AzureADSamples / **WebApp-WebAPI-OAuth2-AppIdentity-DotNet**

Watch 37 Star 21 Fork 6

In contrast to the WebApp-WebAPI-OpenIDConnect-DotNet sample, this sample shows how to build an MVC web application that uses Azure AD for sign-in using OpenID Connect, and then calls a web API under the application's identity (instead of the user's identity) using tokens obtained via OAuth 2.0. This sample uses the OpenID Connect ASP.Net OWIN m…

23 commits **2** branches **2** releases **2** contributors

Code
Issues 5
Pull requests 0
Pulse
Graphs

Branch: **master** WebApp-WebAPI-OAuth2-AppIdentity-DotNet / +

Error Handling

dstrockis authored on 13 Apr latest commit 9c084938fa

TodoListService	Switching to new AAD endpoint	5 months ago
TodoListWebApp	Error Handling	5 months ago
.gitignore	Working application and README.MD	a year ago
LICENSE	Initial commit	a year ago
README.md	oAuth2Permissions Auto Created	8 months ago
WebApp-WebAPI-OAuth2-AppIdentity-Dot…	Working application and README.MD	a year ago

HTTPS clone URL

`https://github.com/A`

You can clone with HTTPS or Subversion.

Clone in Desktop

Download ZIP

README.md

WebApp-WebAPI-OAuth2-AppIdentity-DotNet

In contrast to the WebApp-WebAPI-OpenIDConnect-DotNet sample, this sample shows how to build an MVC web application that uses Azure AD for sign-in using OpenID Connect, and then calls a web API under the application's identity (instead of the user's identity) using tokens obtained via OAuth 2.0. This sample uses the OpenID Connect ASP.Net OWIN middleware and ADAL .Net.

This sample is an example of the trusted sub-system model, where the web API trusts the web application to have authenticated the user, and receives no direct evidence from Azure AD that the user was authenticated. If you want to build a web app that calls a web API using a delegated user identity, please check out the sample named WebApp-WebAPI-OpenIDConnect-DotNet.

For more information about how the protocols work in this scenario and other scenarios, see Authentication Scenarios for Azure AD.

How To Run This Sample

To run this sample you will need:

- Visual Studio 2013
- An Internet connection
- An Azure subscription (a free trial is sufficient)

Every Azure subscription has an associated Azure Active Directory tenant. If you don't already have an Azure subscription, you can get a free subscription by signing up at http://wwww.windowsazure.com. All of the Azure AD features used by this sample are available free of charge.

Step 1: Clone or download this repository

From your shell or command line:

```
git clone https://github.com/AzureADSamples/WebApp-WebAPI-OAuth2-AppIdentity-DotNet.git
```

Step 2: Create a user account in your Azure Active Directory tenant

If you already have a user account in your Azure Active Directory tenant, you can skip to the next step. This sample will not work with a Microsoft account, so if you signed in to the Azure portal with a Microsoft account and have never created a user account in your directory before, you need to do that now. If you create an account and want to use it to sign-in to the Azure portal, don't forget to add the user account as a co-administrator of your Azure subscription.

Step 3: Register the sample with your Azure Active Directory tenant

There are two projects in this sample. Each needs to be separately registered in your Azure AD tenant.

Register the TodoListService web API

1. Sign in to the Azure management portal.
2. Click on Active Directory in the left hand nav.
3. Click the directory tenant where you wish to register the sample application.
4. Click the Applications tab.
5. In the drawer, click Add.
6. Click "Add an application my organization is developing".
7. Enter a friendly name for the application, for example "TodoListService", select "Web Application and/or Web API", and click next.
8. For the sign-on URL, enter the base URL for the sample, which is by default `https://localhost:44321` .
9. For the App ID URI, enter `https://<your_tenant_name>/TodoListService` , replacing `<your_tenant_name>` with the name of your Azure AD tenant. Click OK to complete the registration.
10. While still in the Azure portal, click the Configure tab of your application.
11. Find the Client ID value and copy it aside, you will need this later when configuring your application.

Register the TodoListWebApp web app

1. Sign in to the Azure management portal.
2. Click on Active Directory in the left hand nav.
3. Click the directory tenant where you wish to register the sample application.
4. Click the Applications tab.
5. In the drawer, click Add.
6. Click "Add an application my organization is developing".
7. Enter a friendly name for the application, for example "TodoListWebApp-AppIdentity", select "Web Application and/or Web API", and click next.
8. For the sign-on URL, enter the base URL for the sample, which is by default `https://localhost:44322/` .
9. For the App ID URI, enter `https://<your_tenant_name>/TodoListWebApp-AppIdentity` , replacing `<your_tenant_name>` with the name of your Azure AD tenant. Click OK to complete the registration.
10. While still in the Azure portal, click the Configure tab of your application.
11. Find the Client ID value and copy it aside, you will need this later when configuring your application.
12. Create a new key for the application. Save the configuration so you can view the key value. Save this aside for when you configure the project in Visual Studio.
13. In "Permissions to Other Applications", click "Add Application." Select "Other" in the "Show"

dropdown, and click the upper check mark. Locate & click on the TodoListService, and click the bottom check mark to add the application. Select "Access TodoListService" from the "Delegated Permissions" dropdown, and save the configuration.

Step 4: Configure the sample to use your Azure AD tenant

Configure the TodoListService project

1. Open the solution in Visual Studio 2013.
2. Open the `web.config` file.
3. Find the app key `ida:Tenant` and replace the value with your AAD tenant name.
4. Find the app key `ida:Audience` and replace the value with the App ID URI you registered earlier, for example `https://<your_tenant_name>/TodoListService` .
5. Find the app key `ida:ClientId` and replace the value with the Client ID for the TodoListService from the Azure portal.

Configure the TodoListWebApp project

1. Open the solution in Visual Studio 2013.
2. Open the `web.config` file.
3. Find the app key `ida:Tenant` and replace the value with your AAD tenant name.
4. Find the app key `ida:ClientId` and replace the value with the Client ID for the TodoListWebApp-AppIdentity from the Azure portal.
5. Find the app key `ida:AppKey` and replace the value with the key for the TodoListWebApp-AppIdentity from the Azure portal.
6. If you changed the base URL of the TodoListWebApp sample, find the app key `ida:PostLogoutRedirectUri` and replace the value with the new base URL of the sample.
7. Find the app key `todo:TodoListBaseAdress` ane make sure it has the correct value for the address of the TodoListService project.
8. Find the app key `todo:TodoListResourceId` and replace the value with the App ID URI registered for the TodoListService, for example `https://<your_tenant_name>/TodoListService` .

Configure the TodoListWebApp as a trusted caller to the TodoListService

1. Go back to the TodoListService project and open the `web.config` file.
2. Find the app key `todo:TrustedCallerClientId` and replace the value with the Client ID of the TodoListWebApp. This is used to tell the service to trust the web app.

Step 5: Trust the IIS Express SSL certificate

Since the web API is SSL protected, the client of the API (the web app) will refuse the SSL connection to the web API unless it trusts the API's SSL certificate. Use the following steps in Windows Powershell to trust the IIS Express SSL certificate. You only need to do this once. If you fail to do this step, calls to the TodoListService will always throw an unhandled exception where the inner exception message is:

"The underlying connection was closed: Could not establish trust relationship for the SSL/TLS secure channel."

To configure your computer to trust the IIS Express SSL certificate, begin by opening a Windows Powershell command window as Administrator.

Query your personal certificate store to find the thumbprint of the certificate for `CN=localhost` :

```
PS C:\windows\system32> dir Cert:\LocalMachine\My


    Directory: Microsoft.PowerShell.Security\Certificate::LocalMachine\My


Thumbprint                                Subject
```

```
    ----------                                 -------
    C24798908DA71693C1053F42A462327543B38042   CN=localhost
```

Next, add the certificate to the Trusted Root store:

```
PS C:\windows\system32> $cert = (get-item cert:\LocalMachine\My\C24798908DA71693C1053F42A462327
PS C:\windows\system32> $store = (get-item cert:\Localmachine\Root)
PS C:\windows\system32> $store.Open("ReadWrite")
PS C:\windows\system32> $store.Add($cert)
PS C:\windows\system32> $store.Close()
```

You can verify the certificate is in the Trusted Root store by running this command:

```
PS C:\windows\system32> dir Cert:\LocalMachine\Root
```

Step 6: Run the sample

Clean the solution, rebuild the solution, and run it. You might want to go into the solution properties and set both projects as startup projects, with the service project starting first.

Explore the sample by signing in, clicking the To Do List link, adding items to the To Do list, signing out, and starting again.

How To Deploy This Sample to Azure

Coming soon.

About The Code

Coming soon.

How To Recreate This Sample

First, in Visual Studio 2013 create an empty solution to host the projects. Then, follow these steps to create each project.

Creating the TodoListService Project

1. In the solution, create a new ASP.Net MVC web API project called TodoListService and while creating the project, click the Change Authentication button, select Organizational Accounts, Cloud - Single Organization, enter the name of your Azure AD tenant, and set the Access Level to Single Sign On. You will be prompted to sign-in to your Azure AD tenant. NOTE: You must sign-in with a user that is in the tenant; you cannot, during this step, sign-in with a Microsoft account.
2. In the `Models` folder add a new class called `TodoItem.cs` . Copy the implementation of TodoItem from this sample into the class.
3. Add a new, empty, Web API 2 controller called `TodoListController` .
4. Copy the implementation of the TodoListController from this sample into the controller. Don't forget to add the `[Authorize]` attribute to the class.
5. In `TodoListController` resolving missing references by adding `using` statements for `System.Collections.Concurrent` , `TodoListService.Models` , `System.Security.Claims` .
6. In `web.config` , in `<appSettings>` , create a key `todo:TrustedCallerClientId` and set the value accordingly.

Creating the TodoListWebApp Project

1. In the solution, create a new ASP.Net MVC web application called TodoListWebApp with Authentication set to No Authentication.

2. Set SSL Enabled to be True. Note the SSL URL.
3. In the project properties, Web properties, set the Project Url to be the SSL URL.
4. Add the following ASP.Net OWIN middleware NuGets: Microsoft.IdentityModel.Protocol.Extensions, System.IdentityModel.Tokens.Jwt, Microsoft.Owin.Security.OpenIdConnect, Microsoft.Owin.Security.Cookies, Microsoft.Owin.Host.SystemWeb.
5. Add the Active Directory Authentication Library NuGet (`Microsoft.IdentityModel.Clients.ActiveDirectory`).
6. In the `App_Start` folder, create a class `Startup.Auth.cs` . You will need to remove `.App_Start` from the namespace name. Replace the code for the `Startup` class with the code from the same file of the sample app. Be sure to take the whole class definition! The definition changes from `public class Startup` to `public partial class Startup` .
7. Right-click on the project, select Add, select "OWIN Startup class", and name the class "Startup". If "OWIN Startup Class" doesn't appear in the menu, instead select "Class", and in the search box enter "OWIN". "OWIN Startup class" will appear as a selection; select it, and name the class `Startup.cs` .
8. In `Startup.cs` , replace the code for the `Startup` class with the code from the same file of the sample app. Again, note the definition changes from `public class Startup` to `public partial class Startup` .
9. In the `Views` --> `Shared` folder, create a new partial view `_LoginPartial.cshtml` . Replace the contents of the file with the contents of the file of same name from the sample.
10. In the `Views` --> `Shared` folder, replace the contents of `_Layout.cshtml` with the contents of the file of same name from the sample. Effectively, all this will do is add a single line, `@Html.Partial("_LoginPartial")` , that lights up the previously added `_LoginPartial` view.
11. Create a new empty controller called `AccountController` . Replace the implementation with the contents of the file of same name from the sample.
12. If you want the user to be required to sign-in before they can see any page of the app, then in the `HomeController` , decorate the `HomeController` class with the `[Authorize]` attribute. If you leave this out, the user will be able to see the home page of the app without having to sign-in first, and can click the sign-in link on that page to get signed in.
13. In the `Models` folder add a new class called `TodoItem.cs` . Copy the implementation of TodoItem from this sample into the class.
14. Add a new empty MVC5 controller TodoListController to the project. Copy the implementation of the controller from the sample. Remember to include the [Authorize] attribute on the class definition.
15. In `Views` --> `TodoList` create a new view, `Index.cshtml` , and copy the implementation from this sample.
16. In the shared `_Layout` view, add the Action Links for Profile and To Do List that are in the sample.
17. In `web.config` , in `<appSettings>` , create keys for `ida:ClientId` , `ida:AppKey` , `ida:AADInstance` , `ida:Tenant` , and `ida:PostLogoutRedirectUri` , and set the values accordingly. For the public Azure AD, the value of `ida:AADInstance` is `https://login.windows.net/{0}` .
18. In `web.config` in `<appSettings>` , create keys for `todo:TodoListResourceId` and `todo:TodoListBaseAddress` and set the values accordinly.

Finally, in the properties of the solution itself, set both projects as startup projects.