.NET Identity allows you to do the following.

- Create new users and passwords.
- Store membership information (user names, passwords and supporting data) in Microsoft SQL Server, Active Directory or some other data store, including *NoSQL* data stores.
- Authenticate users who visit a web site. Users can be authenticated programmatically or the Login controls can be used to create a complete authentication system that requires little or no code.
- Manage passwords, which includes creating, changing, and resetting them. Depending on the selected options, ASP.NET Identity can also provide an automated password-reset system that takes a user-supplied question and response.
- Expose a unique identification for authenticated users that can be used in your own applications and that also integrates with the ASP.NET personalization and role-management (authorization) systems.
- Specify a custom membership provider, which allows you to substitute your own code to manage membership and maintain membership data in a custom data store.

To use ASP.NET Identity membership you must do the following.

1. Configure the web application to use membership¹.
2. Configure the web application to use forms authentication².

¹ By default, ASP.NET Identity is enabled for a web application that uses the Web Forms templates. The default provider stores identity information in a Microsoft SQL Server database. However, you can use other providers including a custom provider.

² Form authenticated is configured by default when you create a new web site in ASP.NET.

3. Define user accounts for membership.
4. Optionally specify that some pages or folders in the web application are protected and are accessible only to authenticated (i.e., logged in) users.

You can then use ASP.NET Identity to authenticate users in your application. Typically, you will provide a page where users can log in, a registration page where new users can sign up and a change-password page where users can change their password. The Login controls (see the Appendix) encapsulate virtually all of the logic required to collect user credentials and validate them. When you create a Web Forms web application in Visual Studio, the `Account` folder contains template pages that include much of this basic login functionality as shown in Figure 8. Alternatively, you can use the same Login controls that are in the template pages to create custom login pages.

AUTHORIZATION TYPES

User-based Authorization

If you use the templates provided in the `Account` folder or use the Login controls, they will automatically use membership capabilities to validate a user. After the user is validated, information about the user will be persisted in an encrypted cookie if the user's browser accepts cookies. The Login controls perform this task automatically, but if you create custom login controls, you can call methods of the `FormsAuthentication` class to create the cookie and write it to the user's computer. If a user has forgotten his or her password, the login page can call membership functions that help the user recover the password or reset the password. After a user has been authenticated, the membership system makes available an object that contains information about the current user. For example, you can get properties of the membership user object to determine the user's name and email address, when the user last logged into your web site, and so on.

Each time the user requests a page, forms authentication checks whether the user is authenticated. If a page is restricted, users who are authenticated and/or are members of the approved roles (see the next section) are allowed to view the page. Anonymous users (users who are not logged in) are directed to the login page. By default, the authentication cookie remains valid for the user's session.

Role-based Authorization

User-based authorization allows you to grant access to web pages on a user-by-user basis. However, granting permission for page access or functionality on a user-by-user basis can become a maintenance nightmare in scenarios where there are many user accounts or when users' privileges change often. Any time a user gains or loses authorization to access a particular page, the administrator needs to update the appropriate authorization. To better manage authorization, it usually helps to classify users into groups or *roles* and then to apply permissions on a role-by-role basis. For example, most web applications have a certain set of pages that can be accessed only by administrative users (i.e., those in the Admin role).

ASP.NET Identity allows you to define roles and associate them with user accounts. You can create and delete roles, add users to or remove users from a role, determine the set of users that belong to a particular role, and tell whether a user belongs to a particular role. Using roles you can limit access to pages on a role-by-role basis through URL authorization rules and show or hide additional information or functionality on a page based on the currently logged on user's roles.

Create Roles and Users

We will create two roles and two users using *Identity Manager*³ (see Appendix A).

1. Create two roles "Admin" and "Student" as shown in Figure 1(a). **Be patient as creating the first role requires Visual Studio to also create the tables used by ASP.NET Identity membership** (see Figure 7).
2. Create two users "marvin" and "winnie" and assign the roles to them as shown in Table 1.

User Name	Password	Role(s)
marvin	marvin#	Admin
winnie	winnie#	Admin Student

Table 1: User name, password and roles.

Figure 1(b) and (c) show what the role information should look like in *Identity Manager* when you are done.

³ See Appendix A for how to install *Identity Manager* in Visual Studio.



Figure 1: Users and roles defined using *Identity Manager* tool.

CUSTOMIZING USER NAME AND PASSWORD POLICIES

To meet the requirement that the password should have at least 6 characters and should contain at least one non-alphanumeric character, we previously used a CustomValidator. However, this requirement can now be met by setting some variables of the ASP.NET UserValidator and PasswordValidator classes.

Expand the App_Start folder under the ASUWebApplication node in the Solution Explorer, open the IdentityConfig.cs file and scroll down to the line public static ApplicationUserManager Create... Here you can set the following user name and password policies as shown in Figure 2.

```
// Configure validation logic for usernames
manager.UserValidator = new UserValidator<ApplicationUser>(manager)
{
    AllowOnlyAlphanumericUserNames = true,
    RequireUniqueEmail = true
};

// Configure validation logic for passwords
manager.PasswordValidator = new PasswordValidator
{
    RequiredLength = 6,
    RequireNonLetterOrDigit = true,
    RequireDigit = false,
    RequireLowercase = false,
    RequireUppercase = false,
};
```

Figure 2: User name and password policy settings in IdentityConfig.cs file.

For user name: AllowOnlyAlphanumericUserNames - allow only [A-Za-z0-9@_.] in user names.

RequireUniqueEmail - Enforce emails to be non-empty, valid and unique.

For password: RequiredLength - Minimum required length of a password.

RequireNonLetterOrDigit - Require a non-letter or digit character.

RequireDigit - Require a digit [0-9].

RequireLowercase - Require a lower case letter [a-z].

RequireUppercase - Require an upper case letter [A-Z].

By default, the "RequiredLength" variable is set to "6" and the "RequireNonLetterOrDigit" variable is set to "true" as shown in Figure 2.

MODIFYING THE ACCOUNT/LOGIN.ASPX WEB FORM

By default, the "Login.aspx" web form requires an email and password to log in. To change it to require a user name instead, do the following.

- Open the "Login.aspx" web form in the Account folder and do the following.
 - For the "Email" TextBox control, change the ID property to "UserName" and the TextMode property to "SingleLine".
 - For the Label control associated with the "Email" TextBox control (now the "UserName" TextBox control), change the ControlToValidate property to "UserName" and the Text property to "User Name".
 - For the RequiredFieldValidator control associated with the "Email" TextBox control (now the "UserName" TextBox control), change the ControlToValidate property to "UserName" and the ErrorMessage property to "The user name field is required."

```

<asp:Content runat="server" ID="BodyContent" ContentPlaceHolderID="MainContent">
  <h2><%: Title %></h2>
  <p class="text-danger">
    <asp:Literal runat="server" ID="ErrorMessage" />
  </p>

  <div class="form-horizontal">
    <h4>Create a new account</h4>
    <hr />
    <asp:ValidationSummary runat="server" CssClass="text-danger" />

    <div class="form-group">
      <asp:Label runat="server" AssociatedControlID="Email" CssClass="col-md-2 control-label">Email</asp:Label>
      <div class="col-md-10">
        <asp:TextBox runat="server" ID="Email" CssClass="form-control" TextMode="Email" />
        <asp:RequiredFieldValidator runat="server" ControlToValidate="Email"
          CssClass="text-danger" ErrorMessage="The email field is required." />
      </div>
    </div>

    <div class="form-group">
      <asp:Label runat="server" AssociatedControlID="Password" CssClass="col-md-2 control-label">Password</asp:Label>
      <div class="col-md-10">
        <asp:TextBox runat="server" ID="Password" TextMode="Password" CssClass="form-control" />
        <asp:RequiredFieldValidator runat="server" ControlToValidate="Password"
          CssClass="text-danger" ErrorMessage="The password field is required." />
      </div>
    </div>

    <div class="form-group">
      <asp:Label runat="server" AssociatedControlID="ConfirmPassword" CssClass="col-md-2 control-label">Confirm password</asp:Label>
      <div class="col-md-10">
        <asp:TextBox runat="server" ID="ConfirmPassword" TextMode="Password" CssClass="form-control" />
        <asp:RequiredFieldValidator runat="server" ControlToValidate="ConfirmPassword"
          CssClass="text-danger" Display="Dynamic" ErrorMessage="The confirm password field is required." />
        <asp:CompareValidator runat="server" ControlToCompare="Password" ControlToValidate="ConfirmPassword"
          CssClass="text-danger" Display="Dynamic" ErrorMessage="The password and confirmation password do not match." />
      </div>
    </div>

    <div class="form-group">
      <div class="col-md-offset-2 col-md-10">
        <asp:Button runat="server" OnClick="CreateUser_Click" Text="Register" CssClass="btn btn-default" />
      </div>
    </div>
  </div>
</asp:Content>
  
```

Figure 3: Code to delete in the "Register.aspx" web form in the Account folder.

CREATING A CUSTOMIZED REGISTRATION PAGE

We will demonstrate how to use the web forms available in the Account folder of a web application to create a customized registration web page for the ASU application. Doing so allows us to use the existing ASP.NET Login controls and authentication infrastructure and not to have to create our own. You will need the "StudentRegistration.aspx" web form constructed in the first tutorial and modified in the validation tutorial.

Customize the Account/Register.aspx Web Form

IMPORTANT: Backup your ASUWebApplication before proceeding to carry out the following steps.

CAUTION: Perform the following 4 steps with extreme care and double check what you are copying and deleting!

1. In the Source tab of the "StudentRegistration.aspx" web form's Document window, select from the first <div class="form-group"> tag up to the closing </div> tag located just above the <div> tag for the Button control and **copy it**. **Be careful not to include the Button control!**
2. Expand the Account folder in the Solution Explorer and open the "Register.aspx" web form.
3. In the Source tab of the "Register.aspx" web form's Document window, select from the first <div class="form-group"> tag up to the closing </div> tag located just above the <div> tag for the Button control as shown in Figure 3 and **delete it**. **Be careful not to delete the Button control!**

Property	Value
ID	cvStudentId
ControlToValidate	StudentId
CssClass	text-danger
Display	Static
EnableClientScript	False
ErrorMessage	The student id already exists.
Text	*

Table 2: StudentId TextBox control CustomValidator properties.

4. In the Source tab of the “Register.aspx” web form’s Document window, **paste the text copied from the “StudentRegistration.aspx” web form** at the location just above the <div> tag for the Button control as shown in Figure 3. The modified “Register.aspx” web form should look as shown in Figure 4 when viewed in a browser.
5. Delete the CustomValidator control for the Password TextBox since the ASP.NET authentication infrastructure will handle password validation.
6. For the “Student Id” field, add a CustomValidator control and set its properties as shown in Table 2.
7. Set “EnableClientScript” to “False” for all Validation controls.

Figure 4: Modified “Register.aspx” web form viewed in a browser.

```
protected void CreateUser_Click(object sender, EventArgs e)
{
    if (Page.IsValid)
    {
        var manager = Context.GetOwinContext().GetUserManager<ApplicationUserManager>();
        var signInManager = Context.GetOwinContext().Get<ApplicationSignInManager>();
        var user = new ApplicationUser() { UserName = UserName.Text, Email = Email.Text };
        IdentityResult result = manager.Create(user, Password.Text);
        if (result.Succeeded)
        {
            // For more information on how to enable account confirmation and password reset please
            //string code = manager.GenerateEmailConfirmationToken(user.Id);
            //string callbackUrl = IdentityHelper.GetUserConfirmationRedirectUrl(code, user.Id, Request.QueryString["ReturnUrl"]);
            //manager.SendEmail(user.Id, "Confirm your account", "Please confirm your account by clicking the link below: " + callbackUrl);

            // Assign the user to the Student role.
            IdentityResult roleResult = manager.AddToRole(user.Id, "Student");
            if (!roleResult.Succeeded)
            {
                ErrorMessage.Text = roleResult.Errors.FirstOrDefault();
            }
            // Add the student information to the Student table.
            AddStudentRecord(StudentId.Text.Trim(), FirstName.Text.Trim(), LastName.Text.Trim(),
                Address.Text.Trim(), State.Text.Trim(), Country.Text.Trim(), UserName.Text.Trim());
            signInManager.SignIn(user, isPersistent: false, rememberBrowser: false);
            IdentityHelper.RedirectToReturnUrl(Request.QueryString["ReturnUrl"], Response);
        }
        else
        {
            ErrorMessage.Text = result.Errors.FirstOrDefault();
        }
    }
}
```

Figure 5: Modified “CreateUser_Click” event handler.

Customize the Account/Register.aspx.cs Code-behind File

1. Add the following two `using` statements at the top of the “Account/Register.aspx.cs” code-behind file.

```
using ASUWebApplication.Code_File;  
using System.Data.SqlClient;
```

2. The “CreateUser_Click” event handler will execute whenever the “Register” button is clicked regardless of whether the data entered in the “Register” web page is valid or not. To execute the code in the event handler only when all the validation controls pass validation, wrap all the code in the “CreateUser_Click” event handler in an `if` statement as shown in Figure 5.
3. By default, the user name is set to the email when a new account is created. Therefore, change the text “UserName = Email.Text” to “UserName = UserName.Text” on the third line of the “CreateUser_Click” event handler as shown by the small red rectangle in Figure 5.
4. Add the code from the “Add to CreateUser_Click.txt” file into the “CreateUser_Click” event handler immediately after the comments as shown by the large red rectangle in Figure 5⁴.
5. Add the code in the “AddStudentRecord.txt” file after the “CreateUser_Click” event handler. This code will insert a new student record into the *Enrollment.mdf* database if the user login account is successfully created.
6. Double click the “Country” DropDownList control to create the “Country_SelectedIndexChanged” event handler in the “Register.aspx.cs” code-behind file and add the code in the “Country_SelectedIndexChanged.txt” file from the “ASPNETSecurityTutorial-Download” folder into the “Country_SelectedIndexChanged” event handler.
7. Create the “cvStudentId_ServerValidate” event handler and add the code from the “cvStudentId_ServerValidate.txt” file in the “ASPNETSecurityTutorial-Download” folder into the “cvStudentId_ServerValidate” event handler.

MODIFY THE ENROLLMENT DATABASE

Modify the *Enrollment.mdf* database by adding a column “userName” with data type `nchar(15)` and null allowed to the *Student* table. This column is required to relate a logged in user to her student information.

TEST THE REGISTRATION PAGE

View the “Register.aspx” web form in a browser. Create an account with user name “typical” and password “typical#”. Use any values for the other fields. If you are successfully registered, then you will be redirected to the *ASUWebApplication* directory (default web page) as no URL has been specified for redirection after log in. Moreover, the web page will show a greeting that includes the user name of the logged in user as shown in the upper right of Figure 6.

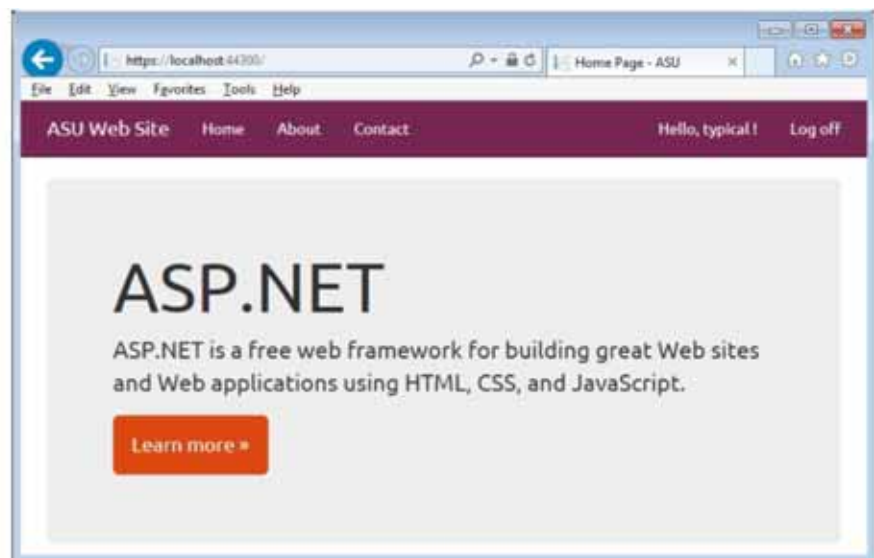


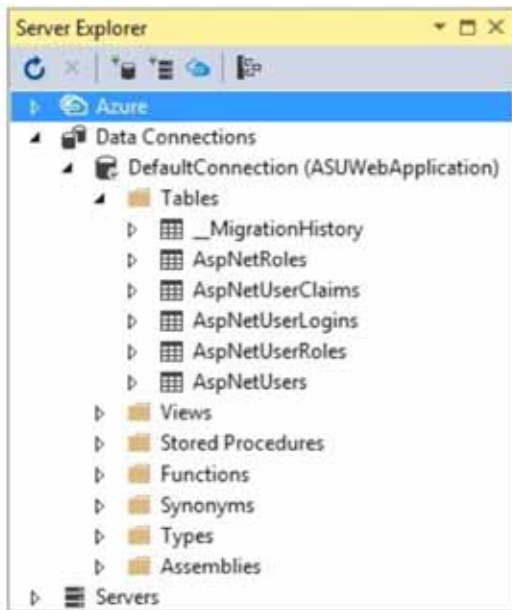
Figure 6: Web page showing logged in user “typical”.

ASP.NET IDENTITY DATABASE

The default database in which ASP.NET stores user credentials and roles can be found in the *App_Data* folder in the *Solution Explorer* and can be examined using the *Server Explorer* window as shown in Figure 7(a). Moreover, if you examine the data in the *AspNetUsers* (Figure 7(b)), *AspNetUserRoles* (Figure 7(c)) and *AspNetRoles* (Figure 7(d)) tables, you can see that there are three users, “marvin”, “typical” and

⁴ As *Identity Manager* uses claims to assign user to roles, users assigned to roles by the code in Figure 5 will not appear as assigned to the role in *Identity Manager*. However, they are assigned to the role in the ASP.NET Identity Database as shown for user “typical” in Figure 7.

“winnie”. By examining the user and role ids, you can see that the user “marvin” is in the role “Admin”, user “winnie” is in the role “Admin” and “Student” and user “typical” is in the role “Student”. You can access and



(a) Default ASP.NET Identity database.

Id	UserName
4e759a3a-2ac2-4586-8c77-198e70937fe6	marvin
ab82e77c-b2d6-4c2e-a160-480b33db6616	typical
fb0a5c91-3e17-4349-b498-431c80e99da9	winnie

(b) Registered users.

Userid	Roleid
ab82e77c-b2d6-4c2e-a160-480b33db6616	528ebd58-337f-4c97-be33-bc1218ba5cc6
fb0a5c91-3e17-4349-b498-431c80e99da9	528ebd58-337f-4c97-be33-bc1218ba5cc6
4e759a3a-2ac2-4586-8c77-198e70937fe6	914801b9-c9f7-49c3-95fd-ea039811152c
fb0a5c91-3e17-4349-b498-431c80e99da9	914801b9-c9f7-49c3-95fd-ea039811152c

(c) Roles assigned to users.

Id	Name
914801b9-c9f7-49c3-95fd-ea039811152c	Admin
528ebd58-337f-4c97-be33-bc1218ba5cc6	Student

(d) Defined roles.

Figure 7: Tables used by ASP.NET Identity.

manipulate the data in these tables directly in your application, if necessary.

CREATING SECURE WEB PAGES

We will demonstrate how to create a web page in ASP.NET that is available only to logged in users using roles. When a user that is not logged in tries to access a secured web page, she will be redirected to a login page where she can either log in or register for an account.

1. Right-click the ASUWebApplication node in the Solution Explorer, select “New Folder” from the pop up menu and name the folder StudentOnly as shown in Figure 8.
2. Right-click the StudentOnly folder in the Solution Explorer, add a new web form and name it “StudentPage.aspx” as shown in Figure 8.
3. Add two HyperLink controls to the “StudentPage.aspx” web form as shown in Figure 10(a) and set their properties as shown in the table beside the web form.
4. Drag the following files *all together at one time*
 - “EditStudentInformation.aspx”
 - “EditStudentInformation.aspx.cs”
 - “EditStudentInformation.aspx.designer.cs”
 - “ViewStudentInformation.aspx”
 - “ViewStudentInformation.aspx.cs”
 - “ViewStudentInformation.aspx.designer.cs”
 from the “ASPNETSecurityTutorial-Download” folder into the StudentOnly folder in the Solution Explorer as shown in Figure 8.
8. Right-click the ASUWebApplication node in the Solution Explorer, select “New Folder” from the pop up menu and name the folder AdminOnly as shown in Figure 8.
9. In the Solution Explorer, drag the following files, constructed in previous tutorials, *all together at one time*
 - “StudentDetails.aspx”
 - “StudentDetails.aspx.cs”,

- “StudentDetails.aspx.designer.cs”
- “StudentRecords.aspx”
- “StudentRecords.aspx.cs”
- “StudentRecords.aspx.designer.cs”
- “StudentSearch.aspx”
- “StudentSearch.aspx.cs”
- “StudentSearch.aspx.designer.cs”

into the AdminOnly folder as shown in Figure 8.

10. Right-click the AdminOnly folder in the Solution Explorer, add a new web form and name it “AdminPage.aspx” as shown in Figure 8.
11. Add two HyperLink controls to the “AdminPage.aspx” web form as shown in Figure 10(b) and set their properties as shown in the table beside the web form.
12. Add two HyperLink controls to the “Default.aspx” web form as shown in Figure 10(c) and set their properties as shown in the table beside the web form.

Securing a Web Page

Web pages are secured by setting some properties in the Web.config file of the folder in which the web pages reside.

1. In the Solution Explorer, right-click the AdminOnly folder and select “Add→New Item” from the pop up menu.
2. In the Add New Item dialog box, select “Visual C#→Web” in the “Installed” pane and “Web Configuration File” in the middle pane.
3. In the Web.config file in the AdminOnly folder, add the following code after the <system.web> tag as shown by the highlighted section in Figure 9.

```
<authorization>
  <deny users="?" />
</authorization>
```

This code secures all pages in the AdminOnly folder by denying access to anonymous users (i.e., users who are not authenticated), which are represented by the <deny users="?" /> tag. The result is that any user that tries to access any web page in the AdminOnly folder is now forced to log in.

4. Add a similar Web.config file in the StudentOnly folder with the same code as in the AdminOnly folder to also require users to log in to access the web pages in the StudentOnly folder.
5. Set the “Default.aspx” web form as the start page for the web application by right-clicking on the file in the Solution Explorer and selecting “Set As Start Page” from the popup menu.
6. View the “Default.aspx” web form in a browser.

The web page will appear as shown in Figure 10(c). Clicking the “For Students Only” link in the page will redirect you to the “Log in” page shown in Figure 11(a). This page is located in the Account folder of the web application and is named “Login.aspx”. Login with the user name and password registered previously (i.e., user name “typical” and password “typical#”), you will be automatically redirected to the “Student Information” web page shown in Figure 10(a). Note that the LoginName control, located in the “Site.Master” master page, now displays the user name of the logged in user as shown in Figure 11(b). Moreover, since you successfully logged in, “Logout” is now displayed by the LoginStatus control as shown in Figure 11(b) rather than “Login”.

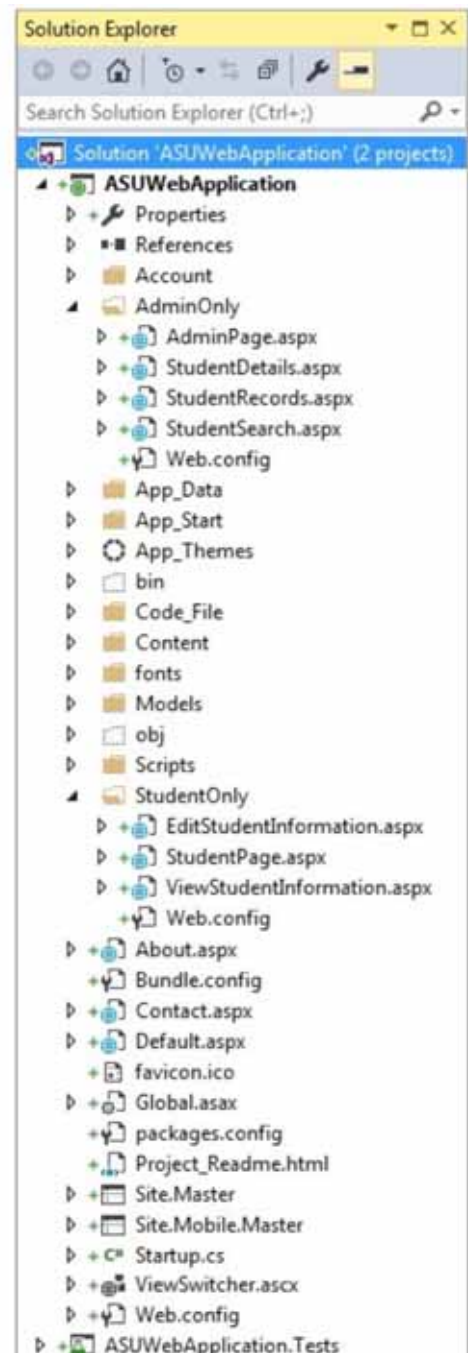


Figure 8: Web application files.



Figure 9: Web.config file in StudentOnly folder



(a) StudentPage.aspx

Control	Property	Value
HyperLink1	Text	View Your Information
	NavigateUrl	~/StudentOnly/ViewStudentInformation.aspx
	ID	
HyperLink2	Text	Edit Your Information
	NavigateUrl	~/StudentOnly/EditStudentInformation.aspx
	ID	

Properties of HyperLink controls for "StudentPage.aspx"



(b) AdminPage.aspx

Control	Property	Value
HyperLink1	Text	Search Student Records
	NavigateUrl ¹	~/AdminOnly/StudentSearch.aspx
	ID	
HyperLink2	Text	Manage Student Records
	NavigateUrl	~/AdminOnly/StudentRecords.aspx
	ID	

Properties of HyperLink controls for "AdminPage.aspx"



(c) Default.aspx

Control	Property	Value
HyperLink1	Text	For Students Only
	NavigateUrl	~/StudentOnly/StudentPage.aspx
	ID	
HyperLink2	Text	For Admin Only
	NavigateUrl	~/AdminOnly/AdminPage.aspx
	ID	

Properties of HyperLink controls for "Default.aspx"

Figure 10: StudentPage.aspx, AdminPage.aspx and Default.aspx layout and HyperLink controls.

Return to the "Default.aspx" web page and click the link "For Admin Only". Since you are logged in, you also have access to the web pages in the `AdminOnly` folder as we have only specified that a user be logged in to access the web pages in the `AdminOnly` folder. To restrict access to the web pages in the `AdminOnly` folder to users assigned to the "Admin" role it is necessary to specify additional access rules in a folder's `Web.config` file.

Specifying Role Access Rules

We will demonstrate how to restrict access to the `AdminOnly` pages only to users assigned to the "Admin" role (i.e., to "marvin" and "winnie") and to the `StudentOnly` page only to users assigned to the "Student" role (i.e., "typical" and "winnie").

1. Open the `Web.config` file in the `AdminOnly` folder and add the following two lines after the line `<deny users="?" />` as shown in Figure 12(a).


```
<allow roles="Admin" />
<deny users="*" />
```
2. Similarly, open the `Web.config` file in the `StudentOnly` folder and add the following two lines after the line `<deny users="?" />` as shown in Figure 12(b).


```
<allow roles="Student" />
<deny users="*" />
```

(a) Log In page.

(b) LoginName and LoginStatus controls after successful log in.

Figure 11: Web site log in.

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <authorization>
      <deny users="?" />
      <allow roles="Admin" />
      <deny users="*" />
    </authorization>
  </system.web>
</configuration>
```

(a) Access rules in Web.config file in AdminOnly folder.

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <authorization>
      <deny users="?" />
      <allow roles="Student" />
      <deny users="*" />
    </authorization>
  </system.web>
</configuration>
```

(b) Access rules in Web.config file in StudentOnly folder.

Figure 12: Access rules for web application folders.

In Figure 12(a), the first rule in the highlight box, as before, denies access to anonymous users requiring them to login to access web pages in the folder. The second rule allows access to all users who are logged in as the “Admin” role. The third rule denies access to all other users. Rules are processed in the order encountered. **Thus, the order of the rules is important.** For example, if the third rule were placed before the second rule, then access to all users would be denied, even those in the “Admin” role. The rules for the StudentOnly folder have a similar meaning.

In summary, the authorization rules as set in Figure 12(a) and (b) will require a login to access both the AdminOnly web pages and the StudentOnly web pages by denying access to anonymous users. Access to the AdminOnly web pages is restricted to only users assigned to the “Admin” role (by denying access to all other users). Access to the StudentOnly web pages is restricted to only users assigned to the “Student” role (by denying access to all other users).

Assigning Users to Roles Programmatically

The Role class in the .NET Framework provides programmatic access to the role service through a number of methods some of which are:

- CreateRole(roleName) – adds a new role to the system.
- DeleteRole(roleName) – deletes a role from the system.
- AddUserToRole(userName, roleName) – adds a particular user to a particular role.
- IsUserInRole(roleName) / IsUserInRole(userName, roleName) – returns true or false, depending on whether the currently logged in user or the user specified is in the specified role.

- `GetAllRoles()` – returns a string array of all of the roles in the system.
- `GetRolesForUser()` / `GetRolesForUser(userName)` – returns a string array of all of the roles to which either the currently logged in user or the user specified belongs.

These methods can be used to display the roles in the system on a web page, list the roles to which a user belongs or determine whether or not a user belongs to a particular role.

For the ASU application, to assign a user to the “Student” role when she logs in, the following code was added to the “CreateUser_Click” event handler of the “Register.aspx.cs” file (see Figure 5).

```
// Assign the user to the Student role.  
IdentityResult roleResult = manager.AddToRole(User.Id, "Student");
```

Testing the Role-based Security

View the “Default.aspx” web form in a browser.

- Log in as user “winnie”. In the “Default.aspx” web page, click the link “For Admin Only”. You are granted access. Return to the “Default.aspx” web page and click the link “For Students Only”. You are also granted access to this page.
- Log out and log in as user “marvin”. In the “Default.aspx” web page, click the link “For Admin Only”. You are granted access. Return to the “Default.aspx” web page and click the link “For Students Only”. You are redirected to the “LogIn.aspx” web page since you are not in the role “Student” and so are not allowed to access this page.
- Log out and log in as user “typical”. In the “Default.aspx” web page, click the link “For Admin Only”. You are redirected to the “LogIn.aspx” web page since you are not in the role “Admin” and so are not allowed to access this page. Return to the “Default.aspx” web page and click the link “For Students Only”. You are granted access.

APPENDIX A: IDENTITY MANAGER

Installing Identity Manager

Identity Manager is a third party tool that can be used for creating roles and managing users in Visual Studio⁵. This tool needs to be installed and configured for use in Visual Studio.

1. In the Visual Studio toolbar, select “Tools→NuGet Package Manager→Package Manager Console”.
2. At the “PM>” prompt, enter the following commands *one at a time*. Wait for the first command to finish execution before entering the second one.

Be patient as it can take several minutes for each package to install.

```
Install-Package IdentityManager -Pre
```

```
Install-Package IdentityManager.AspNetIdentity -Pre
```

3. From the “IdentityManager-Download” folder, do the following.

Note: As the download code for *Identity Manager* is customized according to the name of the web application, you must download the “IdentityManager-Download.zip” file that is specific for a web application.

- a. Drop the “Startup.cs” file onto the root web application node (e.g., ASUWebApplication) and select “Yes” when prompted to replace the existing “Startup.cs” file. This file executes when a web application starts and has been modified to include the *Identity Manager* tool.
 - b. Drop the “IdentityConfig.cs” file onto the App_Start folder. This file has been modified to work with the modified “Startup.cs” file.
4. Open the Web.config file, find the <modules> tag inside the <system.webServer> tag and add the following property inside the <modules> tag as shown in Figure 13.

```
runAllManagedModulesForAllRequests=
"true"
```

Figure 13: Configuring the Identity Manager tool.

5. Open a Visual Studio browser page (i.e., start debugging without any web form open) and enter “idm” at the end of the URL in the browser address field to access *Identity Manager* (see Figure 14).

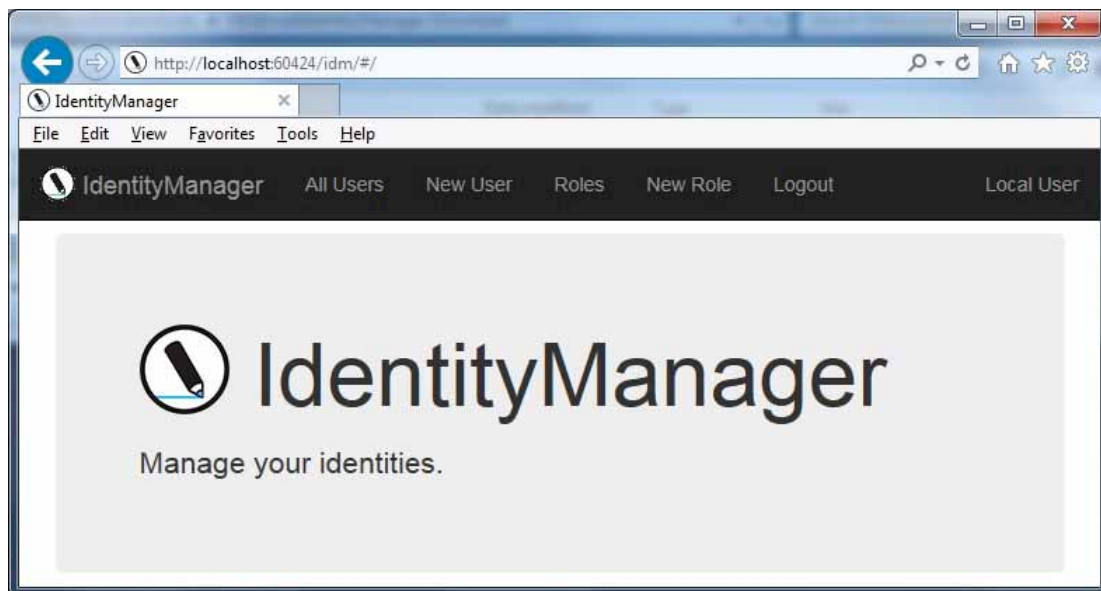


Figure 14: *Identity Manager* tool.

Creating Roles

To create a role, select the “New Role” tab shown in Figure 14 and enter a role name into the textbox shown in Figure 15(a).

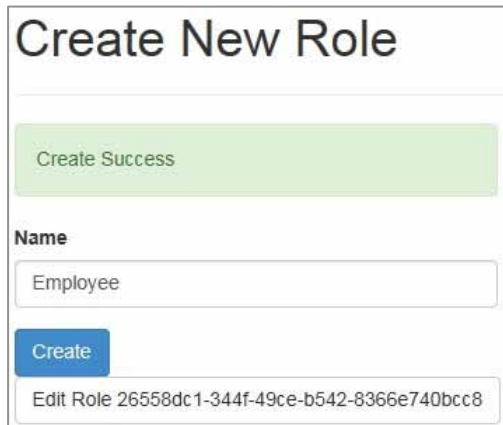
⁵ Visual Studio does not have a tool for creating roles and managing users.

Creating Users

To create a user, select the “New User” tab shown in Figure 14, and enter a User name and password into the text boxes shown in Figure 15(b).

Assigning Roles to Users

A role can be assigned to a user after she has been created. Select the “All Users” tab shown in Figure 14 and then select the “Edit” button to the left of the user as shown in the top figure of Figure 15(c). In the “Edit User” page select the “Roles” tab and finally check the checkbox to the left of a role name as shown in the bottom figure of Figure 15(c).



Create New Role

Create Success


Name

Employee

Create

Edit Role 26558dc1-344f-49ce-b542-8366e740bcc8

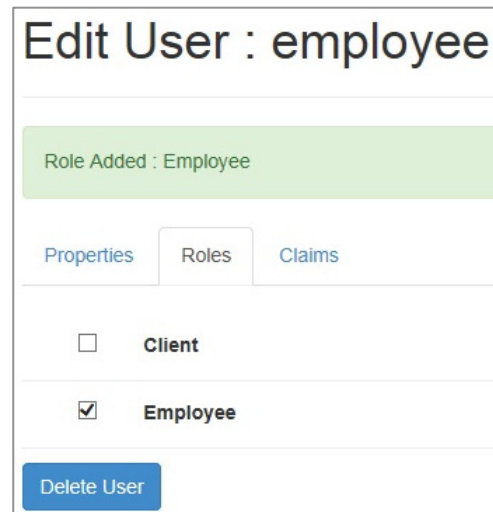
(a) Creating a role.



Users 1 found

Search

Name	Username	Subject
edit	employee	8bc8626d-c8



Edit User : employee

Role Added : Employee

Properties Roles Claims

☐ Client

☒ Employee

Delete User

(c) Assigning a role to a user.



Create New User

Create Success

UserName

employee

Password

Confirm Password

Create

Edit User 8bc8626d-c89a-4e7a-a618-afe90c692045

(b) Creating a user.

Figure 15: Creating roles, users and assigning roles to users.

APPENDIX B: ASP.NET LOGIN CONTROLS⁶

The Login controls are used to allow only registered users to access protected contents on a web site. A registration process and Login system can be created using these controls.

ChangePassword

The ChangePassword control is used to change the current password. In order to change the password, the current password must be provided. The new password must be typed two times. The DisplayUsername property allows the username to be shown, which is useful if authentication of users is not required before they can change their password. The ChangePassword control can be configured to send the new password to the user by email.

CreateUserWizard Control

The CreateUserWizard control can collect the following information from potential users.

- User name
- Password
- Confirmation of password
- E-mail address
- Security question
- Security answer

By default, the CreateUserWizard control collects only a user name and password from the Web site visitor and adds the new user to the ASP.NET membership system. Based on the requirements of the site's MembershipProvider object, the CreateUserWizard control can be configured to collect the additional information. The CreateUserWizard control can optionally send email messages to new users if you have configured an SMTP mail server to send email and the fields of the MailDefinition property have been set. Upon registering, the user's details are added to the web site membership database. An important property of this control is the ContinueDestinationPageUrl, which contains the name of the page where the user will be redirected to after successfully submitting the registration form.

Login

The Login control is a composite control (i.e., it is composed of several controls) that displays a standard user login form with two text boxes and a button. A user can use this control to input their username and password in order to access private content on a web site. If you try to access a page that requires authentication, ASP.NET will automatically redirect you to the Login page and, after you have been authenticated, you will be redirected back to the original page that you wanted to access. The Login control provides the following optional UI elements that support additional functions.

- A link for a password reminder.
- A Remember Me checkbox for retaining the login information between sessions.
- A Help link for users who are having trouble logging in.
- A Register New User link that redirects users to a registration page.
- Instruction text that appears on the login form.
- Custom error text that appears when the user clicks the login button without filling in the user name or password fields.
- Custom error text that appears if the login fails.
- A custom action that occurs when login succeeds.
- A way to hide the login control if the user is already logged in to the site.

LoginName

The LoginName control displays the name contained in the User property of the Page class when the user is logged in (i.e., authenticated). If the user is not logged in, then it displays nothing.

LoginStatus

The LoginStatus control displays a login link for users who are not authenticated and a logout link for users who are authenticated. The login link takes the user to a login page. The logout link resets the current user's identity to be an anonymous user and, when using cookies, will clear the cookie from the user's client

⁶ More information about the Login controls can be obtained at <https://msdn.microsoft.com/en-us/library/ms178329%28v=vs.100%29.aspx>.

computer. The LoginStatus control's appearance can be customized by setting the LoginText and LoginImageUrl properties. The state of the LoginStatus control, logged in or logged out, is determined by the IsAuthenticated property of the Page object's Request property. The LogOutAction property of the LoginStatus control can be used to specify what happens when a user clicks on the Log Out link. The three options are:

- *Redirect* – Redirects the user to the URL contained in the LogoutPageUrl property. If LogoutPageUrl is empty, the user is redirected to the login page defined in the application configuration settings.
- *RedirectToLoginPage* – Redirects the user to the login page defined in the application configuration settings.
- *Refresh* – Refreshes the current page.

LoginView

The LoginView control displays different information to anonymous and logged-in users. The following templates can be defined.

- *AnonymousTemplate* – Specifies the template to display to users who are not logged in to the web site. Users who are logged in will never see this template.
- *LoggedInTemplate* – Specifies the default template to display to users who are logged in to the web site (i.e., authenticated users). This template is displayed to users when both the Name property of the Page User property is not null and the user does not belong to a role group defined in the RoleGroups property.

PasswordRecovery

The PasswordRecovery control assists users who have forgotten their passwords by enabling a user to request an email message containing either a new password or the password already associated with his or her user name. Users can recover passwords only when the membership provider defined in the MembershipProvider property supports clear text or encrypted passwords. Since hashed passwords cannot be recovered, users at sites that use hashed passwords can only reset their passwords. You can configure membership to include a security question that the user must answer to recover a password. If you do, the PasswordRecovery control asks the question and checks the answer before recovering the password. The PasswordRecovery control requires that your application can forward email messages to a Simple Mail Transfer Protocol (SMTP) server. You can customize the text and format of the email message sent to the user by setting the MailDefinition property.