**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: |  |  |  |  |
| Approved By: |  |  |  |  |

Contents

[1. Introduction 5](#_Toc22314988)

[1.1. Purpose 5](#_Toc22314989)

[1.2. Scope 5](#_Toc22314990)

[1.3. Project Vision and Scope Summary 5](#_Toc22314991)

[1.4. Intended Audience 6](#_Toc22314992)

[1.5. Acronyms and Definitions 7](#_Toc22314993)

[2. Design Assumption 7](#_Toc22314994)

[3. Integrate Azure AD into existing web application using OpenID Connect 8](#_Toc22314995)

[3.1. Modification of an application environment that can support Azure AD Authentication 8](#_Toc22314996)

[3.1.1. SSL Enabling 8](#_Toc22314997)

[3.1.2. Installing required NuGet packages 9](#_Toc22314998)

[3.1.3. Configuration in Web.config 9](#_Toc22314999)

[3.1.4. Adding OWIN Startup class 10](#_Toc22315000)

[3.1.5. Adding Startup Auth class 10](#_Toc22315001)

[3.1.6. Modifying AccountController to the Controllers folder 12](#_Toc22315002)

[3.1.7. Adding to HomeController 13](#_Toc22315003)

[3.1.8. Adding to Views 14](#_Toc22315004)

[3.2. Configuration of Azure AD to register the Application 15](#_Toc22315005)

[3.2.1. Register a new application using the Azure portal 15](#_Toc22315006)

[4. Run the Application 19](#_Toc22315007)

[5. Authentication flow using OpenID Connect 20](#_Toc22315008)

[6. Appendix 21](#_Toc22315009)

[6.1. Reference URLS 21](#_Toc22315010)

# Introduction

## Purpose

The purpose of this document is to outline requirement integration ASP.Net openid authentication with Active Directory (Azure AD)

## Scope

The scope of this document is to detail out the complete design and architecture for the ASP.Net MVC 4.7 OpenID authentication with Active Directory (Azure AD).

This document covers how to integrate existing .Net MVC web application that uses OpenID Connect to sign in users from a single Azure Active Directory tenant, using the ASP.Net OpenID Connect OWIN middleware. 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 OpenID Connect. 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 OpenID Connect
  + **Development**
* Steps for configuring single sign-on for ASP.NET MVC 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 |

# 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 4.7 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 OpenID Connect

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

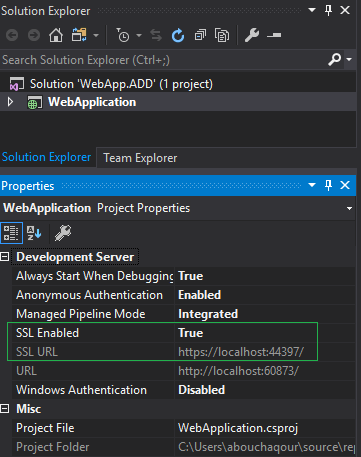
1. Modification of an application environment that can support Azure AD Authentication
2. Configuration of Azure AD to register the Application

## 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

* The Health 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”.



## Installing required NuGet packages

Install all necessary libraries to make our application support Azure AD Authentication.

* To install all necessary libraries open “Package Manager Console” to open it, navigate to  
  Tools > NuGet Package Manager > Package Manager Console
* Then install the following packages one by one (copy, paste, and hit enter).
* Install-Package Microsoft.Owin
* Install-Package Microsoft.Owin.Security.OpenIdConnect
* Install-Package Microsoft.Owin.Security.Cookies
* Install-Package Microsoft.Owin.Host.SystemWeb
* Install-Package Microsoft.IdentityModel.Protocol.Extensions
* Install-Package System.IdentityModel.Tokens.Jwt

## Configuration in Web.config

Open your “Web.config” xml file and then copy the xml lines below and paste them inside <appSettings> tag which lives inside <configuration> tag

<appSettings>

<add key="ida:ClientId" value="" />

<add key="ida:Tenant" value=" " />

<add key="ida:AADInstance" value="https://login.microsoftonline.com/{0}" />

<add key="ida:PostLogoutRedirectUri" value=<https://localhost:44369/>

</appSettings>

Notice, that the “ida:ClientId” and “ida:Tenant” have empty values because we don’t know their values yet. We will know their values when we register our MVC Web Application in Azure AD.

* “ida:ClientId” represents the Application ID
* “ida:Tenant” represents your domain name on Azure AD
* "ida:AADInstance" represents login URL that allows accessing Azure AD for Authentication
* "ida:PostLogoutRedirectUri" represents your local URL

## Adding OWIN Startup class

* 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
* 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
* ConfigureAuth(app) method will be unrecognized because you don’t have a definition of it yet, its definition will be implemented in “Startup.Auth.cs” file.

The code below illustrates how the Startup class should be implemented

using Owin;

namespace SSO

{

public partial class Startup

{

public void Configuration(IAppBuilder app)

{

ConfigureAuth(app);

}

}

}

## Adding Startup Auth class

* In the App\_Start folder, create a class Startup.Auth.cs.
* When you create Startup.Auth.cs file visual studio will concatenate the name of the “App\_Start” folder with the name of your project to form the namespace.
* Make sure to remove the “.App\_start” from your namespace.
* The definition changes from public class Startup to public partial class Startup.
* In Startup.Auth.cs resolve missing references by adding using statements for Owin, Microsoft.Owin.Security, Microsoft.Owin.Security.Cookies, Microsoft.Owin.Security.OpenIdConnect, System.Configuration, and System.Globalization.
* It should look like “namespace <WebAppName>” as illustrated in the code below.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using Microsoft.Owin.Security;

using Microsoft.Owin.Security.Cookies;

using Microsoft.Owin.Security.OpenIdConnect;

using Owin;

using System.Configuration;

using System.Globalization;

using System.Threading.Tasks;

namespace SSO

{

public partial class Startup

{

// Calling the keys values from Web.config file

private static string clientId = ConfigurationManager.AppSettings["ida:ClientId"];// represents the Application ID

private static string tenant = ConfigurationManager.AppSettings["ida:Tenant"];// represents your domain name on Azure AD

private static string aadInstance = ConfigurationManager.AppSettings["ida:AADInstance"];// represents login URL that allows accessing Azure AD for Authentication

* private static string postLogoutRedirectUri = ConfigurationManager.AppSettings["ida:PostLogoutRedirectUri"];// represents your local URL

// Concatenate aadInstance, tenant to form authority value

private string authority = string.Format(CultureInfo.InvariantCulture, aadInstance, tenant);

// ConfigureAuth method

public void ConfigureAuth(IAppBuilder app)

{

app.SetDefaultSignInAsAuthenticationType(CookieAuthenticationDefaults.AuthenticationType);

app.UseCookieAuthentication(new CookieAuthenticationOptions());

app.UseOpenIdConnectAuthentication(

new OpenIdConnectAuthenticationOptions

{

ClientId = clientId,

Authority = authority,

PostLogoutRedirectUri = postLogoutRedirectUri,

Notifications = new OpenIdConnectAuthenticationNotifications

{

AuthenticationFailed = (context) =>

{

context.HandleResponse();

context.OwinContext.Response.Redirect("/Home/Index");

return Task.FromResult(0);

}

}

});

}

// end - ConfigureAuth method

}

}

## Modifying AccountController to the Controllers folder

* In the AccountController to the Controllers folder. The controller AccountController will have two methods SignIn and SingOut.
* Replace the implementation with the contents of the file.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Remoting.Contexts;

using System.Web;

using System.Web.Mvc;

using Microsoft.Owin.Security;

using Microsoft.Owin.Security.Cookies;

using Microsoft.Owin.Security.OpenIdConnect;

using SSO.Models;

namespace SSO.Controllers

{

public class AccountController : Controller

{

// Sends an OpenIDConnect Sign-In Request.

public void SignIn()

{

if (!Request.IsAuthenticated)

{

HttpContext.GetOwinContext()

.Authentication.Challenge(new AuthenticationProperties { RedirectUri = "/" },

OpenIdConnectAuthenticationDefaults.AuthenticationType);

}

}

// Signs the user out and clears the cache of access tokens.

public void SignOut()

{

HttpContext.GetOwinContext().Authentication.SignOut(

OpenIdConnectAuthenticationDefaults.AuthenticationType, CookieAuthenticationDefaults.AuthenticationType);

}

}

}

## Adding to HomeController

* 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.
* After the Azure AD authentication, we check the user across the database as well. If the user present in both AAD and database then only access granted to the application.

using System;

using System.Web;

using System.Linq;

using System.Web.Mvc;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.IdentityModel.Claims;

using System.Security.Claims;

using Microsoft.Owin.Security;

using Microsoft.Owin.Security.Cookies;

using Microsoft.Owin.Security.OpenIdConnect;

using SSO.Models;

namespace SSO.Controllers

{

public class HomeController : Controller

{

public ActionResult Index()

{

if (Request.IsAuthenticated)

{

using (var context = new UserTable())

{

var userDetails = context.users.Where(x => x.email == User.Identity.Name).FirstOrDefault();

if (userDetails != null)

{

System.Security.Claims.Claim displayName = ClaimsPrincipal.Current.FindFirst(ClaimsPrincipal.Current.Identities.First().NameClaimType);

ViewBag.DisplayName = displayName != null ? displayName.Value : string.Empty;

ViewBag.data = userDetails;

ViewBag.Message = "You are logged in!!";

}

else

{

HttpContext.GetOwinContext().Authentication.SignOut(

OpenIdConnectAuthenticationDefaults.AuthenticationType, CookieAuthenticationDefaults.AuthenticationType);

ViewBag.Message = "You cannot logged in!!";

}

}

}

else

{

ViewBag.Message = "You are NOT logged in. Please Log in to Continue";

}

return View();

}

public ActionResult Contact()

{

ViewBag.Message = "You are logged in!!... showing database data...!!!";

using (var context = new UserTable())

{

var userDetails = context.users.Where(x => x.email == User.Identity.Name).FirstOrDefault();

if (userDetails != null)

{

user rec = new user

{

username = userDetails.username,

email = userDetails.email,

role = userDetails.role

};

ViewBag.Message = rec;

}

}

return View();

}

}

}

## Adding to Views

* In the Views --> Shared folder, create a new partial view \_LoginPartial.cshtml

@if (Request.IsAuthenticated)

{

<text>

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

<li>

@Html.ActionLink(User.Identity.Name, "About", "Home", null, new { id = "about" })

</li>

<li>

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

</li>

</ul>

</text>

}

else

{

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

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

</ul>

}

* Open \_Layout partial view and  call your partial view @Html.Partial("\_LoginPartial")
* Effectively, all this will do is add a single line, @Html.Partial("\_LoginPartial"), that lights up the previously added \_LoginPartial view.

@Html.Partial("\_LoginPartial")

## Configuration of Azure AD to register the Application

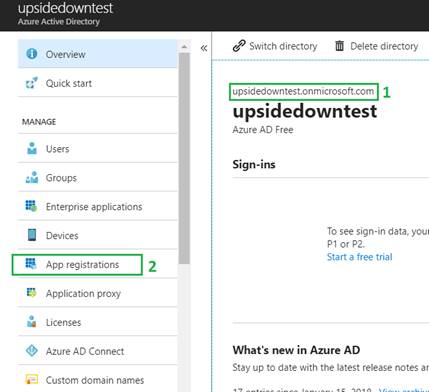
Before your web app can use Azure AD as the identity back-end it needs to be registered in Azure AD. This is done both to ensure that not every random app out there can hook into an AAD tenant, and to configure some of the mechanics needed for it to actually work with the necessary redirects.

We need to register an application for the following reasons:

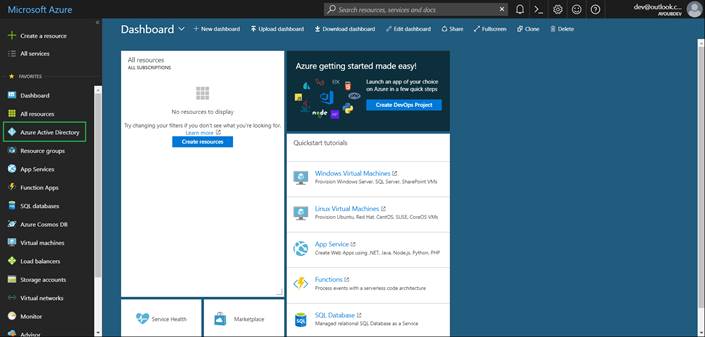
* It is used to integrate the application and service with Azure AD.
* Using Azure App, we can generate the token to authenticate the application.
* If we want to use the Azure AD capabilities, we must register the app.
* After we register the app, we can get the “Client ID, Secret key”.

## Register a new application using the Azure portal

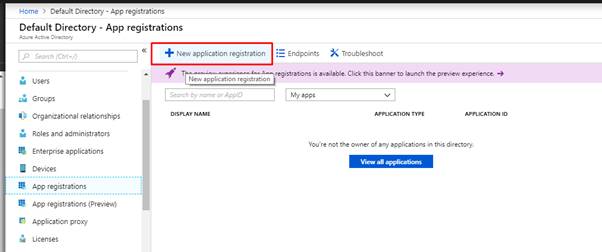
* Sign in to the [Azure portal](https://portal.azure.com/) using a work Microsoft account.
* If your account gives you access to more than one tenant, select your account in the top right corner, and set your portal session to the Azure AD tenant that you want.
* Copy the tenant URL and save it in a notepad because it represents the value of “ida:Tenant” key in the Web.config xml file. For example, #1, it is “upsidedowntest.onmicrosoft.com” refer below sample screenshot.



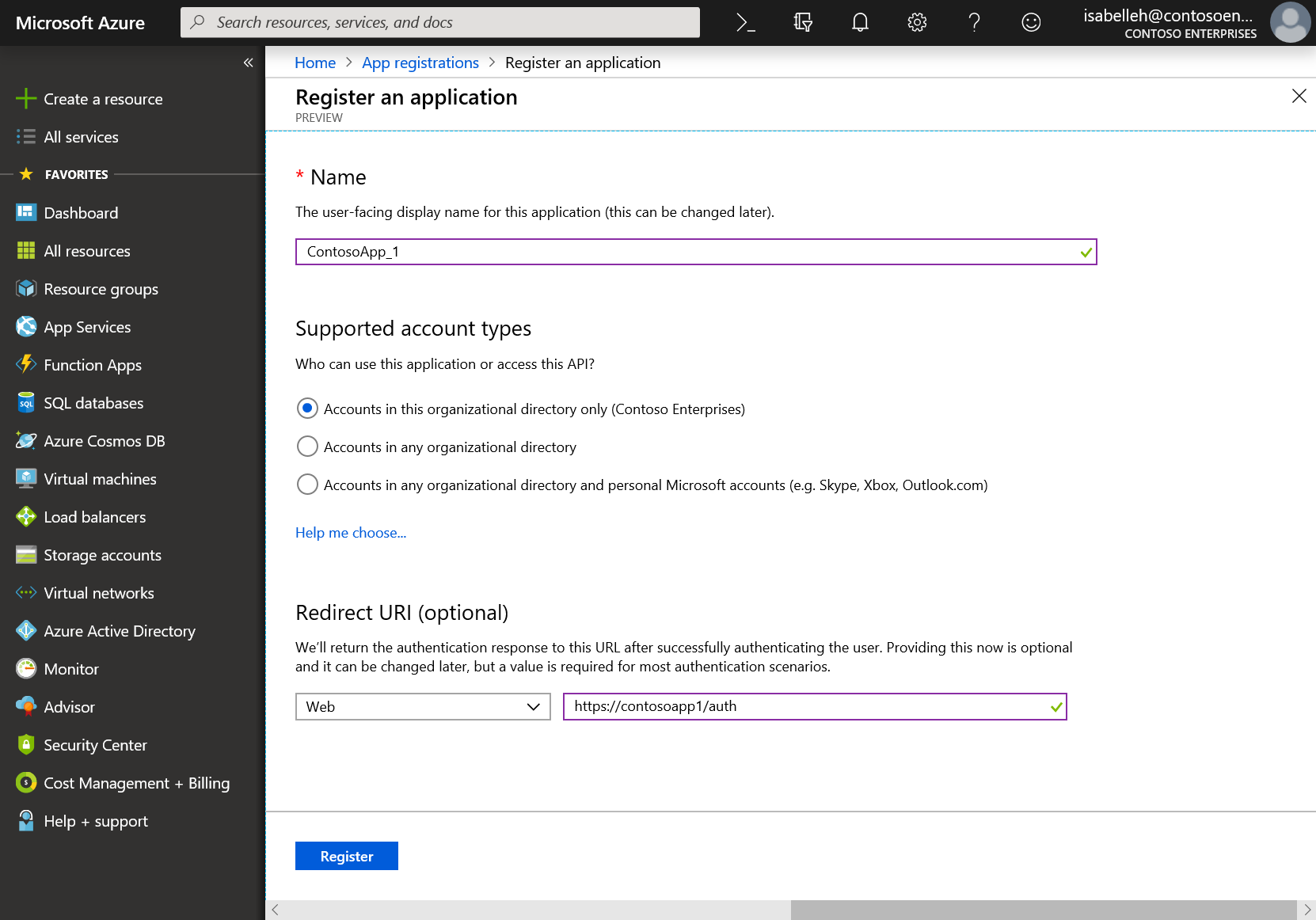
* In the left-hand navigation pane, select the Azure Active Directory service.



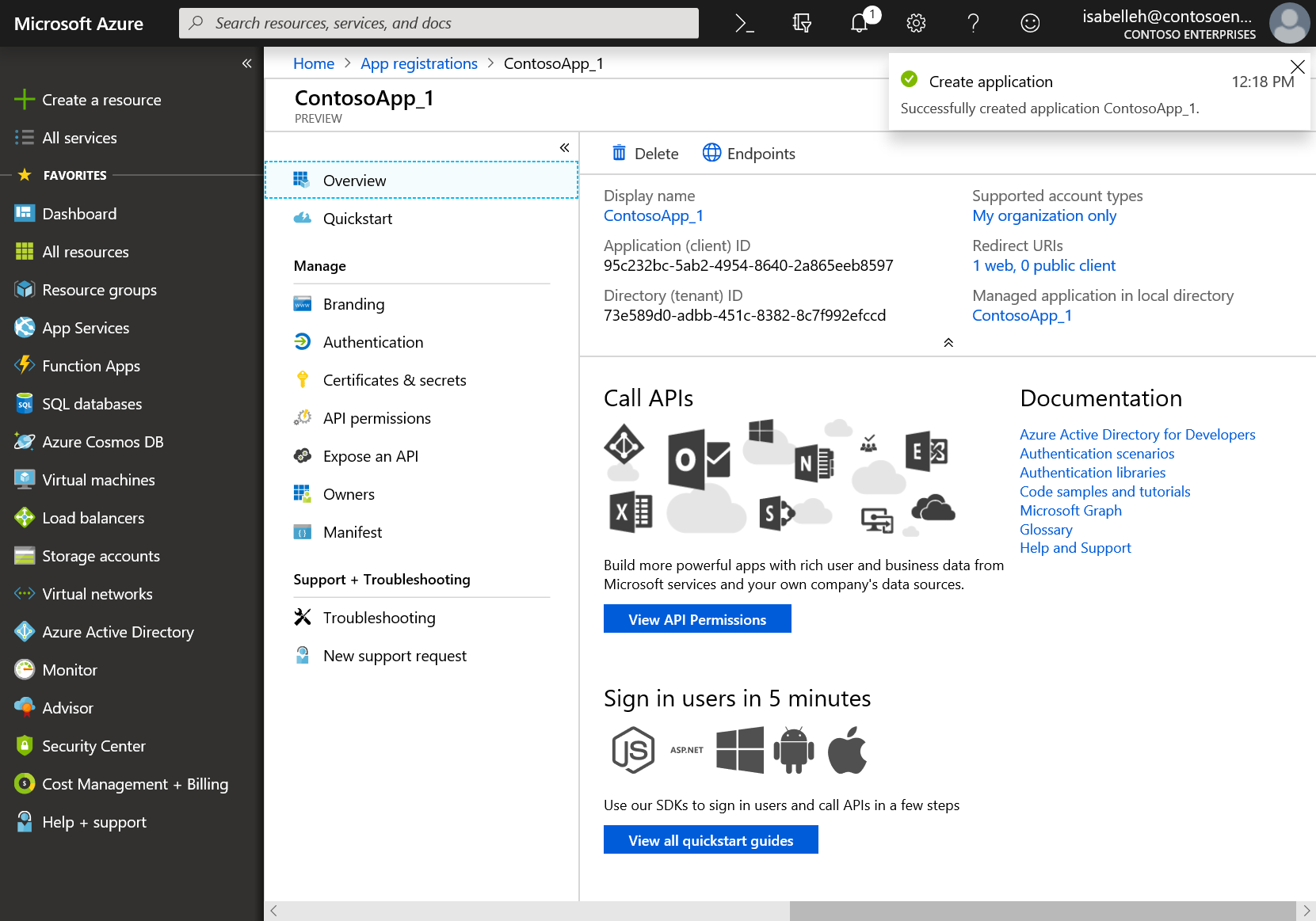
* Then select App registrations > New registration.



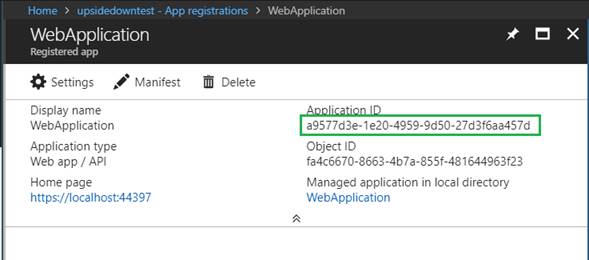
* When we Register an application page appears, enter your application's registration information:
  + **Name** - Enter a meaningful application name that will be displayed to users of the app.
  + **Supported account types** - Select which accounts you would like your application to support.
  + **Redirect 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**.



* After creation you will see the screen below that has information about your application. Copy the Application ID and save it in notepad because it represents the value of “ida:ClientId” key in the web.config xml file.
* After you copy and save your Application ID and domain name (in our example is “upsidedowntest.onmicrosoft.com”) it is time to go back to our MVC Web Application and paste the values copied from Azure AD environment in the web.config xml file.



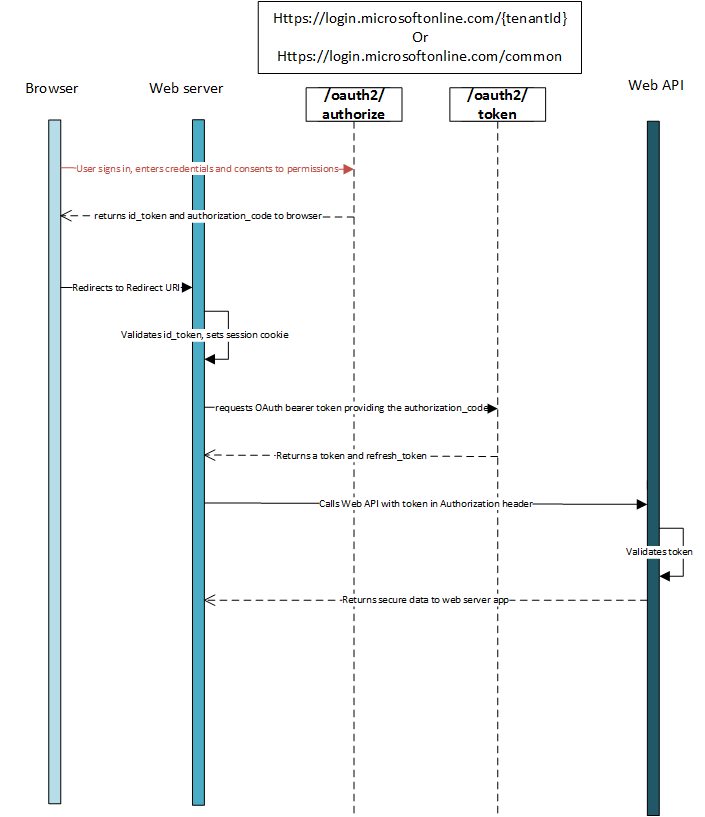
# Run the Application

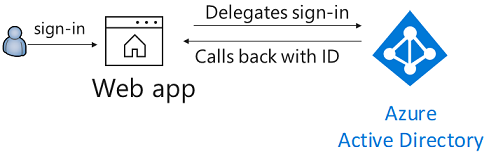
Clean the solution, rebuild the solution, and run it.

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 OpenID Connect

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/azure/active-directory/develop/v1-protocols-openid-connect-code>

<https://docs.microsoft.com/en-us/graph/auth-register-app-v2>