**Adding ASP.NET Identity to an Empty or Existing Web Forms Project**

This tutorial shows you how to add [ASP.NET Identity](https://docs.microsoft.com/en-us/aspnet/identity/overview/getting-started/introduction-to-aspnet-identity) (the new membership system for ASP.NET) to an ASP.NET application.

When you create a new Web Forms or MVC project in Visual Studio 2017 RTM with Individual Accounts, Visual Studio will install all the required packages and add all necessary classes for you. This tutorial will illustrate the steps to add ASP.NET Identity support to your existing Web Forms project or a new empty project. I will outline all the NuGet packages you need to install, and classes you need to add. I will go over sample Web Forms for registering new users and logging in while highlighting all main entry point APIs for user management and authentication. This sample will use the ASP.NET Identity default implementation for SQL data storage which is built on Entity Framework. This tutorial, we will use LocalDB for the SQL database.

**Get started with ASP.NET Identity**

1. Start by installing and running [Visual Studio 2017](https://visualstudio.microsoft.com/downloads/).
2. Select **New Project** from the Start page, or you can use the menu and select **File**, and then **New Project**.
3. In the left pane, expand **Visual C#**, then select **Web**, then **ASP.NET Web Application (.Net Framework)**. Name your project "WebFormsIdentity" and select **OK**.
4. 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.

**Add Identity packages to your app**

In Solution Explorer, right-click your project and select **Manage NuGet Packages**. Search for and install the **Microsoft.AspNet.Identity.EntityFramework** package.

Note that this package will install the dependency packages: **EntityFramework** and **Microsoft ASP.NET Identity Core**.

**Add a web form to register users**

1. In **Solution Explorer**, right-click your project and select **Add**, and then **Web Form**.
2. In the **Specify Name for Item** dialog box, name the new web form **Register**, and then select **OK**
3. Replace the markup in the generated *Register.aspx* file with the code below. The code changes are highlighted.

HTMLCopy

<%@ 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 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.

1. Open the *Register.aspx.cs* file and replace the contents of the file with the following code:

C#Copy

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();

}

}

}

}

**Note**

* 1. 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.
  2. 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.
  3. 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*.
  4. The *UserManager* class exposes user related APIs which will automatically save changes to the *UserStore*.
  5. The *IdentityResult* class represents the result of an identity operation.

1. In **Solution Explorer**, right-click your project and select **Add**, **Add ASP.NET Folder** and then **App\_Data**.
2. 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. The highlighted code shows the markup you should add:

XMLCopy

<?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;Initial 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>

**Note**

For Visual Studio 2015 or higher, replace (localdb)\v11.0 with (localdb)\MSSQLLocalDB in your connection string.

1. 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 select **Register**.

**Note**

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,

**Verify the LocalDb Identity database and tables generated by Entity Framework**

1. In the **View** menu, select **Server Explorer**.
2. Expand **DefaultConnection (WebFormsIdentity)**, expand **Tables**, right-click **AspNetUsers** and then select **Show Table Data**.

**Configure 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](https://msdn.microsoft.com/magazine/dn451439.aspx) or IIS. With this model, the same authentication packages can be used across multiple frameworks including ASP.NET MVC and Web Forms. For more information on project Katana and how to run middleware in a host agnostic see [Getting Started with the Katana Project](https://msdn.microsoft.com/magazine/dn451439.aspx).

**Install authentication packages to your application**

1. In Solution Explorer, right-click your project and select **Manage NuGet Packages**. Search for and install the ***Microsoft.AspNet.Identity.Owin*** package.
2. Search for and install the ***Microsoft.Owin.Host.SystemWeb*** package.

**Note**

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**](https://docs.microsoft.com/en-us/aspnet/aspnet/overview/owin-and-katana/owin-middleware-in-the-iis-integrated-pipeline).

**Add OWIN startup and authentication configuration classes**

1. In **Solution Explorer**, right-click your project, select **Add**, and then **Add New Item**. In the search text box dialog, type "*owin*". Name the class "*Startup*" and select **Add**.
2. In the Startup.cs file, add the highlighted code shown below to configure OWIN cookie authentication.

C#Copy

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")

});

}

}

}

**Note**

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. See [**OWIN Startup Class Detection**](https://docs.microsoft.com/en-us/aspnet/aspnet/overview/owin-and-katana/owin-startup-class-detection) for more info on this model.

**Add web forms for registering and signing in users**

1. Open the *Register.aspx.cs* file and add the following code which signs in the user when registration succeeds.

C#Copy

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();

}

}

}

}

**Note**

* + Since ASP.NET Identity and OWIN Cookie Authentication are claims based system, the framework requires the app developer to generate a **[ClaimsIdentity](https://msdn.microsoft.com/library/microsoft.identitymodel.claims.claimsidentity.aspx)** 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](https://msdn.microsoft.com/library/system.web.security.formsauthentication.setauthcookie.aspx)** used by the **[FormsAuthentication](https://msdn.microsoft.com/library/system.web.security.formsauthenticationmodule.aspx)**module.

1. In **Solution Explorer**, right-click your project, select **Add**, and then **Web Form**. Name the web form **Login**.
2. Replace the contents of the *Login.aspx* file with the following code:

aspxCopy

<%@ 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>

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

C#Copy

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");

}

}

}

**Note**

* + The Page\_Load now checks for the status of current user and takes action 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](https://msdn.microsoft.com/library/system.security.principal.iidentity.aspx)** 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](https://msdn.microsoft.com/library/system.web.httpcontext.user.aspx)** .
  + SignIn method: This method replaces the previous CreateUser\_Clickmethod 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 SignInand 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](https://msdn.microsoft.com/library/system.web.security.formsauthentication.setauthcookie.aspx)** used by the **[FormsAuthentication](https://msdn.microsoft.com/library/system.web.security.formsauthenticationmodule.aspx)** module.
  + SignOut method: Gets a reference to the AuthenticationManager from OWIN and calls SignOut. This is analogous to **[FormsAuthentication.SignOut](https://msdn.microsoft.com/library/system.web.security.formsauthentication.signout.aspx)**method used by the **[FormsAuthentication](https://msdn.microsoft.com/library/system.web.security.formsauthenticationmodule.aspx)** module.

1. Press **Ctrl + F5** to build and run the web application. Enter a new user name and password and then select **Register**.

Note: At this point, the new user is created and logged in.

1. Select the **Log out** button. You are redirected to the Log in form.
2. Enter an invalid user name or password and select the **Log in** button. The UserManager.Find method will return null and the error message: " *Invalid user name or password* " will be displayed.