



ASP.NET – 4.5 Lab Book

Table of Contents

Getting Started	4
Overview	4
Setup Checklist for ASP.NET	4
Instructions	4
Lab 1. Creating Sign in and Sign up pages	6
Develop an Asp.Net application having web pages for:	6
1.1. Creating a Mail Account	6
a. Mail User Name should not be empty	6
b. Password should be at least 7 characters long. The first alphabet should be capital. Should have at least 2 digits somewhere in it.	6
c. Accept confirm password as well. And the Password and Confirm Password entries should match	6
d. Account Creation Date should be current date.	6
e. Accept hobbies using Checkboxes (like Reading, Drawing, Sports ...) and check whether at least one checkbox is selected.	6
1.2. Sign In.....	6
a. Mail User Id and Password should not be empty.	6
b. Password should be at least 7 characters long. The first alphabet should be capital. Should have at least 2 digits somewhere in it.	6
c. Add a hyperlink to this page, so that if user is new he can navigate to "Create Mail Account" page.	6
Create the above pages based on "MailUsers" table.	6
1.3. Modify the SignIn Page and use HTML5 Controls	6
1.4. View the pages with the Page Inspector.	6
Lab 2. Creating Master Page and Themes and applying	7
2.2. Create 2 Themes, Royal Blue and Royal Red.	7
2.3 Create an Asp.Net page based on the Master page created in 2.1 to Send Mail. 7	
Lab 3. Creating a Business Service Layer in form of DLL.....	8
3.1: Create a separate DLL	8
Lab 4. Using State Management Techniques	9



Lab 5.	Using Application_Error Event.....	10
Lab 6.	Using Data Bind Controls	13
Lab 7.	ASP.NET Identity	14
Lab 8.	Deployment in ASP.NET	31



Getting Started

Overview

This lab book is a guided tour for learning ASP.NET version 4.5. Follow the steps provided and work out the assignments given.

Setup Checklist for ASP.NET

Here is what is expected on your machine for the lab to work.

Minimum System Requirements

- Intel Pentium 4 and above
- Microsoft Windows 7 or 8
- Memory: 1 GB of RAM (2GB or more recommended)
- Internet Explorer 9.0 or higher, Google Chrome
- SQL Server 2012 client and access to SQL Server 2012 server
- Visual Studio 2013/2015.

Instructions

- Create a directory by your name in drive <drive>. In this directory, create a subdirectory Asp.Net_assgn. For each lab exercise create a directory as lab <lab number>.
- You may also look up the on-line help provided in the MSDN library.

Create the following database Tables:

Table Name: MailUsers

Column Name	Data Type
MailUserId	Char (PK)
MailUserName	Char
MailUserPwd	Char
AccountCreationDate	DateTime
Hobbies	Char

Table Name: MailsInBox

Column Name	Data Type
MailTranNo	Int (PK) Identity
FromMailUserId	Char (FK)
ToMailUserId	Char (FK)
Subject	Char
MailText	Char
IsMailNew	Char
MailDateTime	DateTime

Lab 1. Creating Sign in and Sign up pages

Goals	To learn <ol style="list-style-type: none"> 1. Creating asp.net web pages 2. Using built in asp.net controls 3. Using Validation Controls to validate Input.
Time	2 Hrs.

Develop an Asp.Net application having web pages for:

1.1. Creating a Mail Account

- a. Mail User Name should not be empty
- b. Password should be at least 7 characters long. The first alphabet should be capital. Should have at least 2 digits somewhere in it.
- c. Accept confirm password as well. And the Password and Confirm Password entries should match
- d. Account Creation Date should be current date.
- e. Accept hobbies using Checkboxes (like Reading, Drawing, Sports ...) and check whether at least one checkbox is selected.

1.2. Sign In

- a. Mail User Id and Password should not be empty.
- b. Password should be at least 7 characters long. The first alphabet should be capital. Should have at least 2 digits somewhere in it.
- c. Add a hyperlink to this page, so that if user is new he can navigate to “Create Mail Account” page.

Create the above pages based on “MailUsers” table.

1.3. Modify the SignIn Page and use HTML5 Controls

1.4. View the pages with the Page Inspector.

Lab 2. Creating Master Page and Themes and applying

Goals	To learn 1. Creating asp.net web page based on Master 2. Creating asp.net themes 3. Applying Master page and theme to application
Time	4 Hrs.

2.1. Create a Master Page which has

Top Header section showing the Company Logo and Name.

Left Side Bar and Section to display Content Page Information.

2.2. Create 2 Themes, Royal Blue and Royal Red.

Royal Blue Theme:

- Textboxes, Labels, Checkbox, Dropdown, Radio buttons should have forecolor: Dark Blue
- Buttons should have Back color: Dark Blue and fore color: White.
- For all controls set font to verdana, size 11 pt.
- Top Header section and Left Side Bar section should have dark blue as background color.

Royal Red Theme:

- Textboxes, Labels, Checkbox, Dropdown, Radio buttons should have fore color: Dark Red
- Buttons should have Back color: Dark Red and fore color: White.
- For all controls set font to verdana, size 11 pt.
- Top Header section and Left Side Bar section should have dark red as background color.

Apply the theme at application level using web.config.

2.3 Create an Asp.Net page based on the Master page created in 2.1 to Send Mail.

This page will accept

- "To" Mail User Id
- Subject
- Mail Body Text

The page will have send Mail Button.

Validate that all the entries should be mandatory.

Modify the above page with Html5 Validation

Lab 3. Creating a Business Service Layer in form of DLL

Goals	<p>To Learn:</p> <ul style="list-style-type: none"> • Creating BL DLL • Creating BL service class. Identify the service signature • Using Entity Classes and Collection classes to exchange data between layers. • Use BL service.
Time	10 Hrs.

3.1: Create a separate DLL

- Define Entity classes corresponding to the database tables
- Create a Business Layer (BL) class "MailService" which will have functions to SignIn, SignUp, Send Mail and Get the Mails from Inbox.

Identify what will be the function names and their signatures

For example:

SignIn function will accept MailUserID and Password as parameters.

Signup function will accept parameters one for each column in "MailUsers" tables except for AccountCreationDate column.

3.2: Modify the SigIn, Signup, and SendMail pages created in earlier lab so that the pages now make calls to the corresponding functions written in the BL class of Lab 3.1

3.3: Use JQuery concepts wherever required.

3.4: Implement Strongly Typed Data Controls for the Send Mail.aspx

Lab 4. Using State Management Techniques

Goals	<p>To Learn:</p> <ul style="list-style-type: none"> Using server side & client-side state management Using GridView Control to show multiple details and customize grid. Link multiple asp.net pages with each other
Time	10Hrs

4.1: Update the SignIn page so that once a user successfully signs in, store the user id and user name either in session OR cookie and redirect the user to Inbox Page

4.2: Create one more asp.net page "InBox.aspx" page to show Mails from Inbox for currently logged in user.

- The Inbox should show only the Sender User Id, Mail Subject, and Mail Sent DateTime in a Grid View.
- Show Mail Subject column as hyperlink.

4.3: Create an asp.net page to ShowMailText.aspx.

When user clicks on the Mail Subject hyperlink column displayed in Lab 4.2, he should be directed to this page and the mail contents of the mail should be shown here.

4.4: Update the SignIn page so that once user successfully signs in, store the user id and user name in Local Storage or Session Storage and redirect user to Inbox Page

Lab 5. Using Application_Error Event

Goals	To Learn: <ul style="list-style-type: none"> Using Application_Error Event
Time	1 Hr.

5.1 : Implementing Application_Error Event

1. Start Visual Studio 2013/2015
2. Click New Project from the Start page, or you can use the menu and select File, and then New Project.
3. Select Visual C# in the left pane, then Web and then select ASP.NET Web Application. Name your project "ApplicationErrorDemo" and then click OK.
4. In the New ASP.NET Project dialog, select the Empty template. Click OK
5. Now create Home.aspx Page and write a code in Home.aspx as below :

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Home.aspx.cs"
Inherits="ApplicationErrorDemo.Home" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
  <head runat="server">
    <title></title>
  </head>
  <body>
    <form id="form1" runat="server">
      <div>
        <asp:Button ID="btnSubmit" runat="server" Text="Submit"
OnClick="btnSubmit_Click" />
      </div>
    </form>
  </body>
</html>
```

6. Write a following code on button click event of Submit in Home.aspx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace ApplicationErrorDemo
{
    public partial class Home : System.Web.UI.Page
    {
        protected void btnSubmit_Click(object sender, EventArgs e)
        {
            int num1, num2;
            num1 = 12;
            num2 = 0;
            int div = num1 / num2;
        }
    }
}
```

7. Create one more page ErrorPage.aspx as below:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="ErrorPage.aspx.cs" Inherits="ApplicationErrorDemo.ErrorPage" %>

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
<div>
<h1>Error Page</h1>
</div>
</form>
</body>
</html>
```

8. Now add the Global.asax file to your application. Add the following code to Application_Error event handler

```
protected void Application_Error(object sender, EventArgs e)
{
    Exception ex = Server.GetLastError();
```

```
Server.ClearError();  
Server.Transfer("ErrorPage.aspx");  
}
```

9. Now open Visual Studio Developer command prompt and run following commands

```
C:\Program Files (x86)\Microsoft Visual Studio 13.0>cd..  
C:\Program Files (x86)>cd IIS Express  
C:\Program Files (x86)\IIS Express> appcmd set config  
/section:system.webServer/directoryBrowse /enabled:true
```

10. Now run your application without debugging.

Lab 6. Using Data Bind Controls

Goals	<p>To Learn:</p> <ul style="list-style-type: none"> Using DetailsView /GridView Control to show multiple details and customize data display. Changing Themes at runtime.
Time	6 Hrs.

6.1: In the Lab 4.2, add an option for user to delete certain existing mails.
In the MailUsers table add one more Colum Display Preference (Char Data Type) which will have value either “tabular” or “block”. Default Value should be “tabular”

6.2: Create an asp.net page “Change Profile” to let user modify this profile setting.

- This page should ask for Display Format from user like tabular or block, and accordingly save this setting in the Mail Users table.
- Add a new function into BL service class MailService which will be called from this page to change the profile setting.

If user sets “tabular” display style in profile, then use Grid View control in Lab 4.2 asp .net page to display mails list. For “block” display style use DetailsView control to display the mails list.

Hint: You can use 2 different pages to show the Mails List in 2 different formats (Grid View Control / DetailsView Control) and redirect the user to one of these pages from the sign in page based on this profile setting.

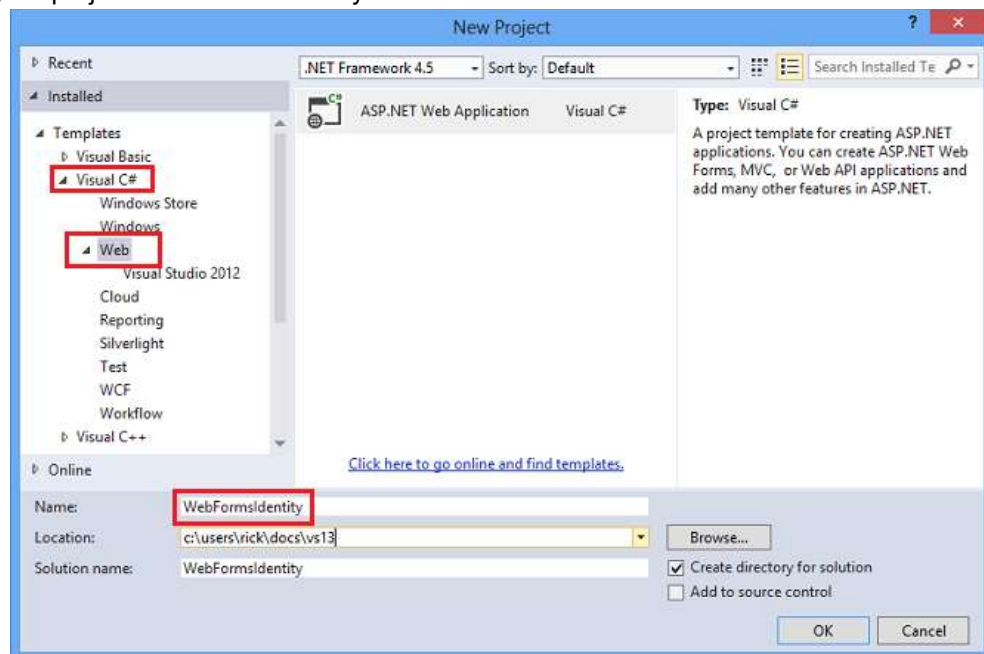
6.3: In the Master Page created in Lab 2.1, add a hyperlink to Left Side Bar, which points to the Change Profile asp.net page.
Also add 2 Buttons with text “Royal Blue Theme” and “Royal Red Theme” in the Left Side Bar. When user clicks these buttons change the theme applied.
Include SiteMap control to show the Navigation path

Lab 7. ASP.NET Identity

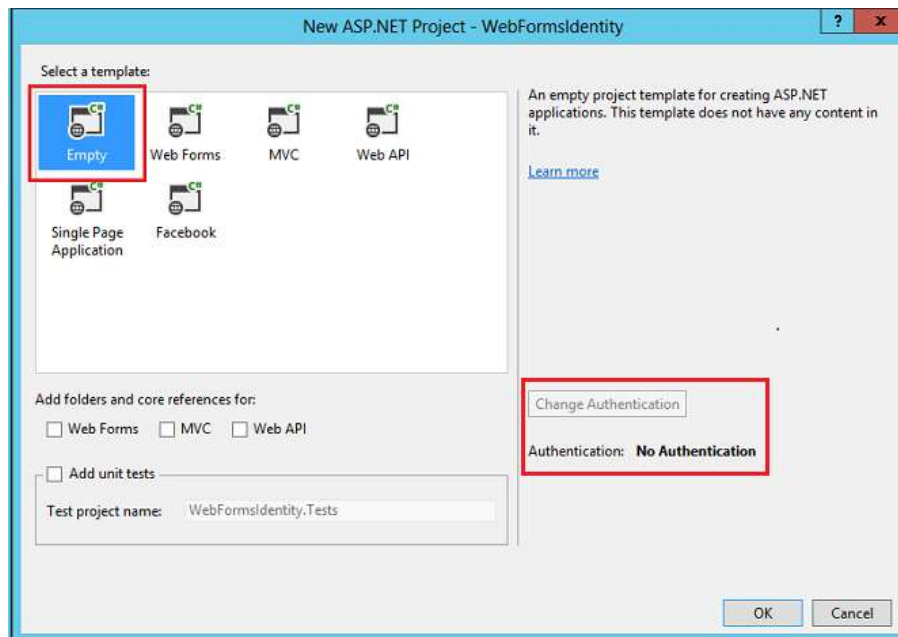
Goals	To Learn: <ul style="list-style-type: none"> Using ASP.NET Identity
Time	1 Hrs.

7.1: Implement Forms Based Authentication for SignIn page

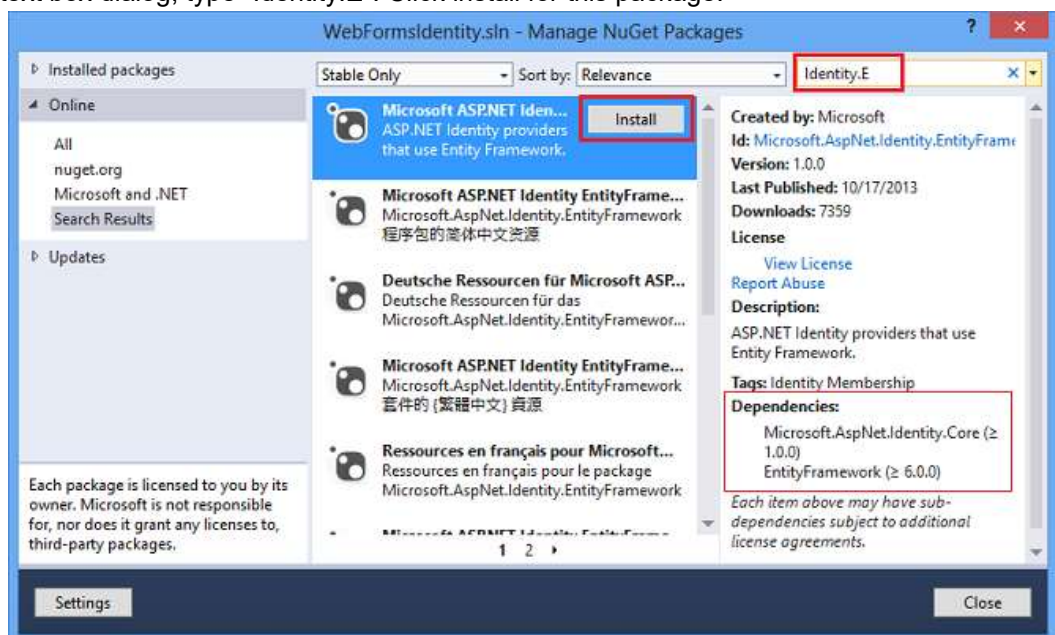
- Start running Visual Studio 2013.
- Click New Project from the Start page, or you can use the menu and select File, and then New Project.
- Select Visual C# in the left pane, then Web and then select ASP.NET Web Application. Name your project "WebFormsIdentity" and then click OK.



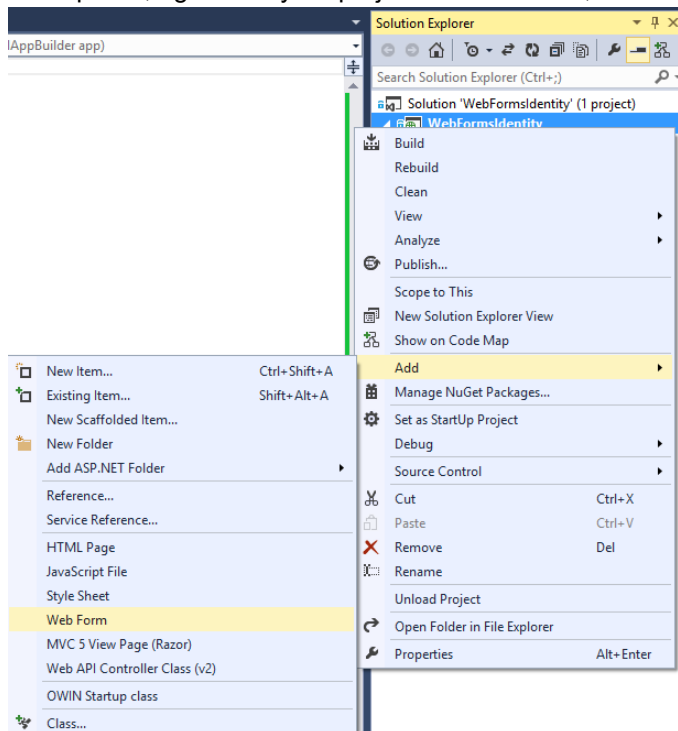
- In the New ASP.NET Project dialog, select the Empty template.



- Notice the Change Authentication button is disabled, and no authentication support is provided in this template. The Web Forms, MVC and Web API templates allow you to select the authentication approach.
- Adding Identity Packages to your App
- In Solution Explorer, right-click your project and select Manage NuGet Packages. In the search text box dialog, type "Identity.E". Click install for this package.



- Note that this package will install the dependency packages: EntityFramework and Microsoft ASP.NET Identity Core.
- Adding Web Forms to Register Users
- In Solution Explorer, right-click your project and click Add, and then Web Form.



- In the Specify Name for Item dialog box, name the new web form Register, and then click OK
- Replace the mark-up in the generated Register.aspx file with the code below.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind ="Register.aspx.cs"
Inherits="WebFormsIdentity.Register" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body style="font-family: Arial, Helvetica, sans-serif; font-size: small">
    <form id="form1" runat="server">
        <div>
            <h4 style="font-size: medium">Register a new user</h4>
            <hr />
            <p>
                <asp.Literal runat="server" ID="StatusMessage" />
            </p>
        </div>
    </form>
</body>
</html>
```



```
</p>
<div style="margin-bottom:10px">
    <asp:Label runat="server" AssociatedControlID ="UserName">User
name</asp:Label>
    <div>
        <asp:TextBox runat="server" ID="UserName" />
    </div>
</div>
<div style="margin-bottom:10px">
    <asp:Label runat="server" AssociatedControlID
="Password">Password</asp:Label>
    <div>
        <asp:TextBox runat="server" ID="Password" TextMode="Password" />
    </div>
</div>
<div style="margin-bottom:10px">
    <asp:Label runat="server" AssociatedControlID="ConfirmPassword">Confirm
password</asp:Label>
    <div>
        <asp:TextBox runat="server" ID="ConfirmPassword" TextMode="Password" />
    </div>
</div>
<div>
    <div>
        <asp:Button runat="server" OnClick="CreateUser_Click" Text="Register" />
    </div>
</div>
</div>
</form>
</body>
</html>
```

- Note: This is just a simplified version of the Register.aspx file that is created when you create a new ASP.NET Web Forms project. The markup above adds form fields and a button to register a new user.
- Open the Register.aspx.cs file and replace the contents of the file with the following code:

```
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.EntityFramework;
using System;
using System.Linq;

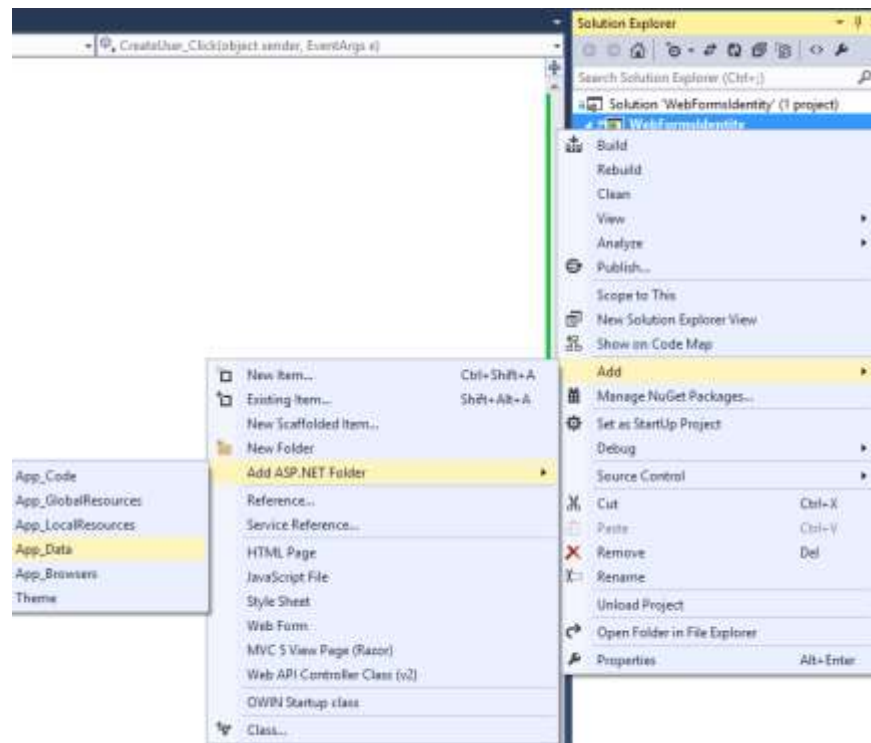
namespace WebFormsIdentity
{
```

```
public partial class Register : System.Web.UI.Page
{
    protected void CreateUser_Click(object sender, EventArgs e)
    {
        // Default UserStore constructor uses the default connection string named:
        DefaultConnection
        var userStore = new UserStore<IdentityUser>();
        var manager = new UserManager<IdentityUser>(userStore);

        var user = new IdentityUser() { UserName = UserName.Text };
        IdentityResult result = manager.Create(user, Password.Text);

        if (result.Succeeded)
        {
            StatusMessage.Text = string.Format("User {0} was created successfully!",
user.UserName);
        }
        else
        {
            StatusMessage.Text = result.Errors.FirstOrDefault();
        }
    }
}
```

- The code above is a simplified version of the Register.aspx.cs file that is created when you create a new ASP.NET Web Forms project.
- The IdentityUser class is the default EntityFramework implementation of the IUser interface. IUser interface is the minimal interface for a user on ASP.NET Identity Core.
- The UserStore class is the default EntityFramework implementation of a user store. This class implements the ASP.NET Identity Core's minimal interfaces: IUserStore, IUserLoginStore, IUserClaimStore and IUserRoleStore.
- The UserManager class exposes user related APIs which will automatically save changes to the UserStore.
- The IdentityResult class represents the result of an identity operation.
- In Solution Explorer, right-click your project and click Add, Add ASP.NET Folder and then App_Data.

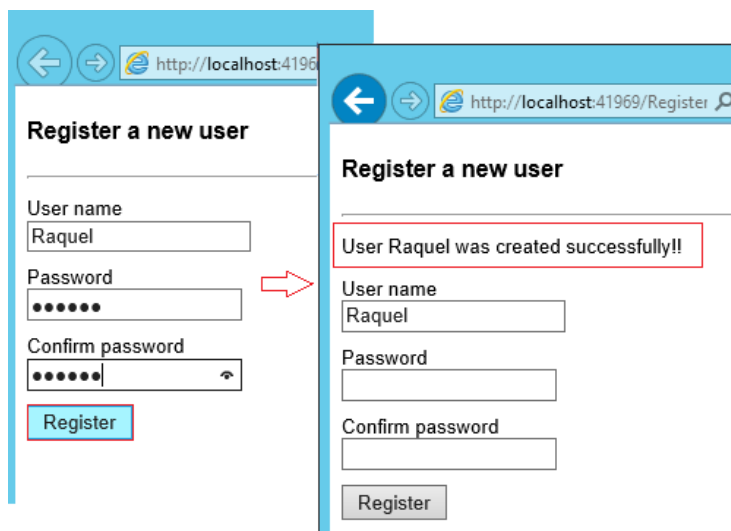


- Open the Web.config file and add a connection string entry for the database we will use to store user information. The database will be created at runtime by EntityFramework for the Identity entities. The connection string is similar to one created for you when you create a new Web Forms project.

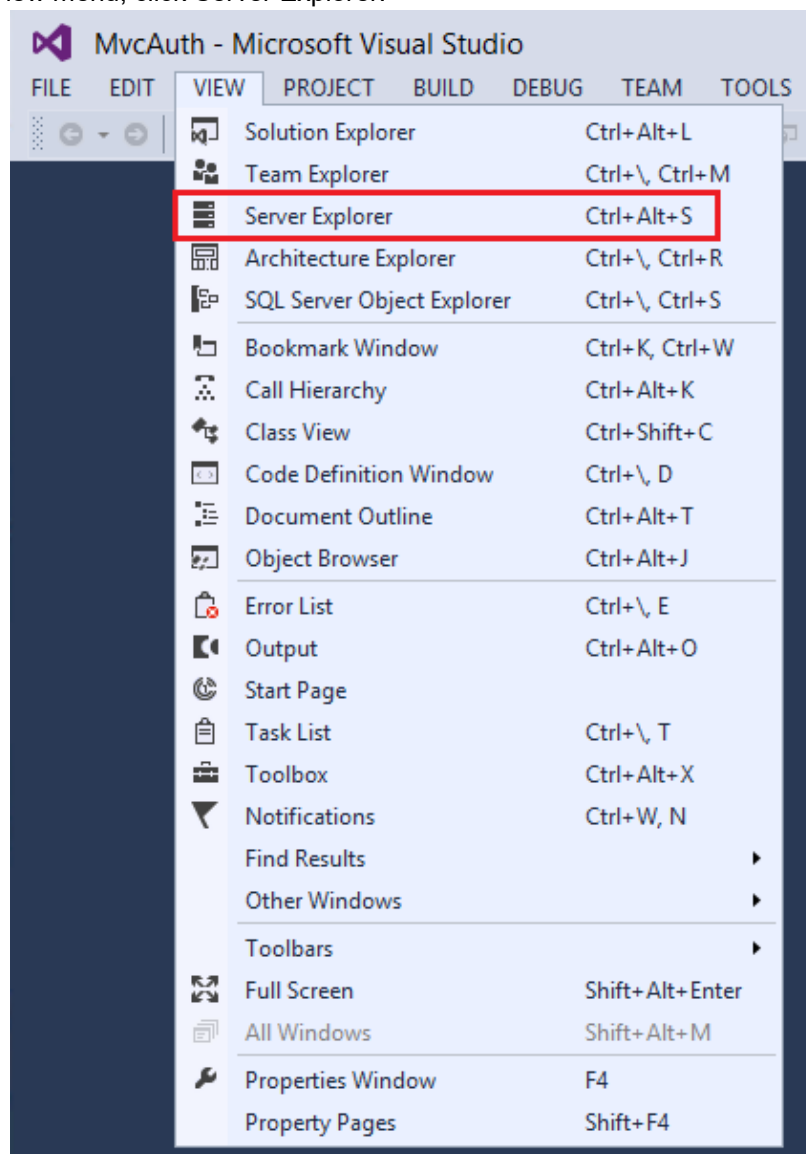
```
<?xml version="1.0" encoding="utf-8"?>
<!--
  For more information on how to configure your ASP.NET application, please visit
  http://go.microsoft.com/fwlink/?LinkId=169433
  -->
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit
    http://go.microsoft.com/fwlink/?LinkId=237468 -->
    <section name="entityFramework"
    type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
    EntityFramework, Version=6.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" requirePermission="false" />
  </configSections>
  <connectionStrings>
```

```
<add name="DefaultConnection" connectionString="Data
Source=(LocalDb)\v11.0;AttachDbFilename=|DataDirectory|\WebFormsIdentity.mdf;Init
ial Catalog=WebFormsIdentity;Integrated Security=True"
providerName="System.Data.SqlClient" />
</connectionStrings>
<system.web>
<compilation debug="true" targetFramework="4.5" />
<httpRuntime targetFramework="4.5" />
</system.web>
<entityFramework>
<defaultConnectionFactory
type="System.Data.Entity.Infrastructure.LocalDbConnectionFactory,
EntityFramework">
<parameters>
<parameter value="v11.0" />
</parameters>
</defaultConnectionFactory>
<providers>
<provider invariantName="System.Data.SqlClient"
type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer"
/>
</providers>
</entityFramework>
</configuration>
```

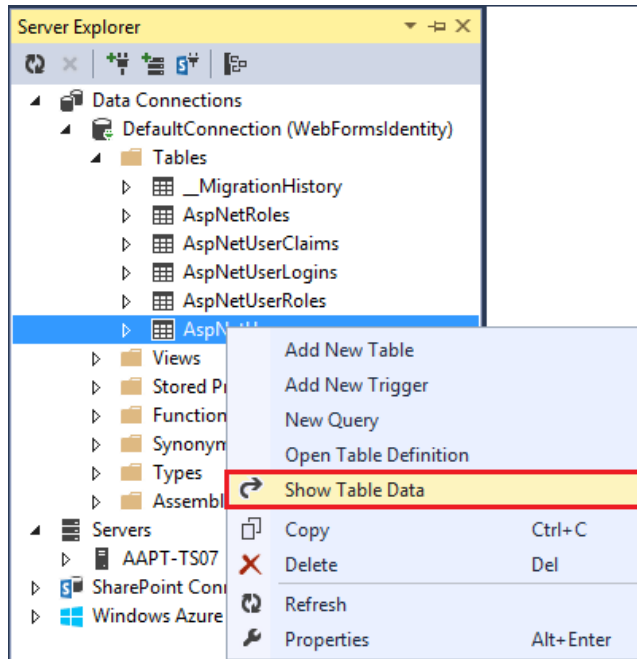
- Right click file Register.aspx in your project and select Set as Start Page. Press Ctrl + F5 to build and run the web application. Enter a new user name and password and then click on Register.



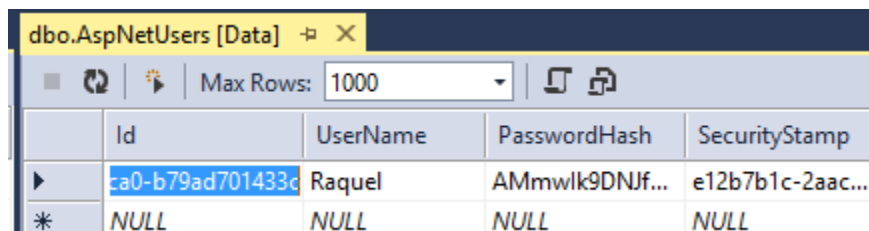
- ASP.NET Identity has support for validation and in this sample you can verify the default behavior on User and Password validators that come from the Identity Core package. The default validator for User (UserValidator) has a property AllowOnlyAlphanumericUserNames that has default value set to true. The default validator for Password (MinimumLengthValidator) ensures that password has at least 6 characters. These validators are properties on UserManager that can be overridden if you want to have custom validation,
- Verifying the LocalDb Identity Database and Tables Generated by Entity Framework
- In the View menu, click Server Explorer.



- Expand DefaultConnection (WebFormsIdentity), expand Tables, right click AspNetUsers and click Show Table Data.



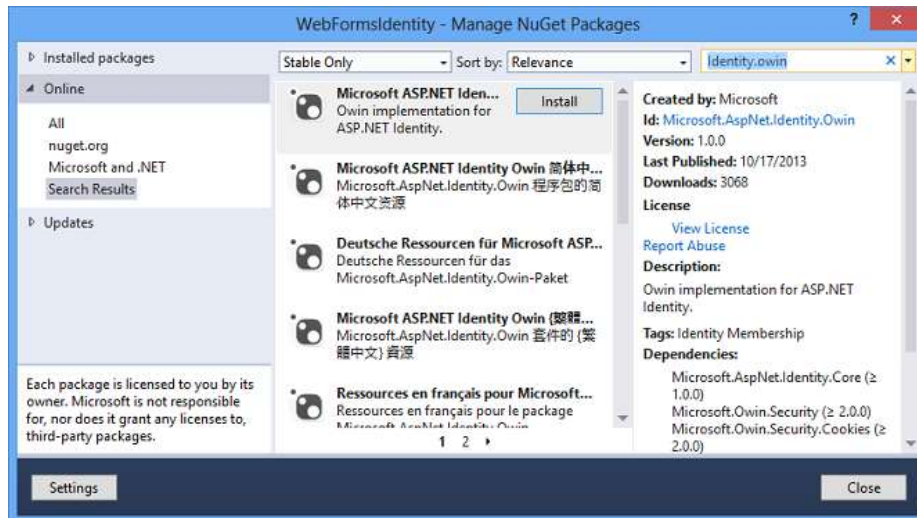
The screenshot shows the 'Server Explorer' window in SQL Server Enterprise Manager. The tree view is expanded to 'DefaultConnection (WebFormsIdentity)' > 'Tables'. The 'AspNetUsers' table is selected, and a right-click context menu is displayed. The 'Show Table Data' option is highlighted with a red rectangle. Other options include 'Add New Table', 'Add New Trigger', 'New Query', 'Open Table Definition', 'Copy', 'Delete', 'Refresh', and 'Properties'.



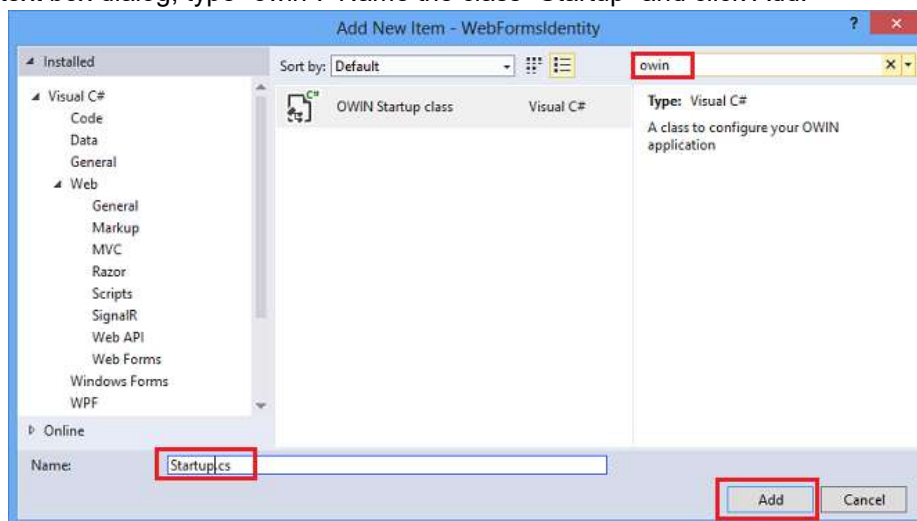
The screenshot shows the 'dbo.AspNetUsers [Data]' table view. The table has five columns: 'Id', 'UserName', 'PasswordHash', and 'SecurityStamp'. The 'Max Rows' is set to 1000. The first row of data is highlighted, showing the following values:

	Id	UserName	PasswordHash	SecurityStamp
	ca0-b79ad701433d	Raquel	AMmwIk9DNJf...	e12b7b1c-2aac...
*	NULL	NULL	NULL	NULL

- Configuring the application for OWIN authentication
- At this point we have only added support for creating users. Now, we are going to demonstrate how we can 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. With this model, the same authentication packages can be used across multiple frameworks including ASP.NET MVC and Web Forms.
- Installing authentication packages to your application
- In Solution Explorer, right-click your project and select Manage NuGet Packages. In the search text box dialog, type "Identity.Owin". Click install for this package.



- Search for package Microsoft.Owin.Host.SystemWeb and install it.
- 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.
- 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. For more information see OWIN Middleware in the IIS integrated pipeline.
- Adding OWIN Startup and Authentication Configuration Classes
- In Solution Explorer, right-click your project, click Add, and then Add New Item. In the search text box dialog, type "owin". Name the class "Startup" and click Add.



In the Startup.cs file, add the code shown below to configure OWIN cookie authentication.

```
using Microsoft.AspNet.Identity;
using Microsoft.Owin;
using Microsoft.Owin.Security.Cookies;
using Owin;

[assembly: OwinStartup (typeof(WebFormsIdentity.Startup))]
namespace WebFormsIdentity
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            // For more information on how to configure your application, visit
            http://go.microsoft.com/fwlink/?LinkID=316888
            app.UseCookieAuthentication(new CookieAuthenticationOptions
            {
                AuthenticationType = DefaultAuthenticationTypes.ApplicationCookie,
                LoginPath = new PathString("/Login")
            });
        }
    }
}
```

- 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.
- Adding Web Forms for Registering and Logging in Users
- Open the Register.cs file and add the following code which will log in the user when registration succeeds.

```
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.EntityFramework;
using Microsoft.Owin.Security;
using System;
using System.Linq;
using System.Web;

namespace WebFormsIdentity
{
    public partial class Register : System.Web.UI.Page
    {
        protected void CreateUser_Click(object sender, EventArgs e)
        {

```

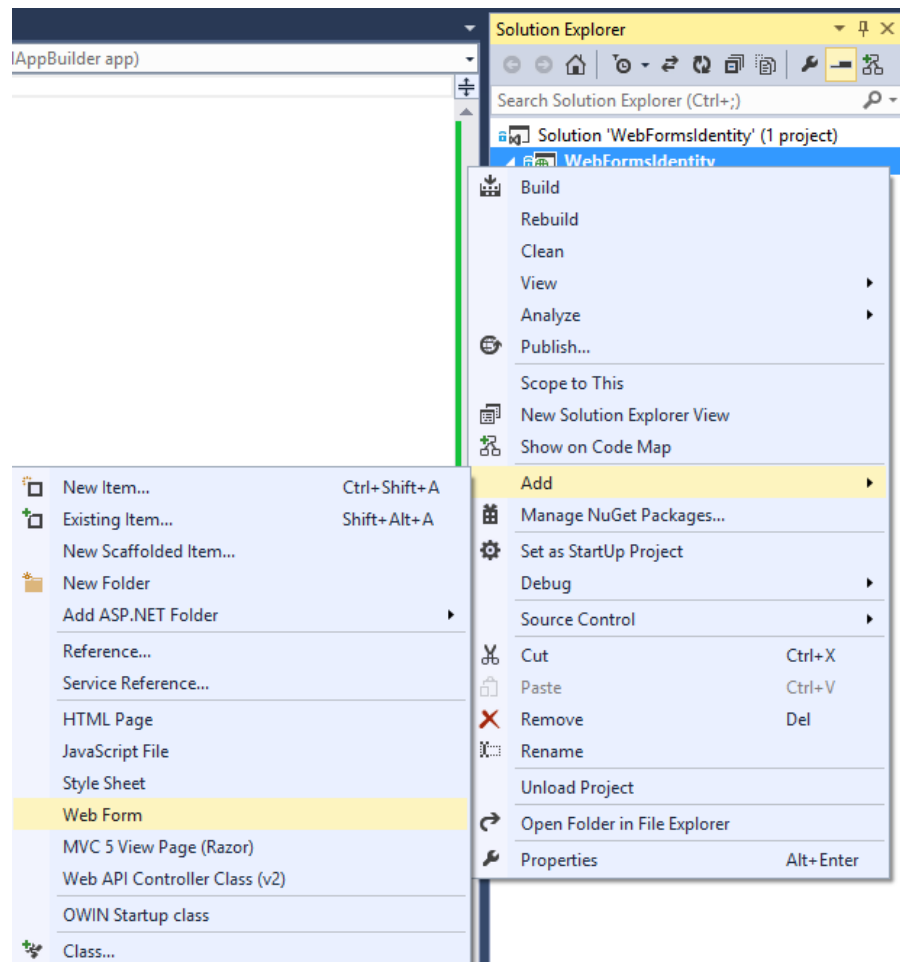


```
// Default UserStore constructor uses the default connection string named:
DefaultConnection
var userStore = new UserStore<IdentityUser>();
var manager = new UserManager<IdentityUser>(userStore);
var user = new IdentityUser() { UserName = UserName.Text };

IdentityResult result = manager.Create(user, Password.Text);

if (result.Succeeded)
{
    var authenticationManager =
HttpContext.Current.GetOwinContext().Authentication;
    var userIdentity = manager.CreateIdentity(user,
DefaultAuthenticationTypes.ApplicationCookie);
    authenticationManager.SignIn(new AuthenticationProperties() { }, userIdentity);
    Response.Redirect("~/Login.aspx");
}
else
{
    StatusMessage.Text = result.Errors.FirstOrDefault();
}
}
}
```

- Since ASP.NET Identity and OWIN Cookie Authentication are claims based system, the framework requires the app developer to generate a ClaimsIdentity for the user. ClaimsIdentity has information about all the claims for the user such as what Roles the user belongs to. You can also add more claims for the user at this stage.
- You can sign in the user by using the AuthenticationManager from OWIN and calling SignIn and passing in the ClaimsIdentity as shown above. This code will sign in the user and generate a cookie as well. This call is analogous to FormAuthentication.SetAuthCookie used by the FormsAuthentication module.
- In Solution Explorer, right-click your project click Add, and then Web Form. Name the web form Login.



- Replace the contents of the Login.aspx file with the following code:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Login.aspx.cs"
Inherits="WebFormsIdentity.Login" %>

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title></title>
</head>
<body style="font-family: Arial, Helvetica, sans-serif; font-size: small">
  <form id="form1" runat="server">
    <div>
      <h4 style="font-size: medium">Log In</h4>
      <hr />
      <asp:Placeholder runat="server" ID="LoginStatus" Visible="false">
```

```
<p>
  <asp:Literal runat="server" ID="StatusText" />
</p>
</asp:Placeholder>
<asp:Placeholder runat="server" ID="LoginForm" Visible="false">
  <div style="margin-bottom: 10px">
    <asp:Label runat="server" AssociatedControlID="UserName">User
name</asp:Label>
    <div>
      <asp:TextBox runat="server" ID="UserName" />
    </div>
  </div>
  <div style="margin-bottom: 10px">
    <asp:Label runat="server"
AssociatedControlID="Password">Password</asp:Label>
    <div>
      <asp:TextBox runat="server" ID="Password" TextMode="Password" />
    </div>
  </div>
  <div style="margin-bottom: 10px">
    <div>
      <asp:Button runat="server" OnClick ="SignIn" Text="Log in" />
    </div>
  </div>
</asp:Placeholder>
<asp:Placeholder runat="server" ID="LogoutButton" Visible="false">
  <div>
    <div>
      <asp:Button runat="server" OnClick ="SignOut" Text="Log out" />
    </div>
  </div>
</asp:Placeholder>
</div>
</form>
</body>
</html>
```

- Replace the contents of the Login.aspx.cs file with the following:

```
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.EntityFramework;
using Microsoft.Owin.Security;
using System;
using System.Web;
```

```
using System.Web.UI.WebControls;

namespace WebFormsIdentity
{
    public partial class Login : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                if (User.Identity.IsAuthenticated)
                {
                    StatusText.Text = string.Format("Hello {0}!!", User.Identity.GetUserName());
                    LoginStatus.Visible = true;
                    LogoutButton.Visible = true;
                }
                else
                {
                    LoginForm.Visible = true;
                }
            }
        }

        protected void SignIn(object sender, EventArgs e)
        {
            var userStore = new UserStore <IdentityUser>();
            var userManager = new UserManager <IdentityUser>(userStore);
            var user = userManager.Find (UserName.Text, Password.Text);

            if (user != null)
            {
                var authenticationManager =
                    HttpContext.Current.GetOwinContext().Authentication;
                var userIdentity = userManager.CreateIdentity(user,
                    DefaultAuthenticationTypes.ApplicationCookie);

                authenticationManager.SignIn(new AuthenticationProperties() { IsPersistent =
                    false }, userIdentity);
                Response.Redirect("~/Login.aspx");
            }
            else
            {
            }
        }
    }
}
```

```

        StatusText.Text = "Invalid username or password.";
        LoginStatus.Visible = true;
    }
}

protected void SignOut(object sender, EventArgs e)
{
    var authenticationManager =
HttpContext.Current.GetOwinContext().Authentication;
    authenticationManager.SignOut();
    Response.Redirect("~/Login.aspx");
}
}
}

```

- The Page_Load now checks for the status of current user and acts based on its Context.User.Identity.IsAuthenticated status.
- Display Logged in User Name: The Microsoft ASP.NET Identity Framework has added extension methods on System.Security.Principal.IIdentity that allows you to get the UserName and UserId for the logged in User. These extension methods are defined in the Microsoft.AspNet.Identity.Core assembly. These extension methods are the replacement for HttpContext.User.Identity.Name.
- SignIn method: This method replaces the previous CreateUser_Click method in this sample and now signs in the user after successfully creating the user.
- The Microsoft OWIN Framework has added extension methods on System.Web.HttpContext that allows you to get a reference to an IOwinContext. These extension methods are defined in Microsoft.Owin.Host.SystemWeb assembly. The OwinContext class exposes an IAuthenticationManager property that represents the Authentication middleware functionality available on the current request.
- You can sign in the user by using the AuthenticationManager from OWIN and calling SignIn and passing in the ClaimsIdentity as shown above.
- Because ASP.NET Identity and OWIN Cookie Authentication are claims-based system, the framework requires the app to generate a ClaimsIdentity for the user.
- The ClaimsIdentity has information about all the claims for the user, such as what roles the user belongs to. You can also add more claims for the user at this stage
- This code will sign in the user and generate a cookie as well. This call is analogous to FormAuthentication.SetAuthCookie used by the FormsAuthentication module.
- SignOut method: Gets a reference to the AuthenticationManager from OWIN and calls SignOut. This is analogous to FormsAuthentication.SignOut method used by the FormsAuthentication module.
- Press Ctrl + F5 to build and run the web application. Enter a new user name and password and then click on Register.

The image shows two browser windows side-by-side. The left window is titled 'Register a new user' and contains three input fields: 'User name' with the text 'Raquel', 'Password' with masked characters, and 'Confirm password' with masked characters. A red box highlights the 'Register' button at the bottom. A red arrow points from the 'Register' button to the right window. The right window is titled 'Log In' and shows a message 'Hello Raquel!!!' in a red box, with a 'Log out' button below it.

- Note: At this point, the new user is created and logged in.
- Click on Log out button. You will be redirected to the Log in form.
- Enter an invalid user name or password and Click on Log in button.
- The UserManager.Find method will return null and the error message: "Invalid user name or password" will be displayed.

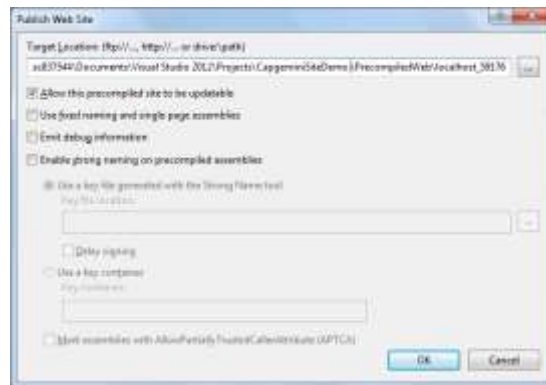
The image shows a browser window titled 'Log In'. It displays an error message 'Invalid username or password.' in a red box above the 'User name' input field. The 'User name' field contains the text 'RaquelS'. The 'Password' field is empty. A 'Log in' button is located below the password field.

Lab 8. Deployment in ASP.NET

Goals	<p>To Learn:</p> <ul style="list-style-type: none"> Deploying an ASP.NET Website ClickOnce Technique Aspnet_compiler
Time	0.5Hr

8.1: Deploy using Setup (Publish web site)

- Go to Build menu, select Publish Website or Right Click on the Website in the Solution Explorer Window and click on Publish Website.



- The Publish Website wizard opens, set the Target Location for the Site to be published and check the checkboxes if required.
- Click OK to publish, navigate to the Target Location specified where you can notice a folder named "PreCompiledWeb" because of Deployment.

8.2: Deploy using aspnet_compiler

- Create a separate folder to store the build output of the asp.net web site. (For example: D:\WebSiteBuildOutput)
- Go to Visual Studio 2012 command prompt. Execute the compiler utility as shown:
aspnet_compiler -v /AspNetDemo -p C:\AspNetDemo D:\WebSiteBuildOutput
where "C:\AspNetDemo" is the folder where asp.net web site source files exist and "D:\WebSiteBuildOutput" is where build output needs to be generated
- Move to this above output folder to see the build output to be deployed on the Web Server.

Note: In the build output all ".aspx.cs" as well as ".aspx" files are also compiled so that application is completely precompiled and there is not additionally delay at runtime.