

Create an ASP.NET MVC 5 App with Facebook, Twitter, LinkedIn and Google OAuth2 Sign-on (C#)

04/03/2015 • 11 minutes to read • Contributors  all

In this article

[Getting Started](#)

[Creating Your First Application](#)

[Setting up SSL in the Project](#)

[Creating a Google app for OAuth 2 and connecting the app to the project](#)

[Creating the app in Facebook and connecting the app to the project](#)

[Examine the Membership Data](#)

[Adding Profile Data to the User Class](#)

[Examine the Membership Data](#)

[Logging off your App and Logging in With Another Account](#)

[Next Steps](#)

by [Rick Anderson](#)

This tutorial shows you how to build an ASP.NET MVC 5 web application that enables users to log in using [OAuth 2.0](#) with credentials from an external authentication provider, such as Facebook, Twitter, LinkedIn, Microsoft, or Google. For simplicity, this tutorial focuses on working with credentials from Facebook and Google.

Enabling these credentials in your web sites provides a significant advantage because millions of users already have accounts with these external providers. These users may be more inclined to sign up for your site if they do not have to create and remember a new set of credentials.

See also [ASP.NET MVC 5 app with SMS and email Two-Factor Authentication](#).

The tutorial also shows how to add profile data for the user, and how to use the Membership API to add roles. This tutorial was written by [Rick Anderson](#) (Please follow me on Twitter: [@RickAndMSFT](#)).

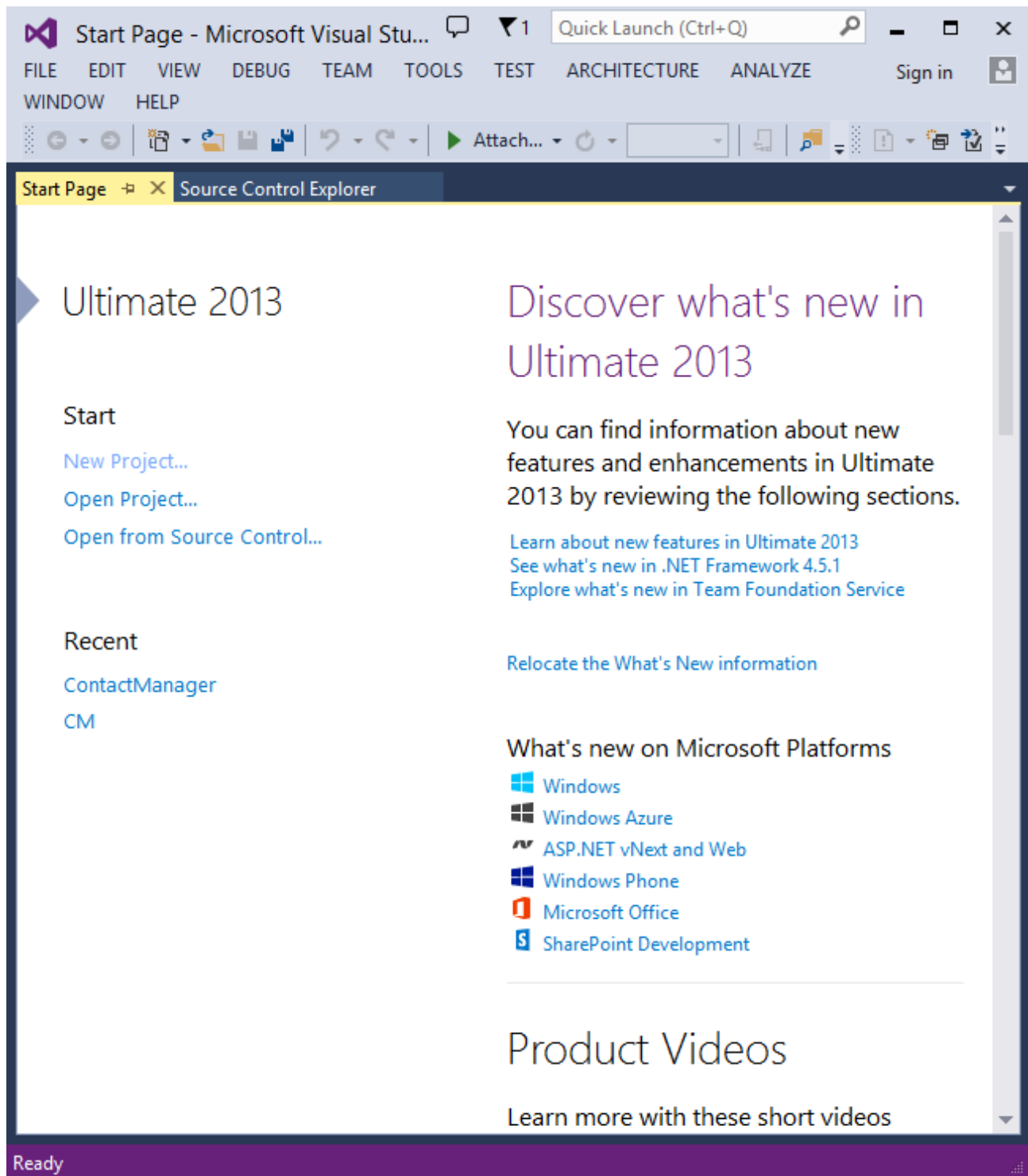
Getting Started

Start by installing and running [Visual Studio Express 2013 for Web](#) or [Visual Studio 2013](#). Install Visual Studio [2013 Update 3](#) or higher. For help with Dropbox, GitHub, LinkedIn, Instagram, Buffer, Salesforce, STEAM, Stack Exchange, Tripit, Twitch, Twitter, Yahoo!, and more, see this [sample project](#).

ⓘ Note

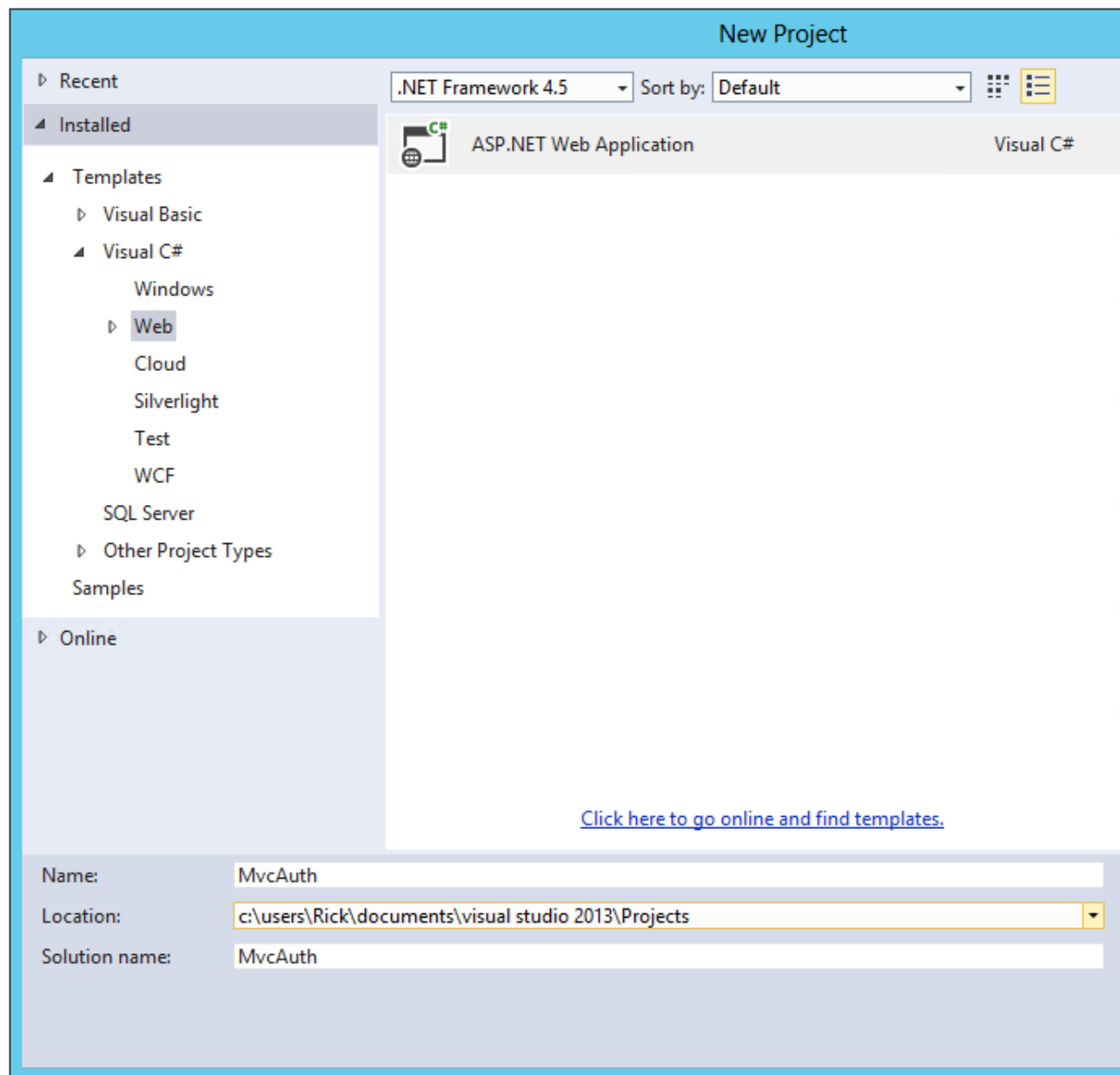
You must install Visual Studio [2013 Update 3](#) or higher to use Google OAuth 2 and to debug locally without SSL warnings.

Click **New Project** from the **Start** page, or you can use the menu and select **File**, and then **New Project**.

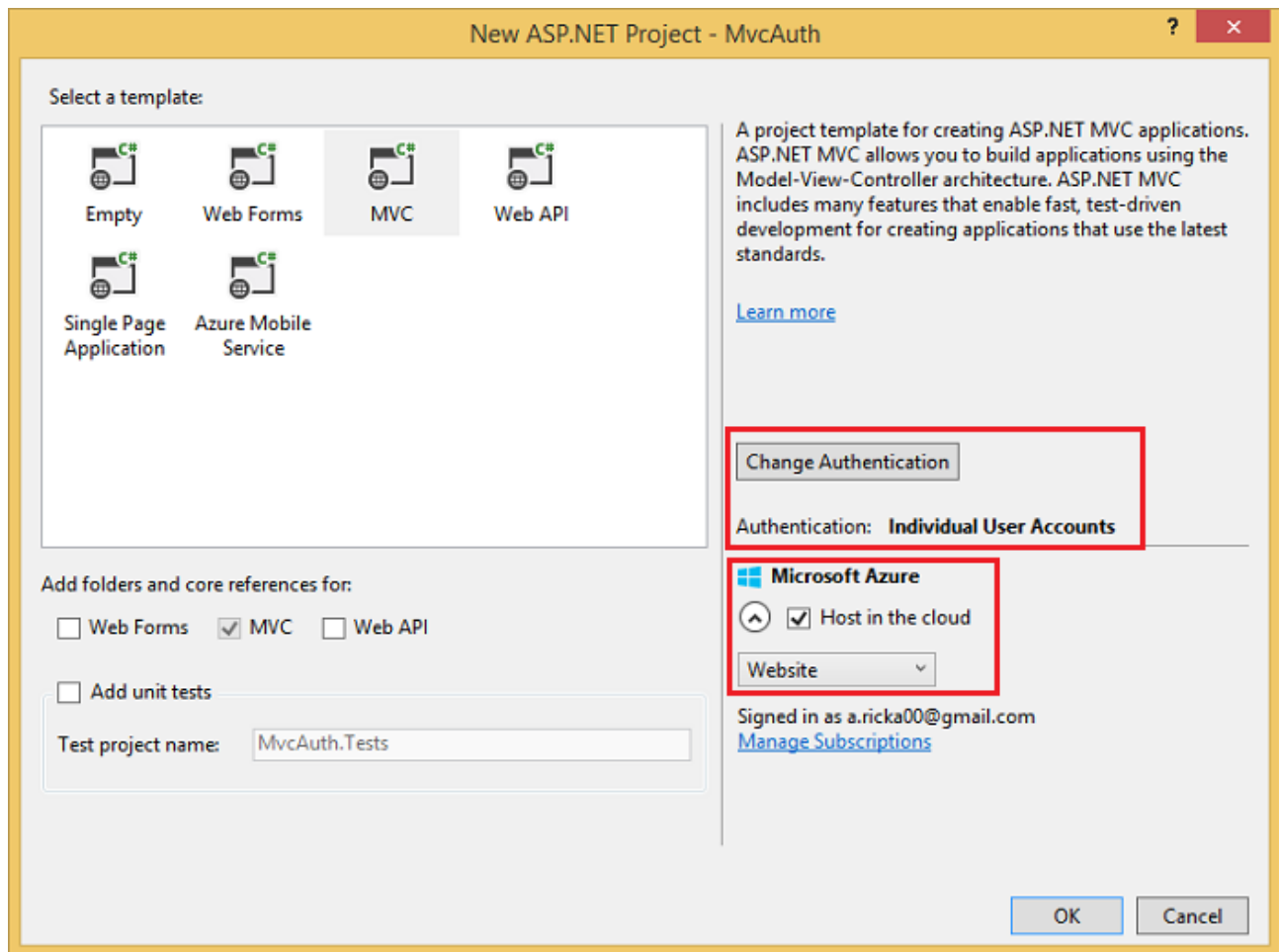


Creating Your First Application

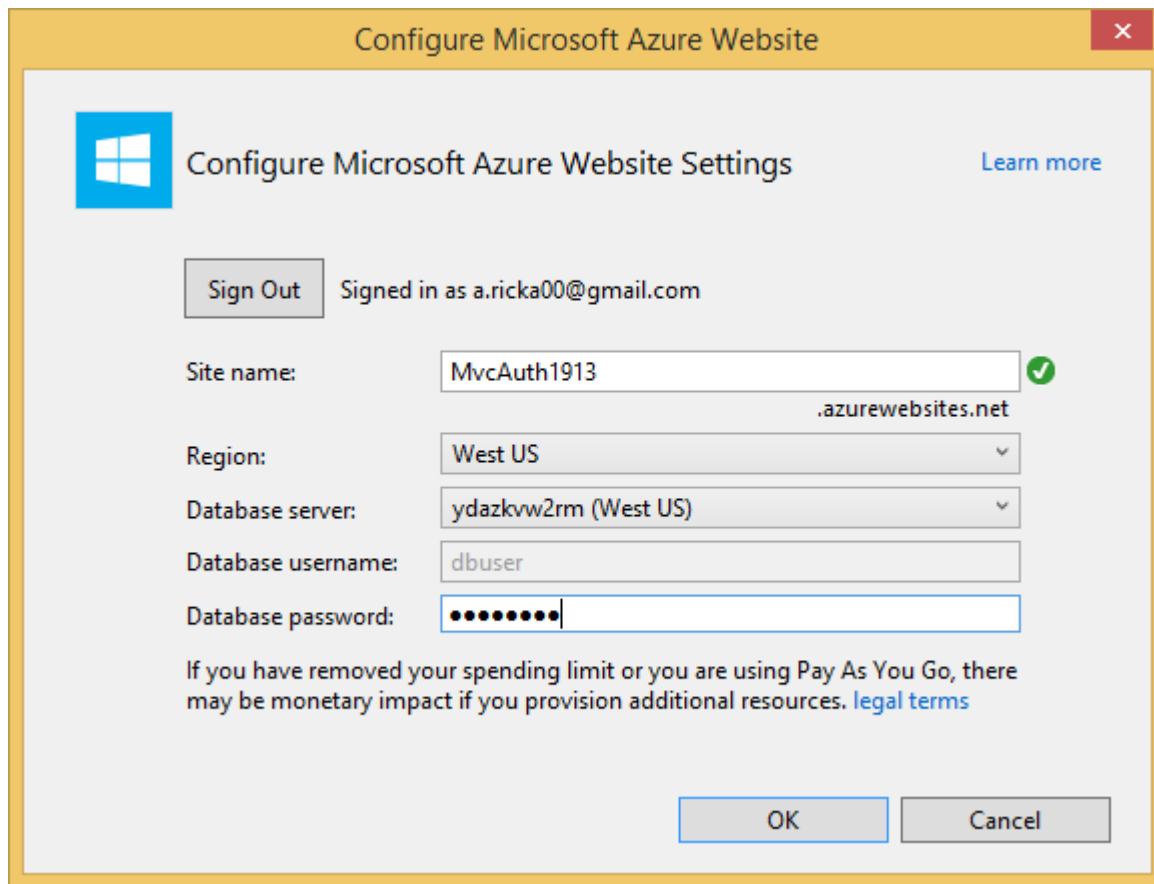
Click **New Project**, then select **Visual C#** on the left, then **Web** and then select **ASP.NET Web Application**. Name your project "MvcAuth" and then click **OK**.



In the **New ASP.NET Project** dialog, click **MVC**. If the Authentication is not **Individual User Accounts**, click the **Change Authentication** button and select **Individual User Accounts**. By checking **Host in the cloud**, the app will be very easy to host in Azure.



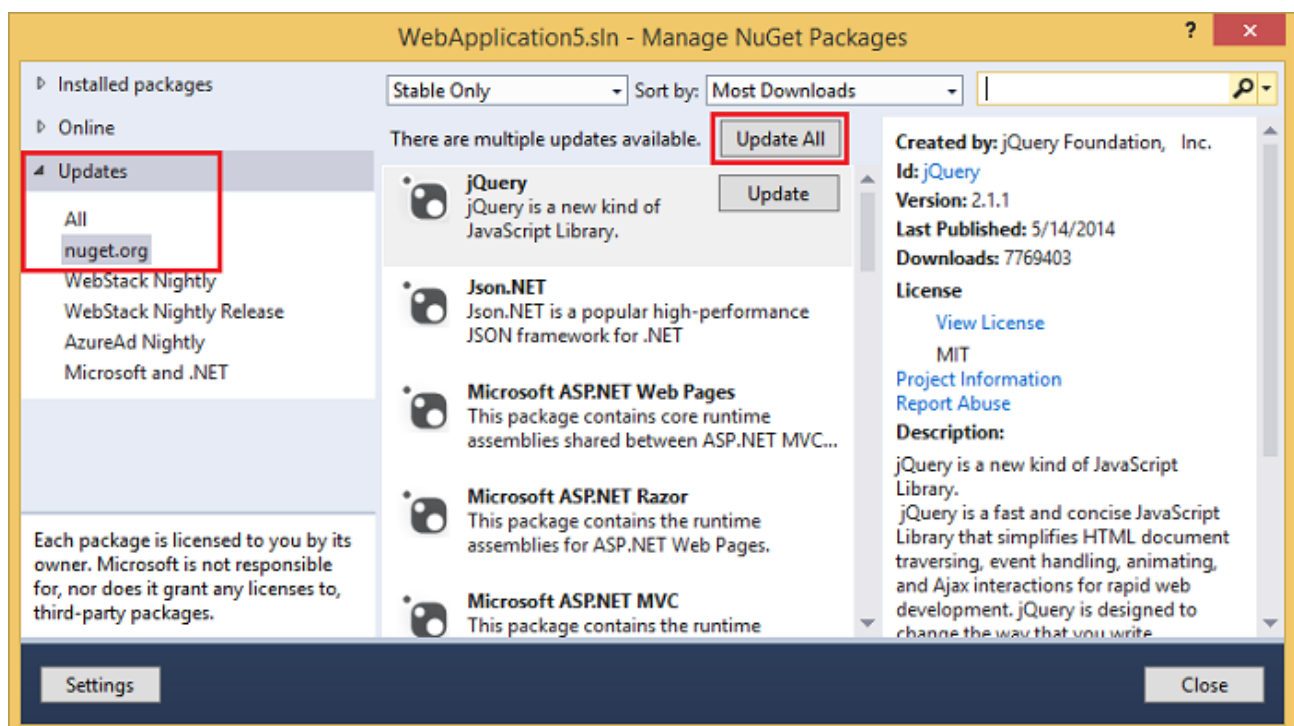
If you selected **Host in the cloud**, complete the configure dialog.



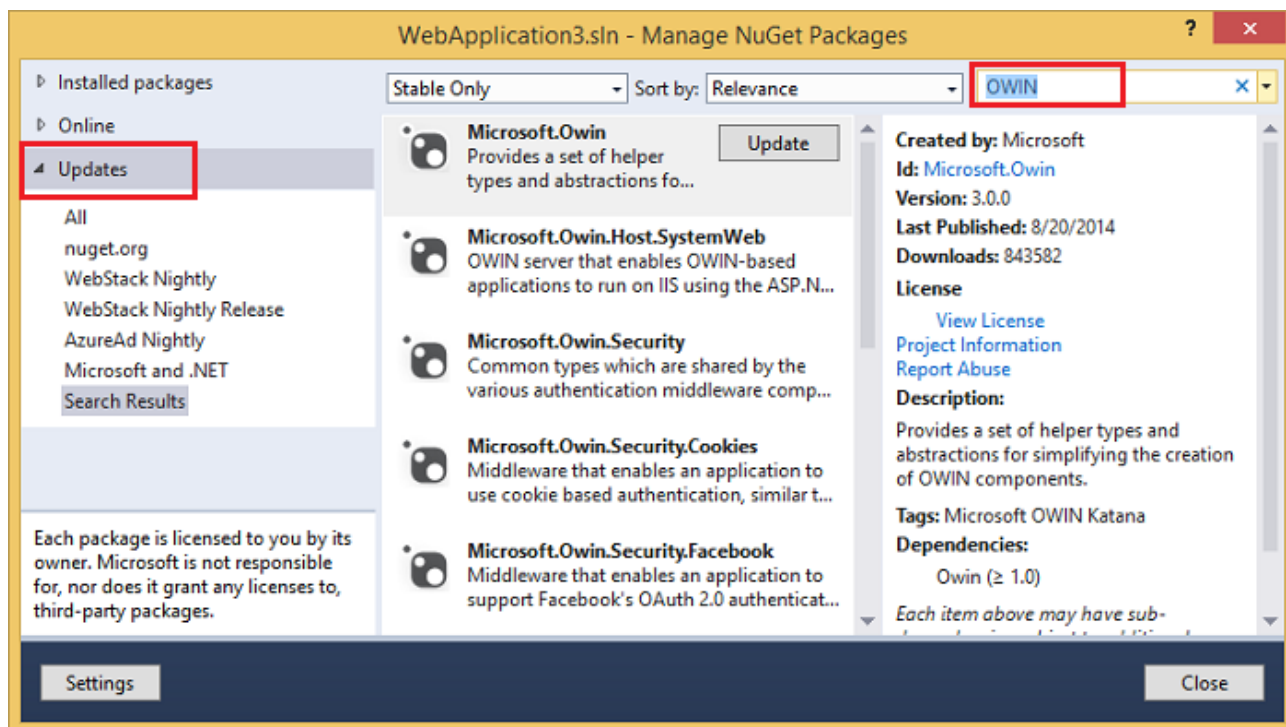
The image shows a 'Configure Microsoft Azure Website' dialog box. It has a title bar with a close button. Inside, there's a Windows logo and the title 'Configure Microsoft Azure Website Settings' with a 'Learn more' link. Below this is a 'Sign Out' button and the text 'Signed in as a.ricka00@gmail.com'. The main section contains several fields: 'Site name' with the value 'MvcAuth1913' and a green checkmark, followed by '.azurewebsites.net'; 'Region' with a dropdown set to 'West US'; 'Database server' with a dropdown set to 'ydazkvw2rm (West US)'; 'Database username' with the value 'dbuser'; and 'Database password' with a masked input field. At the bottom, there's a note about spending limits and a 'legal terms' link, followed by 'OK' and 'Cancel' buttons.

Use NuGet to update to the latest OWIN middleware

Use the NuGet package manager to update the [OWIN middleware](#). Select **Updates** in the left menu. You can click on the **Update All** button or you can search for only OWIN packages (shown in the next image):



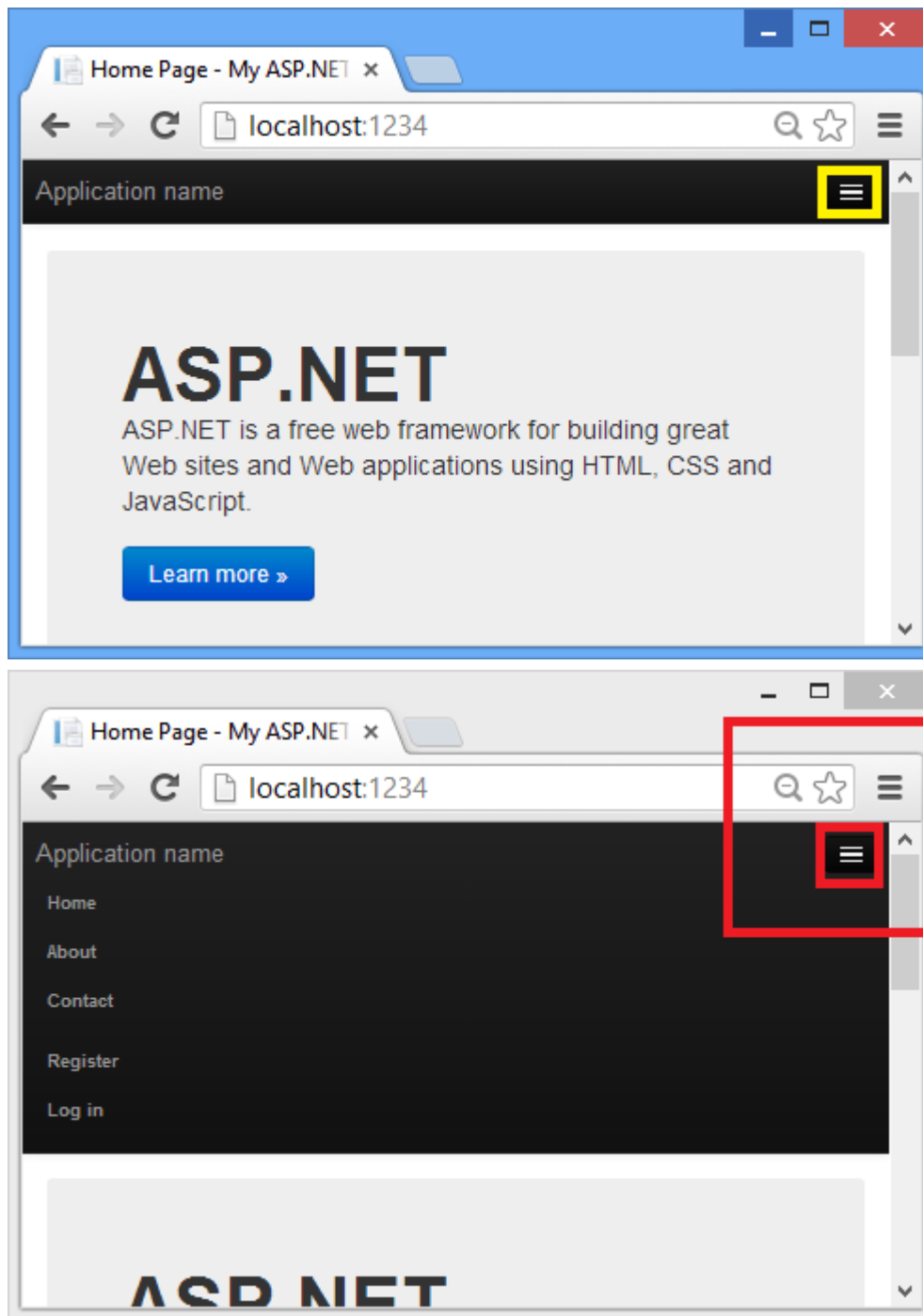
In the image below, only OWIN packages are shown:



From the Package Manager Console (PMC), you can enter the `Update-Package` command, which will update all packages.

Press **F5** or **Ctrl+F5** to run the application. In the image below, the port number is 1234. When you run the application, you'll see a different port number.

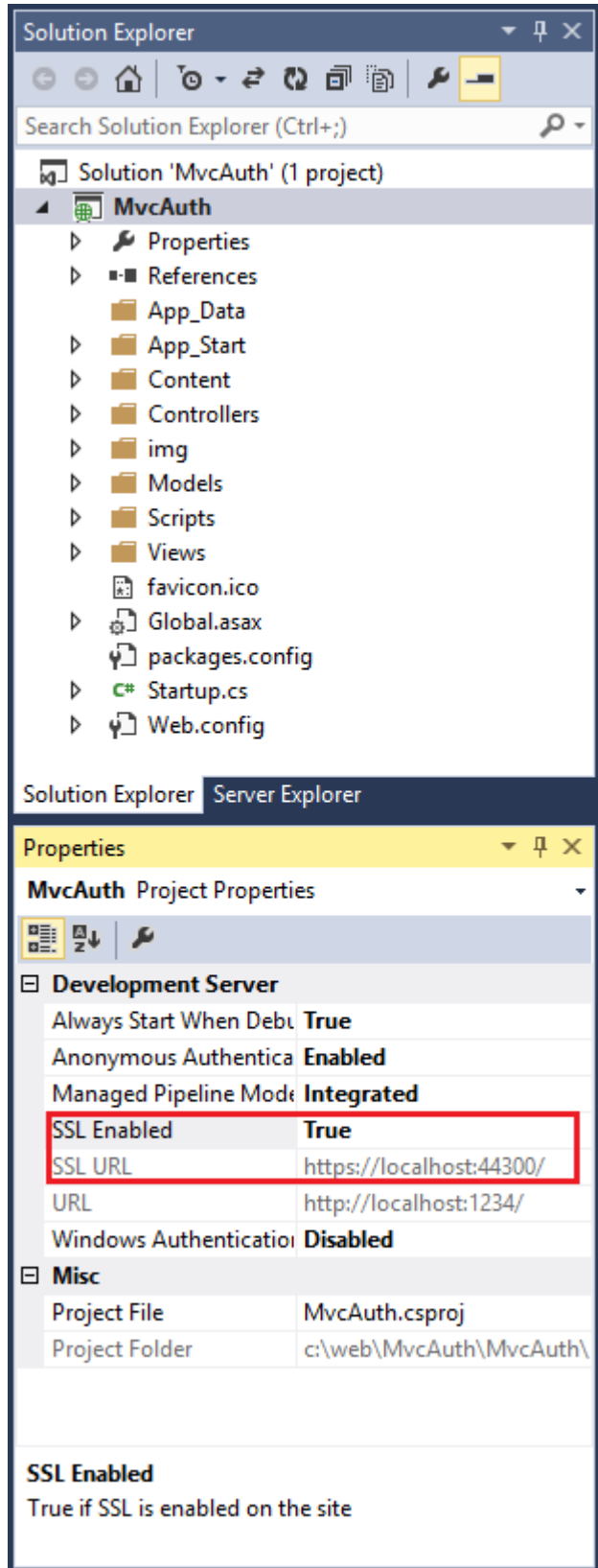
Depending on the size of your browser window, you might need to click the navigation icon to see the **Home**, **About**, **Contact**, **Register** and **Log in** links.



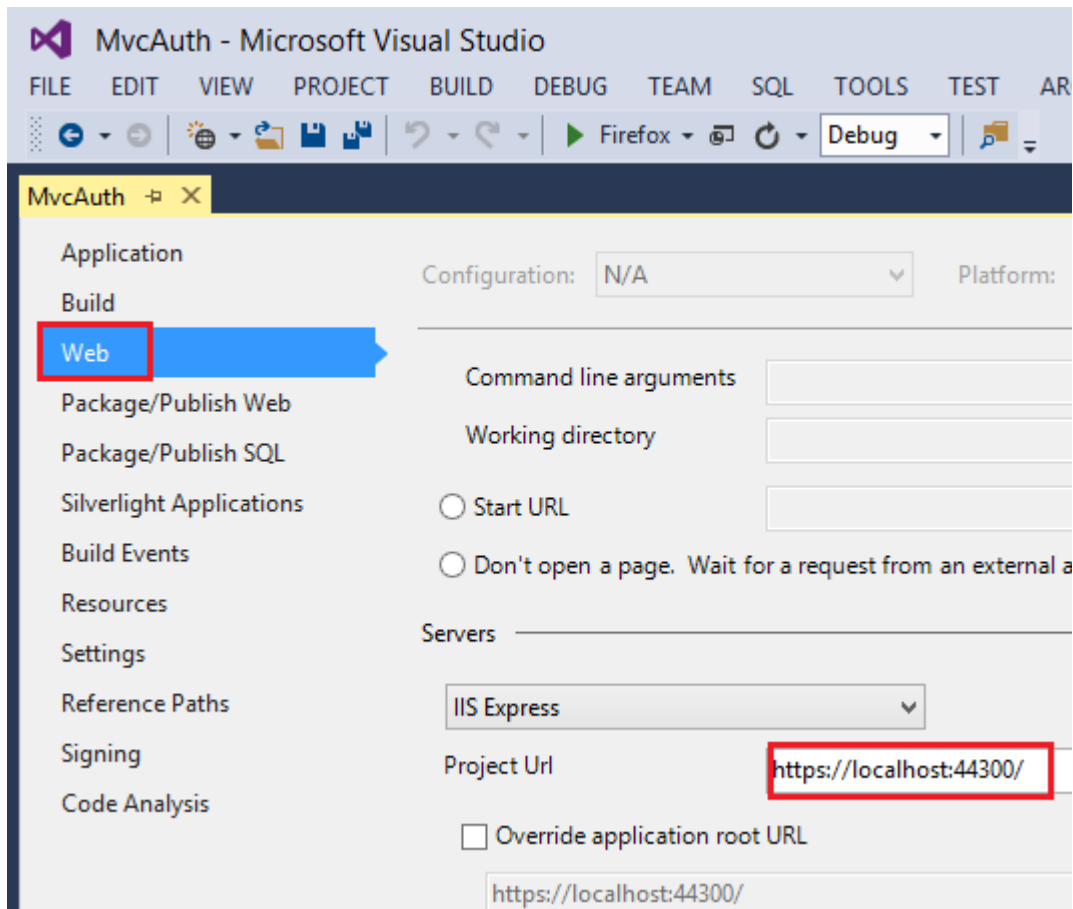
Setting up SSL in the Project

To connect to authentication providers like Google and Facebook, you will need to set up IIS-Express to use SSL. It's important to keep using SSL after login and not drop back to HTTP, your login cookie is just as secret as your username and password, and without using SSL you're sending it in clear-text across the wire. Besides, you've already taken the time to perform the handshake and secure the channel (which is the bulk of what makes HTTPS slower than HTTP) before the MVC pipeline is run, so redirecting back to HTTP after you're logged in won't make the current request or future requests much faster.

1. In **Solution Explorer**, click the **MvcAuth** project.
2. Hit the F4 key to show the project properties. Alternatively, from the **View** menu you can select **Properties Window**.
3. Change **SSL Enabled** to True.



4. Copy the SSL URL (which will be `https://localhost:44300/` unless you've created other SSL projects).
5. In **Solution Explorer**, right click the **MvcAuth** project and select **Properties**.
6. Select the **Web** tab, and then paste the SSL URL into the **Project Url** box. Save the file (Ctl+S). You will need this URL to configure Facebook and Google authentication apps.

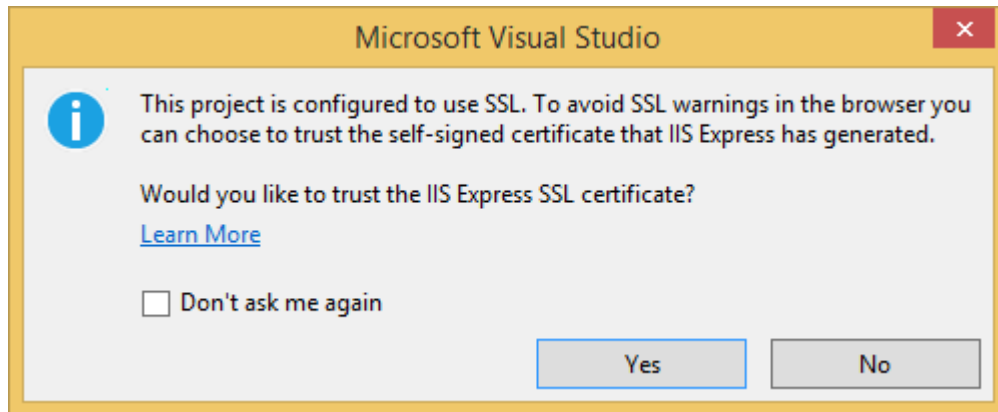


7. Add the [RequireHttps](#) attribute to the `Home` controller to require all requests must use HTTPS. A more secure approach is to add the [RequireHttps](#) filter to the application. See the section "Protect the Application with SSL and the Authorize Attribute" in my tutorial [Create an ASP.NET MVC app with auth and SQL DB and deploy to Azure App Service](#). A portion of the Home controller is shown below.

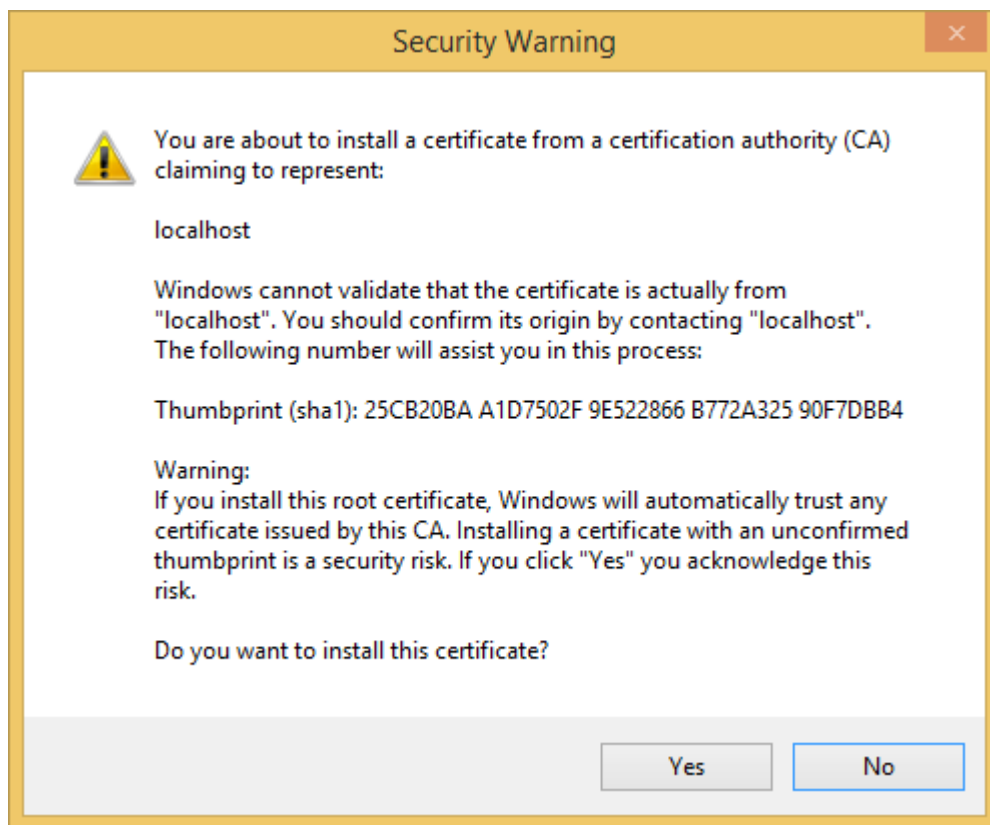
```
C# Copy  
  
[RequireHttps]  
public class HomeController : Controller  
{  
    public ActionResult Index()  
    {  
          
    }  
}
```

```
return View();  
}
```

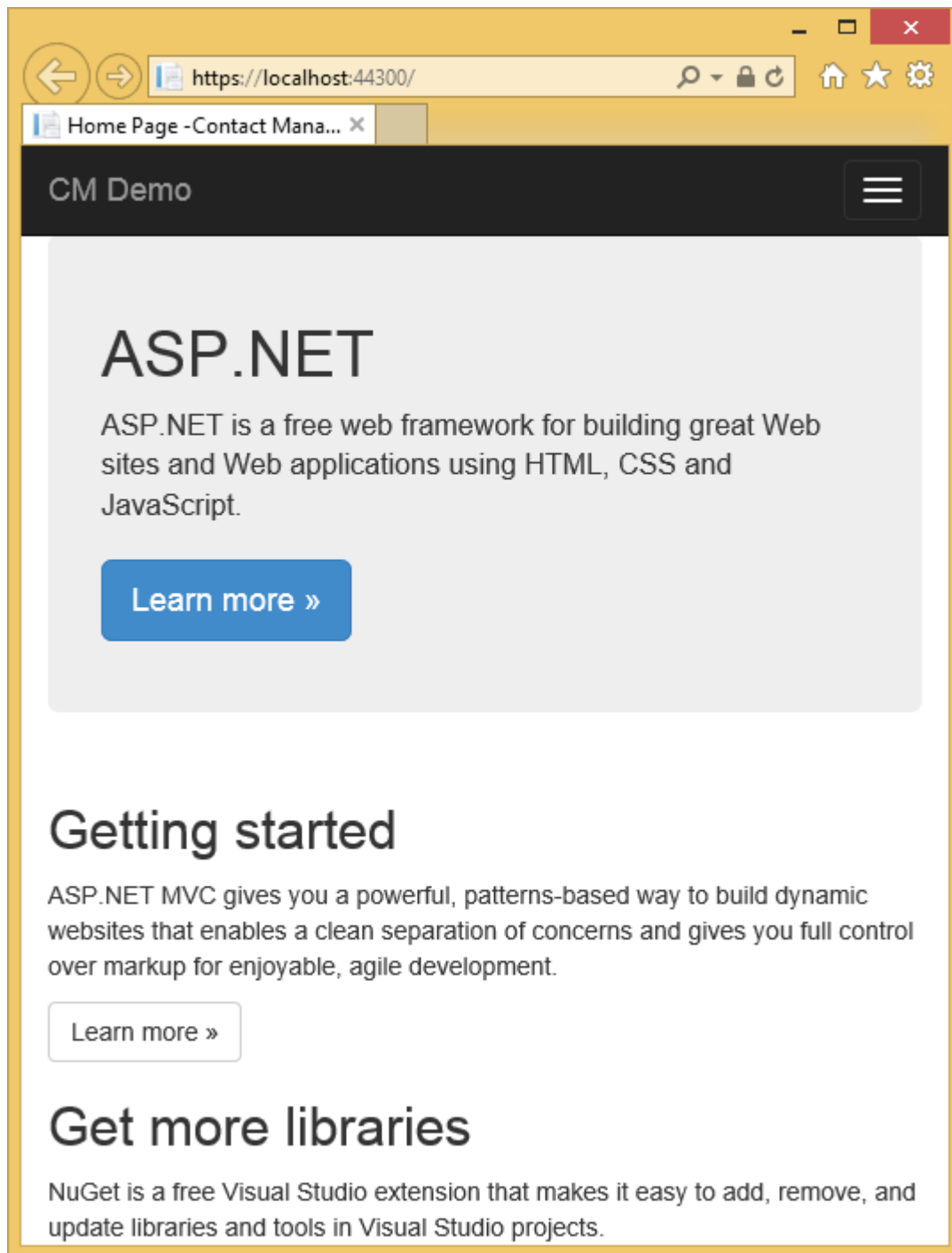
8. Press CTRL+F5 to run the application. If you've installed the certificate in the past, you can skip the rest of this section and jump to [Creating a Google app for OAuth 2 and connecting the app to the project](#), otherwise, follow the instructions to trust the self-signed certificate that IIS Express has generated.



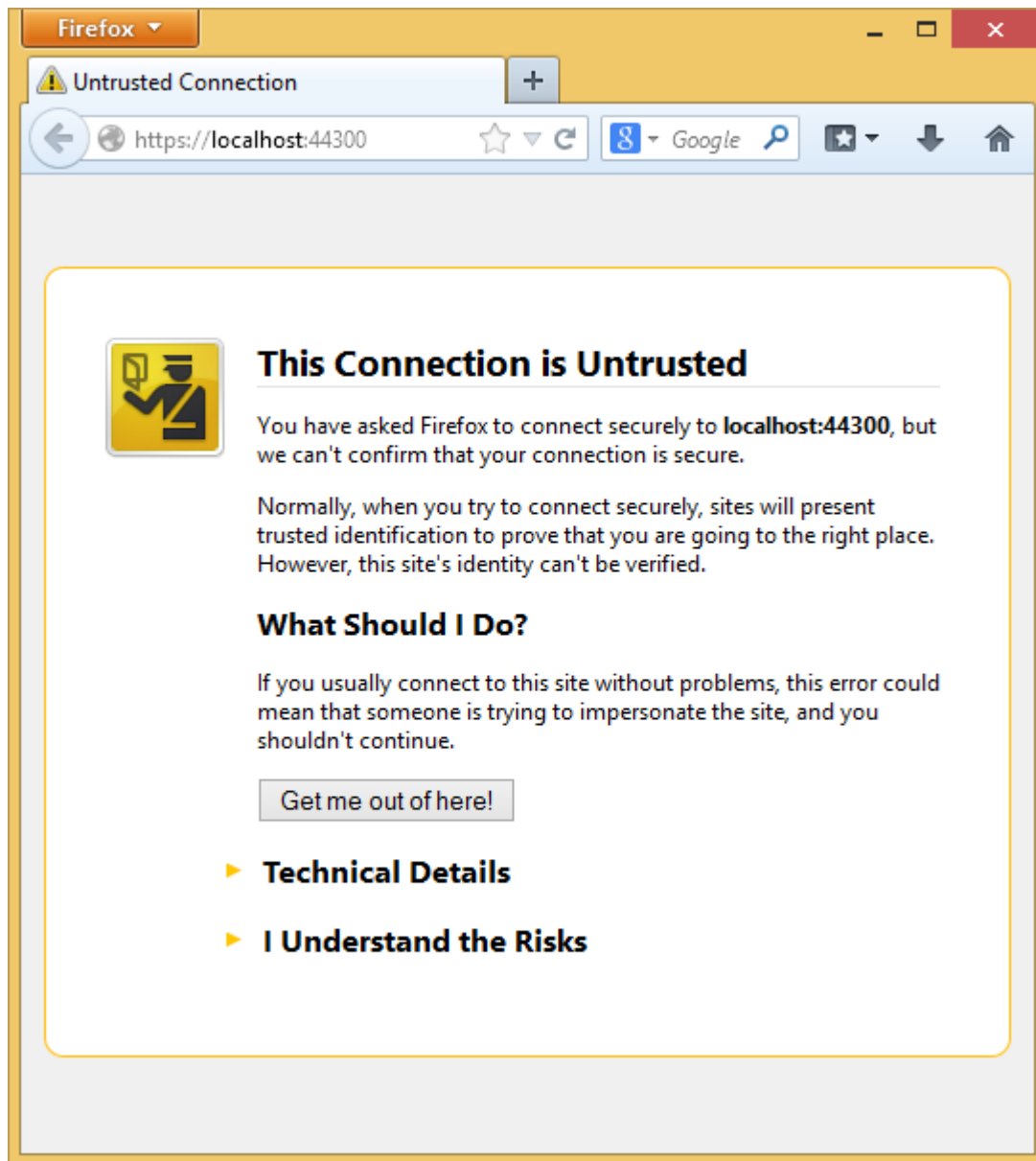
9. Read the **Security Warning** dialog and then click **Yes** if you want to install the certificate representing localhost.



10. IE shows the *Home* page and there are no SSL warnings.



11. Google Chrome also accepts the certificate and will show HTTPS content without a warning. Firefox uses its own certificate store, so it will display a warning. For our application you can safely click **I Understand the Risks**.



Creating a Google app for OAuth 2 and connecting the app to the project

Warning

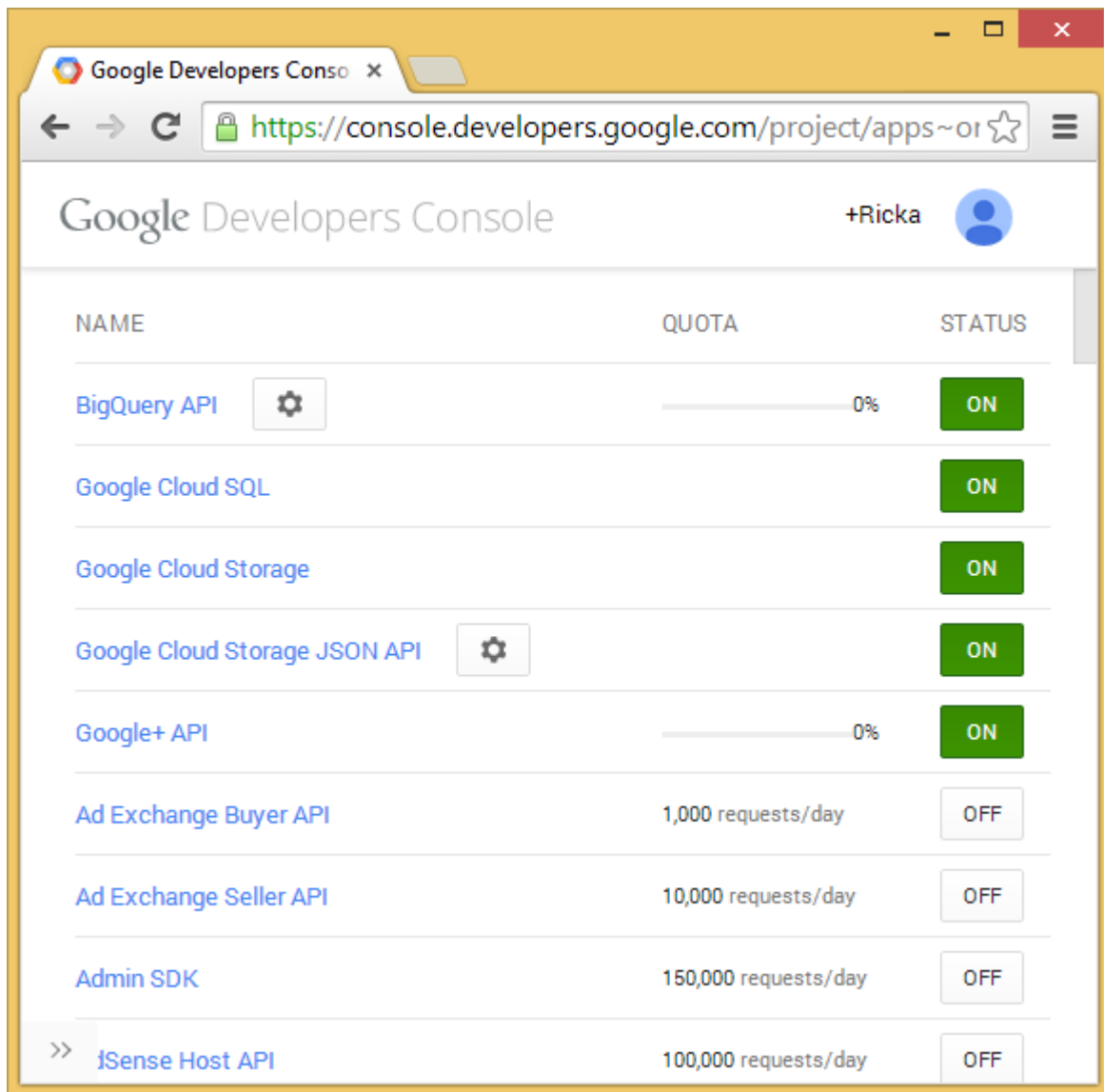
For current Google OAuth instructions, see [Configuring Google authentication in ASP.NET Core](#).

1. Navigate to the [Google Developers Console](#).
2. If you haven't created a project before, select **Credentials** in the left tab, and then select **Create**.

3. In the left tab, click **Credentials**.
4. Click **Create credentials** then **OAuth client ID**.
 - a. In the **Create Client ID** dialog, keep the default **Web application** for the application type.
 - b. Set the **Authorized JavaScript** origins to the SSL URL you used above (`https://localhost:44300/` unless you've created other SSL projects)
 - c. Set the **Authorized redirect URI** to:
`https://localhost:44300/signin-google`
5. Click the OAuth Consent screen menu item, then set your email address and product name. When you have completed the form click **Save**.
6. Click the Library menu item, search **Google+ API**, click on it then press Enable.

Google Spectrum Database API	1,000 requests/day	OFF
Google Webmaster Tools API	1,000,000 requests/day	OFF
Google+ API	10,000 requests/day	OFF
Google+ Domains API	10,000 requests/day	OFF
Google+ Hangouts API		OFF
Groups Migration API	500,000 requests/day	OFF
Groups Settings API	100,000 requests/day	OFF
Identity Toolkit API	1,000,000 requests/day	OFF

The image below shows the enabled APIs.



7. From the Google APIs API Manager, visit the **Credentials** tab to obtain the **Client ID**. Download to save a JSON file with application secrets. Copy and paste the **ClientId** and **ClientSecret** into the `UseGoogleAuthentication` method found in the `Startup.Auth.cs` file in the `App_Start` folder. The **ClientId** and **ClientSecret** values shown below are samples and don't work.

C# Copy

```

public void ConfigureAuth(IAppBuilder app)
{
    // Configure the db context and user manager to use a single instance
    // per request
    app.CreatePerOwinContext(ApplicationDbContext.Create);
    app.CreatePerOwinContext<ApplicationUserManager>
    (ApplicationUserManager.Create);

    // Enable the application to use a cookie to store information for the
    // signed in user

```

```
// and to use a cookie to temporarily store information about a user
logging in with a third party login provider
// Configure the sign in cookie
app.UseCookieAuthentication(new CookieAuthenticationOptions
{
    AuthenticationType = DefaultAuthenticationTypes.ApplicationCookie,
    LoginPath = new PathString("/Account/Login"),
    Provider = new CookieAuthenticationProvider
    {
        OnValidateIdentity =
SecurityStampValidator.OnValidateIdentity<ApplicationUserManager,
ApplicationUser>(
    validateInterval: TimeSpan.FromMinutes(30),
    regenerateIdentity: (manager, user) =>
user.GenerateUserIdentityAsync(manager))
    }
});

app.UseExternalSignInCookie(DefaultAuthenticationTypes.ExternalCookie);

// Uncomment the following lines to enable logging in with third party
login providers
//app.UseMicrosoftAccountAuthentication(
//    clientId: "",
//    clientSecret: "");

//app.UseTwitterAuthentication(
//    consumerKey: "",
//    consumerSecret: "");

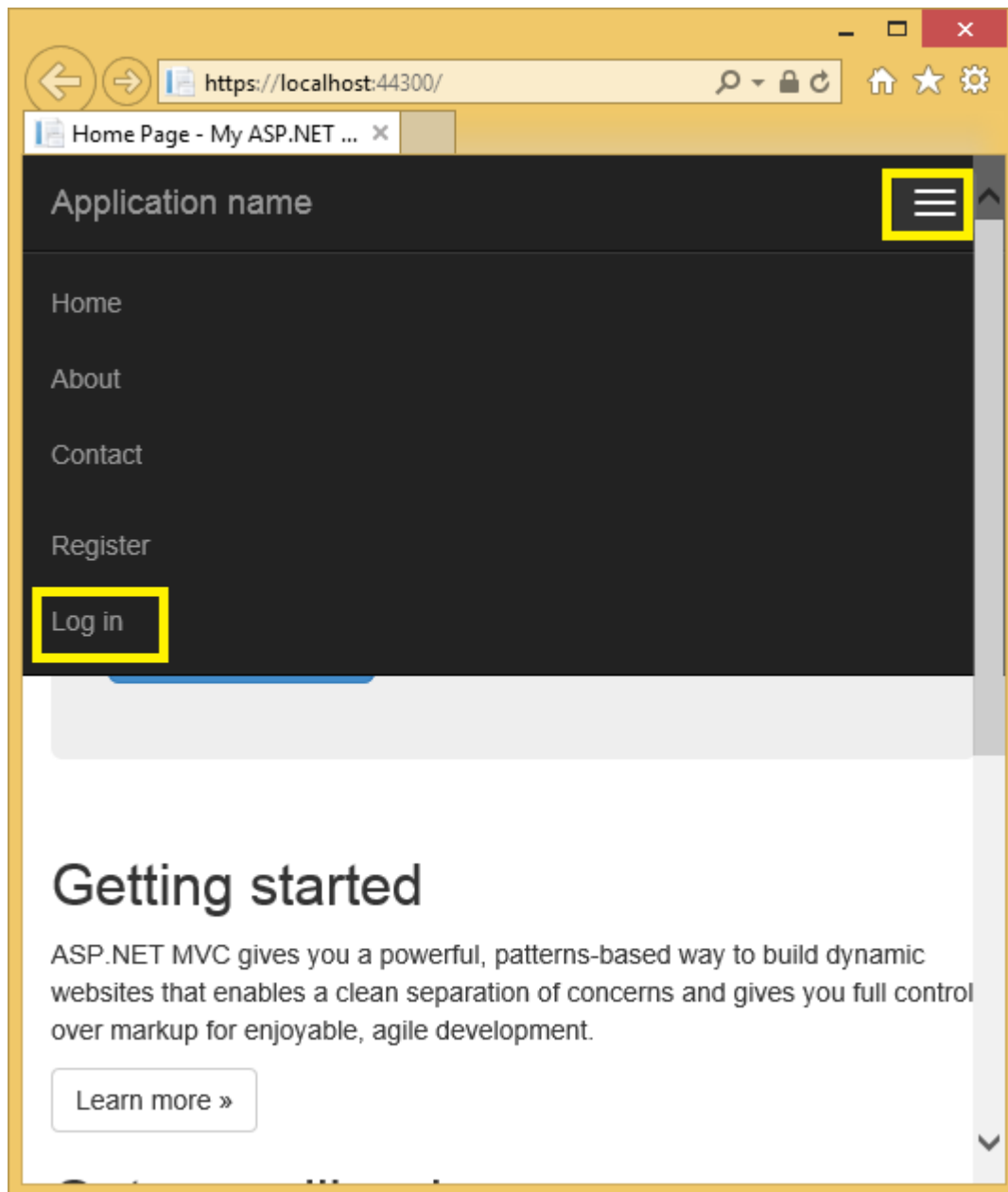
//app.UseFacebookAuthentication(
//    appId: "",
//    appSecret: "");

app.UseGoogleAuthentication(
    clientId: "000-000.apps.googleusercontent.com",
    clientSecret: "000000000000");
}
```

Warning

Security - Never store sensitive data in your source code. The account and credentials are added to the code above to keep the sample simple. See [Best practices for deploying passwords and other sensitive data to ASP.NET and Azure App Service](#).

8. Press **CTRL+F5** to build and run the application. Click the **Log in** link.



9. Under **Use another service to log in**, click **Google**.

Application name

Log in.

Use a local account to log in.

Email

Password

☐ Remember me?

Log in

[Register as a new user](#)

Use another service to log in.

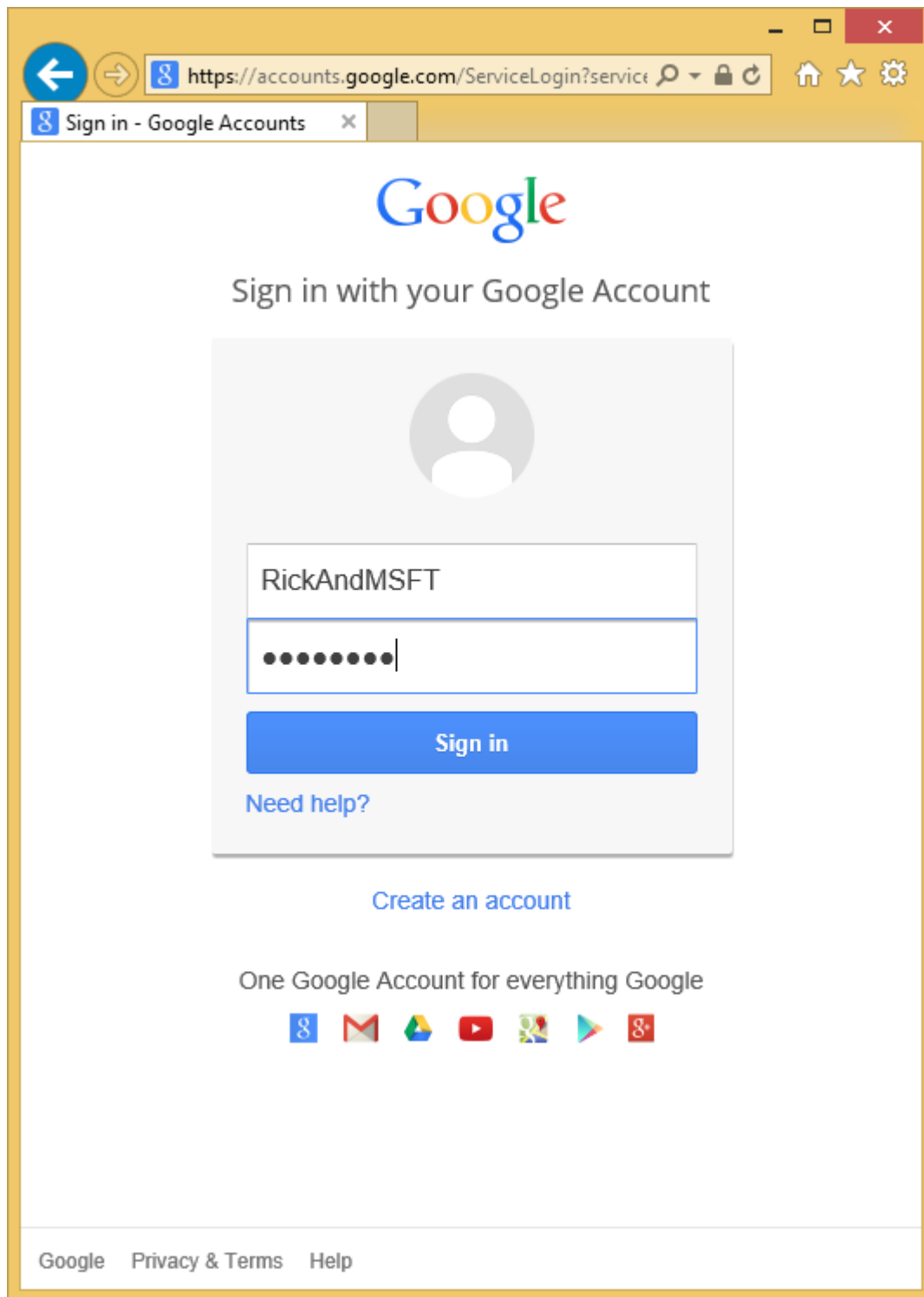
Google

© 2014 - My ASP.NET Application

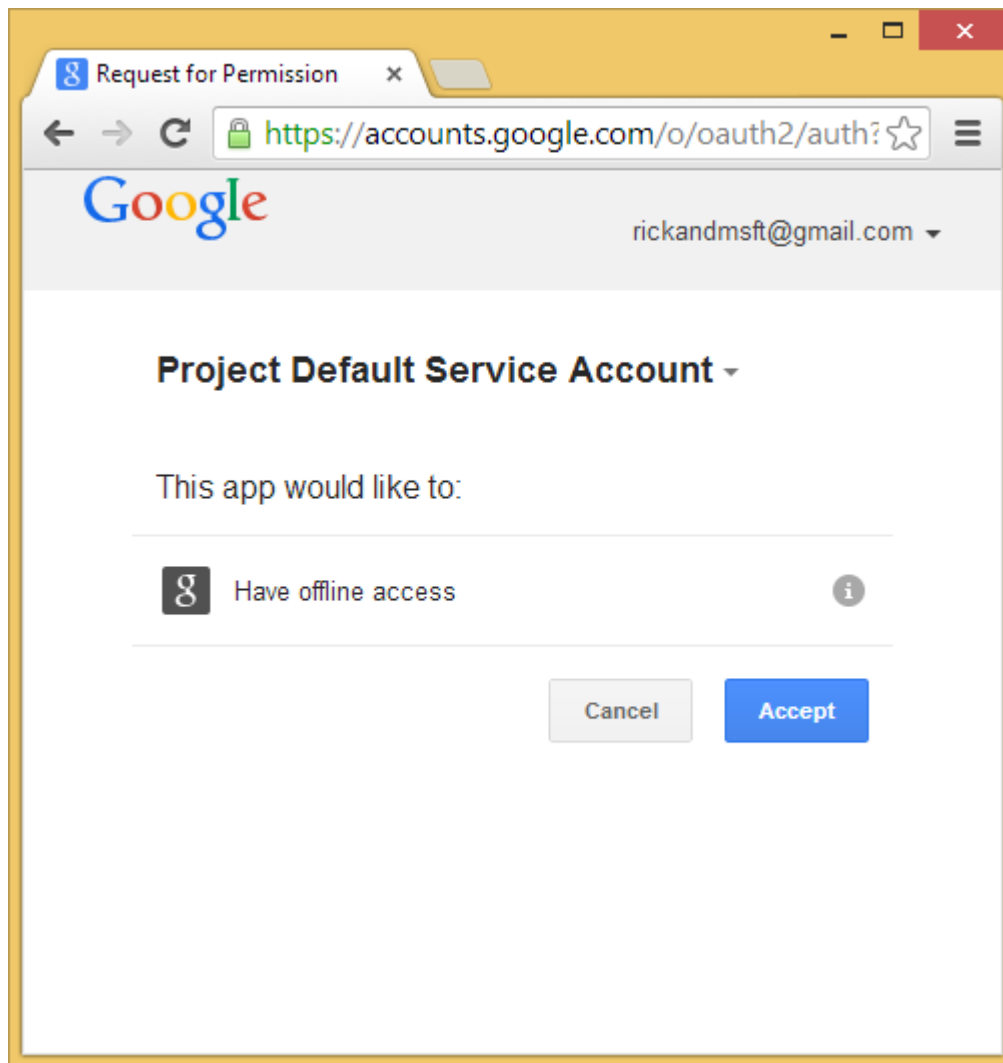
ⓘ Note

If you miss any of the steps above you will get a HTTP 401 error. Recheck your steps above. If you miss a required setting (for example **product name**), add the missing item and save; it can take a few minutes for authentication to work.

10. You will be redirected to the Google site where you will enter your credentials.



11. After you enter your credentials, you will be prompted to give permissions to the web application you just created:



12. Click **Accept**. You will now be redirected back to the **Register** page of the MvcAuth application where you can register your Google account. You have the option of changing the local email registration name used for your Gmail account, but you generally want to keep the default email alias (that is, the one you used for authentication). Click **Register**.

Application name

Register.

Associate your Google account.

Association Form

You've successfully authenticated with **Google**. Please enter a user name for this site below and click the Register button to finish logging in.

Email

Register

© 2014 - My ASP.NET Application

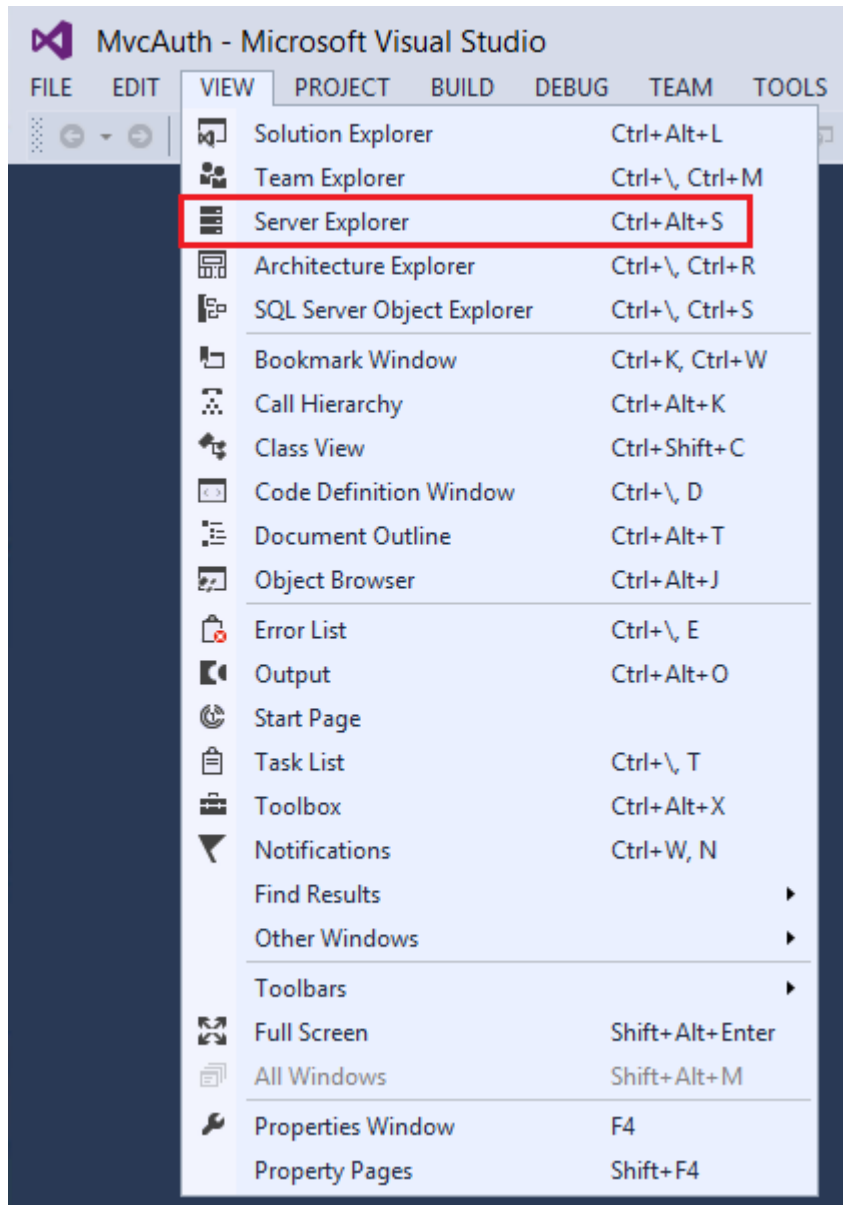
Creating the app in Facebook and connecting the app to the project

 **Warning**

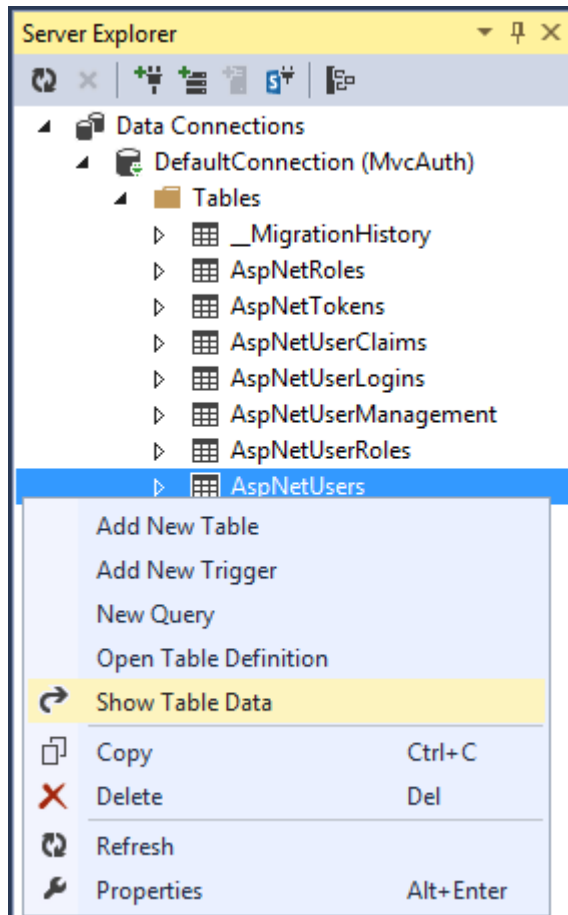
For current Facebook OAuth2 authentication instructions, see [Configuring Facebook authentication](#)

Examine the Membership Data

In the **View** menu, click **Server Explorer**.



Expand **DefaultConnection (MvcAuth)**, expand **Tables**, right click **AspNetUsers** and click **Show Table Data**.



dbo.AspNetUsers [Data]						
Max Rows: 1000						
	Id	Email	EmailConfirmed	PasswordHash	SecurityStamp	PhoneNumber
	8878-475f87d6efaf	rick.FB@contoso.com	False	NULL	4fb9fed9-0028-...	NULL
	727d3bc0-74cf-...	rickandmsft@gmail.com	False	NULL	228178d3-d11f-...	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL

Adding Profile Data to the User Class

In this section you'll add birth date and home town to the user data during registration, as shown in the following image.

Application name

Register.

Associate your Google account.

Association Form

You've successfully authenticated with **Google**. Please enter a user name for this site below and click the Register button to finish logging in.

Email

HomeTown

BirthDate

Register

© 2014 - My ASP.NET Application

Open the *Models\IdentityModels.cs* file and add birth date and home town properties:

C#	Copy
<pre>public class ApplicationUser : IdentityUser { public string HomeTown { get; set; } public System.DateTime? BirthDate { get; set; } public async Task<ClaimsIdentity> GenerateUserIdentityAsync(UserManager<ApplicationUser> manager)</pre>	


```
{  
    // Note the authenticationType must match the one defined in  
    CookieAuthenticationOptions.AuthenticationType  
    var userIdentity = await manager.CreateIdentityAsync(this,  
DefaultAuthenticationTypes.ApplicationCookie);  
    // Add custom user claims here  
    return userIdentity;  
}  
}
```

Open the *Models\AccountViewModels.cs* file and the set birth date and home town properties in `ExternalLoginConfirmationViewModel`.

C#

 Copy

```
public class ExternalLoginConfirmationViewModel  
{  
    [Required]  
    [EmailAddress]  
    [Display(Name = "Email")]  
    public string Email { get; set; }  
  
    public string HomeTown { get; set; }  
    public System.DateTime? BirthDate { get; set; }  
}
```

Open the *Controllers\AccountController.cs* file and add code for birth date and home town in the `ExternalLoginConfirmation` action method as shown:

C#

 Copy

```
[HttpPost]  
[AllowAnonymous]  
[ValidateAntiForgeryToken]  
public async Task<ActionResult>  
ExternalLoginConfirmation(ExternalLoginConfirmationViewModel model, string  
returnUrl)  
{  
    if (User.Identity.IsAuthenticated)  
    {  
        return RedirectToAction("Manage");  
    }  
  
    if (ModelState.IsValid)  
    {  
        // Get the information about the user from the external login provider  
        var info = await AuthenticationManager.GetExternalLoginInfoAsync();
```

```

if (info == null)
{
    return View("ExternalLoginFailure");
}
var user = new ApplicationUser()
{
    UserName = model.Email, Email = model.Email,
    BirthDate = model.BirthDate,
    HomeTown = model.HomeTown
};
IdentityResult result = await UserManager.CreateAsync(user);
if (result.Succeeded)
{
    result = await UserManager.AddLoginAsync(user.Id, info.Login);
    if (result.Succeeded)
    {
        await SignInAsync(user, isPersistent: false);

        // For more information on how to enable account confirmation
        // and password reset please visit http://go.microsoft.com/fwlink/?LinkID=320771
        // Send an email with this link
        // string code = await
        UserManager.GenerateEmailConfirmationTokenAsync(user.Id);
        // var callbackUrl = Url.Action("ConfirmEmail", "Account", new
        { userId = user.Id, code = code }, protocol: Request.Url.Scheme);
        // SendEmail(user.Email, callbackUrl, "Confirm your account",
        "Please confirm your account by clicking this link");

        return RedirectToLocal(returnUrl);
    }
}
AddErrors(result);
}

ViewBag.ReturnUrl = returnUrl;
return View(model);
}

```

Add birth date and home town to the *Views\Account\ExternalLoginConfirmation.cshtml* file:

CSHTML

 Copy

```

@model MvcAuth.Models.ExternalLoginConfirmationViewModel
@{
    ViewBag.Title = "Register";
}
<h2>@ViewBag.Title.</h2>
<h3>Associate your @ViewBag.LoginProvider account.</h3>

```

```

@using (Html.BeginForm("ExternalLoginConfirmation", "Account", new { ReturnUrl
= ViewBag.ReturnUrl }, FormMethod.Post, new { @class = "form-horizontal", role
= "form" }))
{
    @Html.AntiForgeryToken()

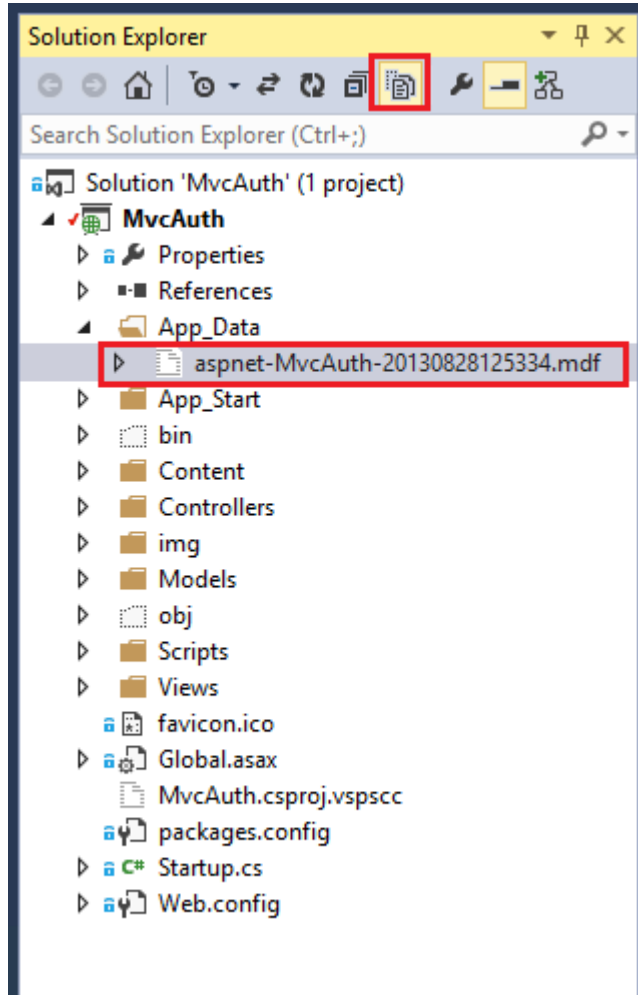
    <h4>Association Form</h4>
    <hr />
    @Html.ValidationSummary(true, "", new { @class = "text-danger" })
    <p class="text-info">
        You've successfully authenticated with
    <strong>@ViewBag.LoginProvider</strong>.
        Please enter a user name for this site below and click the Register
button to finish
        logging in.
    </p>
    <div class="form-group">
        @Html.LabelFor(m => m.Email, new { @class = "col-md-2 control-label" })
        <div class="col-md-10">
            @Html.TextBoxFor(m => m.Email, new { @class = "form-control" })
            @Html.ValidationMessageFor(m => m.Email, "", new { @class = "text-
danger" })
        </div>
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.HomeTown, new { @class = "col-md-2 control-label"
    })
        <div class="col-md-10">
            @Html.TextBoxFor(m => m.HomeTown, new { @class = "form-control" })
            @Html.ValidationMessageFor(m => m.HomeTown)
        </div>
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.BirthDate, new { @class = "col-md-2 control-
label" })
        <div class="col-md-10">
            @Html.TextBoxFor(m => m.BirthDate, new { @class = "form-control" })
            @Html.ValidationMessageFor(m => m.BirthDate)
        </div>
    </div>
    <div class="form-group">
        <div class="col-md-offset-2 col-md-10">
            <input type="submit" class="btn btn-default" value="Register" />
        </div>
    </div>
}

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}

```

Delete the membership database so you can again register your Facebook account with your application and verify you can add the new birth date and home town profile information.

From **Solution Explorer**, click the **Show All Files** icon, then right click *Add_Data\aspnet-MvcAuth-<dateStamp>.mdf* and click **Delete**.



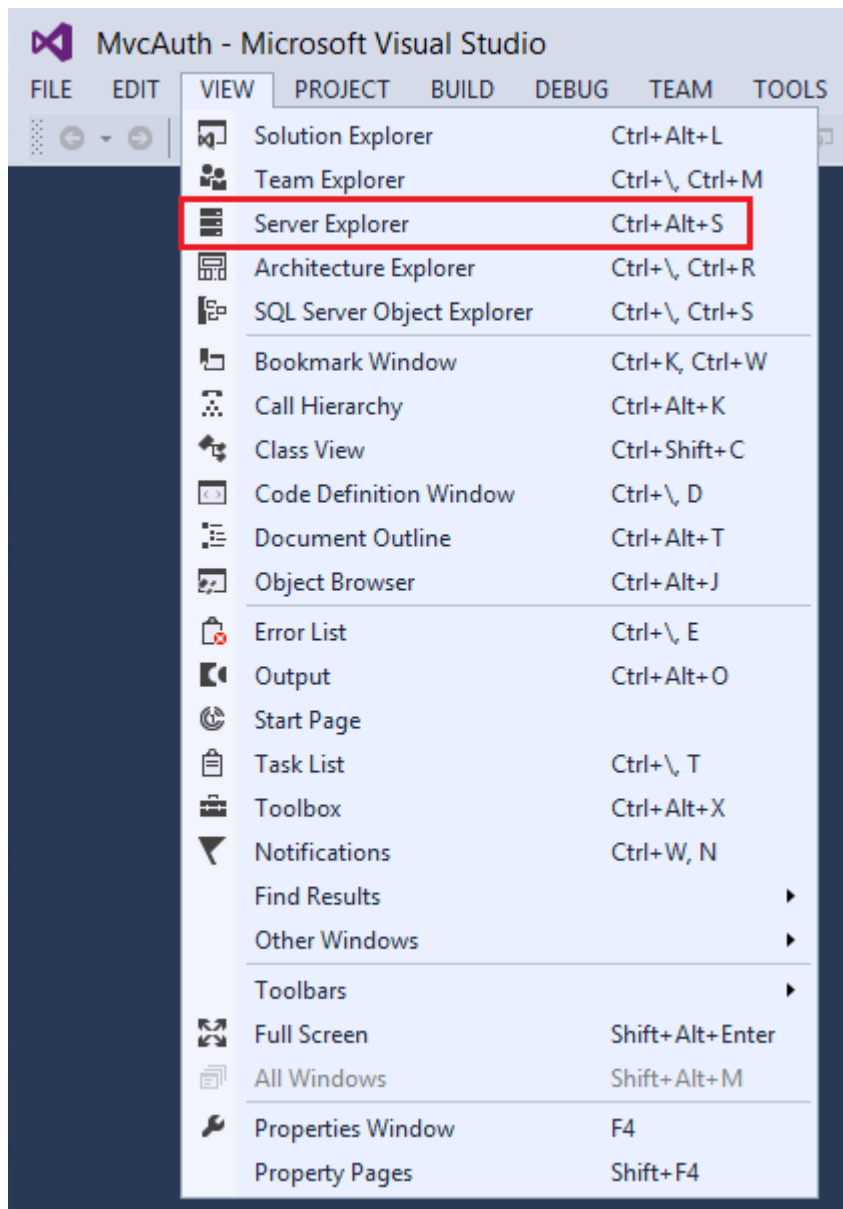
From the **Tools** menu, click **NuGet Package Manager**, then click **Package Manager Console** (PMC). Enter the following commands in the PMC.

1. Enable-Migrations
2. Add-Migration Init
3. Update-Database

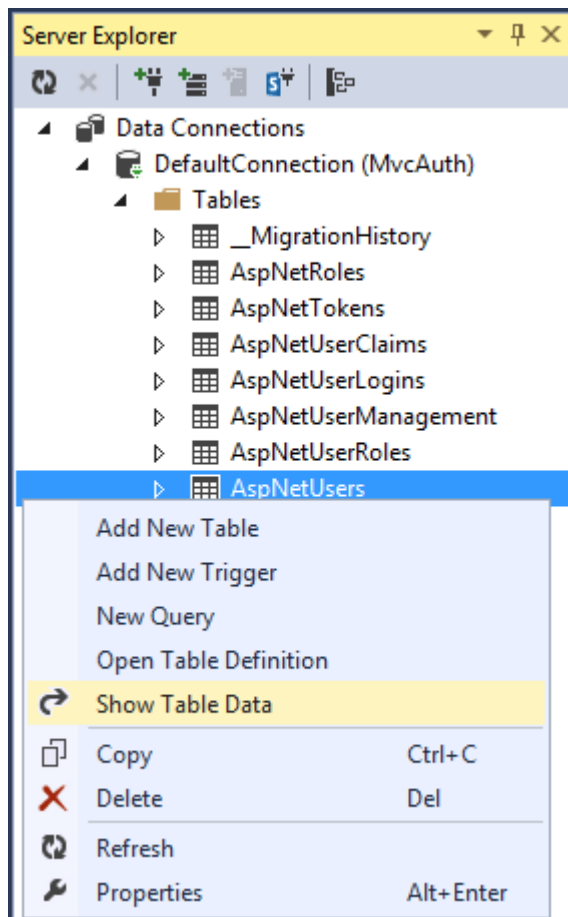
Run the application and use FaceBook and Google to log in and register some users.

Examine the Membership Data

In the **View** menu, click **Server Explorer**.



Right click **AspNetUsers** and click **Show Table Data**.



The `HomeTown` and `BirthDate` fields are shown below.

dbo.AspNetUsers [Data]					
Max Rows: 1000					
	Id	HomeTown	BirthDate	Email	EmailConfirmed
▶	eea1081f...	Captown	7/7/1977 12:00:...	Scott@contoso.com	False
	a64159...	Redmond	1/2/1903 12:00:...	Rick2@Contoso.com	False
	b2d94...	Great Falls	8/8/1988 12:00:...	RickAndMSFT@gm...	False
*	NULL	NULL	NULL	NULL	NULL

Logging off your App and Logging in With Another Account

If you log on to your app with Facebook,, and then log out and try to log in again with a different Facebook account (using the same browser), you will be immediately logged in to the previous Facebook account you used. In order to use another account, you need to navigate to Facebook and log out at Facebook. The same rule applies to any other 3rd

party authentication provider. Alternatively, you can log in with another account by using a different browser.

Next Steps

See [Introducing the Yahoo and LinkedIn OAuth security providers for OWIN](#) by Jerrie Pelsner for Yahoo and LinkedIn instructions. See Jerrie's Pretty social login buttons for ASP.NET MVC 5 to get enable social login buttons.

Follow my tutorial [Create an ASP.NET MVC app with auth and SQL DB and deploy to Azure App Service](#), which continues this tutorial and shows the following:

1. How to deploy your app to Azure.
2. How to secure you app with roles.
3. How to secure your app with the [RequireHttps](#) and [Authorize](#) filters.
4. How to use the membership API to add users and roles.

Please leave feedback on how you liked this tutorial and what we could improve. You can also request new topics at [Show Me How With Code](#). You can even ask for and vote on new features to be added to ASP.NET. For example, you can vote for a tool to [create and manage users and roles](#).

For an good explanation of how ASP.NET External Authentication Services work, see Robert McMurray's [External Authentication Services](#). Robert's article also goes into detail in enabling Microsoft and Twitter authentication. Tom Dykstra's excellent [EF/MVC tutorial](#) shows how to work with the Entity Framework.