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

By Raquel Soares De Almeida | October 23, 2013
408 of 467 people found this helpful

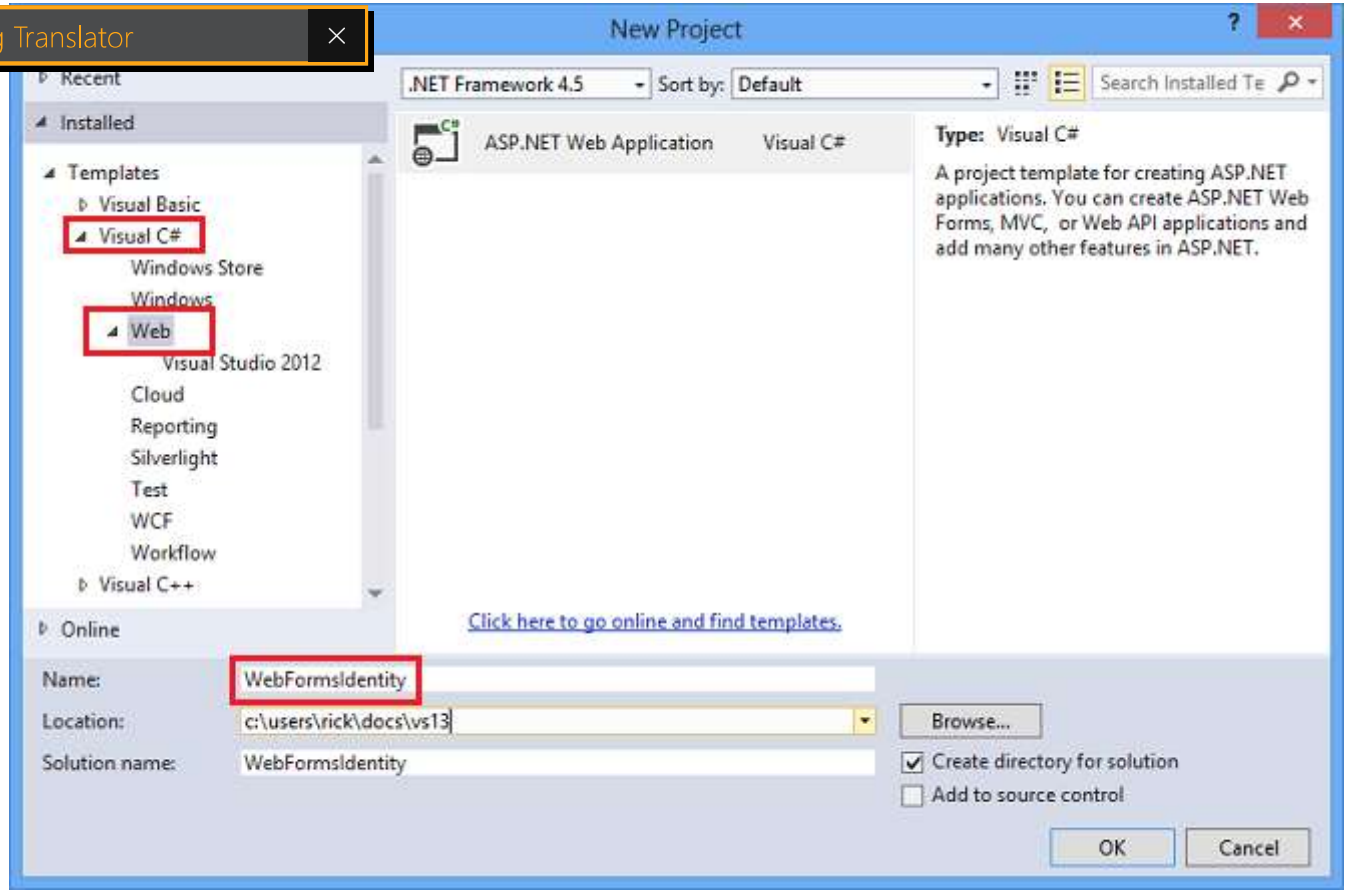
This tutorial shows you how to add **ASP.NET Identity (/identity/overview/getting-started/introduction-to-aspnet-identity)** (the new membership system for ASP.NET) to an ASP.NET application.

When you create a new Web Forms or MVC project in Visual Studio 2013 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.

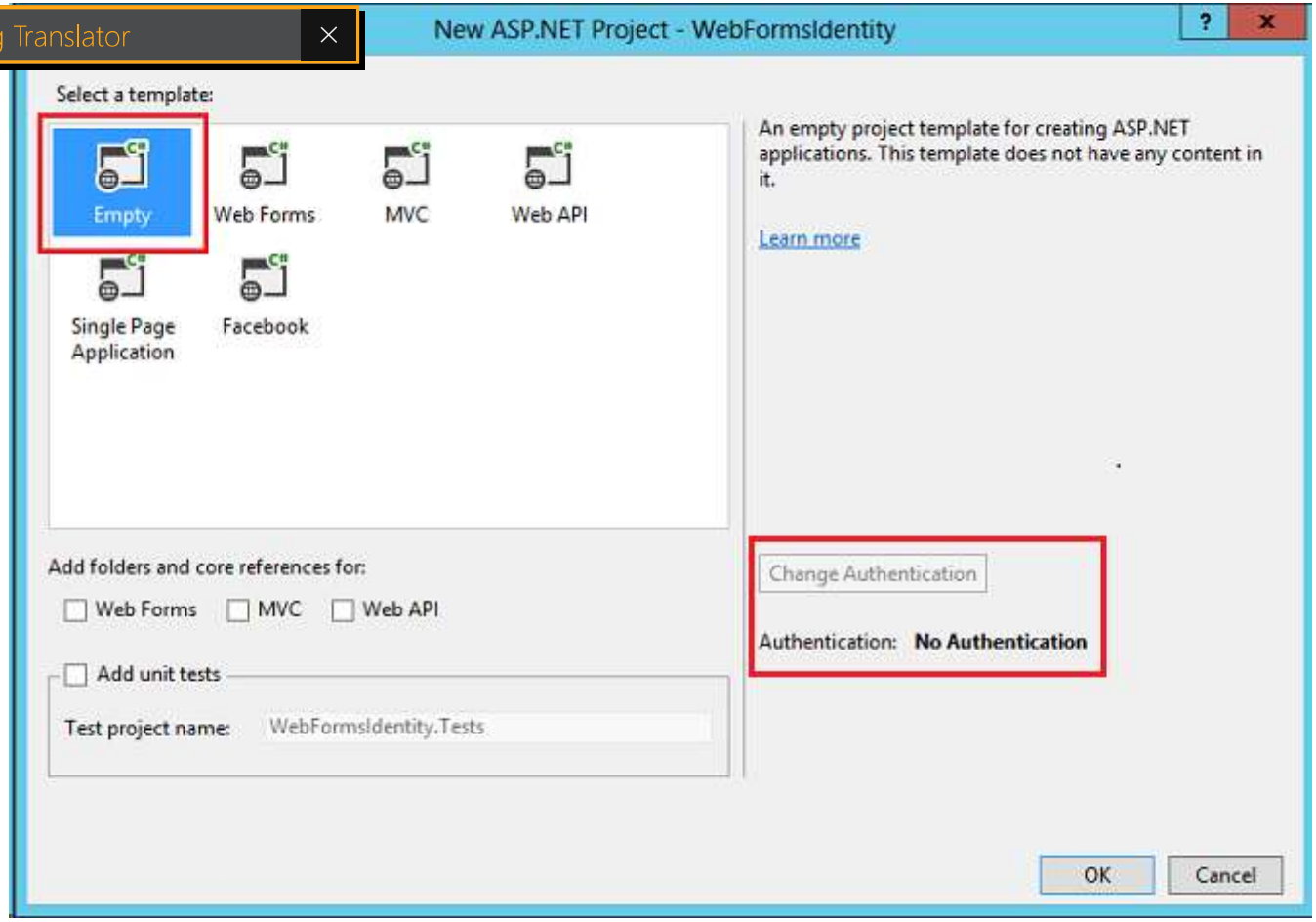
This tutorial was written by Raquel Soares De Almeida and Rick Anderson (**@RickAndMSFT (https://twitter.com/#!/RickAndMSFT)**).

Getting Started ASP.NET Identity

1. Start by installing and running **Visual Studio Express 2013 for Web (http://go.microsoft.com/fwlink/?LinkId=299058)** or **Visual Studio 2013 (http://go.microsoft.com/fwlink/?LinkId=306566)** .
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# i**n the left pane, then **Web** and then select **ASP.NET Web Application**. Name your project "WebFormsIdentity" and then click **OK**.



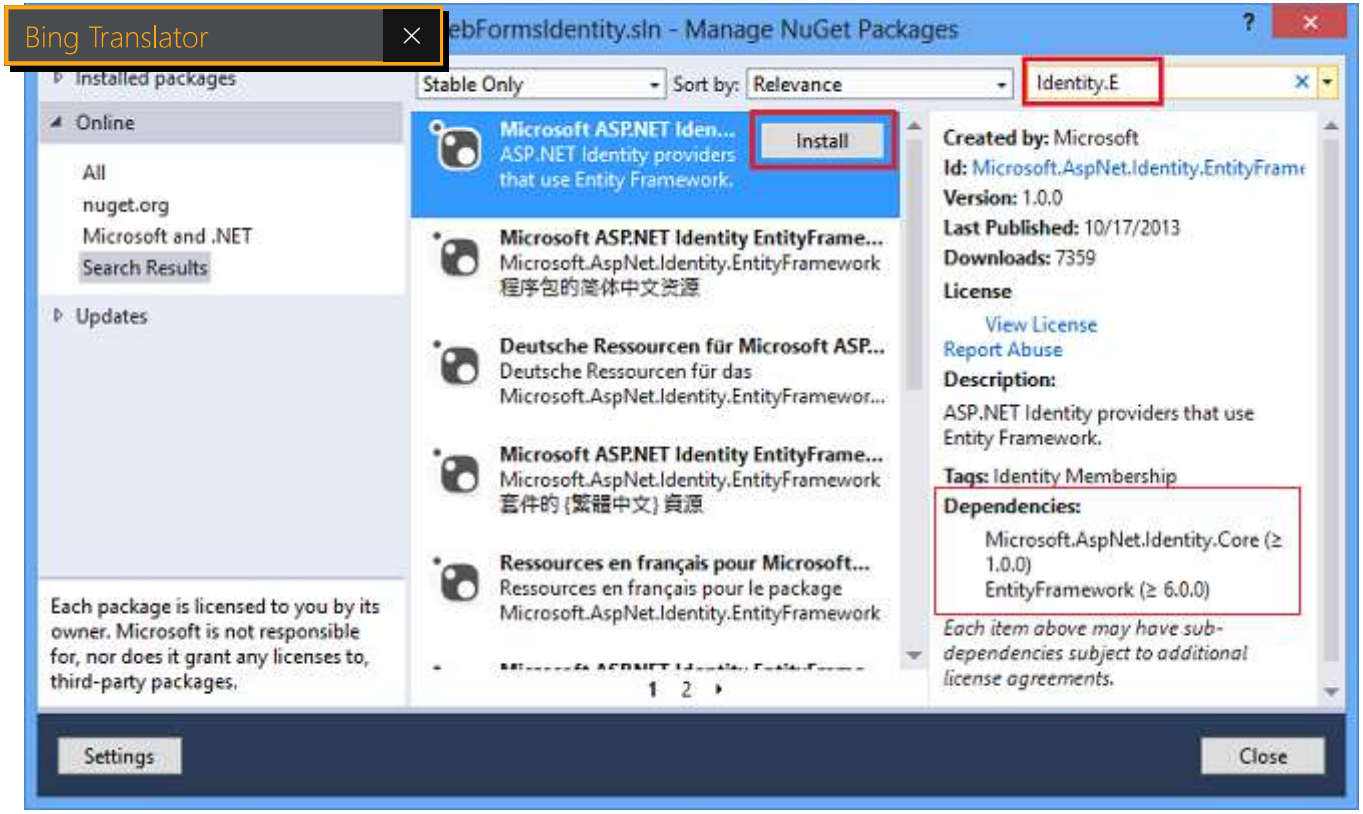
4. In the **New ASP.NET Project** dialog, select the **Empty** template.



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. For more information, see **Overview of Authentication (/visual-studio/overview/2013/creating-web-projects-in-visual-studio#auth)** .

Adding Identity Packages to your App

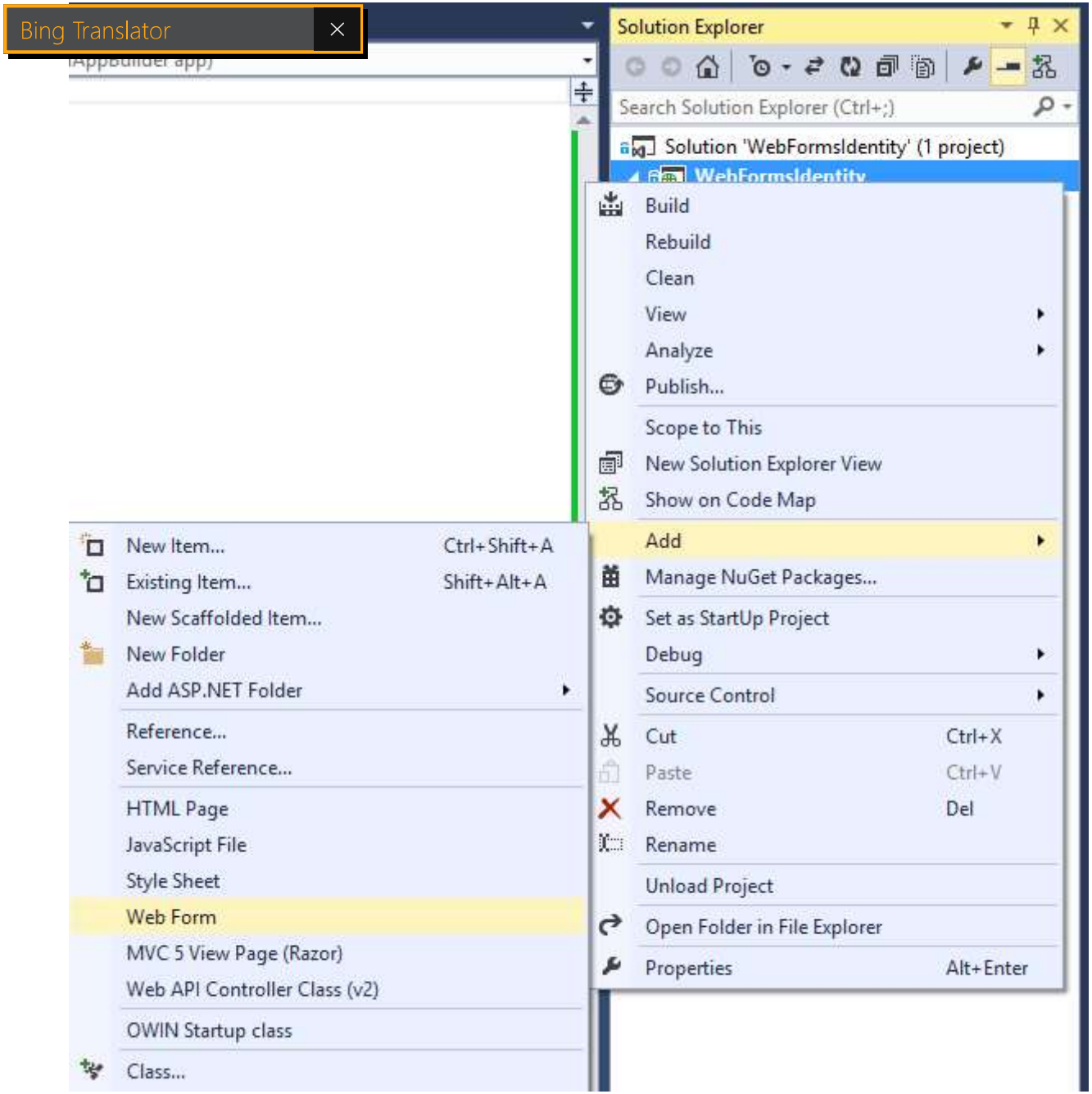
In Solution Explorer, right-click your project and select **Manage NuGet Packages**. In the search text box dialog, type "*Identity.E*". Click install for this package.



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

Adding Web Forms to Register Users

1. In **Solution Explorer**, right-click your project and click **Add**, and then **Web Form**.


Bing Translator
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'WebFormsIdentity' (1 project)
WebFormsIdentity
Build
Rebuild
Clean
View
Analyze
Publish...
Scope to This
New Solution Explorer View
Show on Code Map
Add
Manage NuGet Packages...
Set as StartUp Project
Debug
Source Control
Cut Ctrl+X
Paste Ctrl+V
Remove Del
Rename
Unload Project
Open Folder in File Explorer
Properties Alt+Enter
New Item... Ctrl+Shift+A
Existing Item... Shift+Alt+A
New Scaffolded Item...
New Folder
Add ASP.NET Folder
Reference...
Service Reference...
HTML Page
JavaScript File
Style Sheet
Web Form
MVC 5 View Page (Razor)
Web API Controller Class (v2)
OWIN Startup class
Class...

dialog box, name the new web form **Register**, and then click **OK**

rated *Register.aspx* file with the code below. The code changes are highlighted.

```
<%@ 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>
```

```
        ton runat="server" OnClick="CreateUser_Click" Text="Register" />
        </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.

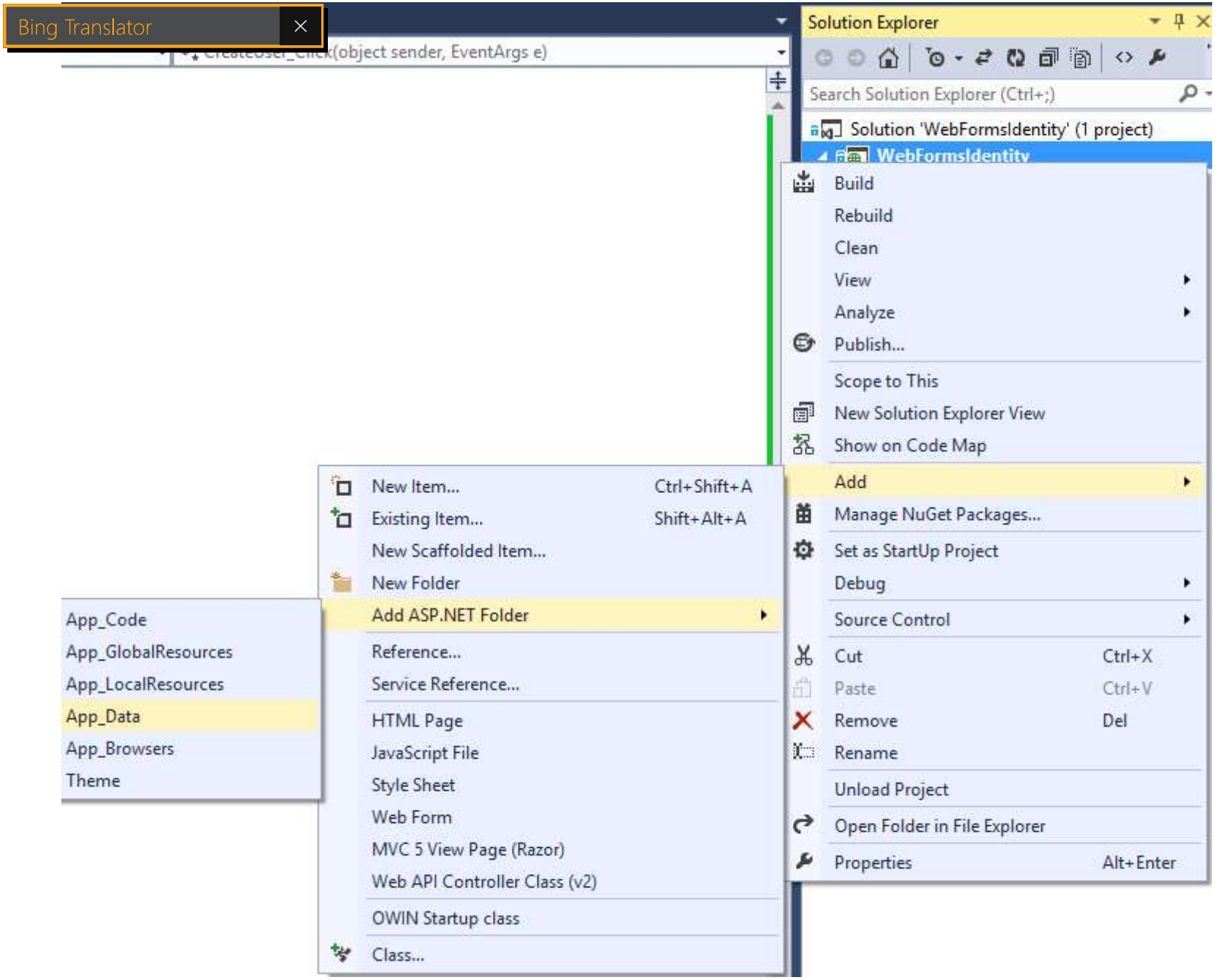
4. Open the *Register.aspx.cs* file and replace the contents of the file with the following code:

```
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);
         }
```

```
            StatusMessage.Text = result.Errors.FirstOrDefault();
        }
    }
}
```

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.

5. In **Solution Explorer**, right-click your project and click **Add**, **Add ASP.NET Folder** and then **App_Data**.


Bing Translator
(object sender, EventArgs e)
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'WebFormsIdentity' (1 project)
WebFormsIdentity
Build
Rebuild
Clean
View
Analyze
Publish...
Scope to This
New Solution Explorer View
Show on Code Map
Add
Manage NuGet Packages...
Set as StartUp Project
Debug
Source Control
Cut Ctrl+X
Paste Ctrl+V
Remove Del
Rename
Unload Project
Open Folder in File Explorer
Properties Alt+Enter
New Item... Ctrl+Shift+A
Existing Item... Shift+Alt+A
New Scaffolded Item...
New Folder
Add ASP.NET Folder
Reference...
Service Reference...
HTML Page
JavaScript File
Style Sheet
Web Form
MVC 5 View Page (Razor)
Web API Controller Class (v2)
OWIN Startup class
Class...
App_Code
App_GlobalResources
App_LocalResources
App_Data
App_Browsers
Theme

d a connection string entry for the database we will use to store user information. The database will be created at runtime by EntityFramework nection 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:

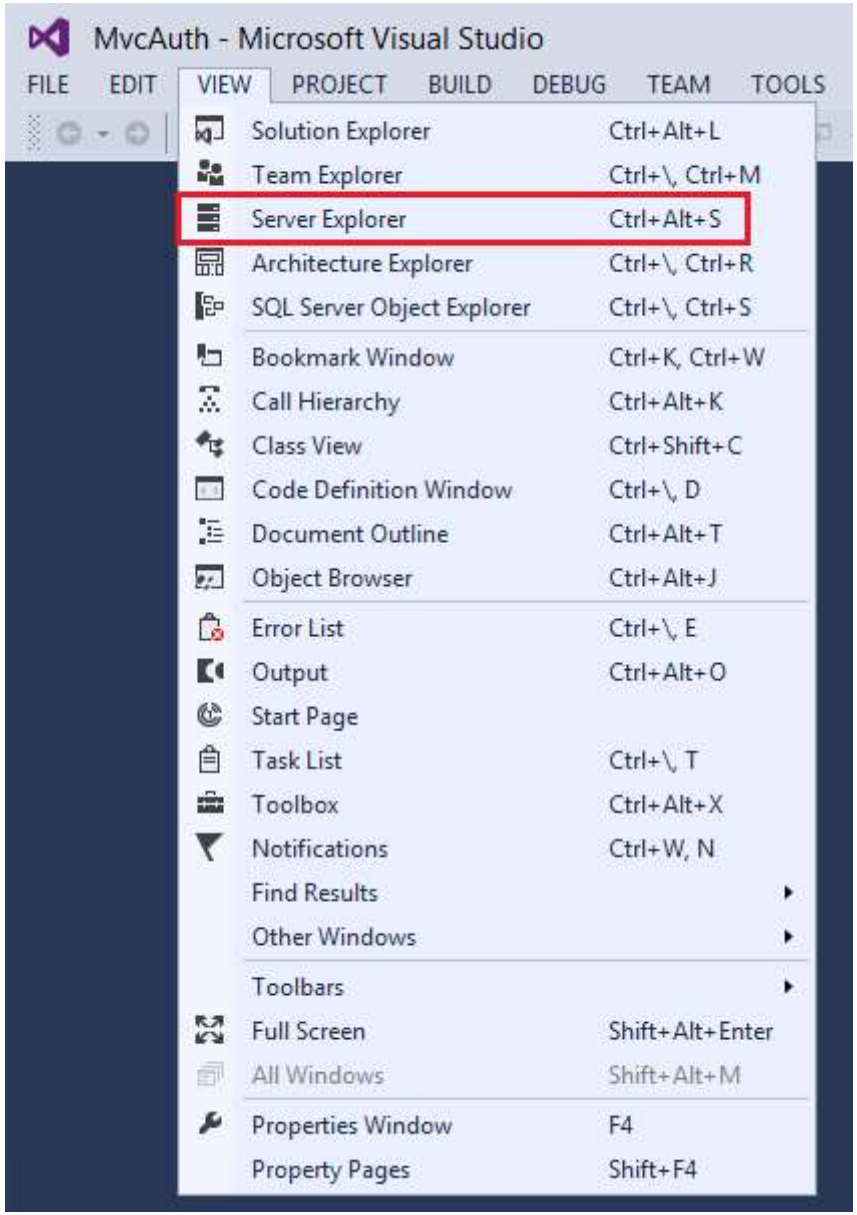
```
<?xml version="1.0" encoding="utf-8"?>
<!--
  For more information on how to configure your ASP.NET application, please visit
  http://go.microsoft.com/fwlink/?LinkId=169433
  -->
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->
    <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>
```

our 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

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,

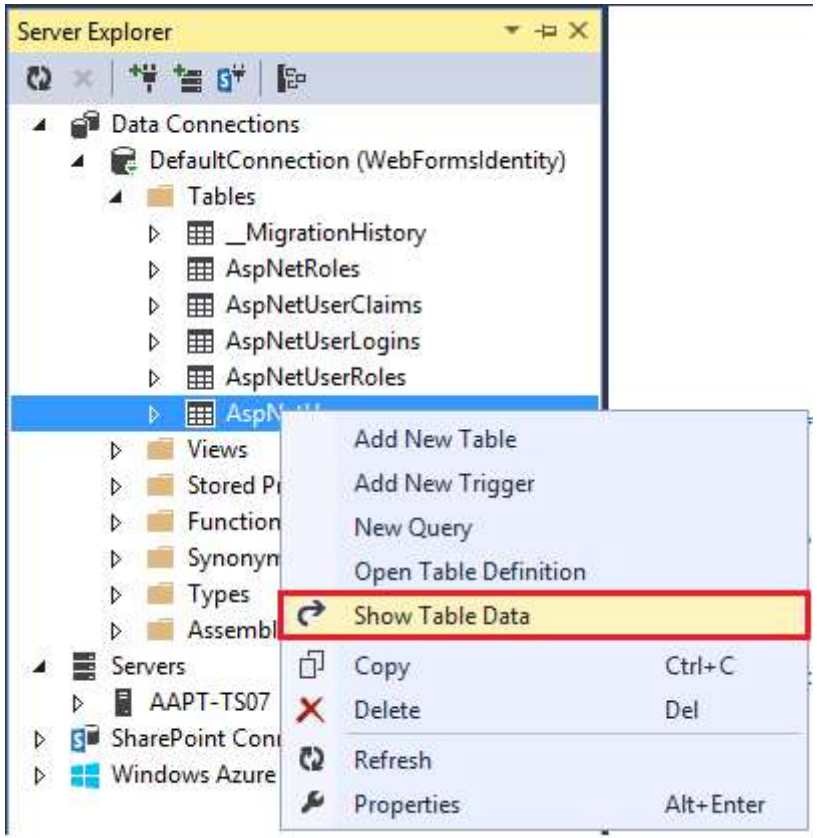
Verifying the LocalDb Identity Database and Tables Generated by Entity Framework

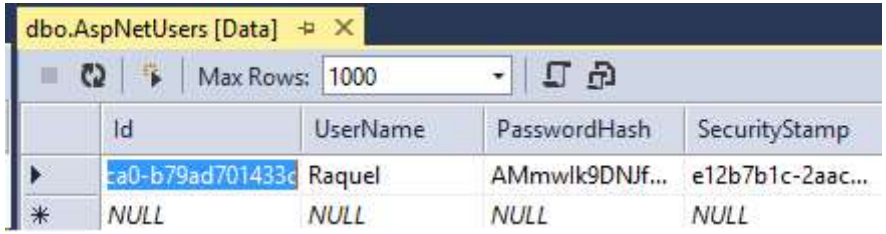
Explorer.



Bing Translator

ebFormsIdentity), expand **Tables**, right click **AspNetUsers** and click **Show Table Data**.



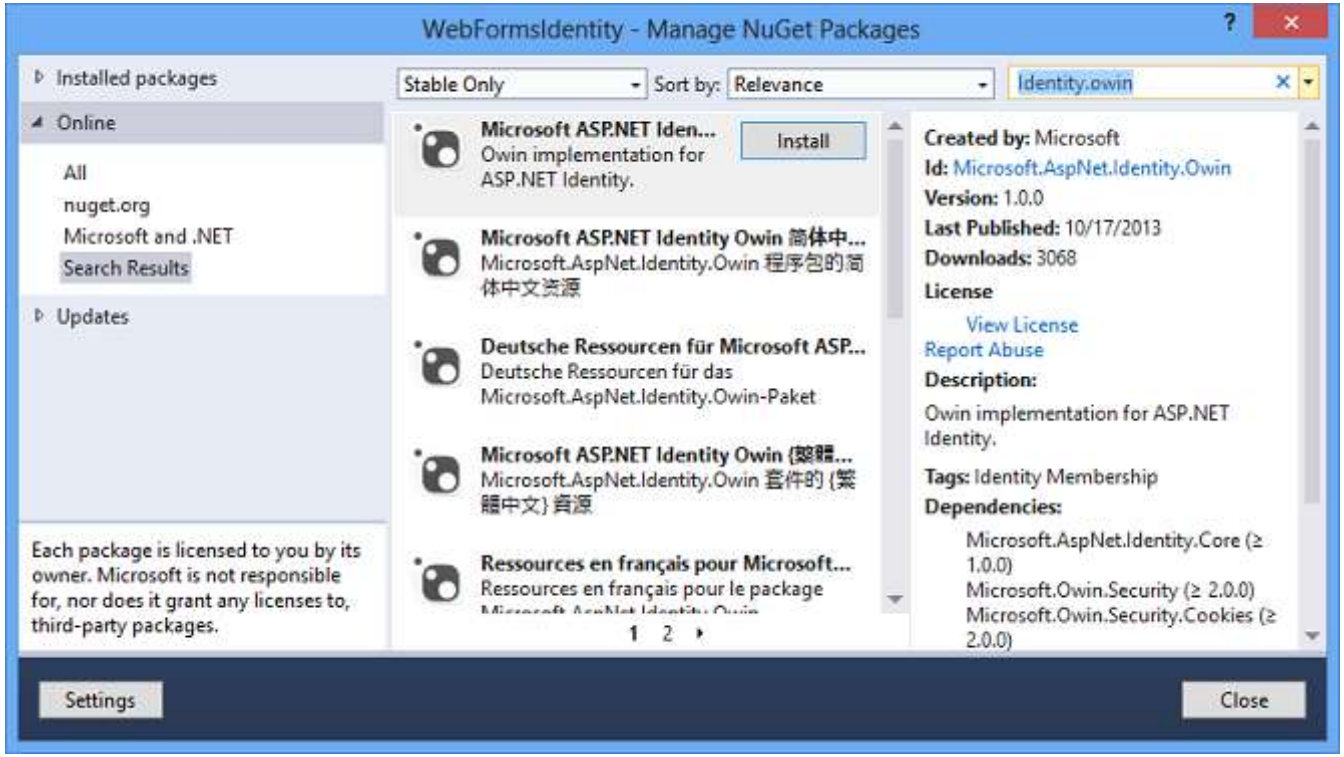


Configuring the application for OWIN authentication

port for creating users. Now, we are going to demonstrate how we can add authentication to login a user. ASP.NET Identity uses Microsoft forms authentication. The OWIN Cookie Authentication is a cookie and claims based authentication mechanism that can be used by any framework hosted on **OWIN (http://msdn.microsoft.com/en-us/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 (http://msdn.microsoft.com/en-us/magazine/dn451439.aspx)** .

Installing authentication packages to your application

1. 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.

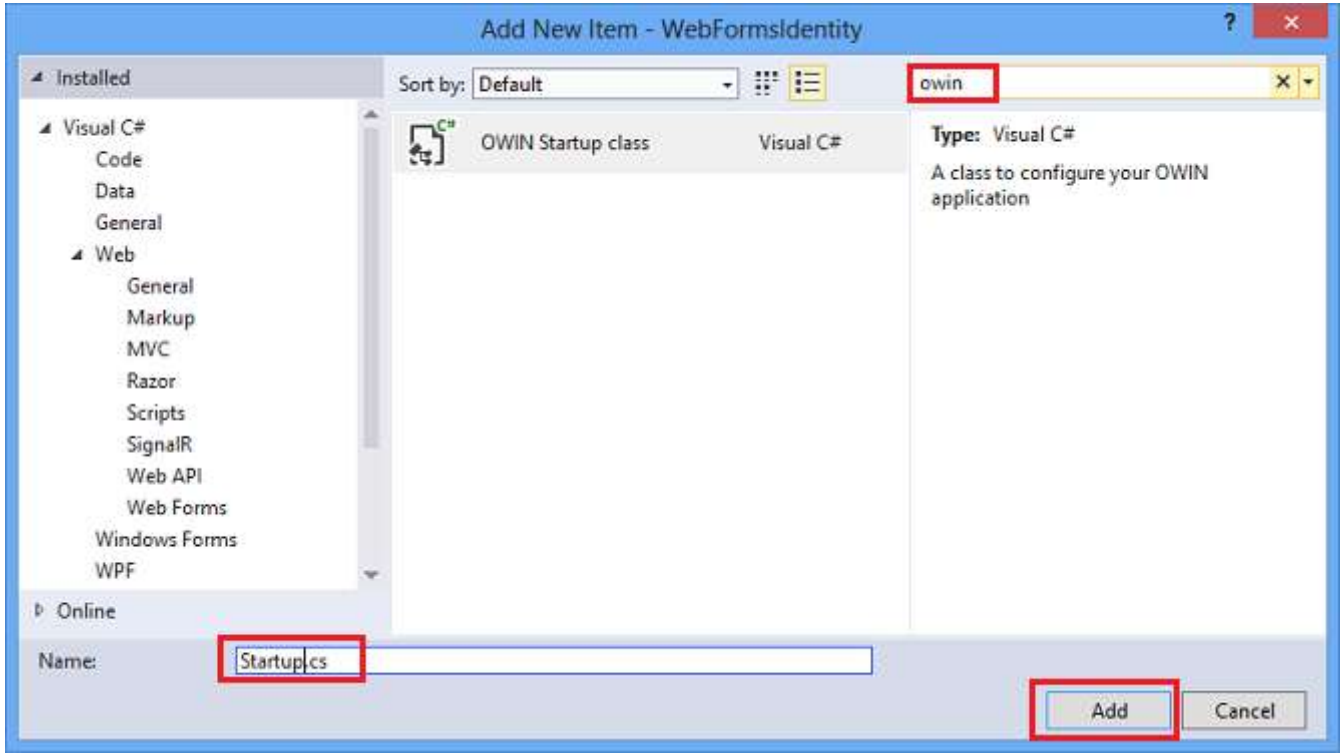


2. Search for package ***Microsoft.Owin.Host.SystemWeb*** and install it.

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 (/aspnet/overview/owin-and-katana/owin-middleware-in-the-iis-integrated-pipeline)** .

Adding OWIN Startup and Authentication Configuration Classes

1. 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**.



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

```
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. See **OWIN Startup Class Detection (/aspnet/overview/owin-and-katana/owin-startup-class-detection)** for more info on this model.

Adding Web Forms for Registering and Logging in Users

1. Open the *Register.cs* file and add the following code which will log in the user when registration succeeds. The changes are highlighted below.

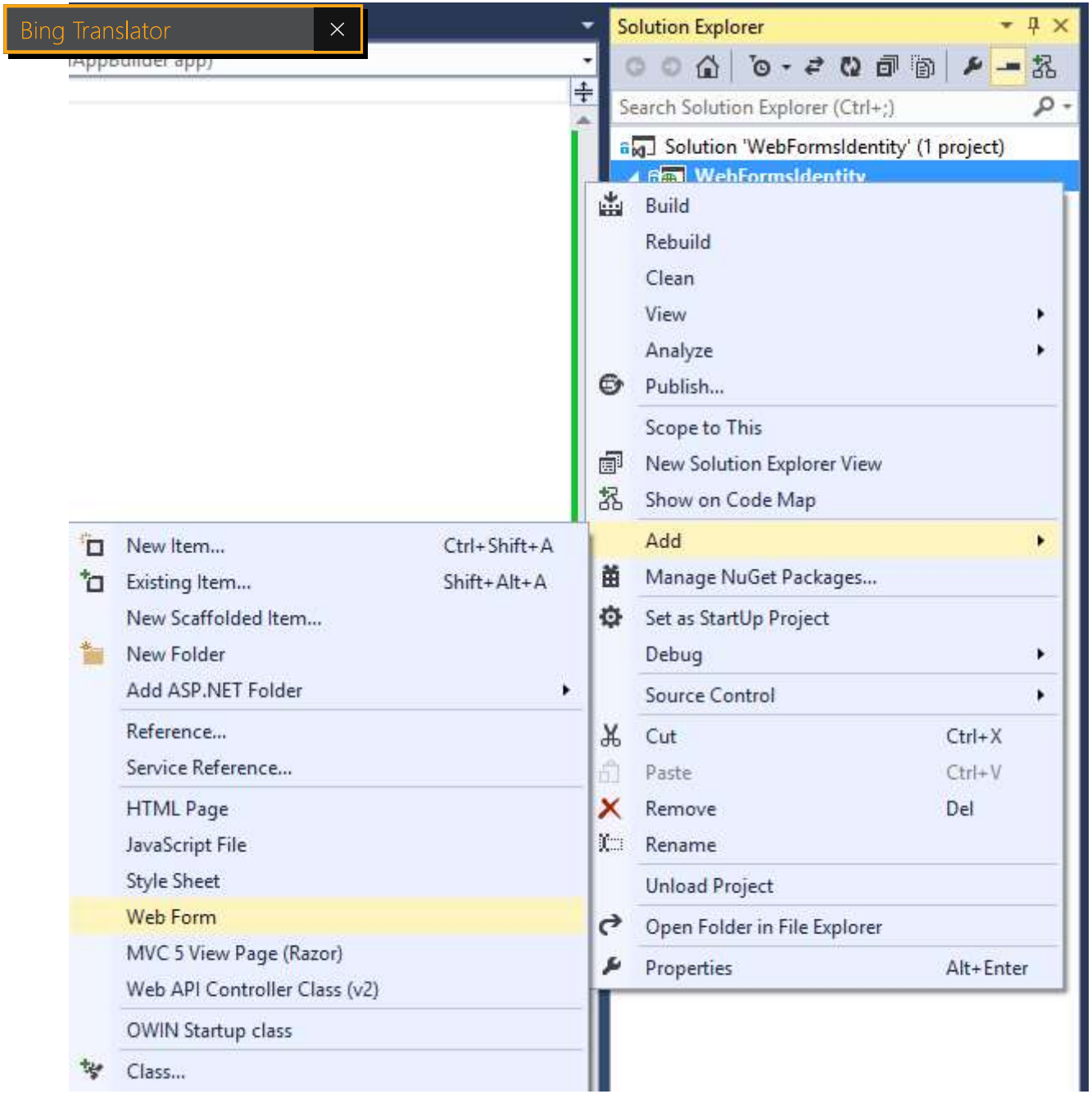
```
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.EntityFramework;
using Microsoft.Owin.Security;
```

```
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 (http://msdn.microsoft.com/en-us/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 (http://msdn.microsoft.com/en-**

m.web.security.formsauthentication.setauthcookie.aspx) used by the **FormsAuthentication (http://msdn.microsoft.com/en-**
m.web.security.formsauthenticationmodule.aspx) module.

2. In **Solution Explorer**, right-click your project click **Add**, and then **Web Form**. Name the web form **Login**.


Bing Translator
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'WebFormsIdentity' (1 project)
WebFormsIdentity
Build
Rebuild
Clean
View
Analyze
Publish...
Scope to This
New Solution Explorer View
Show on Code Map
Add
Manage NuGet Packages...
Set as StartUp Project
Debug
Source Control
Cut Ctrl+X
Paste Ctrl+V
Remove Del
Rename
Unload Project
Open Folder in File Explorer
Properties Alt+Enter
New Item... Ctrl+Shift+A
Existing Item... Shift+Alt+A
New Scaffolded Item...
New Folder
Add ASP.NET Folder
Reference...
Service Reference...
HTML Page
JavaScript File
Style Sheet
Web Form
MVC 5 View Page (Razor)
Web API Controller Class (v2)
OWIN Startup class
Class...

n.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>
```

```
                runat="server" ID="LogoutButton" Visible="false">
                <div>
                    <asp:Button runat="server" OnClick="SignOut" Text="Log out" />
                </div>
            </div>
        </asp:PlaceHolder>
    </div>
  </form>
</body>
</html>
```

4. 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;
```

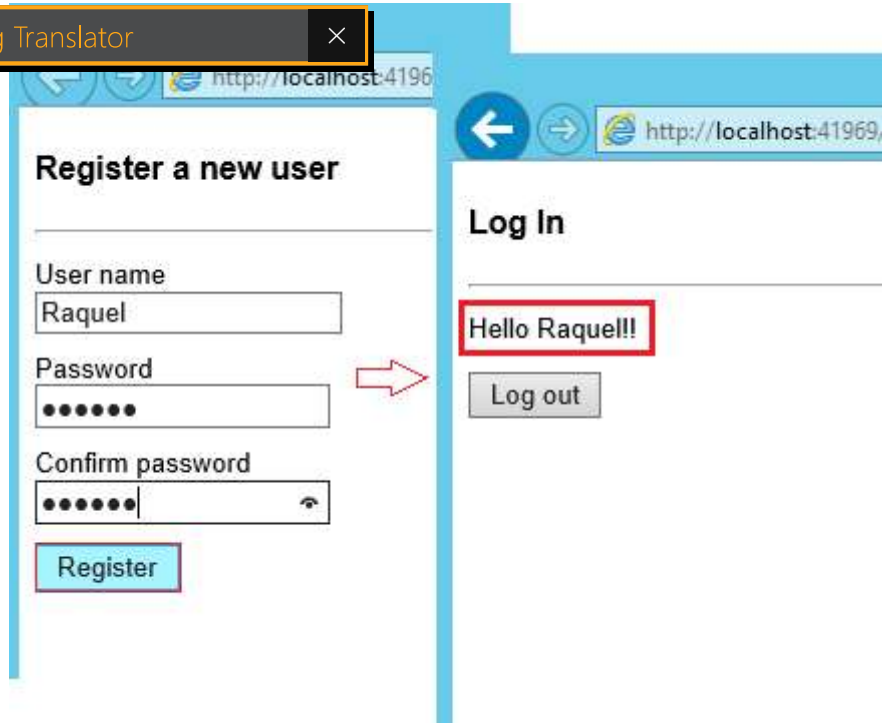
```
    }

    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 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 (http://msdn.microsoft.com/en-us/library/system.security.principal.iidentity.aspx)** that allows you to get the `UserName` and `UserId` for the logged in

ension methods are defined in the `Microsoft.AspNet.Identity.Core` assembly. These extension methods are the replacement for **ser.Identity.Name (http://msdn.microsoft.com/en-us/library/system.web.httpcontext.user.aspx)** .

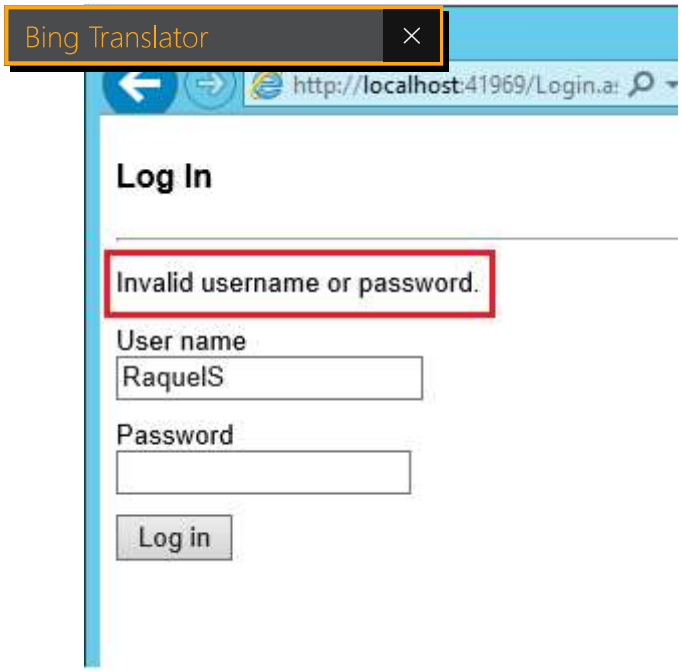
- SignIn method:
  `This` method replaces the previous `CreateUser_Click` method in this sample and now signs in the user after successfully creating the user.
  The Microsoft OWIN Framework has added extension methods on `System.Web.HttpContext` that allows you to get a reference to an `IOwinContext`. 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 (http://msdn.microsoft.com/en-us/library/system.web.security.formsauthentication.setauthcookie.aspx)** used by the **FormsAuthentication (http://msdn.microsoft.com/en-us/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 (http://msdn.microsoft.com/en-us/library/system.web.security.formsauthentication.signout.aspx)** method used by the **FormsAuthentication (http://msdn.microsoft.com/en-us/library/system.web.security.formsauthenticationmodule.aspx)** module.

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



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

6. Click on **Log out** button.You will be redirected to the Log in form.
7. 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.



This article was originally created on October 23, 2013

Author Information

Raquel Soares De Almeida – Raquel Almeida is a Software Developer Engineer in Test on the Azure Application Platform and Tools team where her focus is on ASP.NET and Web Stack Runtime.

Comments (52)