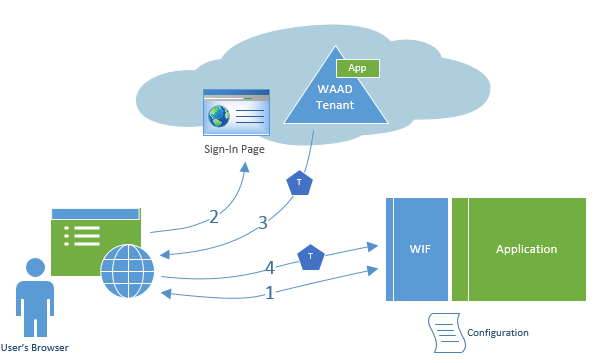
**Introduction to Azure Active Directory**

**Overview**

In this hands-on lab you will learn how to use **Azure Active Directory** to implement web single sign-on in an ASP.NET application. The instructions will focus on taking advantage of the directory tenant associated with your Microsoft Azure subscription, as that constitutes the obvious choice of identity providers for Line of Business (LoB) applications in your own organization. This lab will show you how to create an application with single sign-on enabled with Azure Active Directory using the new Organizational Accounts authentication template that comes with MVC. Then you will explore all the configurations set in the application and in Microsoft Azure that make single sign-on possible. At the end of the lab, you will use the Graph RESTful API to query Active Directory data and display it in the application.



**Objectives**

In this hands-on lab, you will learn how to:

* Create a new Azure Active Directory tenant.
* Provision an MVC application in the AD tenant.
* Explore the configuration of the application Authentication.
* Query Active Directory data using Graph AD API.

**Prerequisites**

The following is required to complete this hands-on lab:

* [Visual Studio Express 2013 for Web](http://www.microsoft.com/visualstudio/) or later
* A Microsoft Azure subscription
  + Sign up for a [Free Trial](http://aka.ms/watk-freetrial)
  + If you are a Visual Studio Professional, Test Professional, Premium or Ultimate with MSDN or MSDN Platforms subscriber, activate your [MSDN benefit](http://aka.ms/watk-msdn) now to start developing and testing on Microsoft Azure.
  + [BizSpark](http://aka.ms/watk-bizspark) members automatically receive the Microsoft Azure benefit through their Visual Studio Ultimate with MSDN subscriptions.
  + Members of the [Microsoft Partner Network](http://aka.ms/watk-mpn) Cloud Essentials program receive monthly Microsoft Azure credits at no charge.

**Setup**

In order to execute the exercises in this hands-on lab, you will need to set up your environment.

1. Open a Windows Explorer window and browse to the lab's **Source** folder.
2. Right-click the **Setup.cmd** file and click **Run as administrator**. This will launch the setup process that will install the Visual Studio code snippets for this lab.
3. If the User Account Control dialog box is shown, confirm the action to proceed.

**Note:** Make sure you have checked all the dependencies for this lab before running the setup.

**Using the Code Snippets**

Throughout the lab document, you will be instructed to insert code blocks. For your convenience, most of this code is provided as Visual Studio Code Snippets, which you can access from within Visual Studio 2013 to avoid having to add it manually.

**Exercises**

This hands-on lab includes the following exercises:

1. [Adding Sign-On to Your Web Application Using Azure Active Directory](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/HOL.md" \l "Exercise1)
2. [Using the Graph API to Query Azure Active Directory](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/HOL.md#Exercise2)

**Note:** Each exercise is accompanied by a starting solution. These solutions are missing some code sections that are completed throughout each exercise and will not necessarily work if you run them directly. Inside each exercise you will also find an end folder with the solution you should obtain after completing the exercises. You can use this solution as a guide if you need additional help working through the exercises.

Estimated time to complete this lab: **45** minutes

**Note:** When you first start Visual Studio, you must select one of the predefined settings collections. Every predefined collection is designed to match a particular development style and determines window layouts, editor behavior, IntelliSense code snippets, and dialog box options. The procedures in this lab describe the actions necessary to accomplish a given task in Visual Studio when using the **General Development Settings** collection. If you choose a different settings collection for your development environment, there may be differences in these procedures that you need to take into account.

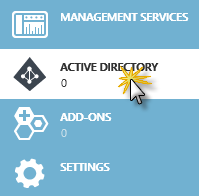
**Exercise 1: Adding Sign-On to Your Web Application Using Azure Active Directory**

In the first exercise you will learn how to provision a new **Azure AD** tenant within your Microsoft Azure subscription and how to operate the **Azure AD Management Portal** features to register an application.

**Task 1 - Creating a New Directory Tenant**

In this task, you will provision a new Azure Active Directory Tenant from Microsoft Azure Management Portal.

1. Navigate to [Microsoft Azure Management Portal](https://manage.windowsazure.com/) using a web browser and sign in using the Microsoft Account associated with your Microsoft Azure account.
2. Select **Active Directory** from the left pane.

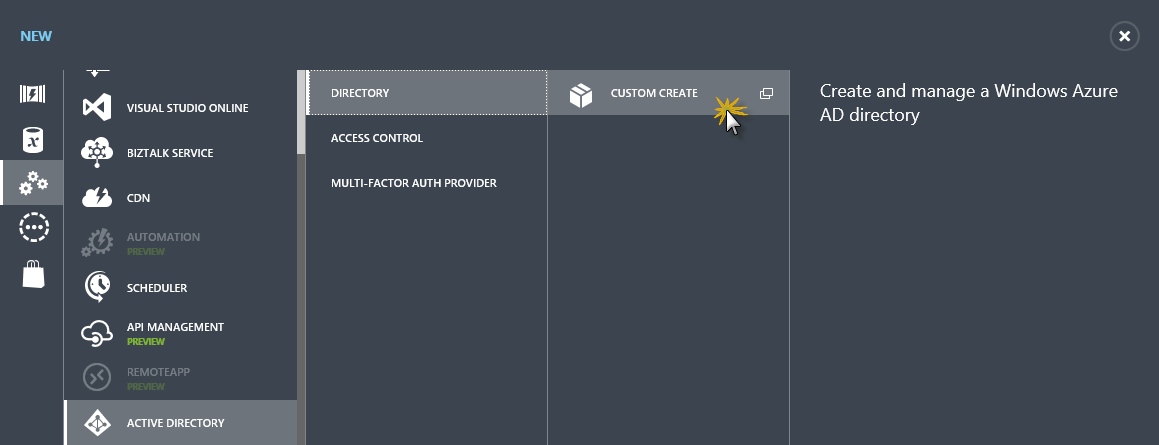
[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/active-directory-menu-panel.png)

*Accessing Azure Active Directory*

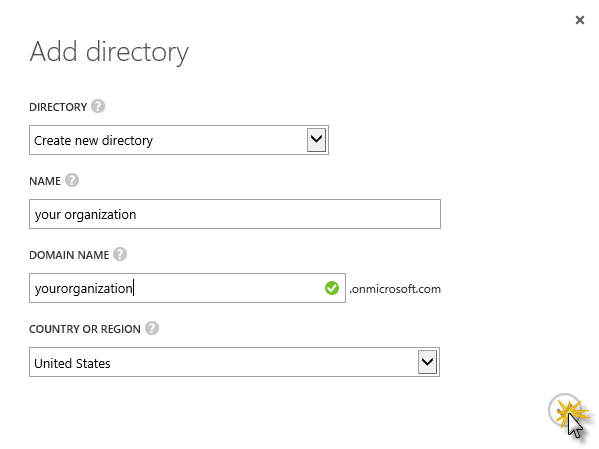
1. Click the **New** button in the bottom toolbar, and select **App Services | Active Directory | Directory | Custom Create**.

[Adding a new Active Directory Tenant](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/add-ad-menu.png?raw=true)

*Adding a new Active Directory Tenant*

[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/adding-a-new-active-directory-tenant.png?raw=true)

1. In the **Add directory** dialog box, make sure that **Create new directory** is selected under **Directory**. Enter a **Name**, type a unique **Domain Name** and select a **Country or Region**. Click the Check button to continue.

[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/create-a-new-directory.png?raw=true)

*Creating a New Directory*

**Note:** This dialog box gathers essential information needed to create a directory tenant for you.

* + **Name**: This is a required field, and its value will be used as a moniker whenever the company name needs to be displayed.
  + **Domain Name**: This field represents a critical piece of information: it is the part of the directory tenant domain name that is specific to your tenant - what distinguishes it from every other directory tenant.

At the time of creation, every directory tenant is identified by a domain of the form .onmicrosoft.com. That domain is used in the UPN of all directory users and in general wherever it is necessary to identify your directory tenant. After creation, it is possible to register additional domains. For more information, see domain management.

The domain name must be unique. The UI validation logic will help you pick a unique value. It is recommended that you choose a handle which refers to your company, as that will help users and partners identify you as they interact with the directory tenant.

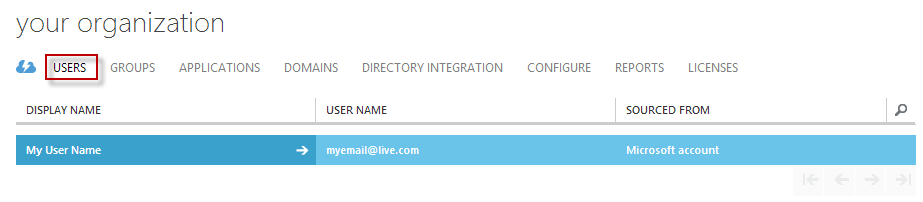
* + **Country or Region**: The value selected in this drop-down menu will determine where your tenant will be created. Given that the directory will store sensitive information, please take into account the normative about privacy of the country in which your company operates.

1. Wait until the Active Directory is created (its status should display **Active**).

*Active Directory Tenant Creation Completed*

**Note:** When a directory tenant is created, it is configured to store users and credentials in the cloud. If you want to integrate your directory tenant with your on-premises deployment of Windows Server Active Directory, you can find detailed instructions in [Directory integration](http://technet.microsoft.com/library/jj151781.aspx).

1. Click on the newly created directory entry and then click the **Users** tab to display the user management UI. The directory tenant is initially empty, except for the Microsoft Account administering the Microsoft Azure subscription in which the new tenant was created.

[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/active-directory-user-list.png?raw=true)

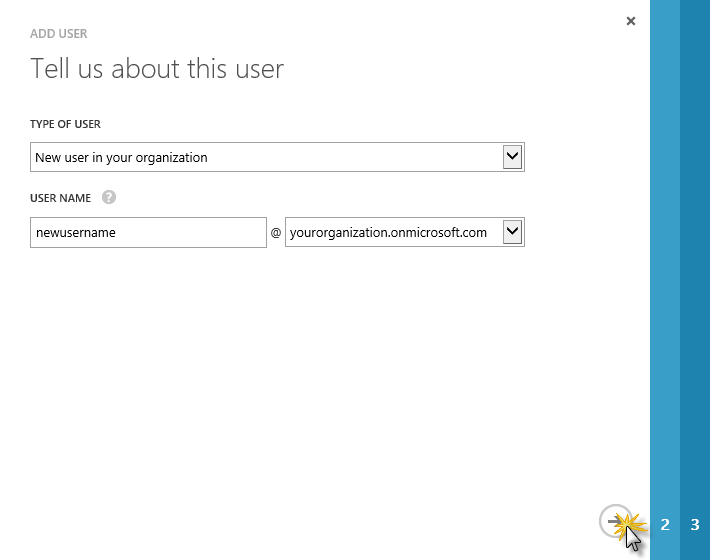
*Active Directory User List*

1. You will now add a new user to the directory. Click the **Add User** button in the bottom bar.

[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/add-new-user-to-active-directory.png?raw=true)

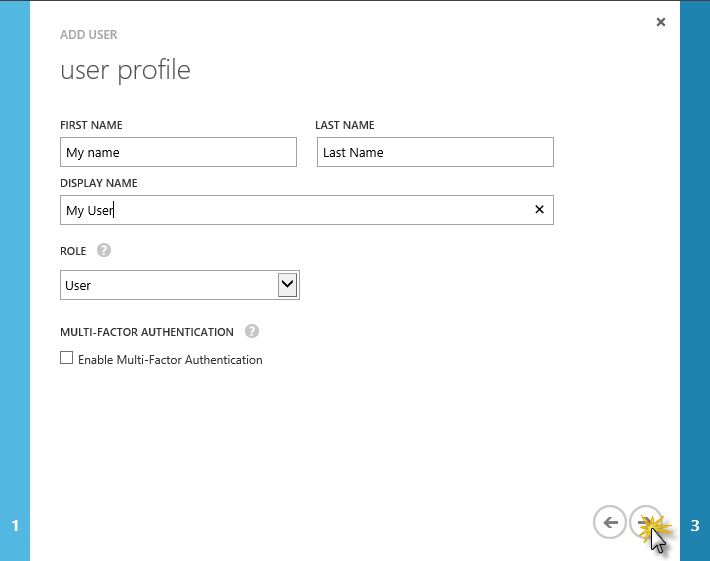
*Adding a new user to Active Directory*

1. In the dialog box, keep the default option of **New user in your organization** selected and type a username (e.g.:*newusername*). Click **Next** to continue.

[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/filling-new-user-name-details.png?raw=true)

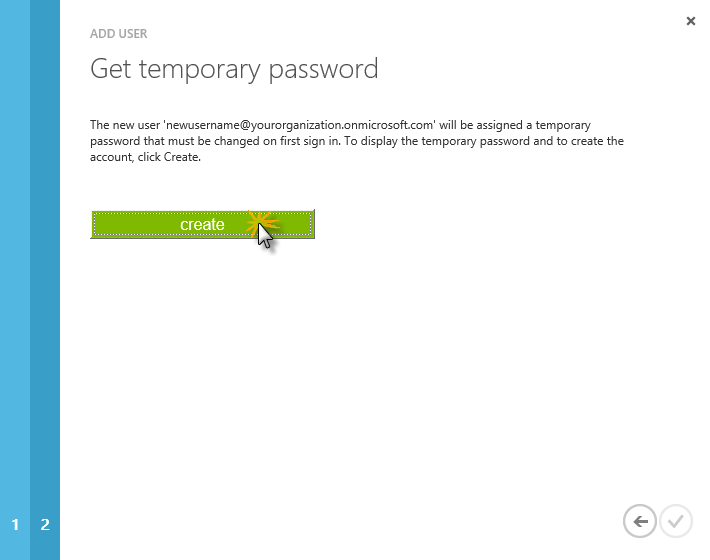
*Entering new user details*

1. Enter the user profile data. Keep the **Role** option of **User** selected. Click **Next** to continue.

[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/filling-user-profile-information.png?raw=true)

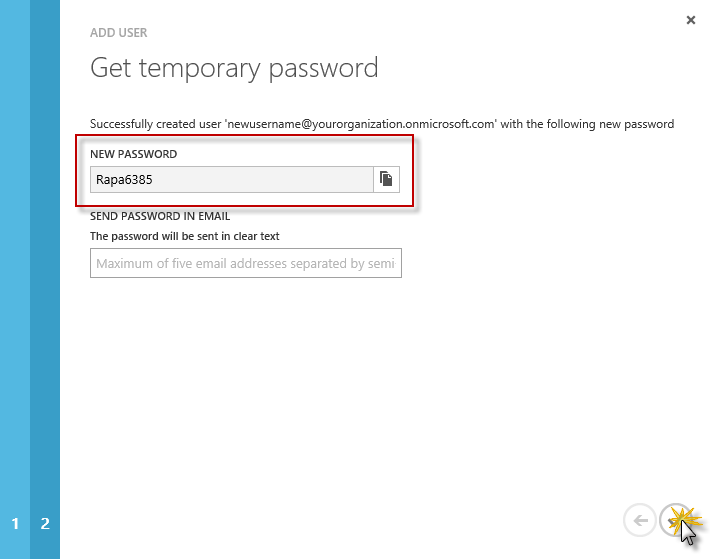
*Entering user profile information*

1. The Microsoft Azure Management Portal generates a temporary password to be used when first logging in. At that time, the user will be forced to change the password. Click the **create** button.

[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/creating-a-temporary-password.png?raw=true)

*Creating a temporary password*

1. Take note of the temporary password, as you will need it in the following tasks. Click the **Check** button to create the user.

[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/creating-the-new-user.png?raw=true)

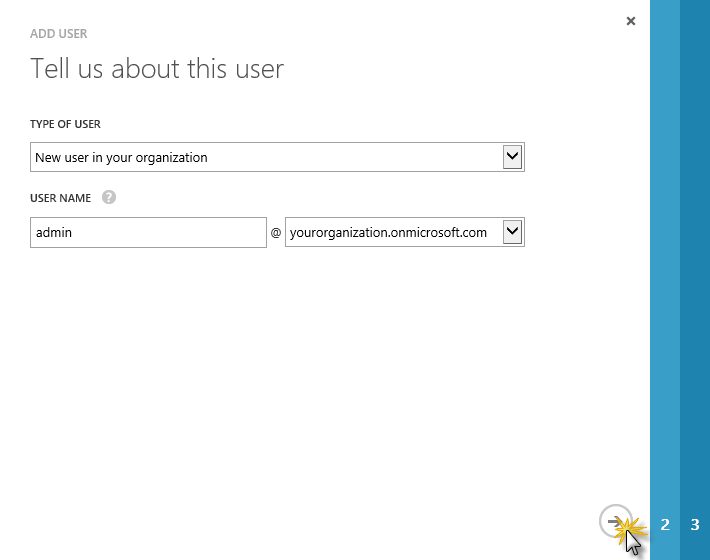
*Creating the new user*

1. You will now repeat the steps to add a new *admin* user to the directory. Click the **Add User** button in the bottom bar.

[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/add-new-user-to-active-directory.png?raw=true)

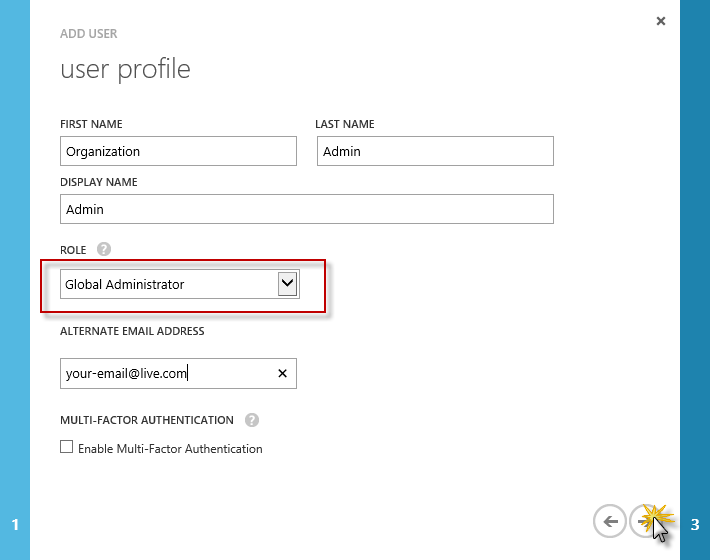
*Adding a new user to Active Directory*

1. In the dialog box, leave the default option of **New user in your organization** selected and type a username (e.g.:*admin*). Click **Next** to continue.

[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/filling-new-admin-user-name-details.png?raw=true)

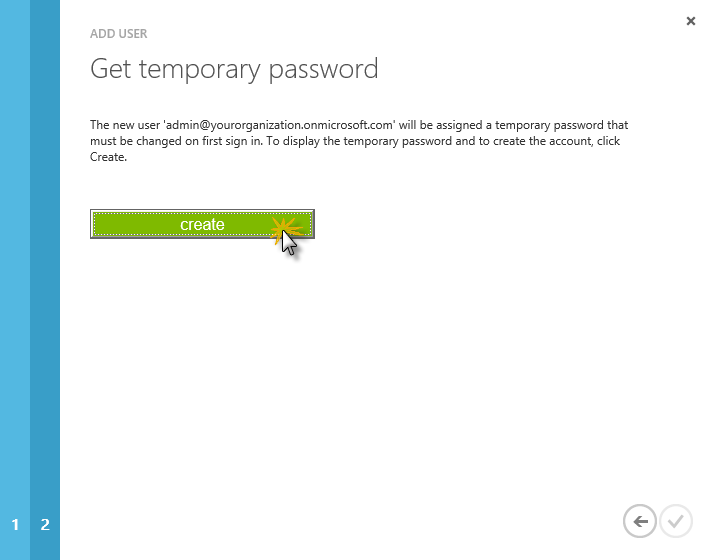
*Entering new admin user name details*

1. Enter the user profile data. This time, change the **Role** option to **Global Administrator**. You will need to provide an alternate email address. Click **Next** to continue.

[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/filling-admin-user-profile.png?raw=true)

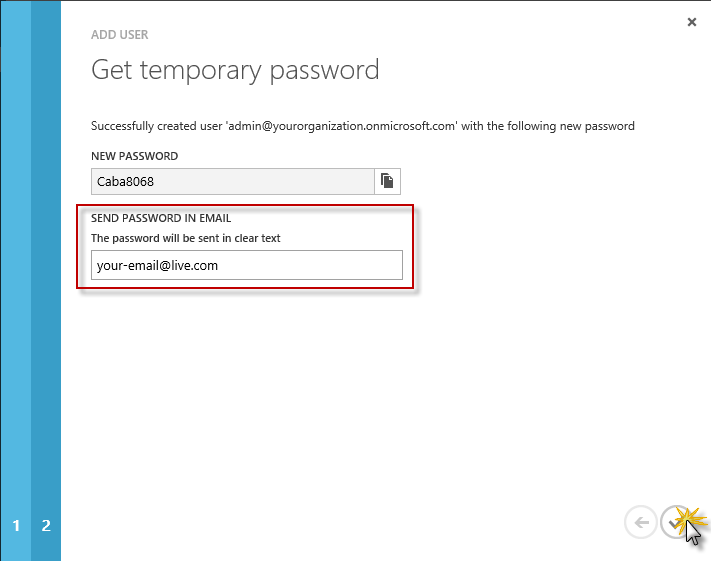
*Completing admin user profile*

1. In the **Get temporary password** step, click the **create** button.

[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/creating-a-temporary-password-for-the-admin.png?raw=true)

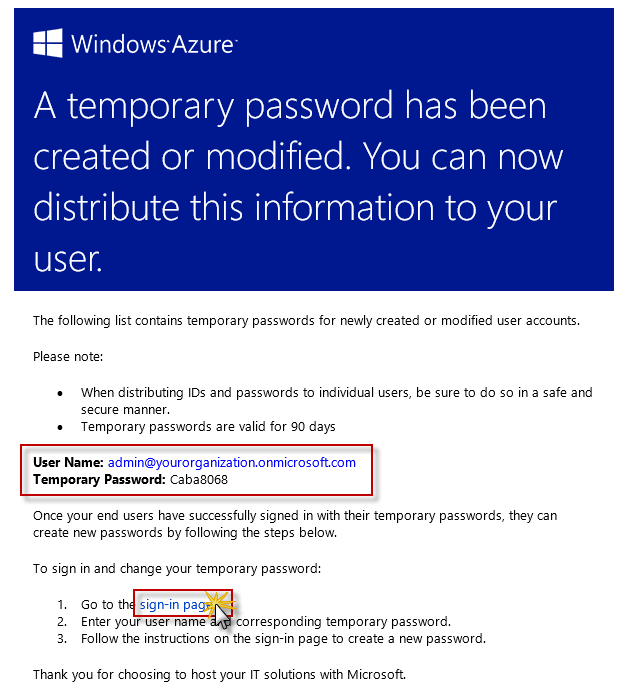
*Creating a temporary password for the admin*

1. You will need to send the password in an email in order to be able to modify it. To do this, type your email in the **SEND PASSWORD IN EMAIL** text box. Click the Check button to create the user.

[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/sending-the-admin-password-by-email.png?raw=true)

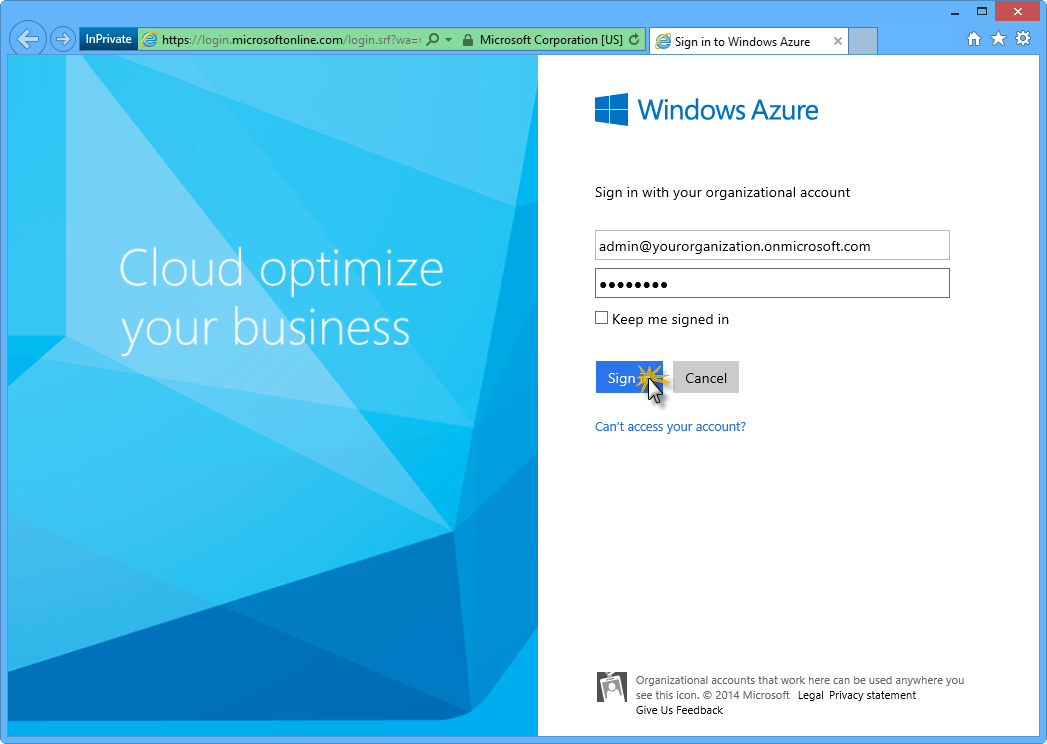
*Sending the admin password by email*

1. Once you get the email, take note of the temporary password and click the sign-in page link provided in the instructions.

[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/temporary-password-email.png?raw=true)

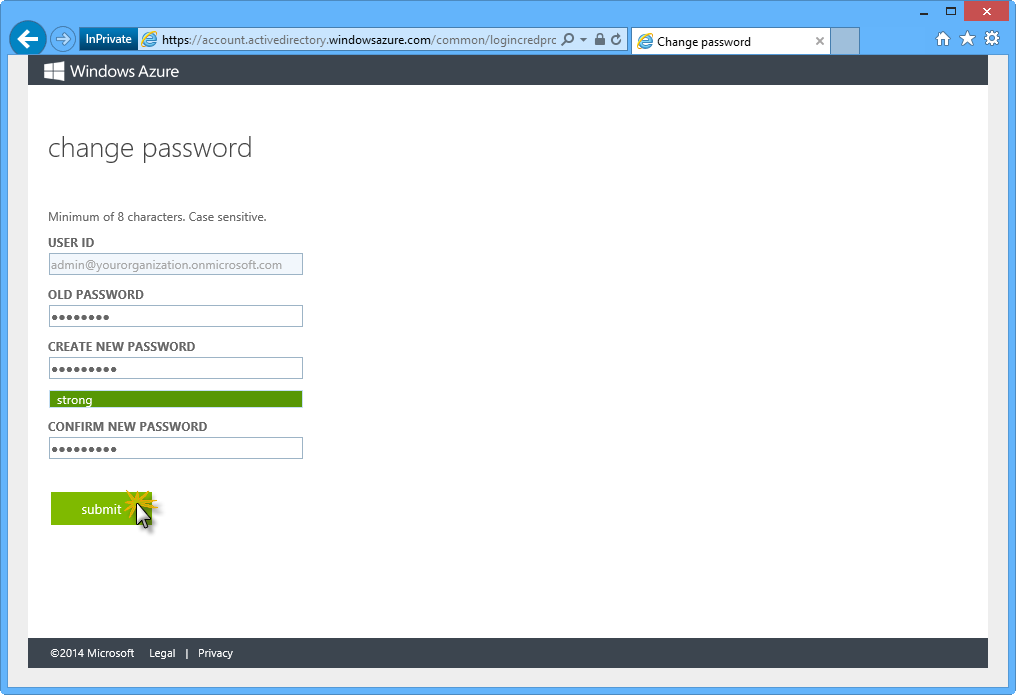
*Temporary password email*

1. Sign in with your temporary credentials.

[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/singing-in-with-your-temporal-credentials.png?raw=true)

*Signing in with your temporary credentials*

1. Finally, change your temporary password and click **submit**.

[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/changing-your-temporary-password.png?raw=true)

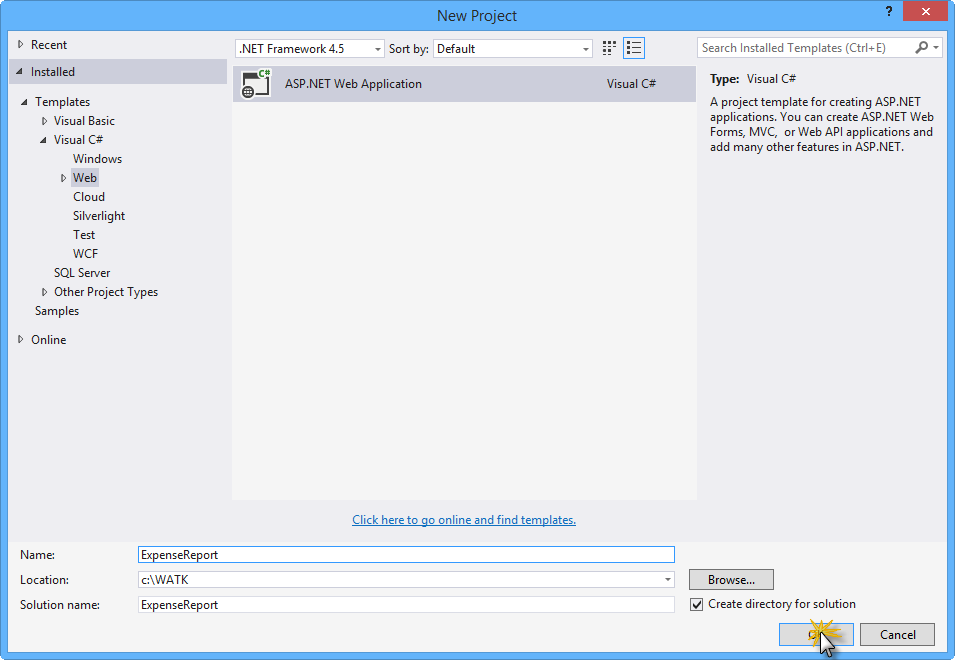
*Changing your temporary password*

At this point you have everything you need to provide authentication authority in your web SSO scenario: a directory tenant, a valid user, and a valid admin.

**Task 2 - Creating and Configuring an MVC App with Organizational Accounts Authentication**

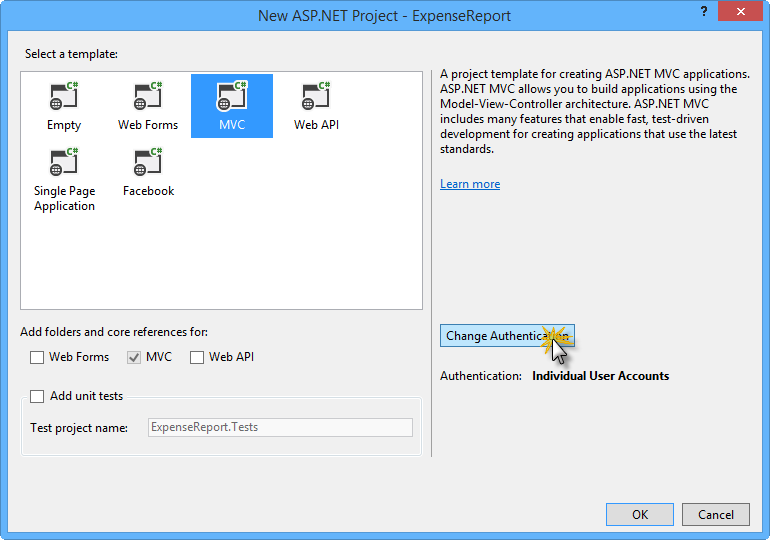
In this task, you will create a new MVC Application using **Visual Studio Express 2013 for Web** and will configure it to use**Organizational Accounts** as an Authentication method using the Active Directory tenant you created in the previous task.

1. Open **Visual Studio Express 2013 for Web**.
2. From the **FILE** menu, choose **New Project...**.
3. In the **New Project** dialog box, expand **Visual C#** in the **Installed** list and select **Web**. Choose the **ASP.NET Web Application** template, set the **Name** of the project to *ExpenseReport* and set a location for the solution. Click **OK** to create the project.



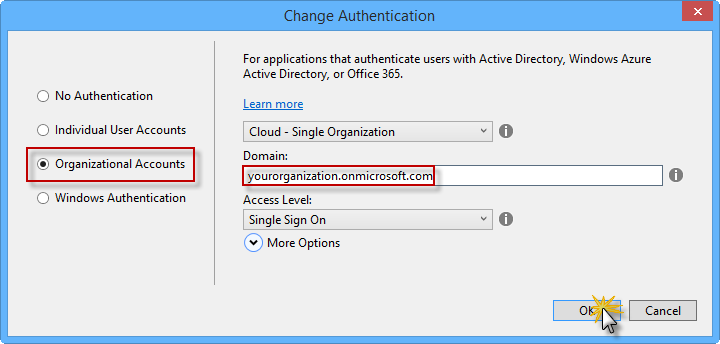
*Creating a new MVC 5 Application*

1. In the **New ASP.NET Project - ExpenseReport** window, select the **MVC** template and then click **Change Authentication**.



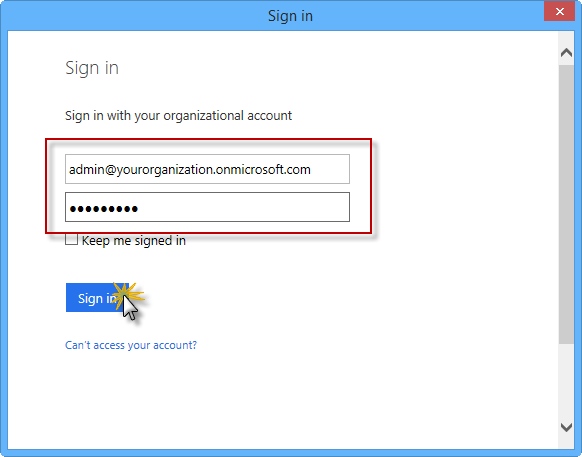
*Selecting the MVC template and changing the authentication method*

1. In the **Change Authentication** dialog box, select **Organizational Accounts**, type the **Domain** created in the previous tasks (e. g. *yourorganization.onmicrosoft.com*) and click **OK**.



*Selecting Organizational Accounts as the Authentication method*

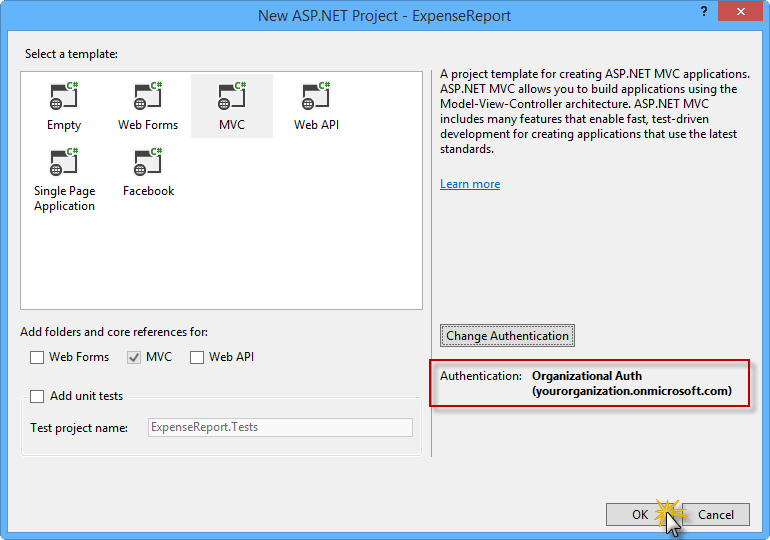
1. In the **Sign in** dialog box, use the credentials for the domain admin you created in the previous tasks (e. g.*[admin@yourorganization.onmicrosoft.com](mailto:admin@yourorganization.onmicrosoft.com)*) and then click **Sign in**.

[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/signing-in-with-the-domain-admininstrator-cre.png?raw=true)

*Signing in with Domain Administrator credentials*

**Note:** No application can take advantage of Azure AD without being registered. This is both for security reasons (only apps that are approved by the administrator should be allowed) and practical considerations (interaction with Azure AD entails the use of specific open protocols, requiring knowledge of key parameters describing the app).

1. In the **New ASP.NET Project - ExpenseReport** window, note that the Authentication is now set to **Organizational Auth**. Click **OK** in order to create the project.

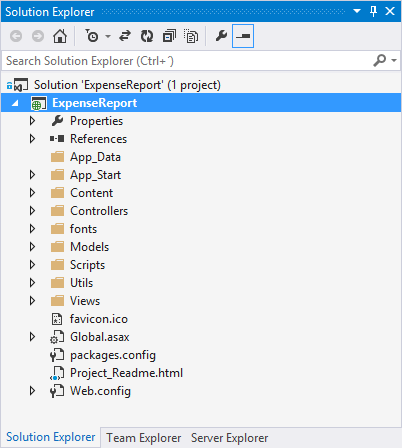
[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/creating-the-mvc-project.png?raw=true)

*Creating the MVC project*

**Task 3 - Exploring the generated MVC project**

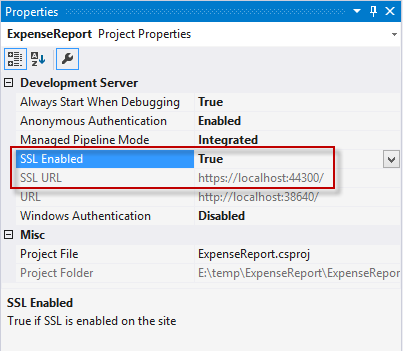
In this task you will explore the code generated by Visual Studio to enable Single Sign-on with Azure Active Directory. You will start by running the solution as is and accessing the application with the account you created in **Task 1**. Then you will go to the Microsoft Azure Management Portal to check that the application was created and go through the most relevant configuration settings in the application. Finally, you will go through all the code and configuration involved in enabling Single Sign-On.

1. Go to **Solution Explorer** and explore the generated project. Notice that the solution has the common MVC structure with some configurations already in place.



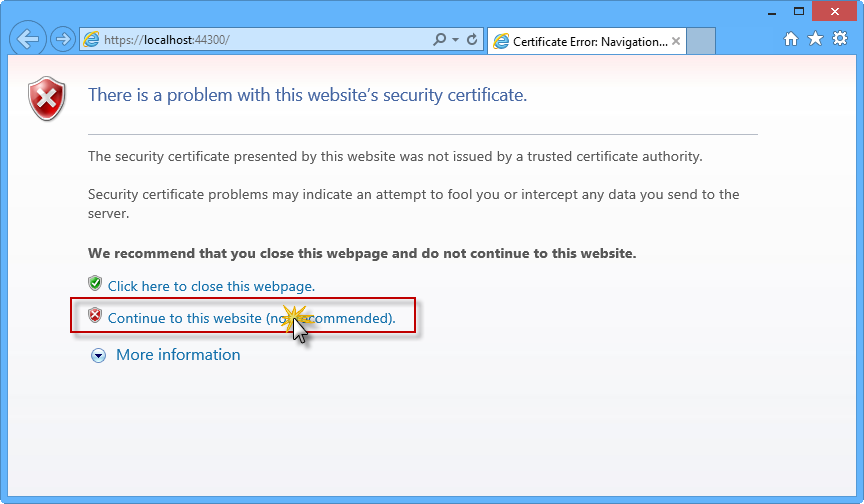
*Exploring the ExpenseReport project in Solution Explorer*

1. Select the **ExpenseReport** project in the **Solution Explorer**. In the **Properties** pane, note that **SSL Enabled** is set to*True*.



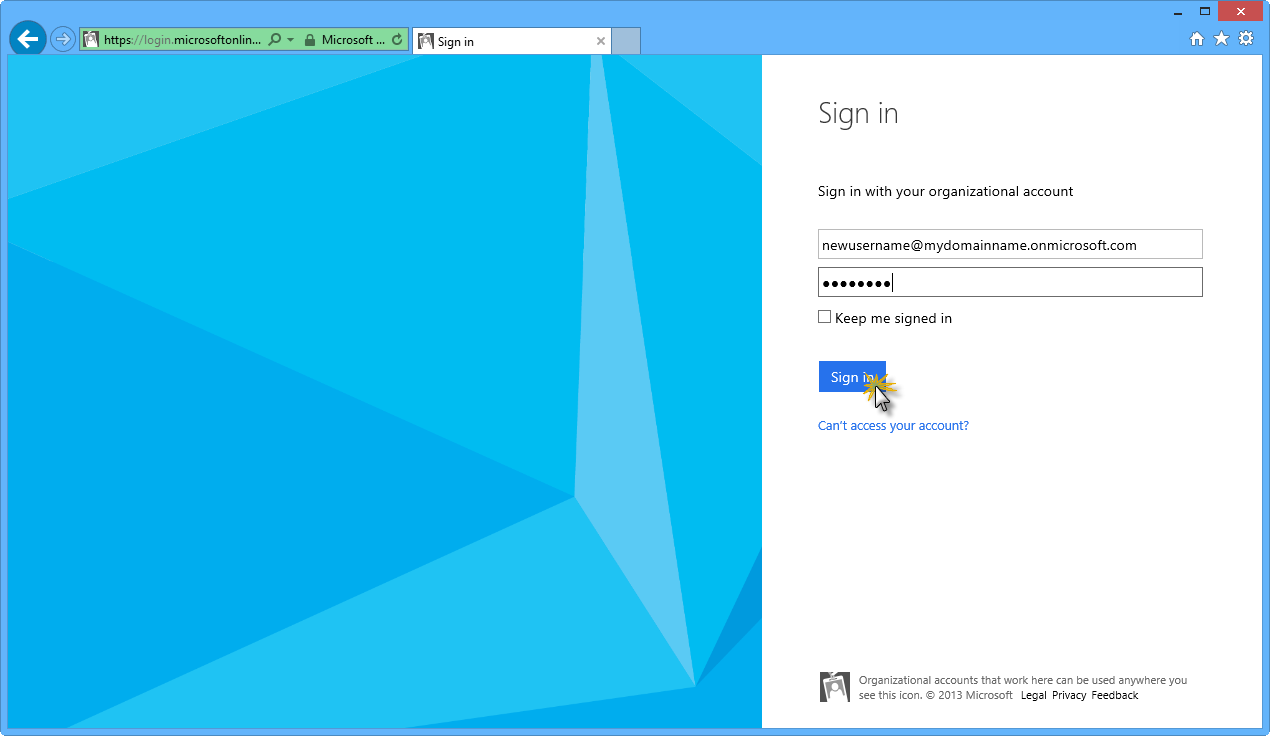
*ExpenseReport project properties*

1. Press **F5** to run the application.
2. A security certificate warning will appear in your browser. This is an expected behavior. Click **Continue to this website (not recommended)**.



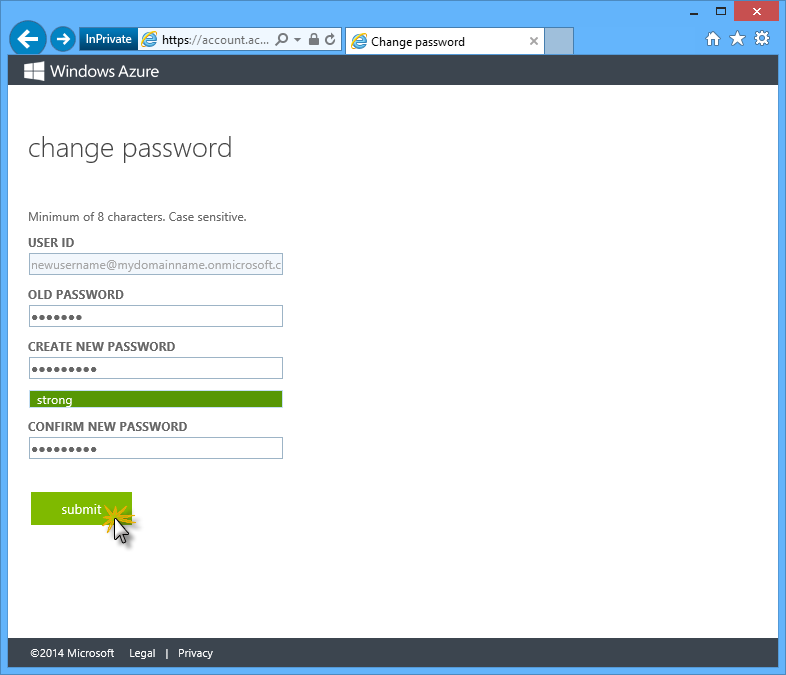
*Browser displaying Security Certificate Warning*

1. The URL address bar is replaced by that of the authority, and the user is prompted to authenticate with the Azure AD UI. Type the user credentials you created previously.



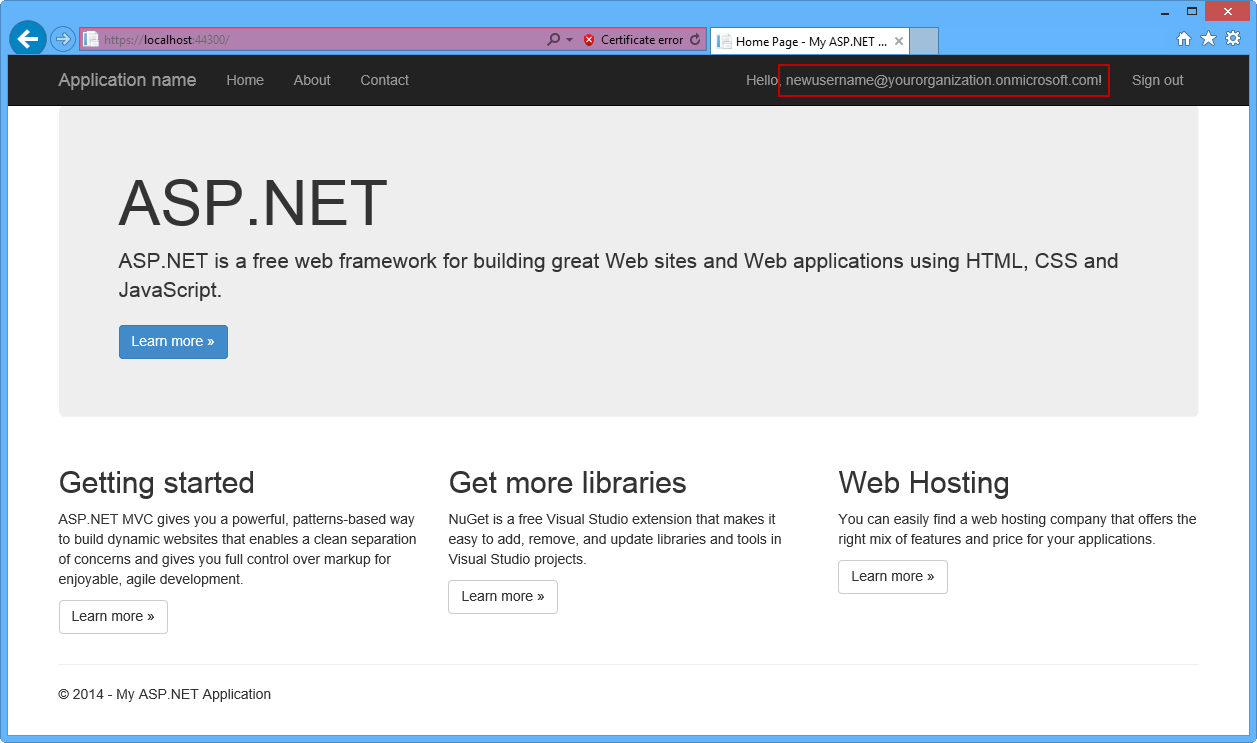
*Logging in to the Application*

1. You may recall that when you created the user in your Azure AD tenant, the Microsoft Azure Management Portal assigned a temporary password for authentication. However, given that this password was meant to be temporary, you will be asked to choose a new user password at this first sign-in attempt before moving forward with the authentication flow.



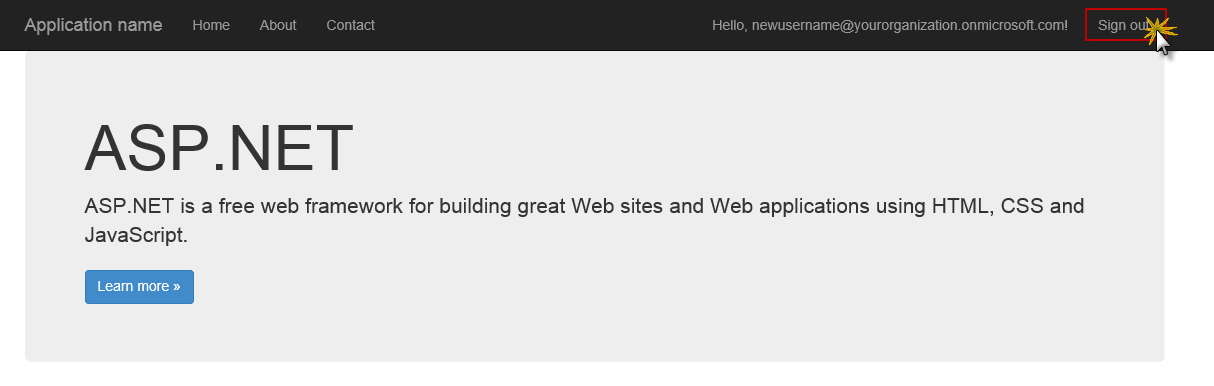
*Typing a New User Password*

1. On the Home page, notice the username displayed in the upper-right corner.



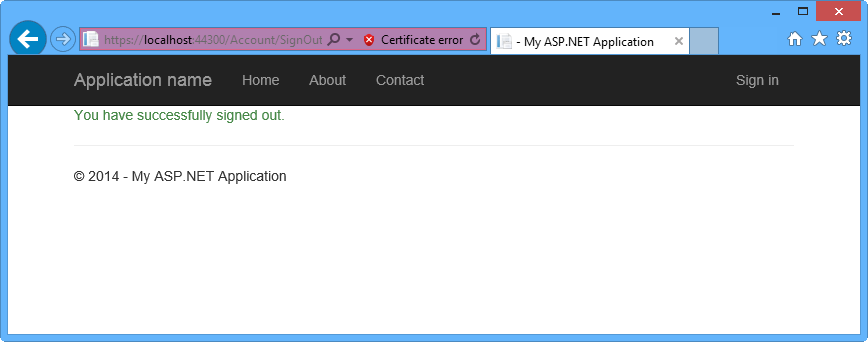
*ExpenseReport Home page*

1. Click the **Sign out** link located in the upper-right corner.



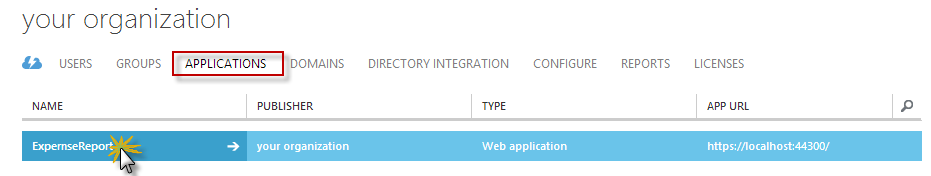
*Signing out*

1. You will see the Sign Out view.



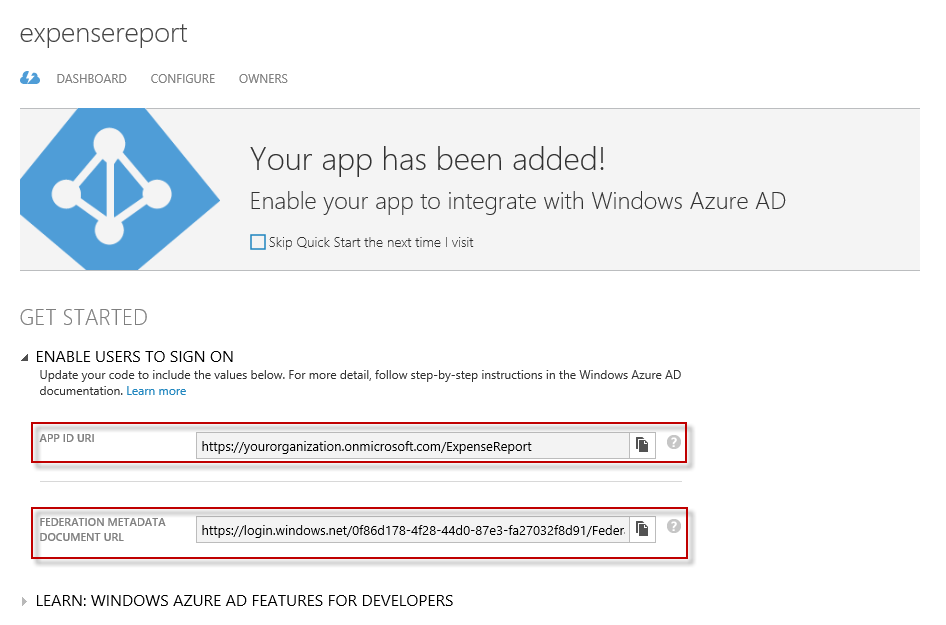
*Sign Out View*

1. Switch back to Visual Studio and press **Shift** + **F5** to stop the solution.
2. Minimize Visual Studio and go back to the **Microsoft Azure Management Portal**. Go to your Active Directory tenant and click the **Applications** tab.
3. Click the arrow next to the **ExpenseReport** application in the **Applications** list to go to the dashboard.



*Selecting the ExpenseReport application*

1. In the application dashboard, click the **Enable Users To Sign On** link to display information to enable single sign-on with Azure AD. You will then see where the **Federation Metadata Document URL** and **App ID URI** were configured by the MVC new project assistant.



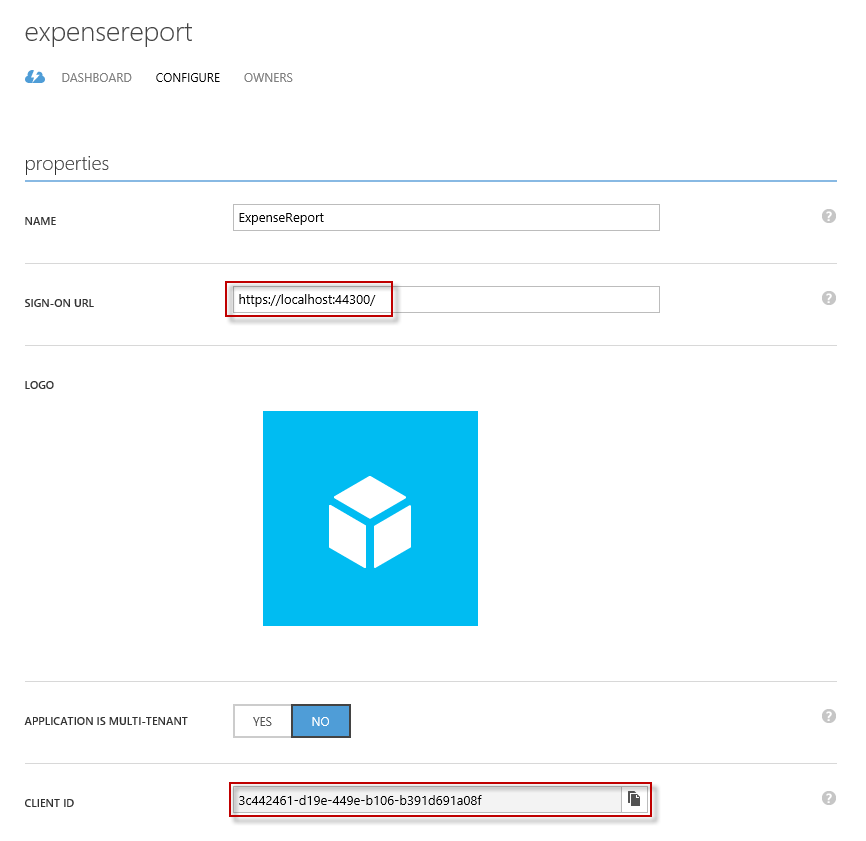
*Application dashboard*

1. Switch to the **Configure** tab by clicking the **Configure** button at the top of the page.



*Configure button*

1. In the **Properties** section, you can verify the **SIGN-ON URL** which is the URL where users can sign in and use your app. You can also see the **CLIENT ID** which is the unique identifier for your app. You will use the **CLIENT ID** property in the next exercise.



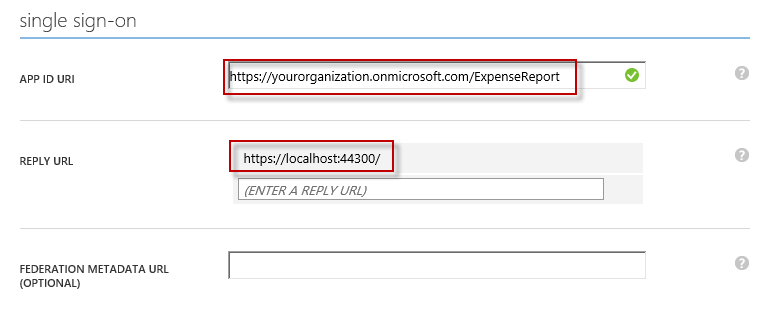
*Properties section in Configure tab*

**Note:** You need to use the **CLIENT ID** if your app calls another service, such as the Azure AD Graph API, to read or write data.

1. The **single sign-on** section shows the required **APP ID URI** and **REPLY URL** properties which the service needs to drive the sign-in protocol flow.

The **APP ID URI** represents the identifier of your web application. Azure AD uses this value at sign-on time to determine that the authentication request is meant to enable a user to access this particular application - among all those registered - so that the correct settings can be applied. The APP ID URI must be unique within the directory tenant. A good default value is the APP URL value itself, although the uniqueness constraint is not always easy to respect. Developing the app on local hosting environments such as IIS Express and the Microsoft Azure Fabric Emulator tends to produce a restricted range of addresses that may be reused by multiple developers or even multiple projects from the same developer.

The **REPLY URL** represents the address of your web application. Azure AD needs to know your application's address so that, after a user is successfully authenticated on Azure AD's pages, it can direct the flow back to your application.



*Single sign-on section in Configure tab*

1. Switch back to Visual Studio.
2. Open the **Web.config** file to check the authentication configuration of your application.
3. In the **appSettings** section there are three new keys: **ida:FederationMetadataLocation**, **ida:Realm** and**ida:AudienceUri**. Those values were configured with the Federation Metadata Document and the App ID Uri of your application. The **IdentityConfig** class reads these values to configure the identity when the application starts.

<appSettings>

<add key="webpages:Version" value="3.0.0.0" />

<add key="webpages:Enabled" value="false" />

<add key="ClientValidationEnabled" value="true" />

<add key="UnobtrusiveJavaScriptEnabled" value="true" />

<add key="ida:FederationMetadataLocation" value="https://login.windows.net/yourorganization.onmicrosoft.com/FederationMetadata/2007-06/FederationMetadata.xml" />

<add key="ida:Realm" value="https://yourorganization.onmicrosoft.com/ExpenseReport" />

<add key="ida:AudienceUri" value="https://yourorganization.onmicrosoft.com/ExpenseReport" />

</appSettings>

1. The **identityConfiguration** element in the **system.identityModel** section of the **Web.config** file determines the behavior of the application during the authentication phase.

<system.identityModel>

<identityConfiguration>

<issuerNameRegistry type="ExpenseReport.Utils.DatabaseIssuerNameRegistry, ExpenseReport" />

<audienceUris>

<add value="https://yourorganization.onmicrosoft.com/ExpenseReport" />

</audienceUris>

<securityTokenHandlers>

<add type="System.IdentityModel.Services.Tokens.MachineKeySessionSecurityTokenHandler, System.IdentityModel.Services, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />

<remove type="System.IdentityModel.Tokens.SessionSecurityTokenHandler, System.IdentityModel, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />

</securityTokenHandlers>

<certificateValidation certificateValidationMode="None" />

</identityConfiguration>

</system.identityModel>

1. The **federationConfiguration** element in the **system.identitymodel.services** section provides the coordinates that are necessary to drive WS-Federation flows: the address of the authority to be used for sign-on requests, the identifier of the app itself to be included in requests, and so on.

<system.identityModel.services>

<federationConfiguration>

<cookieHandler requireSsl="true" />

<wsFederation passiveRedirectEnabled="true" issuer="https://login.windows.net/yourorganization.onmicrosoft.com/wsfed" realm="https://yourorganization.onmicrosoft.com/ExpenseReport" requireHttps="true" />

</federationConfiguration>

</system.identityModel.services>

1. The application is configured to handle authentication via blanket redirects. This means that, if you try to access this View after a successful sign out you will be immediately redirected to Azure AD to sign in again. To avoid this behavior, the **<location>** element in the **Web.config** file is used to create an exception to the authentication policy.

<configuration>

...

</appSettings>

<location path="Account">

<system.web>

<authorization>

<allow users="\*" />

</authorization>

</system.web>

</location>

<system.web>

...

</configuration>

1. Now open the **Global.axax.cs** file. There is a new method called**WSFederationAuthenticationModule\_RedirectingToIdentityProvider**. It is invoked when the module redirects the user to the identity provider. The method updates the **Realm** property of the **SignInRequestMessage** object with that of the **IdentityConfig** class. The **IdentityConfig** class was configured previously in the **Application\_Start** event.

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

IdentityConfig.ConfigureIdentity();

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

RouteConfig.RegisterRoutes(RouteTable.Routes);

BundleConfig.RegisterBundles(BundleTable.Bundles);

}

void WSFederationAuthenticationModule\_RedirectingToIdentityProvider(object sender, RedirectingToIdentityProviderEventArgs e)

{

if (!String.IsNullOrEmpty(IdentityConfig.Realm))

{

e.SignInRequestMessage.Realm = IdentityConfig.Realm;

}

}

1. Open the **IdentityConfig.cs** file located in the **App\_Start** folder.
2. Your application accepts tokens coming from your Azure AD tenant of choice. It is a common security practice to regularly renew cryptographic keys, and Azure AD signing keys are no exception: at fixed time intervals the old keys will be retired, and new ones will take their place in the issuer's signing logic and in your tenant's metadata document. The**RefreshValidationSettings** method called in the **ConfigureIdentity** method saves the validation keys in a database by calling the **RefreshKeys** method of the **DatabaseIssuerNameRegistry** class.

public static void ConfigureIdentity()

{

RefreshValidationSettings();

// Set the realm for the application

Realm = ConfigurationManager.AppSettings["ida:realm"];

// Set the audienceUri for the application

AudienceUri = ConfigurationManager.AppSettings["ida:AudienceUri"];

if (!String.IsNullOrEmpty(AudienceUri))

{

UpdateAudienceUri();

}

AntiForgeryConfig.UniqueClaimTypeIdentifier = ClaimTypes.Name;

}

public static void RefreshValidationSettings()

{

string metadataLocation = ConfigurationManager.AppSettings["ida:FederationMetadataLocation"];

DatabaseIssuerNameRegistry.RefreshKeys(metadataLocation);

}

1. Open the **AccountController.cs** file located in the **Controllers** folder and note the code of the **SignOut** method.

public class AccountController : Controller

{

public ActionResult SignOut()

{

WsFederationConfiguration config = FederatedAuthentication.FederationConfiguration.WsFederationConfiguration;

// Redirect to SignOutCallback after signing out.

string callbackUrl = Url.Action("SignOutCallback", "Account", routeValues: null, protocol: Request.Url.Scheme);

SignOutRequestMessage signoutMessage = new SignOutRequestMessage(new Uri(config.Issuer), callbackUrl);

signoutMessage.SetParameter("wtrealm", IdentityConfig.Realm ?? config.Realm);

FederatedAuthentication.SessionAuthenticationModule.SignOut();

return new RedirectResult(signoutMessage.WriteQueryString());

}

public ActionResult SignOutCallback()

{

if (Request.IsAuthenticated)

{

// Redirect to home page if the user is authenticated.

return RedirectToAction("Index", "Home");

}

return View();

}

}

1. Open the **\_LoginPartial.cshtml** file located in **Views | Shared**. This view is rendered in the page navbar and shows the User's name with the **Sign out** link when authenticated or a link to the **Sign in** page.

@if (Request.IsAuthenticated)

{

<text>

<ul class="nav navbar-nav navbar-right">

<li class="navbar-text">

Hello, @User.Identity.Name!

</li>

<li>

@Html.ActionLink("Sign out", "SignOut", "Account")

</li>

</ul>

</text>

}

else

{

<ul class="nav navbar-nav navbar-right">

<li>@Html.ActionLink("Sign in", "Index", "Home", routeValues: null, htmlAttributes: new { id = "loginLink" })</li>

</ul>

}

**Task 4 - Displaying information of the authenticated user**

In this task you will display the authenticated user information in the Home page of the application. You will use the**ClaimsPrincipal** class to get the **Claims** of the current user and display them in the Home page.

1. Open the **HomeController.cs** file located in the **Controllers** folder.
2. Add the following directive at the top of the class.

using System.Security.Claims;

1. Replace the **Index** method content with the following code.

(Code Snippet - *IntroductionToAzureAD - Ex1 - QueryingClaimsPrincipal*)

public ActionResult Index()

{

ClaimsPrincipal cp = ClaimsPrincipal.Current;

ViewBag.Message = string.Format("Dear \"{0}, {1}\", welcome to the Expense Note App", cp.FindFirst(ClaimTypes.Surname).Value, cp.FindFirst(ClaimTypes.GivenName).Value);

return View();

}

**Note:** Starting from .NET 4.5, every identity in .NET is represented with a **ClaimsPrincipal**. In this case, the current**ClaimsPrincipal** has been constructed during the validation of an authentication token generated by Azure Active Directory and presented by the user at sign-on time.

1. Open the **Index.cshtml** file located in **Views** | **Shared** and replace the content with the following code to display the**Message** property from the **ViewBag**.

(Code Snippet - *IntroductionToAzureAD - Ex1 - DisplayMessageInIndexView*)

@{

ViewBag.Title = "Home Page";

}

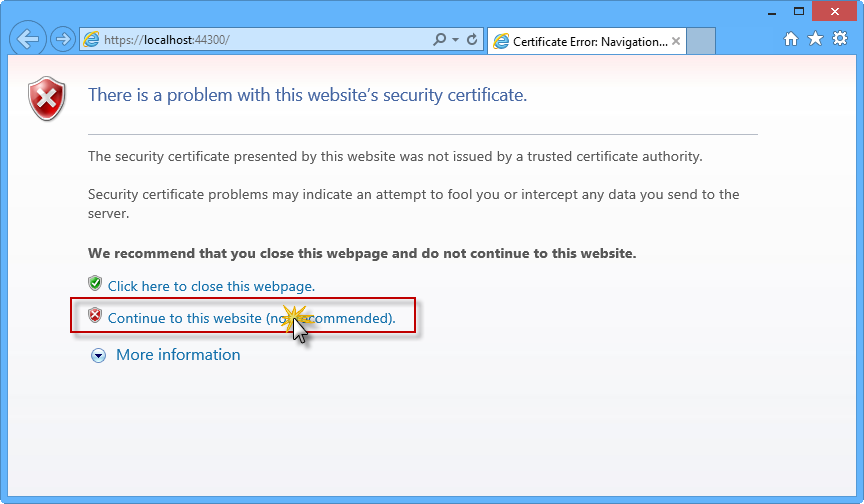
<div class="jumbotron">

<h1>ASP.NET</h1>

<p class="lead">@ViewBag.Message</p>

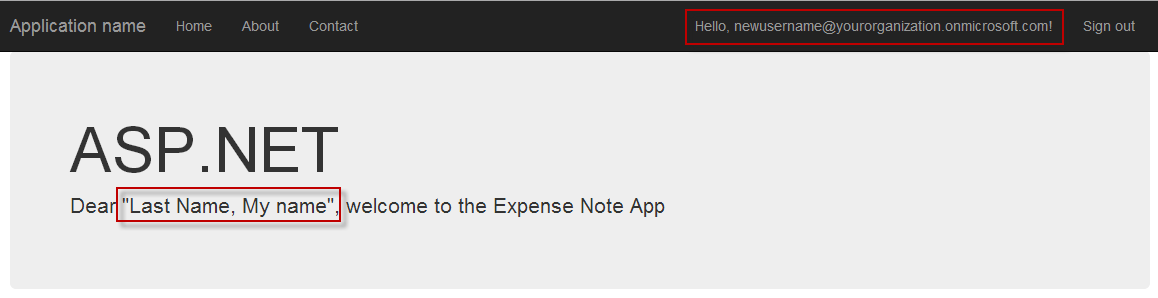
</div>

1. Press **F5** to run the application.
2. A security certificate warning will appear in your browser. This is an expected behavior. Click **Continue to this website (not recommended)**.



*Browser displaying Security Certificate Warning*

1. Log in to the application using the Active Directory user credentials.
2. On the Home page, notice the username displayed in the upper-right corner and the user's first and last name displayed in the center of the page.



*Displaying User Name in Home Page*

1. Switch back to Visual Studio and press **Shift + F5** to stop the solution.

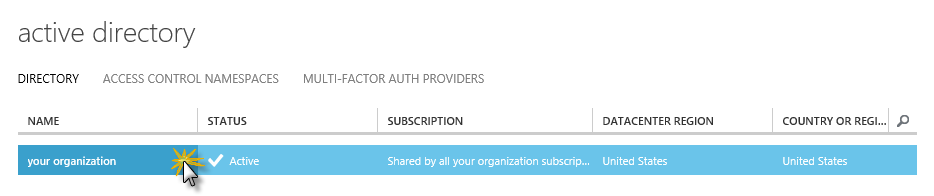
**Exercise 2: Using the Graph API to Query Azure Active Directory**

This builds on the previous exercise and will show how to add capability to read directory data using the **Azure AD Graph API**. The Graph API is a new RESTful API that allows applications to access customers' data in Microsoft Azure directory.

**Task 1 - Configuring Application Authorization and Authentication for the Graph API**

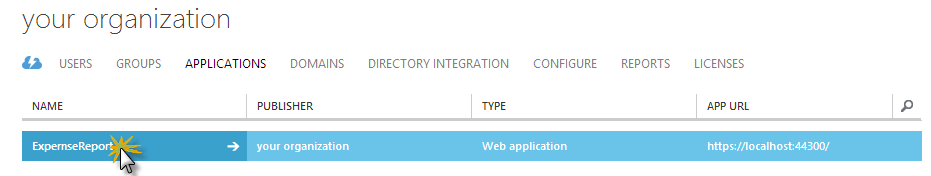
In this task you will update the application configuration in the Management Portal to enable the MVC application to authenticate and be authorized to call the Graph API. With authorization, you configure your app permissions to allow read/write access to the directory. With authentication, you get an Application Key, which is your application's password and will be used to authenticate your application to the Graph API.

1. Log on to the [Microsoft Azure Management Portal](https://manage.windowsazure.com/) and select **Active Directory** in the left pane.
2. In the list of directories, click on the directory you created in the previous exercise.

[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/your-organization-directory.png?raw=true)

*Your organization directory*

1. Click on the **Applications** tab to display the list of applications and click the application arrow you created in the previous exercise to go to its dashboard.

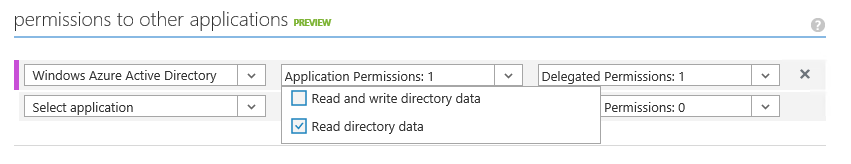
[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/selecting-application-name.png?raw=true)

*Selecting Application Name*

1. In the **CONFIGURE** page, under the **keys** section, add a key by selecting the key's lifespan, and then click **SAVE** at the bottom of the screen. This will generate a key value that is your application's password and will be used in the application configuration.

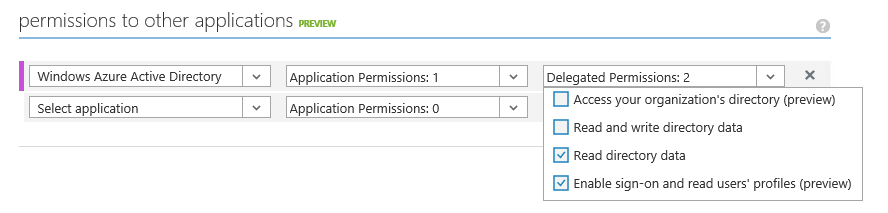
*Manage Access Button*

1. On the **What do you want to do?** screen, click **Change the directory access for this app**.
2. In the **CONFIGURE** page, under the **permissions to other applications** section, select **Read directory data** in the**Application Permissions** drop-down list for the **Windows Azure Active Directory** application.

[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/configure-app-permissions---app-permissions.png?raw=true)

1. Also in the **CONFIGURE** page and under the **permissions to other applications** section, select **Read directory data** in the **Delegated Permissions** drop-down list. for the **Windows Azure Active Directory** application.

**Note:** You can also set the application permissions by downloading, updating and uploading the JSON Manifest file. For more information see [Appendix A: Configuring permissions to other applications by updating the application manifest](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/HOL.md#appendixA).

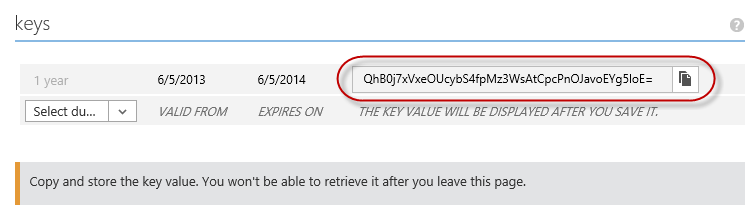
[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/configure-application-permissions-to-other-ap.png?raw=true)

1. Click **Save** in the bottom toolbar and wait until the configuration for the application has been updated successfully.

[Save Application Permissions](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/save-application-permissions.png?raw=true)

1. In the **CONFIGURE** page, under the **keys** section, add a key by selecting the key's lifespan, and then click **SAVE** at the bottom of the screen. This will generate a key value that is your application's password and will be used in the application configuration.

**Note:** The key value is displayed after key creation but cannot be later retrieved. Therefore, you should immediately copy the key value and store it in a secure place for future reference. Also, your application can have multiple keys. For example, you may want more than one for testing and production.

[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/generating-key-for-read-and-write.png?raw=true)

*Generating Key for Read and Write*

**Task 2 - Including Graph API Helper in MVC App**

In this task you will add the Graph API Client library to your MVC app. This library provides easy-to-use functionality to access directory information from Azure Active Directory.

1. Download the **Graph API Helper** project from <http://go.microsoft.com/fwlink/?LinkID=290812> and extract its contents to a known location.
2. If not already open, start **Visual Studio 2013** and continue with the solution obtained in the previous exercise. Alternatively, you can open the **Begin.sln** solution from the **Source\Ex2-UsingGraphAPIWithWAAD\Begin** folder of this lab.

**Note:** If you opened the **Begin.sln** solution, you need to **enable SSL** from the properties of the ExpenseReport project. Update the **project URL** in the **Web** tab of the ExpenseReport project properties in the *Use Local IIS Web server* section with the SSL URL obtained from the previous step. Also update the **App URL** of the configured application in the Microsoft Azure Management Portal with the SSL URL obtained from the first step.

In the **Web.config** file, update the following placeholders:

* + The *[APP-ID-URI]* placeholder of the **audienceUris** element in the **system.identityModel** section with your**App ID URI**.
  + The *[APP-ID-URI]* placeholder of the **realm** attribute from the **federationConfiguration** element in the**system.identityModel.services** section with your **App ID URI**
  + The *[YOUR-DIRECTORY-NAME]* placeholder of the **issuer** attribute from the **federationConfiguration**element in the **system.identityModel.services** section with your directory name (without spaces)
  + The *[FEDERATION-METADATA-DOCUMENT]* placeholder for the **ida:FederationMetadataLocation** key in**AppSettings**
  + The *[APP-ID-URI]* placeholder for the **ida:Realm** key in **AppSettings**
  + The *[APP-ID-URI]* placeholder for the **ida:AudienceUri** key in **AppSettings**

Press **CTRL + SHIFT + B** to build the solution and download the NuGet package dependencies.

1. Open the **Web.config** file of the **ExpenseReport** project. Add the following key values to the **appSettings** section. Make sure you update the *[YOUR-CLIENT-ID]* placeholder with the **Client ID** value obtained from the **Configure** tab of your application in the Microsoft Azure Management Portal and the *[YOUR-APPLICATION-KEY-VALUE]* placeholder with the key that you generated in the previous task.
2. To add the Graph API Helper to the single sign-on project, right-click the solution and click **Add | Existing Project...**.
3. From the **Add Existing Project** dialog box, navigate to the folder where you extracted the Graph API Helper and open the **Microsoft.WindowsAzure.ActiveDirectory.GraphHelper.csproj** project file.
4. Open the **Web.config** file of the **ExpenseReport** project. Add the following key values to the **appSettings** section. Make sure you update the *[YOUR-CLIENT-ID]* placeholder with the **Client ID** value obtained from the **Configure** tab of your application in the Microsoft Azure Management Portal and the *[YOUR-APPLICATION-KEY-VALUE]* placeholder with the key that you generated in the previous task.

<appSettings>

<add key="ClientId" value="[YOUR-CLIENT-ID]"/>

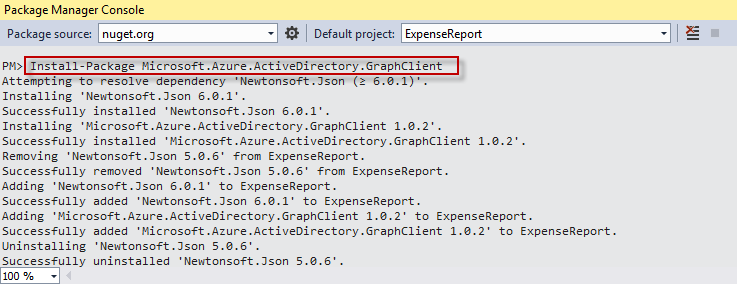
<add key="Password" value="[YOUR-APPLICATION-KEY-VALUE]"/>

...

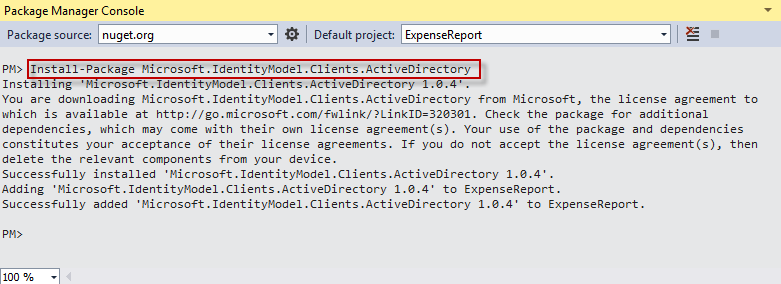
<appSettings>

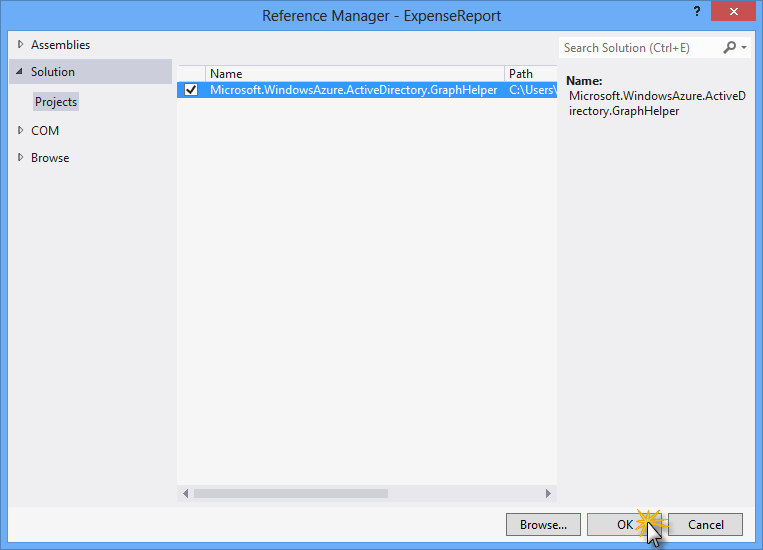
1. Save the **Web.config** file after making the changes.
2. Open the **Package Manager Console** by clicking **View | Other Windows | Package Manager Console**.
3. In the console, type the following command to download and install the **Microsoft Data Services Client** NuGet package and its dependencies. Make sure that **ExpenseReport** is set as the Default project.

Install-Package Microsoft.Data.Services.Client -Version 5.3.0

[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/package-manager-console-install-graph-client.png?raw=true)

*Package Manager Console*

1. In the **ExpenseReport** project, right-click the **References** folder and click **Add Reference...**.
2. In the **Reference Manager** dialog box, expand the **Solution** menu on the left and then select**Microsoft.WindowsAzure.ActiveDirectory.GraphHelper**. Click **OK** to add the references. [](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/package-manager-console-install-adal.png?raw=true)

[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/graph-api-helper-reference.png?raw=true)

*Graph API Helper Reference*

1. Press **Ctrl** + **Shift** + **S** to save the unsaved changes.

**Task 3 - Displaying Active Directory Query Data**

In this task you will update the **HomeController** of your MVC app to query the list of users from the Active Directory tenant using the Graph API Helper, and create a view to display the results.

1. In the **Solution Explorer**, expand the **Controllers** folder of the **ExpenseReport** project and open **HomeController.cs**. Add the following references to assemblies to the file.

(Code Snippet - *IntroductionToAzureAD - Ex2 - HomeControllerReferences*)

using System.Configuration;

using System.Data.Services.Client;

using Microsoft.WindowsAzure.ActiveDirectory;

using Microsoft.WindowsAzure.ActiveDirectory.GraphHelper;

1. Add the following declarations at the beginning of the **HomeController** class.

(Code Snippet - *IntroductionToAzureAD - Ex2 - HomeControllerDeclarations*)

private const string Resource = "https://graph.windows.net";

private const string AuthString = "https://login.windows.net/{0}";

private const string ClaimType = "http://schemas.microsoft.com/identity/claims/tenantid";

(Code Snippet - *IntroductionToAzureAD - Ex2 - UsersActionMethod*)

public class HomeController : Controller

{

...

public ActionResult Users()

{

// get the tenantName

var tenantId = ClaimsPrincipal.Current.FindFirst(ClaimType).Value;

var authenticationContext = new AuthenticationContext(

string.Format(AuthString, tenantId),

false);

authenticationContext.TokenCacheStore.Clear();

// retrieve the clientId and password from the Web.config file

var clientId = ConfigurationManager.AppSettings["ClientId"];

var clientSecret = ConfigurationManager.AppSettings["Password"];

// create the credentials and get a token

var clientCred = new ClientCredential(clientId, clientSecret);

try

{

var token = authenticationContext.AcquireToken(Resource, clientCred).AccessToken;

var users = graphService.users;

QueryOperationResponse<User> response;

response = users.Execute() as QueryOperationResponse<User>;

List<User> userList = response.ToList();

ViewBag.userList = userList;

if (!string.IsNullOrEmpty(token))

// The following checks to see if the existing token is expired or about to expire in 2 mins

// if true, then get a new token and refresh the graphService

int tokenMins = 2;

if (token.IsExpired || token.WillExpireIn(tokenMins))

{

var graphSettings = new GraphSettings

{

ApiVersion = "2013-11-08",

GraphDomainName = "graph.windows.net"

};

// get tenant information

var graphConnection = new GraphConnection(token, graphSettings);

var tenant = graphConnection.Get(typeof(TenantDetail), tenantId);

if (tenant != null)

{

var tenantDetail = (TenantDetail)tenant;

ViewBag.OtherMessage = "User List from tenant: " + tenantDetail.DisplayName;

}

// get the list of users

var pagedReslts = graphConnection.List<User>(null, new FilterGenerator());

return View(pagedReslts.Results);

}

}

catch (ActiveDirectoryAuthenticationException ex)

{

ViewBag.OtherMessage = string.Format("Acquiring a token failed with the following error: {0}", ex.Message);

}

catch (AuthorizationException e)

{

ViewBag.OtherMessage = string.Format("Failed to return the list of Users with Exception: {0}", e.ErrorMessage);

}

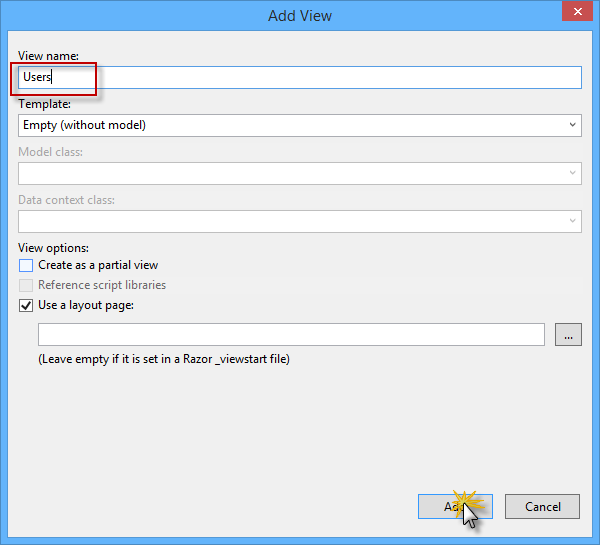
return View();

}

}

**Note:** It is recommended that the JWT token be cached by the application for subsequent calls � in this block, the JWT token expiration is checked before making a second Graph API call. If the token is expired, then a new token is acquired. If a call to the Graph API is made with an expired token, an error response will be returned, and the client must request a new token.

1. You will now add a new view to display the list of users retrieved from the Active Directory tenant. To do this, expand the**Views** folder of the **ExpenseReport** project, right-click the **Home** folder and select **Add | View...**. In the **Add View** dialog box, set the view name to *Users* and click **Add**.



*Adding Users View*

1. Replace the code of the **Users** view with the following block.

(Code Snippet - *IntroductionToAzureAD - Ex2 - UsersView*)

@model IEnumerable<Microsoft.WindowsAzure.ActiveDirectory.User>

@{

ViewBag.Title = "Users";

}

<h1>@ViewBag.Message</h1>

<h2>@ViewBag.OtherMessage</h2>

@if (Model != null)

{

<table class="table table-striped">

<tr>

<th>

DisplayName

</th>

<th>

UPN

</th>

</tr>

@if (User.Identity.IsAuthenticated)

{

foreach (var user in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => user.DisplayName)

</td>

<td>

@Html.DisplayFor(modelItem => user.UserPrincipalName)

</td>

</tr>

}

}

</table>

}

1. In the **Solution Explorer**, expand the **Views/Shared** folder and open **\_Layout.cshtml**. Find the **<nav>** element inside the **<header>** section and add the following highlighted action link to the *Users* action method of the **HomeController**.

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav">

<li>@Html.ActionLink("Home", "Index", "Home")</li>

<li>@Html.ActionLink("About", "About", "Home")</li>

<li>@Html.ActionLink("Contact", "Contact", "Home")</li>

<li>@Html.ActionLink("Users", "Users", "Home")</li>

</ul>

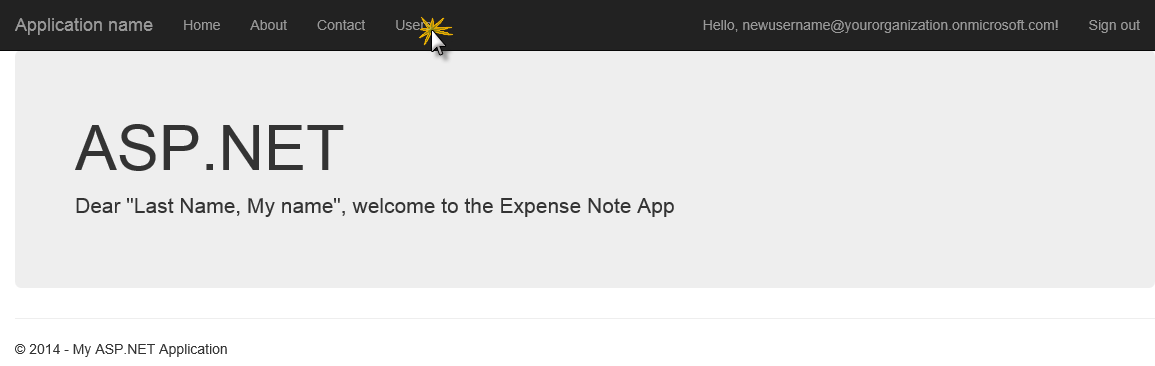
@Html.Partial("\_LoginPartial")

</div>

1. Press **Ctrl** + **Shift** + **S** to save the unsaved changes.

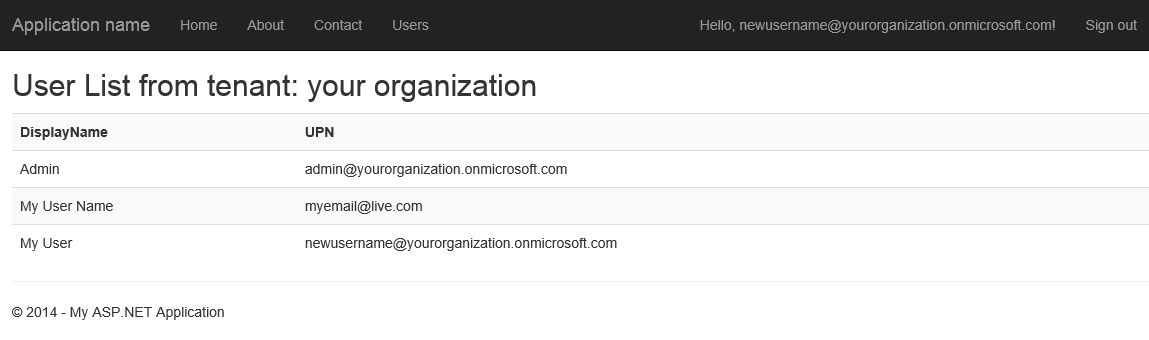
**Task 4 � Verification**

1. Press **F5** to run the application. The single sign-on experience is the same as you saw in the previous exercise, requiring authentication using your Azure AD credentials.
2. Once you have successfully authenticated using your credentials, click the **Users** tab in the top menu bar.

[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/users-action-link.png?raw=true)

*Users Action Link*

1. You should see the *Users* view displaying the list of users from the Active Directory tenant.

[](https://github.com/Azure-Readiness/MicrosoftAzureTrainingKit/blob/master/HOLs/HOL-IntroToAzureAD/Images/displaying-users-from-ad-tenant.png?raw=true)

*Displaying Users from AD Tenant*

**Next Steps**

To learn more about **Azure Active Directory**, please refer to the following articles:

**Technical Reference**

This is a list of articles that expand on the technologies explained in this lab:

* [Directory integration](http://aka.ms/Oe9k89): If your organization uses an on-premises directory service, you can integrate it with your Azure Active Directory (Azure AD) tenant to simplify your cloud-based administrative tasks and even provide your users with a more streamlined sign-in experience.
* [Introducing Single Sign-on and Active Directory Integration (Video)](http://aka.ms/Uhh5bm): In this video you will see learn about the SSO configuration and end-user experience that WAAD offers and how to integrate WAAD into applications developed with Visual Studio 2013.
* [Manage Azure AD using Windows PowerShell](http://aka.ms/Xfpfmr): As an administrator, you can use the Azure Active Directory Module for Windows PowerShell cmdlets to accomplish many Azure AD tenant-based administrative tasks such as user management, domain management and for configuring single sign-on. This topic includes information on how to install these cmdlets to use with your tenant.
* [Microsoft Azure Identity](http://aka.ms/S14yvq): Managing identity is just as important in the public cloud as it is on-premises, and Microsoft Azure supports several different cloud identity technologies.

**Development**

This is a list of developer-oriented articles related to **Azure Active Directory**:

* [Using the Graph API to Query Azure AD](http://aka.ms/Pk9n2r): This document explains how to configure a Microsoft .NET application to use the Azure Active Directory Graph API to access data from a Azure AD tenant directory.
* [Securing a Windows Store Application and REST Web Service Using Azure AD (Preview)](http://aka.ms/t5ejfa): This document will show you how to create a simple web API resource and a Windows Store client application using the Microsoft Azure Authentication Library and Azure AD.

**Summary**

By completing this hands-on lab you learned how to:

* Create a new Azure Active Directory tenant
* Provision an MVC application in the AD tenant
* Explore the configuration of the application Authentication
* Query Active Directory data using Graph AD API

**Appendix A: Configuring Permissions to other Applications by Updating the Application Manifest**

In this appendix you will learn how to set up permissions to other applications by updating the JSON Manifest for an application.

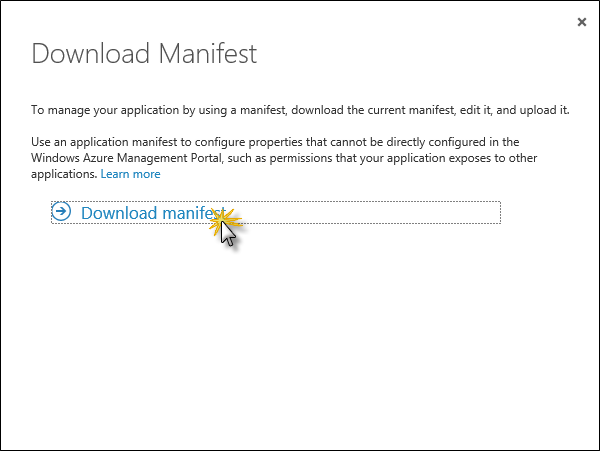
**Configuring permissions to other applications by updating the application manifest**

1. Click **Manage Manifest** in the bottom toolbar and select **Download Manifest**.



*Manage Manifest*

1. In the Download Manifest dialog box, click **Download Manifest**. When prompted, save the file and close the dialog box.



*Download Manifest*

1. Open the application manifest file with **Visual Studio**. Find the **requiredResourceAccess** section and add the section highlighted below. Save the file.

"requiredResourceAccess": [

{

"resourceAppId": "00000002-0000-0000-c000-000000000000",

"requiredAppPermissions": [

{

"permissionId": "5778995a-e1bf-45b8-affa-663a9f3f4d04",

"directAccessGrant": true,

"impersonationAccessGrants": [

"User"

]

},

{

"permissionId": "311a71cc-e848-46a1-bdf8-97ff7156d8e6",

"directAccessGrant": false,

"impersonationAccessGrants": [

"User"

]

}

]

}

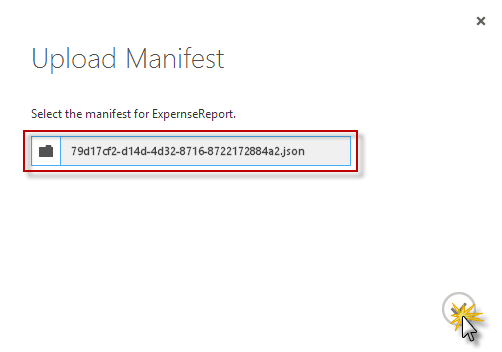
],

1. In the Management Portal, click **Manage Manifest** and select **Upload Manifest**.



*Upload Manifest button*

1. Select the file you just updated and upload the manifest.



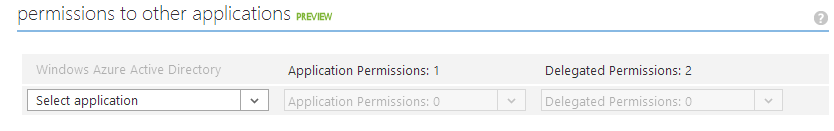
*Upload manifest dialog box*

1. Wait until the operation is complete and the page refreshes.

Upload Manifest completed

*Upload Manifest completed*

1. Notice that in the **Permissions to other applications** section, both permissions have been updated.



*Permissions to other applications*

End of the document.