**Implementing Single Sign-On with Azure AD –**

**Detailed Design Document**

**Version 1.0**

**Monday, 02 March 2020**

**REVISION LOG**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Changes done** | **Comments** |
| 1.0 | Monday, 02 March 2020 |  | First Draft | Initial Draft |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**DETAILED DESIGN DOCUMENT SIGN OFF**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Name** | **Position/Role** | **Date** | **Signature** |
| Reviewed By: |  | Architecture / Project Owner |  |  |
| Approved By: |  | IT Infrastructure Manager / Project Advisor |  |  |

Contents

[1. Introduction 5](#_Toc22818405)

[1.1. Purpose 5](#_Toc22818406)

[1.2. Scope 5](#_Toc22818407)

[1.3. Project Vision and Scope Summary 5](#_Toc22818408)

[1.4. Intended Audience 6](#_Toc22818409)

[1.5. Acronyms and Definitions 7](#_Toc22818410)

[2. Design Assumption 7](#_Toc22818411)

[3. Integrate Azure AD into existing web application using WS Federation authentication 8](#_Toc22818412)

[3.1. Modification of an application environment that can support Azure AD Authentication 8](#_Toc22818413)

[3.1.1. SSL Enabling 8](#_Toc22818414)

[3.1.3. To configure Web config of ASP.NET application for claims-based authentication 10](#_Toc22818415)

[3.1.4. Adding ASP.Net application Code Changes 11](#_Toc22818416)

[3.1.5. Adding to Logout method 13](#_Toc22818417)

[3.2. Configuration of Azure AD to register the Application 13](#_Toc22818418)

[3.2.1. Register a new application using the Azure portal 14](#_Toc22818419)

[4. Run the Application 20](#_Toc22818420)

[5. Authentication flow using WS Federation Authentication 21](#_Toc22818421)

[6. Appendix 22](#_Toc22818422)

[6.1. Reference URLS 22](#_Toc22818423)

# Introduction

## Purpose

The purpose of this document is to outline all the detailed pre-requisites and steps to move an ASP.NET 4.0 Web Forms application from Active Directory based LDAP authentication to claims-aware ASP.NET 4.0 Web Forms application for successful implementation of WS federated authentication with Azure AD

This document is guide to help application owners and vendors to move the apps to SSO on Azure AD

## Scope

The scope of this document is to detail out the complete design and architecture for the ASP.Net 4.0 Web Forms application for successful implementation of WS federated authentication with Active Directory (Azure AD).

This document covers how to integrate existing .Net 4.0 web application that uses WS Federation authentication to sign in users from a single Azure Active Directory. All the design and development changes made will be covered in this document.

This document is a step-by-step guide for implementation and configuration.

## Project Vision and Scope Summary

The vision and primary goal of the solution is to Integrate Azure AD into a web application using WS Federation authentication. Based on the initial project kick-off scope below is the vision and scope summary of this project

* + **Assessment**
* Detailed assessment of tenant’s Azure Active Directory environment
* Assessment of technical requirements around existing applications, frameworks and databases
  + **Designing and development**
* Design will be done for Integrating Azure AD into existing web application using WS Federation authentication
  + **Development**
* Steps for configuring single sign-on for ASP.NET 4.0 Web Forms applications using organizational authentication.
  + **Azure App Registration**
* Registration of the existing application with your Azure Active Directory tenant
  + **Security best practices**
* Single sign-on (SSO) adds security and convenience when users sign-on to applications in Azure Active Directory (Azure AD). Administrators can centralize user account management, and automatically add or remove user access to applications based on group membership.
  + **Single Sign-On in all the sites**
* Azure Active Directory Seamless Single Sign-On (Azure AD Seamless SSO) automatically signs users in when they are on their corporate devices connected to your corporate network.
* When enabled, users don't need to type in their passwords to sign in to Azure AD, and usually, even type in their usernames. This feature provides your users easy access to your cloud-based applications without needing any additional on-premises components.

## Intended Audience

|  |  |  |
| --- | --- | --- |
| **Name** | **Role** | **Department/**  **Team** |
| Application owners |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Acronyms and Definitions

|  |  |
| --- | --- |
| **Acronym** | **Definition** |
| AD | Active Directory |
| AAD | Azure Active Directory |
| SSO | Single Sign-On |
| ASP.Net | Active Server Pages .Net language |
| WS Fed | Web Services Federation |
| SSL | Secure Sockets Layer |
| URL / URI | Uniform Resource Locator / Identifier |

# Design Assumption

The following assumptions have been made while suggesting the proposed Azure Active Directory Seamless Single Sign-On (Azure AD Seamless SSO)

| Assumption |
| --- |
| Dot Net Framework 4.0 web form application |
| Access to code for modification |
| Access to database for testing |
| Microsoft Azure Subscription |
| Microsoft Azure Active Directory |
| Admin access to Azure portal |

# Integrate Azure AD into existing web application using WS Federation authentication

There are two sections in this course of enabling Single Sign-On across applications:

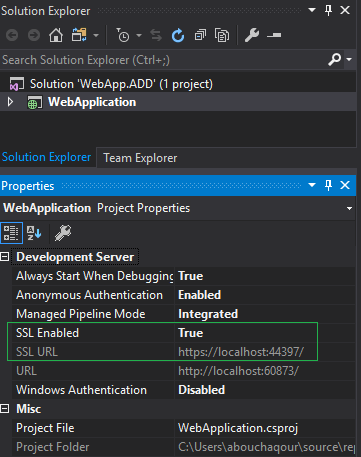
1. Modification of an application environment that can support Azure AD Authentication
2. Configuration of Azure AD to register the Application

## Modification of an application environment that can support Azure AD Authentication

The following modification were made in the existing environment to come out with the best possible design

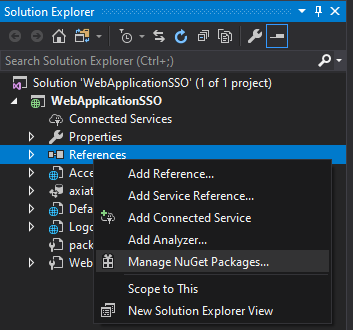
## SSL Enabling

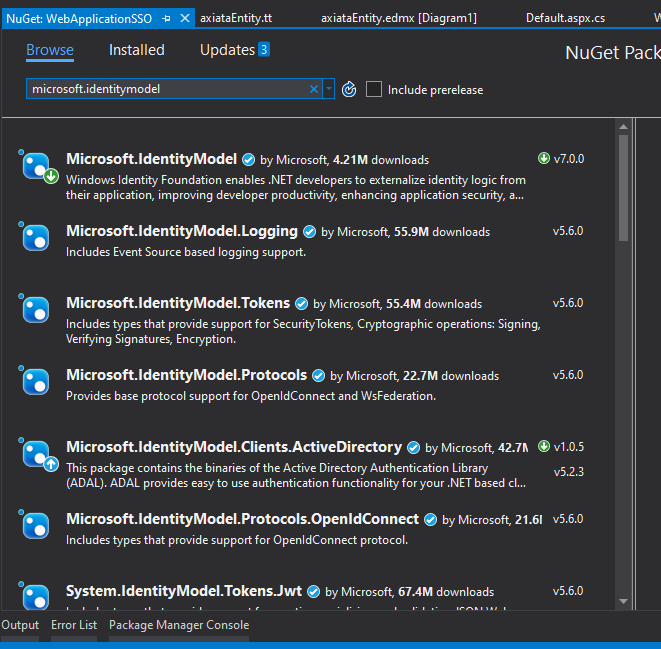
* Select your project under “Solution Explorer” then you must see the “Project Properties” window. If you don't see it press “F4” key on your keyboard then it will appear.
* Change the “SSL Enabled” property value to “True”, then copy the value of “SSL URL” property because you will need it to configure Azure AD.
* The image below illustrates the state of “SSL Enabled” property and “SSL URL” property after changing the value of “SSL Enabled” property from “false” to “true”.



* + 1. Add Authorization Library

1. Go to Solution Explorer -> Project -> References right click on it and select the “Manage NuGet Packages…” from the list. Search for the Microsoft.IdentityModel.dll. Click Install.





## To configure Web config of ASP.NET application for claims-based authentication

* + - * Add the following configuration section entries to the Web.config configuration file immediately after the <**configuration**> opening element:

<configSections>

<section name="microsoft.identityModel" type="Microsoft.IdentityModel.Configuration.MicrosoftIdentityModelSection, Microsoft.IdentityModel, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" />

</configSections>

* + - * Add a **<location>** element that enables access to the application’s federation metadata:

<location path="FederationMetadata">

<system.web>

<authorization>

<allow users="\*" />

</authorization>

</system.web>

</location>

* + - * Add the following configuration entries within the **<system.web>** elements to deny users, disable native authentication, and enable WIF to manage authentication.

<authorization>

<deny users="?" />

</authorization>

<authentication mode="None" />

* + - * Add a **<system.webServer>** element that defines the modules for federated authentication. Note that the PublicKeyToken attribute must be the same as the PublicKeyToken attribute for the <configSections> entries added earlier:

<system.webServer>

<validation validateIntegratedModeConfiguration="false" />

<modules>

<add name="SessionAuthentication" type="Microsoft.IdentityModel.Web.SessionAuthenticationModule, Microsoft.IdentityModel, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" preCondition="managedHandler" />

<add name="WSFederationAuthenticationModule" type="Microsoft.IdentityModel.Web.WSFederationAuthenticationModule, Microsoft.IdentityModel, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" preCondition="managedHandler" />

<remove name="FormsAuthenticationModule" />

<remove name="FormsAuthentication" />

</modules>

</system.webServer>

* + - * Add the following Windows Identity Foundation related configuration entries and ensure that your ASP.NET application’s URL and port number match the values in the **<audienceUris>** entry, **realm** attribute of the **<wsFederation>** element, and the **reply** attribute of the **<wsFederation>** element. Also ensure that the **issuer** value fits your endpoint URL.

<microsoft.identityModel>

<service>

<certificateValidation certificateValidationMode="None" />

<audienceUris>

<add value="api://569d1573-2eaa-4ecb-aabd-a2d8e6b45cf5" />

</audienceUris>

<issuerNameRegistry type="Microsoft.IdentityModel.Tokens.ConfigurationBasedIssuerNameRegistry, Microsoft.IdentityModel, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35">

<trustedIssuers>

<add thumbprint="68F72DC3FA1DBD13A810D8375683A5221DAD884B" name=" Demo Issuer" />

</trustedIssuers>

</issuerNameRegistry>

<federatedAuthentication>

<wsFederation passiveRedirectEnabled="true" issuer="https://login.microsoftonline.com/d1fd9326-4644-4e1c-8685-279bfbe2f5a0/wsfed" realm="api://569d1573-2eaa-4ecb-aabd-a2d8e6b45cf5" requireHttps="true" reply="https://localhost:44377/" />

<cookieHandler requireSsl="false" />

</federatedAuthentication>

</service>

</microsoft.identityModel>

* + - * Add requestValidationMode=”2.0” attribute to <httpRuntime> tag within the <system.web> elements

<httpRuntime requestValidationMode="2.0" />

## Adding ASP.Net application Code Changes

* + - * Add following lines to default Page load method. The code below illustrates how the Page\_Load method should be implemented.

protected void Page\_Load(object sender, EventArgs e)

{

var identity = User.Identity as ClaimsIdentity;

ClaimsPrincipal claimsPrincipal = Thread.CurrentPrincipal as ClaimsPrincipal;

if (claimsPrincipal.Identity.IsAuthenticated)

{

//Created a instance of contextDB.

using (demodbEntities1 context = new demodbEntities1())

{

var userDetails = context.users.Where(x => x.email == User.Identity.Name).Select(x => new { x.email, x.role, x.username }).ToList();

if (userDetails != null)

{

IList listSource = userDetails;

GridView1.DataSource = listSource;

GridView1.DataBind();

}

}

HttpContext.Current.User = claimsPrincipal;

System.Threading.Thread.CurrentPrincipal = System.Web.HttpContext.Current.User;

foreach (Claim claim in identity.Claims)

{

claimType.Text = claim.ClaimType;

claimValue.Text = claim.Value;

claimValueType.Text = claim.ValueType;

claimSubjectName.Text = claim.Subject.Name;

claimIssuer.Text = claim.Issuer;

}

}

else

{

Microsoft.IdentityModel.Web.FederatedAuthentication.SessionAuthenticationModule.SignOut();

Microsoft.IdentityModel.Web.FederatedAuthentication.SessionAuthenticationModule.DeleteSessionTokenCookie();

Microsoft.IdentityModel.Web.FederatedAuthentication.SessionAuthenticationModule.SignOut();

Response.Redirect("~/AccessDenied.aspx");

}

}

* + - * Sample form to display the claims which would not be required in actual deployment

<form id="form1" runat="server">

<div>

<span style="float:right">

<asp:button ID="Timer1" runat="server" Interval="1000" OnClick="lnkbtnlogout\_Click" Text="Logout">

</asp:button>

</span>

<h2> Azure AD User</h2>

<h1><asp:label ID="signedIn" runat="server" /></h1> <hr />

<asp:label ID="claimType" runat="server" /> <hr />

<asp:label ID="claimValue" runat="server" /> <hr />

<asp:label ID="claimValueType" runat="server" /> <hr />

<asp:label ID="claimSubjectName" runat="server" /> <hr />

<asp:label ID="claimIssuer" runat="server" /> <hr />

</div>

<div>

<h2> Database User</h2>

<asp:GridView ID="GridView1" runat="server" BackColor="#FFF" BorderColor="#000" BorderStyle="None" BorderWidth="1px" CellPadding="3" CellSpacing="2">

<FooterStyle BackColor="White" ForeColor="Black" />

<HeaderStyle BackColor="White" Font-Bold="True" ForeColor="Black" />

<PagerStyle ForeColor="Black" HorizontalAlign="Center" />

<RowStyle BackColor="White" ForeColor="Black" />

<SelectedRowStyle BackColor="#fff" Font-Bold="True" ForeColor="black" />

<SortedAscendingCellStyle BackColor="#fff" />

<SortedAscendingHeaderStyle BackColor="#fff" />

<SortedDescendingCellStyle BackColor="#fff" />

<SortedDescendingHeaderStyle BackColor="#fff" /></asp:GridView>

</div>

</form>

## Adding to Logout method

Add the Microsoft.IdentityModel methods to clear session and cookies after SignOut

protected void lnkbtnlogout\_Click(object sender, EventArgs e)

{

Microsoft.IdentityModel.Web.FederatedAuthentication.SessionAuthenticationModule.SignOut();

Microsoft.IdentityModel.Web.FederatedAuthentication.SessionAuthenticationModule.DeleteSessionTokenCookie();

Microsoft.IdentityModel.Web.FederatedAuthentication.SessionAuthenticationModule.SignOut();

Response.Redirect("~/Logout.aspx");

}

## Configuration of Azure AD to register the Application

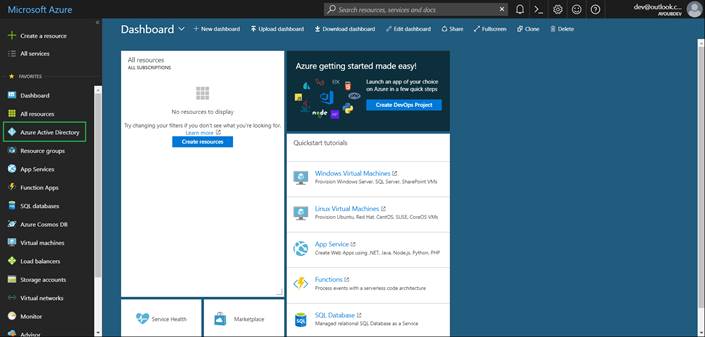
Before your web app can use Azure AD as the identity back-end it needs to be registered in Azure AD. This is done both to ensure that not every random app out there can hook into an AAD tenant, and to configure some of the mechanics needed for it to actually work with the necessary redirects.

We need to register an application for the following reasons:

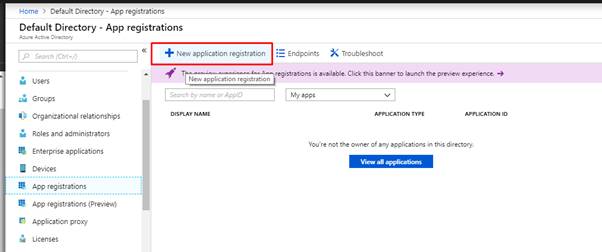
* It is used to integrate the application and service with Azure AD.
* Using Azure App, we can generate the token to authenticate the application.
* If we want to use the Azure AD capabilities, we must register the app.
* After we register the app, we can get the “Client ID, Secret key”.

## Register a new application using the Azure portal

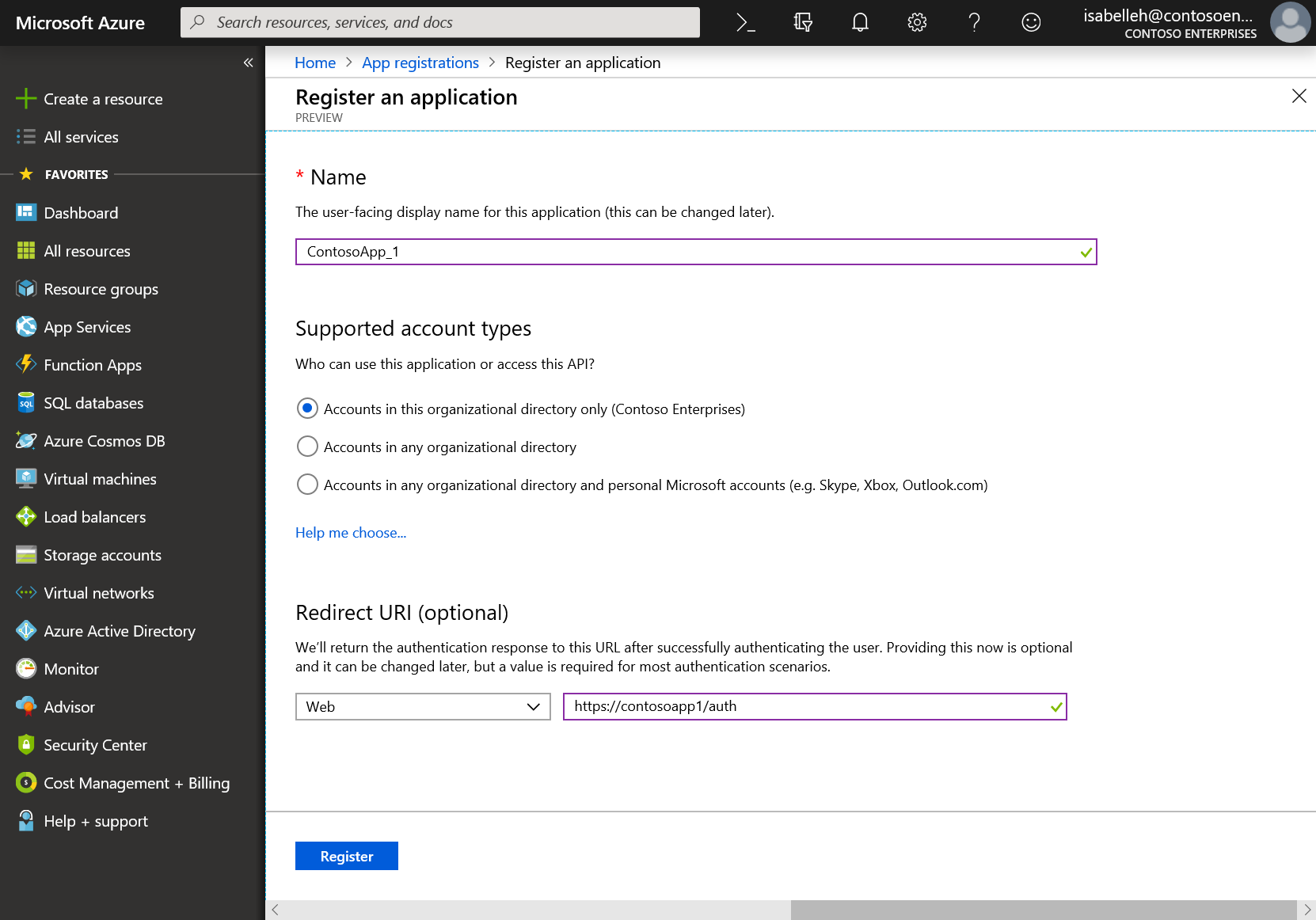
* Sign in to the [Azure portal](https://portal.azure.com/) using a work Microsoft account.
* If your account gives you access to more than one tenant, select your account in the top right corner, and set your portal session to the Azure AD tenant that you want.
* In the left-hand navigation pane, select the Azure Active Directory service.



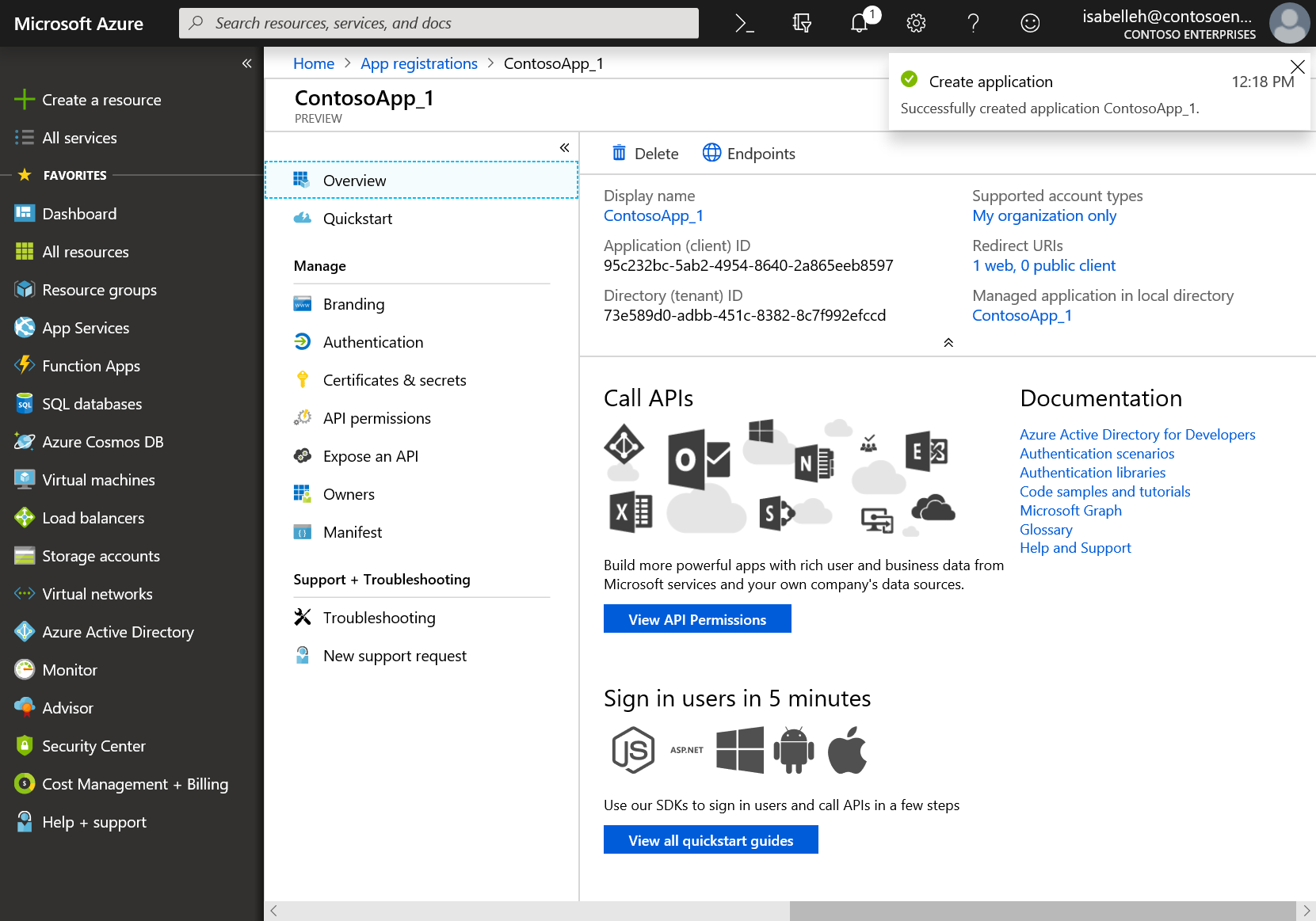
* Then select App registrations > New registration.



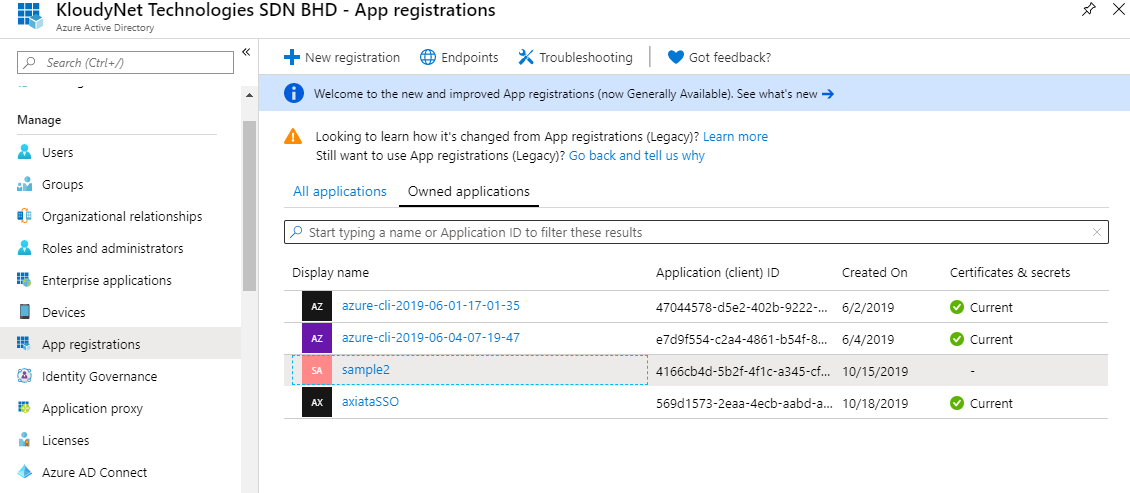
* When we Register an application page appears, enter your application's registration information:
  + **Name** - Enter a meaningful application name that will be displayed to users of the app.
  + **Supported account types** - Select which accounts you would like your application to support.
  + **Redirect URI** (optional) - Select the type of app you're building as Web and then enter the redirect URI (or reply URL) for your application.



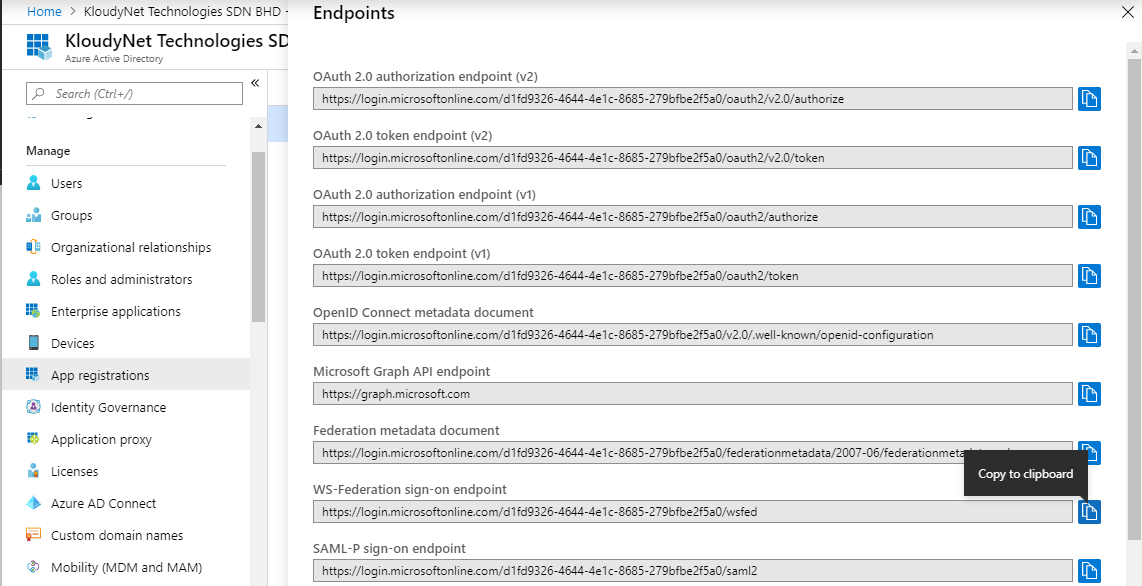
* When finished, select **Register**.



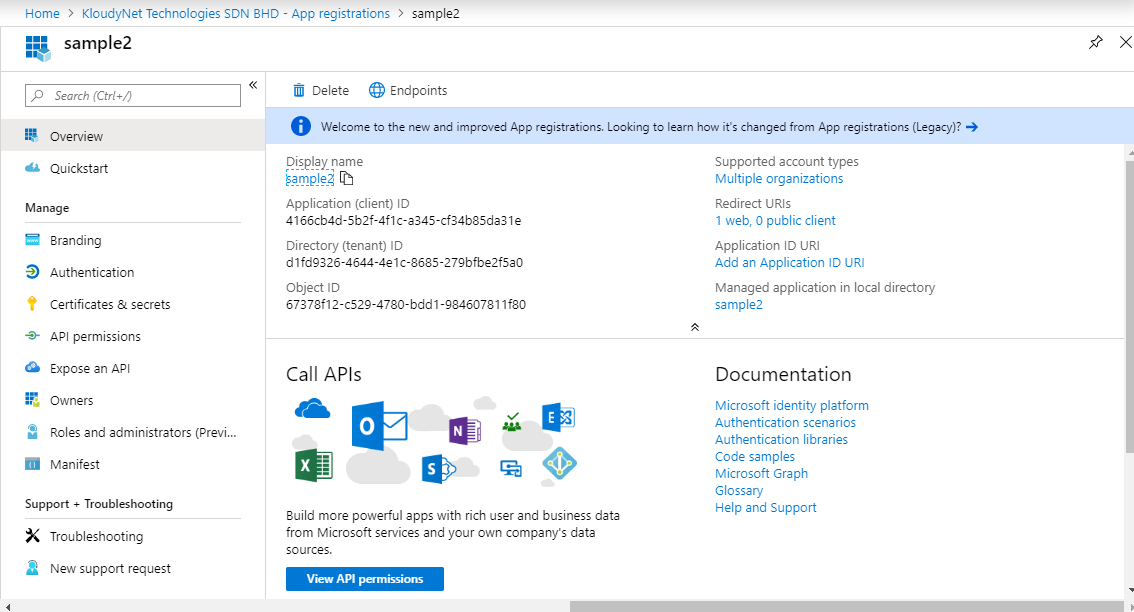
* Now, click on App registration and then click on Endpoints on top bar



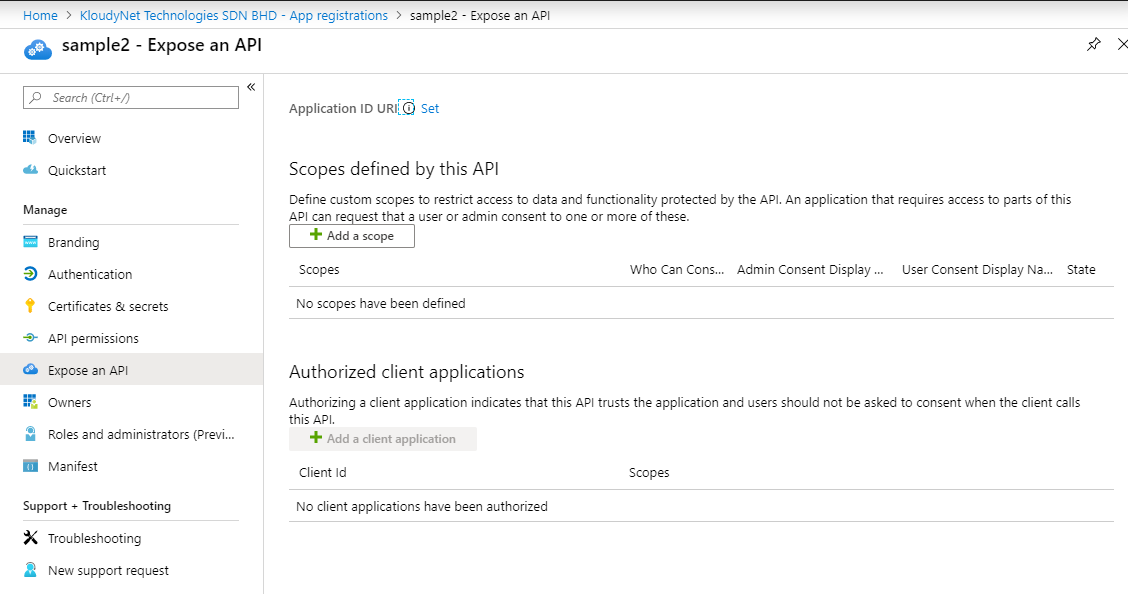
* Note down the endpoint for WS-Federation Sign-On



* On the overview page of the registered application Click on the “Add an Application ID URI”



* Now click on “Add Scope” from within next screen

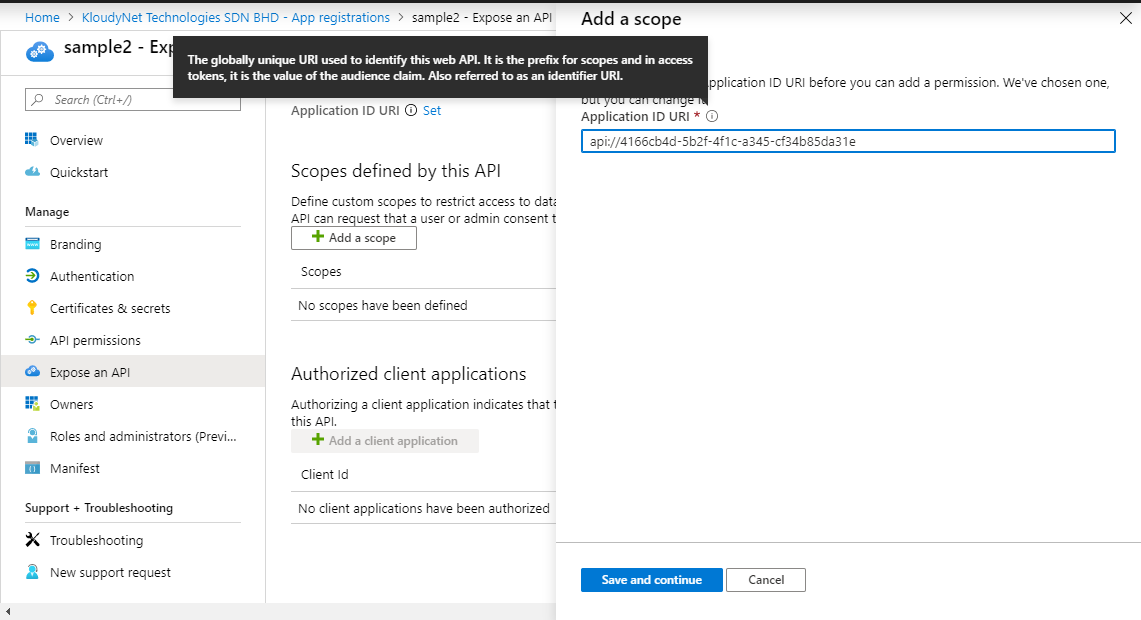


* Note down App ID URI for your application, replace the audienceUris in web config and then click “Save and Continue”

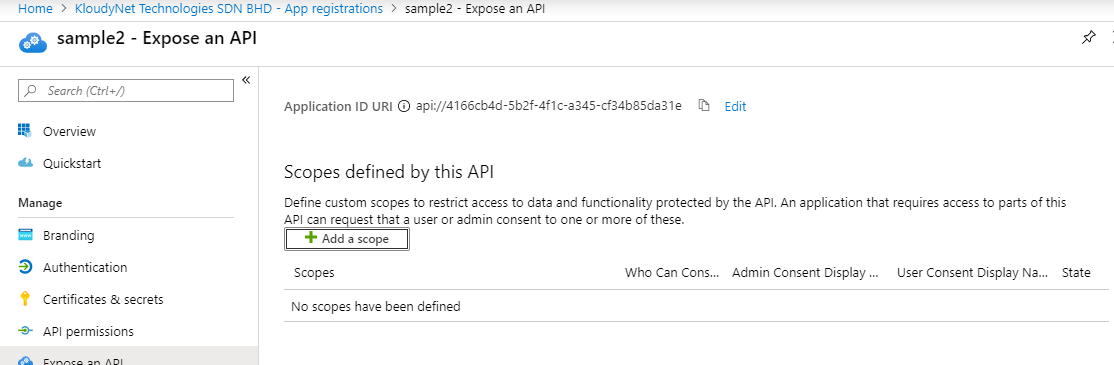
<audienceUris>

<add value="api://569d1573-2eaa-4ecb-aabd-a2d8e6b45cf5" />

</audienceUris>



* You can notice Application UD URI added on the top



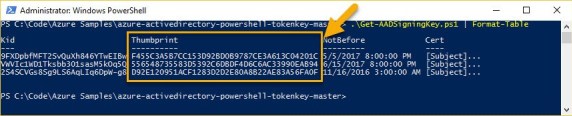
Next step is updating configuration of your application to different values copied above.

* 1. Configuration Windows Identity Foundation (WIF) based Web App

For ASP .NET 4.0 Web Forms application configuration, thumbprints of certificates supported by issuer are typically setup under <trustedIssuers> section of Web.config. We will need to find out thumbprints for AzureAD. Use following steps to retrieve thumbprints:

* Download PowerShell scripts from this repository: https://github.com/AzureAD/azure-activedirectory-powershell-tokenkey
* Run following command from PowerShell, which should give you thumbprints for current active certificates

PS C:\temp\azure-activedirectory-powershell-tokenkey-master> .\Get-AADSigningKey.ps1 | Format-Table



* Once, thumbprints are copied over, modify parameters copied in earlier steps for web.config below to integrate your WIF based application with Azure AD

<trustedIssuers>

<add thumbprint="68F72DC3FA1DBD13A810D8375683A5221DAD884B" name="Demo Issuer" />

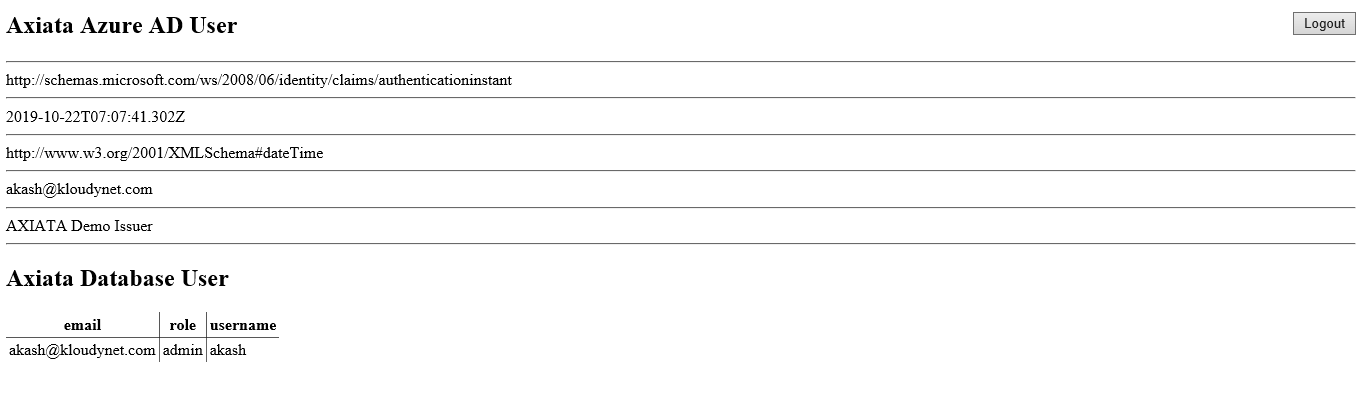
</trustedIssuers>

# Run the Application

Clean the solution, rebuild the solution, and run it.

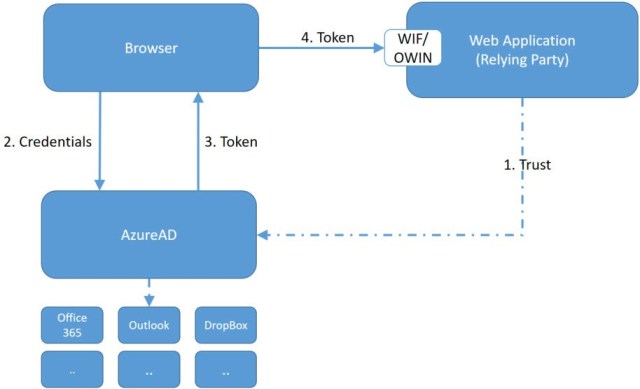
Click the sign-in link on the homepage of the application to sign in. On the Azure AD sign-in page, enter the name and password of a user account that is in your Azure AD tenant.





# Authentication flow using WS Federation Authentication

The most basic sign-in flow contains the following steps - each of them is described in detail below diagram



# Appendix

# Reference URLS

<https://docs.microsoft.com/en-us/dotnet/framework/security/how-to-build-claims-aware-aspnet-web-forms-app-using-wif>

<https://github.com/AzureAD/azure-activedirectory-powershell-tokenkey>