# Exercise 2: Implementing authentication

This exercise will demonstrate the different account types that are used within the Microsoft identity platform.

**Note**: This exercise demonstrates signing into a web application using two different accounts. These two accounts will come from two organizations, one of them being the organization where the Azure AD application is registered. Therefore, in order to complete the exercise, you’ll need access to two user accounts in different Azure AD directories.

## Task 1: Create application that only allows single organization sign in

In this task, you will register an application in the Azure portal that allows users from the current organization to sign in. ### Register a single-tenant Azure AD application

1. From the Azure portal [https://azure.portal.com](https://azure.portal.com/), navigate to **Azure Active Directory**.
2. Select **Manage > App registrations** in the left-hand navigation.
3. On the **App registrations** page, select **New registration**.
4. On the **Register an application** page, set the values as follows:
   * **Name**: Hello ASPNET Core Identity 01
   * **Supported account types**: Accounts in this organizational directory only (Single tenant)

* Register an application for Hello ASPNET Core Identity 01
* Register an application for Hello ASPNET Core Identity 01

1. Select **Register** to create the application.
2. On the **Hello ASPNET Core Identity 01** page, copy the values **Application (client) ID** and **Directory (tenant) ID**; you’ll need these values later in this exercise.

* Hello ASPNET Core Identity 01 overview page with tenant id and client id.
* Hello ASPNET Core Identity 01 overview page with tenant id and client id.

1. On the **Hello ASPNET Core Identity 01** page, select the **Add a Redirect URI** link under the **Redirect URIs**.
2. Locate the section **Platform configurations** and select **+Add a platform**.
3. Select **Web** and add the following URLs:
   * **https://localhost:3007**
4. Select **Configure**
5. Locate the section **Redirect URIs** under **Web**, select **Add URI** and add the following URL:
   * **https://localhost:3007/signin-oidc**
6. Locate the section **Logout URL** and add the following **Logout URL**: **https://localhost:3007/signout-oidc**
7. Locate the section **Implicit grant** and select both **Access tokens** and **ID tokens**. This tells Azure AD to return these tokens the authenticated user if requested.
8. Select **Save** when finished setting these values.

Hello ASPNET Core Identity 01 Authentication settings page.

Hello ASPNET Core Identity 01 Authentication settings page.

## Task 2: Create a single organization ASP.NET core web application

In this first application, you’ll create an ASP.NET Core web application that allows users from the current organization to sign in and display their information. 1. Open your command prompt, navigate to a directory where you want to save your work, create a new folder, and change directory into that folder. For example:

```powershell  
Cd c:/LabFiles  
md SingleOrg  
cd SingleOrg  
```

1. Execute the following command to create a new ASP.NET Core MVC web application:

* dotnet new mvc --auth SingleOrg

1. Open the folder in Visual Studio code by executing from the project directory: code .

### Configure the web application with the Azure AD application you created

1. Locate and open the **./appsettings.json** file in the ASP.NET Core project.
2. Set the **AzureAd.Domain** property to the domain of your Azure AD tenant where you created the Azure AD application (*for example: contoso.onmicrosoft.com*).
3. Set the **AzureAd.TenantId** property to the **Directory (tenant) ID** you copied when creating the Azure AD application in the previous step.
4. Set the **AzureAd.ClientId** property to the **Application (client) ID** you copied when creating the Azure AD application in the previous step.

### Update the web application’s launch configuration

1. Locate and open the **./Properties/launchSettings.json** file in the ASP.NET Core project.
2. Set the **iisSettings.iisExpress.applicationUrl** property to **https://localhost:3007**.
3. Set the **iisSettings.iisExpress.sslPort** property to **3007**.

### Update the user experience

1. Finally, update the user experience of the web application to display all the claims in the OpenID Connect ID token.
2. Locate and open the **./Views/Home/Index.cshtml** file.
3. Add the following code to the end of the file:

* @if (User.Identity.IsAuthenticated)  
  {  
  <div>  
   <table cellpadding="2" cellspacing="2">  
   <tr>  
   <th>Claim</th>  
   <th>Value</th>  
   </tr>  
   @foreach (var claim in User.Claims)  
   {  
   <tr>  
   <td>@claim.Type</td>  
   <td>@claim.Value</td>  
   </tr>  
   }  
   </table>  
  </div>  
  }

## Task 3: Build and test the single organization web app

1. Execute the following command in a command prompt to compile and run the application:

* dotnet build  
  dotnet run

1. Open a browser and navigate to the url **https://localhost:5001**. The web application will redirect you to the Azure AD sign in page.
2. Sign in using a Work and School account from your Azure AD directory. Azure AD will redirect you back to the web application. Notice some of the details from the claims included in the ID token.

* Hello ASPNET Core Identity 01 application running displaying authenticated user informatoin.
* Hello ASPNET Core Identity 01 application running displaying authenticated user informatoin.

1. Take special note of the **tenantid** and **upn** claim. These claims indicate the ID of the Azure AD directory and ID of the user that signed in. Make a note of these values to compare them to other options in a minute.
2. Now try logging in as a user from a different organization. Select the **Sign out** link in the top left. Wait for Azure AD and the web application signs out the current user. When the web application reloads, repeat the sign in process, except this time try signing in as a user from a different organization or use a Microsoft Account.
3. Notice Azure AD will reject the user’s sign in, explaining that the user’s account doesn’t exist in the current tenant.
4. Stop the web server by pressing **CTRL+C** in the command prompt.

## Task 4: Create application that allows any organization’s users to sign in

In this task, you will register an application in the Azure portal that allows users from any organization or Microsoft Accounts to sign in. ### Register a multi-tenant Azure AD application

1. From the Azure portal [https://azure.portal.com](https://azure.portal.com/), navigate to **Azure Active Directory**.
2. Select **Manage > App registrations** in the left-hand navigation.
3. On the **App registrations** page, select **New registration**.
4. On the **Register an application** page, set the values as follows:
   * **Name:** Hello ASPNET Core Identity 02
   * **Supported account types**: Accounts in any organizational directory only (Any Azure AD directory - Multitenant)

* Register an application page with Accounts in any organizational directory selected.
* Register an application page with Accounts in any organizational directory selected.

1. Select **Register** to create the application.
2. On the **Hello ASPNET Core Identity 02** page, copy the values **Application (client) ID** and **Directory (tenant) ID**; you’ll need these values later in this exercise.
3. On the **Hello ASPNET Core Identity 01** page, select the **Add a Redirect URI** link under the **Redirect URIs**.
4. Locate the section **Redirect URIs** and add the following two URLs:
   * **https://localhost:3007**
   * **https://localhost:3007/signin-oidc**
5. Locate the section **Advanced settings** and add the following **Logout URL**: **https://localhost:3007/signout-oidc**
6. Locate the section **Implicit grant** and select both **Access tokens** and **ID tokens**. This tells Azure AD to return these tokens the authenticated user if requested.
7. Select **Save** when finished setting these values.

## Task 5: Create a multiple organization ASP.NET core web application

In this second application, you’ll create an ASP.NET Core web application that allows users from any organization or Microsoft Accounts to sign in and display their information.

1. Open your command prompt, navigate to a directory where you want to save your work, create a new folder, and change directory into that folder. For example:

* Cd c:/LabFiles  
  md MultiOrg  
  cd MultiOrg

1. Execute the following command to create a new ASP.NET Core MVC web application:

* dotnet new mvc --auth MultiOrg

1. Open the folder in Visual Studio code by executing from the project directory: code .

### Configure the web application with the Azure AD application you created

1. Locate and open the **./appsettings.json** file in the ASP.NET Core project.
2. Set the **AzureAd.ClientId** property to the **Application (client) ID** you copied when creating the Azure AD application in the previous step.

### Update the web application’s launch configuration

1. Locate and open the **./Properties/launchSettings.json** file in the ASP.NET Core project.
2. Set the **iisSettings.iisExpress.applicationUrl** property to **https://localhost:3007**.
3. Set the **iisSettings.iisExpress.sslPort** property to **3007**.

### Update the user experience

1. Finally, update the user experience of the web application to display all the claims in the OpenID Connect ID token.
2. Locate and open the **./Views/Home/Index.cshtml** file.
3. Add the following code to the end of the file:

* @if (User.Identity.IsAuthenticated)  
  {  
  <div>  
   <table cellpadding="2" cellspacing="2">  
   <tr>  
   <th>Claim</th>  
   <th>Value</th>  
   </tr>  
   @foreach (var claim in User.Claims)  
   {  
   <tr>  
   <td>@claim.Type</td>  
   <td>@claim.Value</td>  
   </tr>  
   }  
   </table>  
  </div>  
  }

## Task 6: Build and test the multiple organization web app

1. Execute the following command in a command prompt to compile and run the application:

* dotnet build  
  dotnet run

1. Open a browser and navigate to the url **https://localhost:5001**. The web application will redirect you to the Azure AD sign in page.
2. Sign in using a Work and School account from your Azure AD directory. Azure AD will redirect you back to the web application. Notice some of the details from the claims included in the ID token.

* ASP.NET applicaiton running in browser.
* ASP.NET applicaiton running in browser.

1. Take special note of the **tenantid** and **upn** claim. These indicate the ID of the Azure AD directory and ID of the user that signed in. Make a note of these values to compare them to other options in a minute.
2. Now try logging in as a user from a different organization. Select the **Sign out** link in the top left. Wait for Azure AD and the web application signs out the current user.
3. This time, the user is prompted to first trust the application:

* Permission requested accept dialog for external user.
* Permission requested accept dialog for external user.

1. Select **Accept**.
2. Notice the web application’s page loads with different claims, specifically for the **tenantid** and **upn** claim. This indicates the user is not from the current directory where the Azure AD application is registered:

* External organization user logged into web application and displaying user’s information.
* External organization user logged into web application and displaying user’s information.

1. Stop the web server by going back to the running project in Visual Studio Code. From the Terminal, press **CTRL+C** in the command prompt.

## Review

In this exercise, you learned how to create different types of Azure AD applications and use an ASP.NET Core application to support the different sign in options that support different types of accounts.