Getting Started with Azure AD B2C

{IEF - custom policies}

**Lab**

Terms of Use

© 2018 Microsoft Corporation. All rights reserved.

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see **Microsoft Copyright Permissions** at <http://www.microsoft.com/permission>

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

The Microsoft company name and Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

**This document reflects current views and assumptions as of the date of development and is subject to change.  Actual and future results and trends may differ materially from any forward-looking statements.  Microsoft assumes no responsibility for errors or omissions in the materials.**

**THIS DOCUMENT IS FOR INFORMATIONAL AND TRAINING PURPOSES ONLY AND IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT.**

# Overview

Azure Active Directory (Azure AD) B2C is a white-label cloud identity service (Idaas)for your customer-facing web & mobile apps. It is highly-available, secure and scales to millions of customer identities. Customers can use their social accounts (Facebook, Google, Microsoft account, etc.) or create new credentials to access your apps. In this lab, you will create your first web application that will allow that allows customers sign-up & sign-in to your web app using email addresses and passwords. You will also configure custom policies by adding Facebook as an Identity Provider (IdP), connect a third-party RESTful API service, and upload custom policies in the Azure AD B2C admin portal.

# Objectives

Learn how to:

* Create a new Azure AD B2C tenant
* Register a test web application and create sign-up & sign-in experiences
* Download starter pack and modify policies
* Enable Facebook as an Identity Provider with Azure AD B2C
* Integrate Azure AD B2C with a RESTful API
* Policy management with Identity Experience Framework

# Prerequisites

* Windows 10
* Azure Active Directory [with an associated Azure Subscription](https://docs.microsoft.com/en-us/azure/active-directory/fundamentals/active-directory-how-subscriptions-associated-directory)
* Minimum [Contributor](https://docs.microsoft.com/en-us/azure/role-based-access-control/built-in-roles) role for Azure services linked to your subscription.
* Complete the steps in [Set up sign-up and sign-in with a Facebook account](https://docs.microsoft.com/en-us/azure/active-directory-b2c/identity-provider-facebook) to configure a Facebook application. Although a Facebook application is not required for using custom policies, it's used in this walkthrough to demonstrate enabling social login in a custom policy.
* [Visual Studio Code](https://code.visualstudio.com/download) and the [Azure AD B2C extension](https://marketplace.visualstudio.com/items?itemName=AzureADB2CTools.aadb2c)

This is a screenshot of Azure AD B2C extension that is located in Visual Studio Code.

Navigate to the Extension tab and search Azure AD B2C. It is the only one that is available.

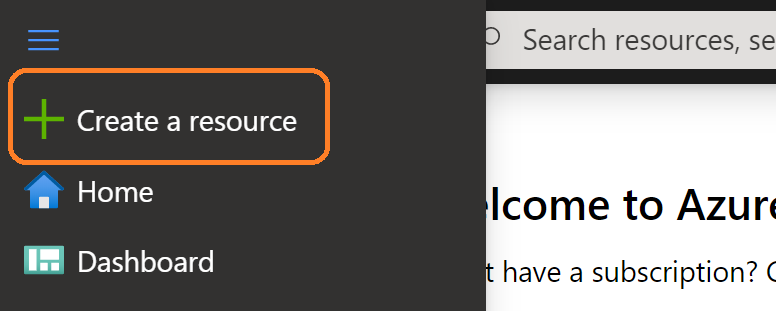
|  |
| --- |
| **IMPORTANT:** Note that you may use other XML editors such as NotePad++ but this will be more difficult which switching between multiple XML files during configuration. We recommend using Visual Studio Code for ease of use. |

# Exercise 1: Create an Azure AD B2C

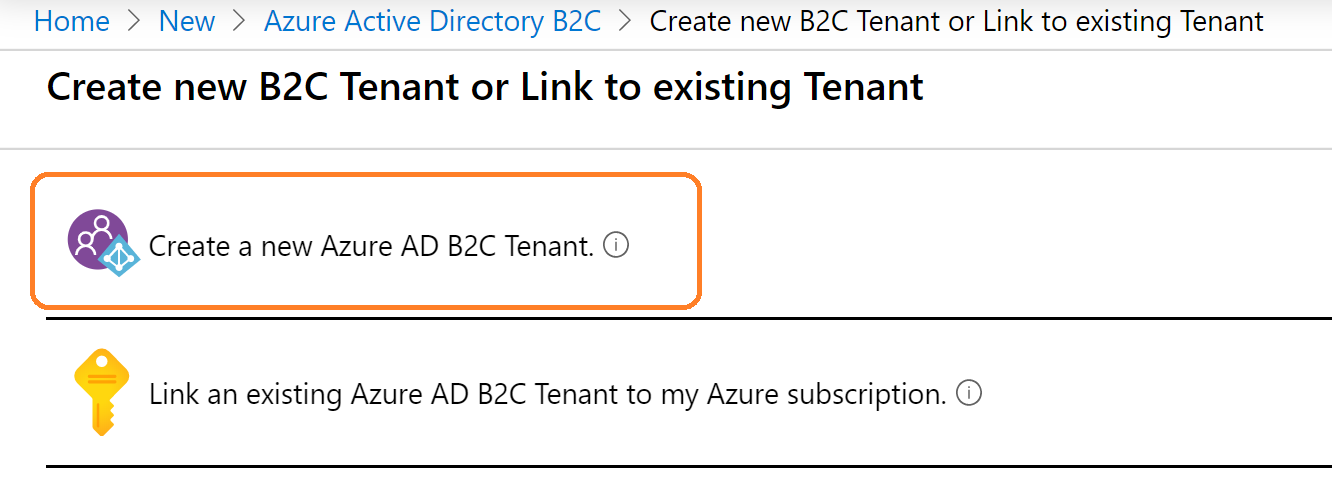
**IMPORTANT:** Note that you may **skip this exercise if** you already have an Azure AD B2C tenant. Simply switch into this tenant from the top-right corner.

Each exercise consists of a scenario and learning objectives, the scenario describes the purpose of the exercices, while the objectives are listed and have bullet points.

1. Open the portal <https://portal.azure.com>
   1. Enter your Hotmail or Outlook credentials
2. In the left-hand navigation bar, Select and then Click on **Create a resource**



1. Search **B2C** and select **Azure Active Directory B2C**. Click **Create**.
2. Click the **Create a new Azure AD B2C Tenant.** Button.



1. Enter an **Organization name** of “Contoso” and an **Initial domain name** that’s globally unique. For example, you may want to include your name, such as **“ContosoFrankDoe”**. This will be the subdomain of the tenant. Which we also called the tenant name, such as **“ContosoFrankDoe.onmicrosoft.com”**, and will be used to configure the application later. Click **Create**.

A screenshot of a cell phone

Description automatically generated

1. It will take a minute or so for the directory to be created. You can track progress using the notification dropdown in the top right corner of the portal.

A screen shot of a smart phone

Description automatically generated

1. Once the directory has been created,  **Refresh** your browser to reflect all the changes

**IMPORTANT:** Note that if you receive the following error, your Azure AD tenant does not have an Azure Subscription associated with it. Stop and [follow these steps before proceeding](https://docs.microsoft.com/en-us/azure/active-directory/fundamentals/active-directory-how-subscriptions-associated-directory).

“You are currently signed into the 'Contoso' directory which does not have any subscriptions. You have other directories you can switch to or you can sign up for a new subscription.”

# Exercise 2: Link a subscription

**IMPORTANT:** Note that you may **skip this exercise** **if** you already have an Azure AD B2C tenant with *an Azure subscription linked to it.*

Each exercise consists of a scenario and learning objectives, the scenario describes the purpose of the exercices, while the objectives are listed and have bullet points.

### Scenario

In this exercise, you will link a subscription to the newly created tenant.

1. If you have not already done so, make sure you  **Refresh** your browser to reflect all the changes.
2. As previously navigate, follow the same path:

A screenshot of a cell phone

Description automatically generated

1. Click on your identity in the upper right and make sure you are in the same tenant that you used to create your Select **Link an existing Azure AD B2C Tenant to my Azure Subscription**.

A screenshot of a cell phone

Description automatically generated

1. From the drop-down menus, select your tenant, **create a new resource group** if you do not already have one, and select the resource group that you consider is closer to you. Click **create**.

A screenshot of a cell phone

Description automatically generated

1. It will take a minute or so for the resource group to be created. You can track progress using the notification dropdown in the top right corner of the portal.
2. After the resource group is created, **Refresh** your browser or **sign out from the portal and sign in again**.
3. Click your name, on the top right corner and Switch directory

A screenshot of a cell phone

Description automatically generated

1. You will see your Default Directory and your newly created Azure AD B2C directory, in this case called **Contoso**. These two directories are completely independent from each other. Select your Azure AD B2C directory.

A screenshot of a cell phone

Description automatically generated

**IMPORTANT:** note that you must be in the Azure AD B2C directory in order to have access to Azure AD B2C functionality and user directory

1. To make it easier to access your new Azure AD B2C tenant, select **All services**.

A screenshot of a cell phone

Description automatically generated

1. Enter **“b2c”** in the search bar. Then, click **on the star symbol** to add Azure AD B2C to your favorites.

A screenshot of a social media post

Description automatically generated

1. Look for **Azure AD B2C** blade under your Favorites menu on the left-hand side of the portal and click on it. For easy access, you can **drag it to the top** of the list.

A screenshot of a cell phone

Description automatically generated

# Exercise 3: Register your web app

|  |
| --- |
| **IMPORTANT:** Note that you may **skip this exercise** **if** you already have an Azure AD B2C tenant with a *functional web app*. |

Each exercise consists of a scenario and learning objectives, the scenario describes the purpose of the exercices, while the objectives are listed and have bullet points.

### Scenario

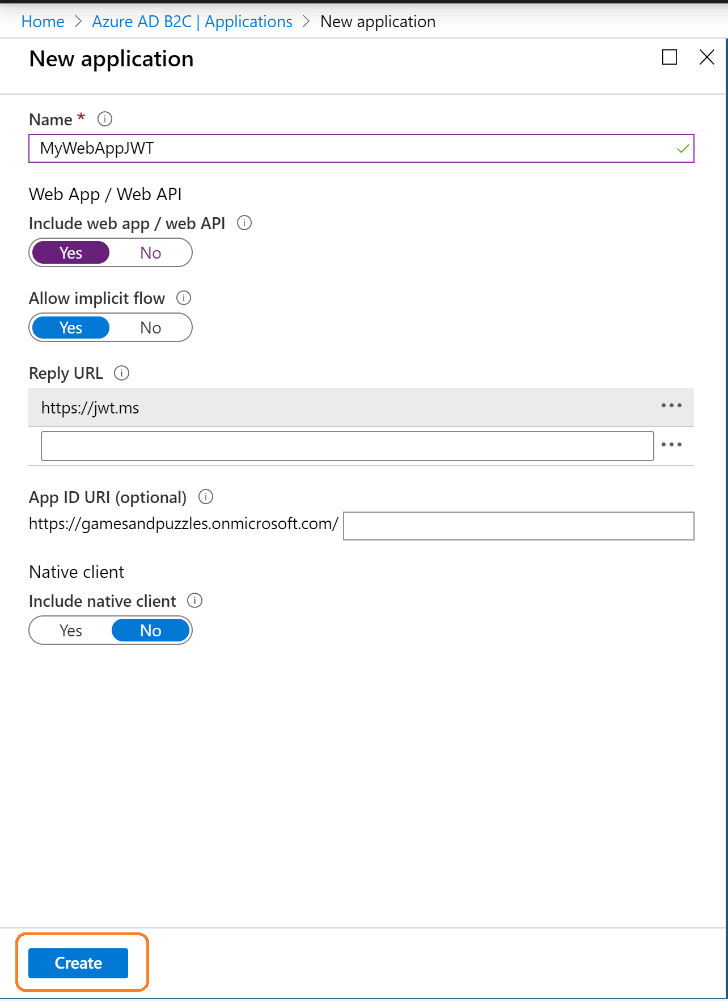
*In this exercise, you will create a B2C application that will represent your real-world application when the applications make a request to Azure AD B2C. The first thing we need to do is register an application. This is necessary for Azure AD B2C to identify which application is making the sign-in request.*

1. Open the **Azure AD B2C** menu by searching for it under **More Services**
2. Select **Applications** tab and then select **Add**

A screenshot of a cell phone

Description automatically generated

1. Enter **“MyWebAppJWT”** as the **Name** of the application.
2. Select **Yes** for **Include web app/web API**.
3. In the **Reply URL**, type in “**https://jwt.ms”**. This is where Azure AD B2C will send tokens back after successful sign up or sign in. Because we do not have an application built yet, we will use “**https://jwt.ms”** temporarily for testing.
4. Leave the rest of the settings to their default values and select **Create**.



# Exercise 4: Add signing and encryption keys

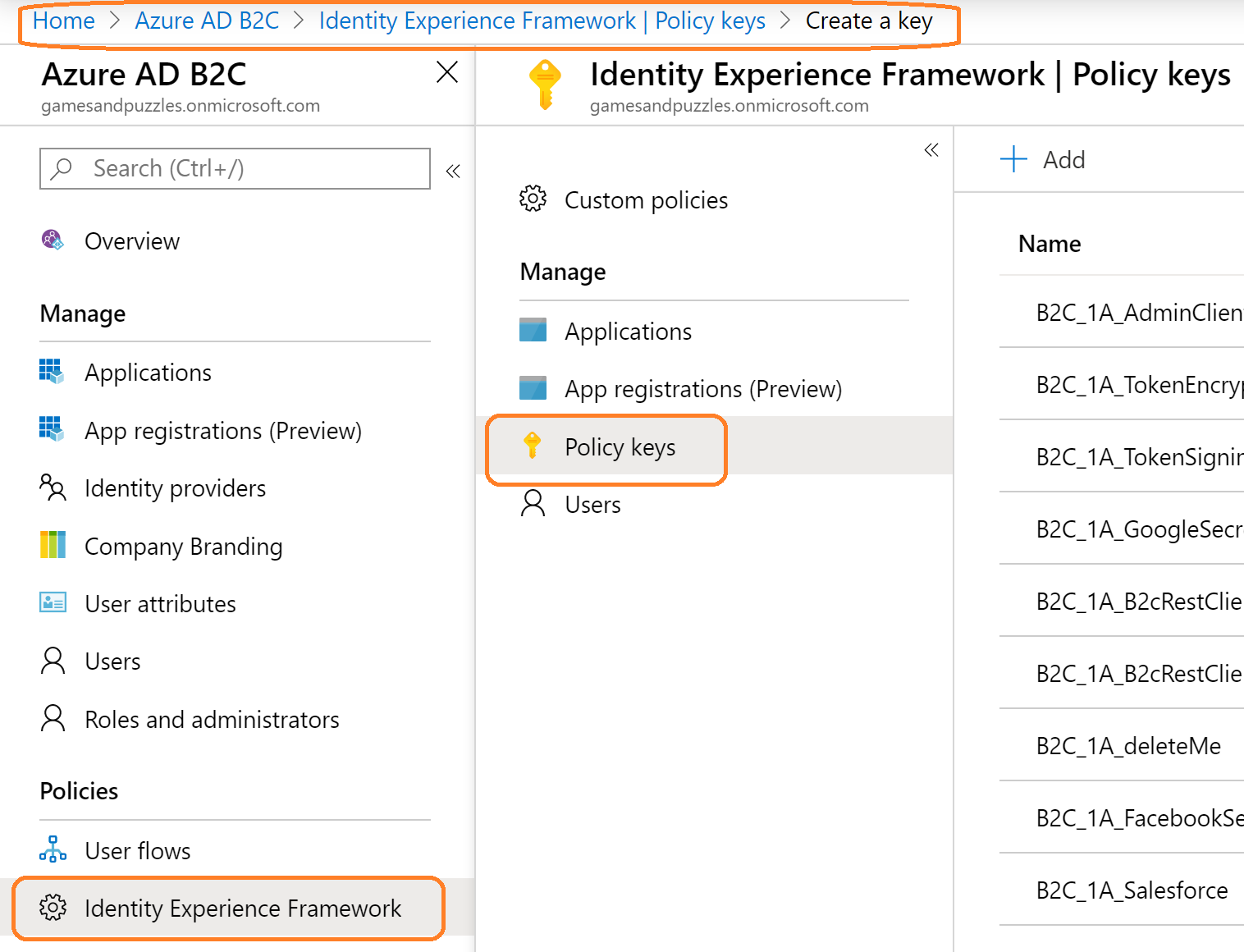
### Scenario

*In this exercise, you will need to create the signing, encryption keys that will be required for the Azure AD B2C custom policy in the Technical Profiles that are used during sign in or sign-up.*

When configuring secrets & certificates, these will be referenced as **Policy keys**. Policy keys stores the secrets that are used to secure authentication or validate signature such as secrets for identity providers or RESTful API’s credentials. Any way to secure information that should not be stored in plain text within the custom policy should be stored in the Policy Keys tab within the Identity Experience Framework.

|  |
| --- |
| **IMPORTANT:** Note that Azure AD B2C does not support Azure Key Vault. |

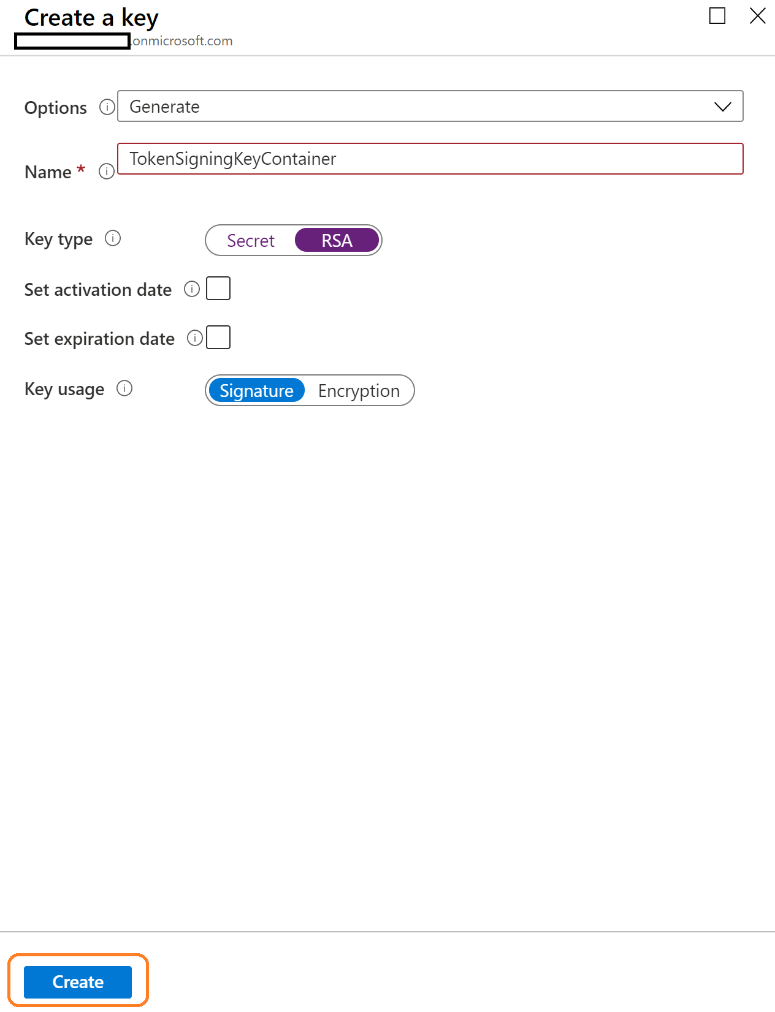
**START HERE:** While on the Azure AD B2C blade, select **Identity Experience Framework** then, select **Policy keys**



### Step 1: Create the signing Key

1. While in the ***Policy keys*** tab, select ![A close up of a logo

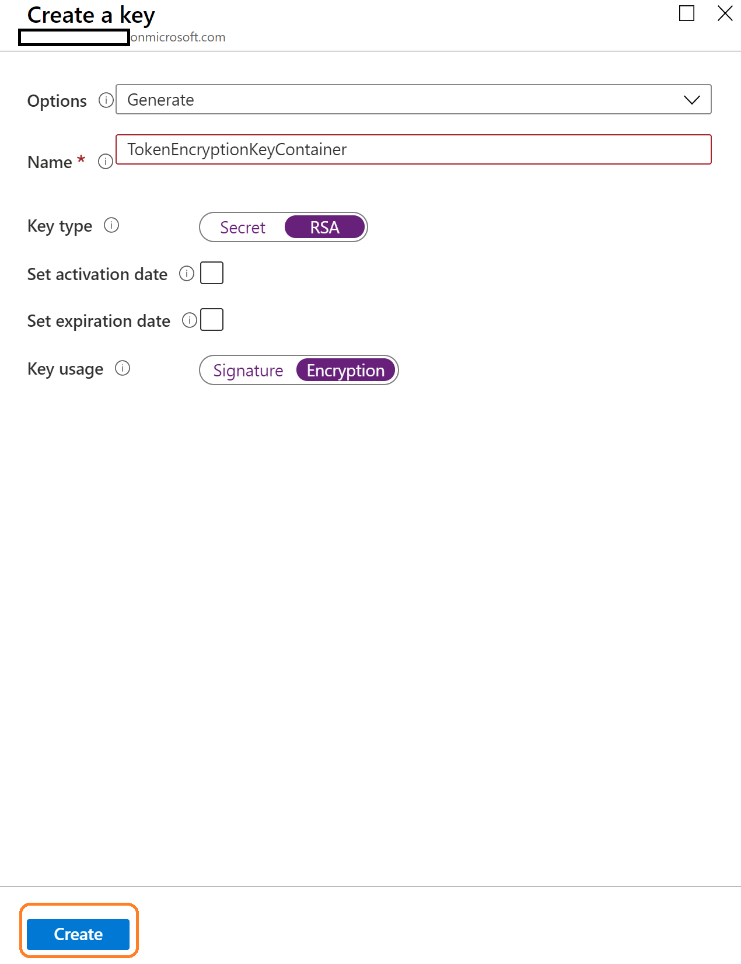
   Description automatically generated](data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAEYAAAA2CAYAAAB6DO9FAAAAAXNSR0IArs4c6QAAAARnQU1BAACxjwv8YQUAAAAJcEhZcwAADsMAAA7DAcdvqGQAAAD4SURBVGhD7duxDsFAAIDhY2RmZ/UQJLyUSU1eiqQewlo7c6245ib8SUtSHf4vae7SrX+uHa5t7/4U9KafRr0wDDAMMAwwDDAMMAwwDDAMMAwwDOh8mGx/Cb31qRrb5IoBhgGGAYYBhgGGAYYBhgGGAYYBhgGGAYYBhgGtvKLdHq5p1lxelNWxmA6r41ub5SjN6mklTNxP+bf7bpZm9bQS5pdNpry4heO5DPNJXDGDdLa5bDVOs3o6/7VDjBpvxXgrNL24X/jwBYYBhgGGAYYBhgGGAYYBhgGGAYYBhgGGAYYB/mQBXDHAMMAwwDDAMMAwwDDAMB+F8AA9Njjq6/DL/gAAAABJRU5ErkJggg==)**Add**.
2. For ***Options***, choose **Generate**.
3. In ***Name***, enter **TokenSigningKeyContainer**..
4. For ***Key type***, select **RSA**.
5. No configuration required for **Set activation date** or **Set expiration date**
6. For ***Key usage***, select **Signature**.
7. Click **Create**.



### Step 2: Create the encryption Key

1. While in the ***Policy Keys*** tab, select ![A close up of a logo

   Description automatically generated](data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAEYAAAA2CAYAAAB6DO9FAAAAAXNSR0IArs4c6QAAAARnQU1BAACxjwv8YQUAAAAJcEhZcwAADsMAAA7DAcdvqGQAAAD4SURBVGhD7duxDsFAAIDhY2RmZ/UQJLyUSU1eiqQewlo7c6245ib8SUtSHf4vae7SrX+uHa5t7/4U9KafRr0wDDAMMAwwDDAMMAwwDDAMMAwwDOh8mGx/Cb31qRrb5IoBhgGGAYYBhgGGAYYBhgGGAYYBhgGGAYYBhgGtvKLdHq5p1lxelNWxmA6r41ub5SjN6mklTNxP+bf7bpZm9bQS5pdNpry4heO5DPNJXDGDdLa5bDVOs3o6/7VDjBpvxXgrNL24X/jwBYYBhgGGAYYBhgGGAYYBhgGGAYYBhgGGAYYB/mQBXDHAMMAwwDDAMMAwwDDAMB+F8AA9Njjq6/DL/gAAAABJRU5ErkJggg==)**Add** .
2. For ***Options***, choose **Generate**.
3. In ***Name***, enter **TokenEncryptionKeyContainer**
4. For ***Key type***, select **RSA**.
5. No configuration required for **Set activation date** or **Set expiration date**
6. For ***Key usage***, select **Encryption**.
7. Click **Create**.

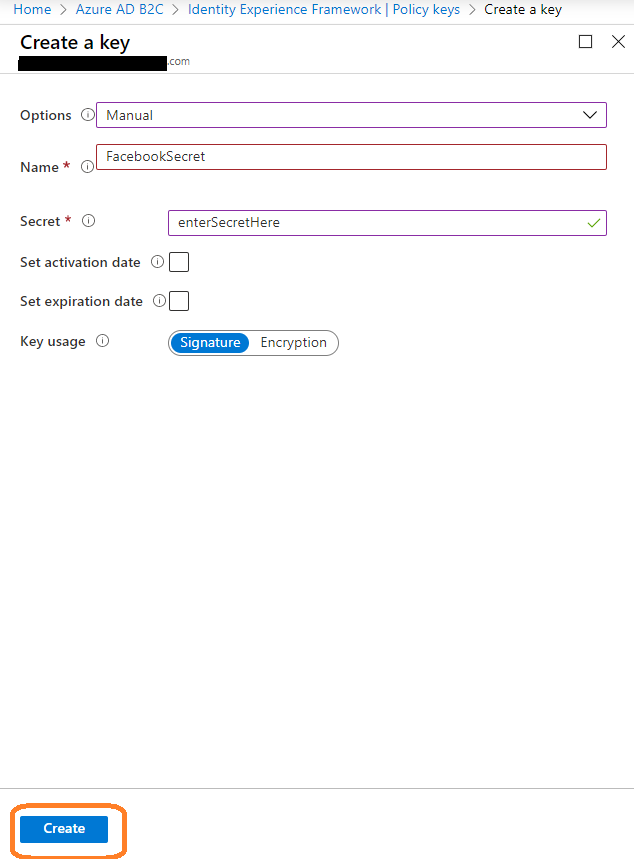


### Step 3: Create Facebook Key

Add your Facebook application's [App Secret](https://docs.microsoft.com/en-us/azure/active-directory-b2c/identity-provider-facebook) as a policy key. You can use the App Secret of the application you created as part of this article's prerequisites.

1. While in the ***Policy keys*** tab, select ![A close up of a logo

   Description automatically generated](data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAEYAAAA2CAYAAAB6DO9FAAAAAXNSR0IArs4c6QAAAARnQU1BAACxjwv8YQUAAAAJcEhZcwAADsMAAA7DAcdvqGQAAAD4SURBVGhD7duxDsFAAIDhY2RmZ/UQJLyUSU1eiqQewlo7c6245ib8SUtSHf4vae7SrX+uHa5t7/4U9KafRr0wDDAMMAwwDDAMMAwwDDAMMAwwDOh8mGx/Cb31qRrb5IoBhgGGAYYBhgGGAYYBhgGGAYYBhgGGAYYBhgGtvKLdHq5p1lxelNWxmA6r41ub5SjN6mklTNxP+bf7bpZm9bQS5pdNpry4heO5DPNJXDGDdLa5bDVOs3o6/7VDjBpvxXgrNL24X/jwBYYBhgGGAYYBhgGGAYYBhgGGAYYBhgGGAYYB/mQBXDHAMMAwwDDAMMAwwDDAMB+F8AA9Njjq6/DL/gAAAABJRU5ErkJggg==)**Add**.
2. For ***Options***, choose **Manual**.
3. In ***Name***, enter **FacebookSecret**.
4. In **Secret**, enter your Facebook application's *App Secret* from developers.facebook.com. This value is the secret, not the application ID
5. For ***Key usage***, select **Signature**.
6. Click **Create**.



|  |
| --- |
| **IMPORTANT:** The prefix **“B2C\_1A\_”** will be added automatically later. This is how Azure AD B2C Identity Experience Framework differentiates **custom policy** from **user flows**. |

# Exercise 5: Register applications

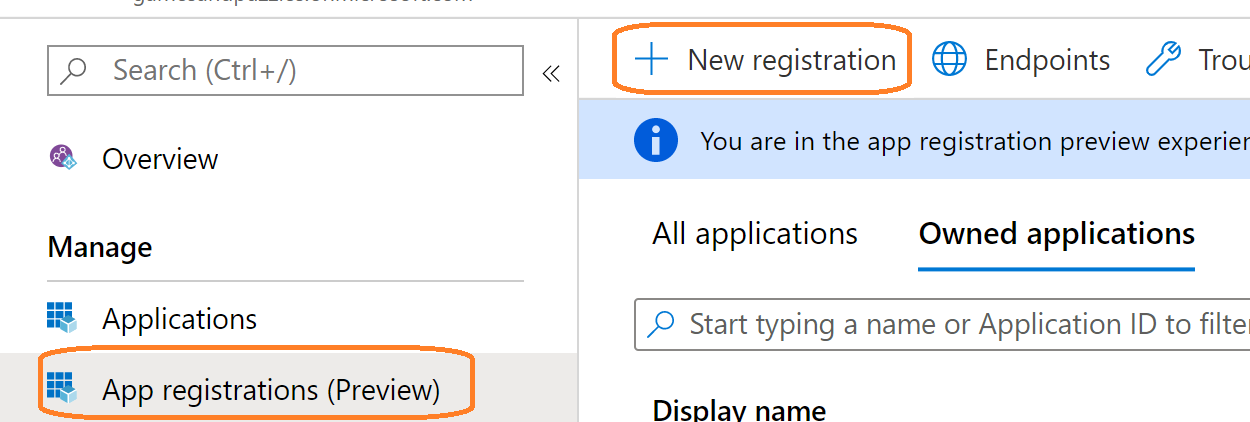
### Scenario

Azure AD B2C requires you to register two applications within Azure AD (not B2C) that are used to sign up and sign in users: **IdentityExperienceFramework** (a web app), and **ProxyIdentityExperienceFramework** (a native app) with delegated permission from the **IdentityExperienceFramework** app. Local accounts exist only in your tenant. Your users sign up with a unique email address/password combination to access your tenant-registered applications.

### Step 1: Register the IdentityExperienceFramework application

1. Open the **Azure AD B2C** menu by searching for it under **More Services**
2. Select **App registrations (Preview)** tab and then select ![A close up of a logo

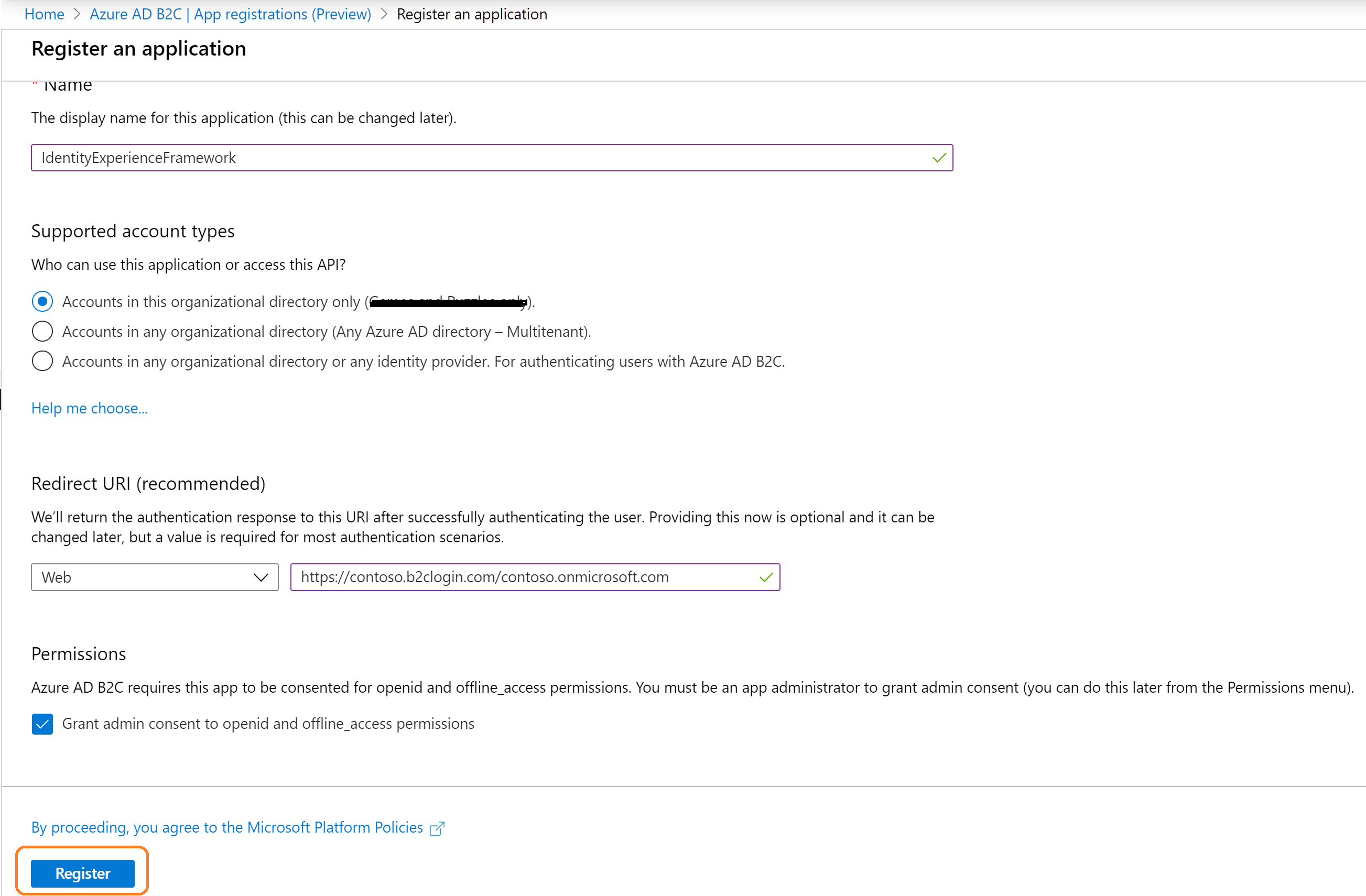
   Description automatically generated](data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAEYAAAA2CAYAAAB6DO9FAAAAAXNSR0IArs4c6QAAAARnQU1BAACxjwv8YQUAAAAJcEhZcwAADsMAAA7DAcdvqGQAAAD4SURBVGhD7duxDsFAAIDhY2RmZ/UQJLyUSU1eiqQewlo7c6245ib8SUtSHf4vae7SrX+uHa5t7/4U9KafRr0wDDAMMAwwDDAMMAwwDDAMMAwwDOh8mGx/Cb31qRrb5IoBhgGGAYYBhgGGAYYBhgGGAYYBhgGGAYYBhgGtvKLdHq5p1lxelNWxmA6r41ub5SjN6mklTNxP+bf7bpZm9bQS5pdNpry4heO5DPNJXDGDdLa5bDVOs3o6/7VDjBpvxXgrNL24X/jwBYYBhgGGAYYBhgGGAYYBhgGGAYYBhgGGAYYB/mQBXDHAMMAwwDDAMMAwwDDAMB+F8AA9Njjq6/DL/gAAAABJRU5ErkJggg==) **New registration**



1. Enter **“IdentityExperienceFramework”** as the ***Name*** of the application.
2. Under ***Supported account types***, select **Accounts in this organizational directory only**.
3. Under ***Redirect URI***, select **Web**, and then enter “**https://your-tenant-name.b2clogin.com/your-tenant-name.onmicrosoft.com**”,

|  |
| --- |
| **IMPORTANT:** Note that **your-tenant-name** is your Azure AD B2C tenant domain name  ***Example:*** https://**contoso**.b2clogin.com/**contoso**.onmicrosoft.com |

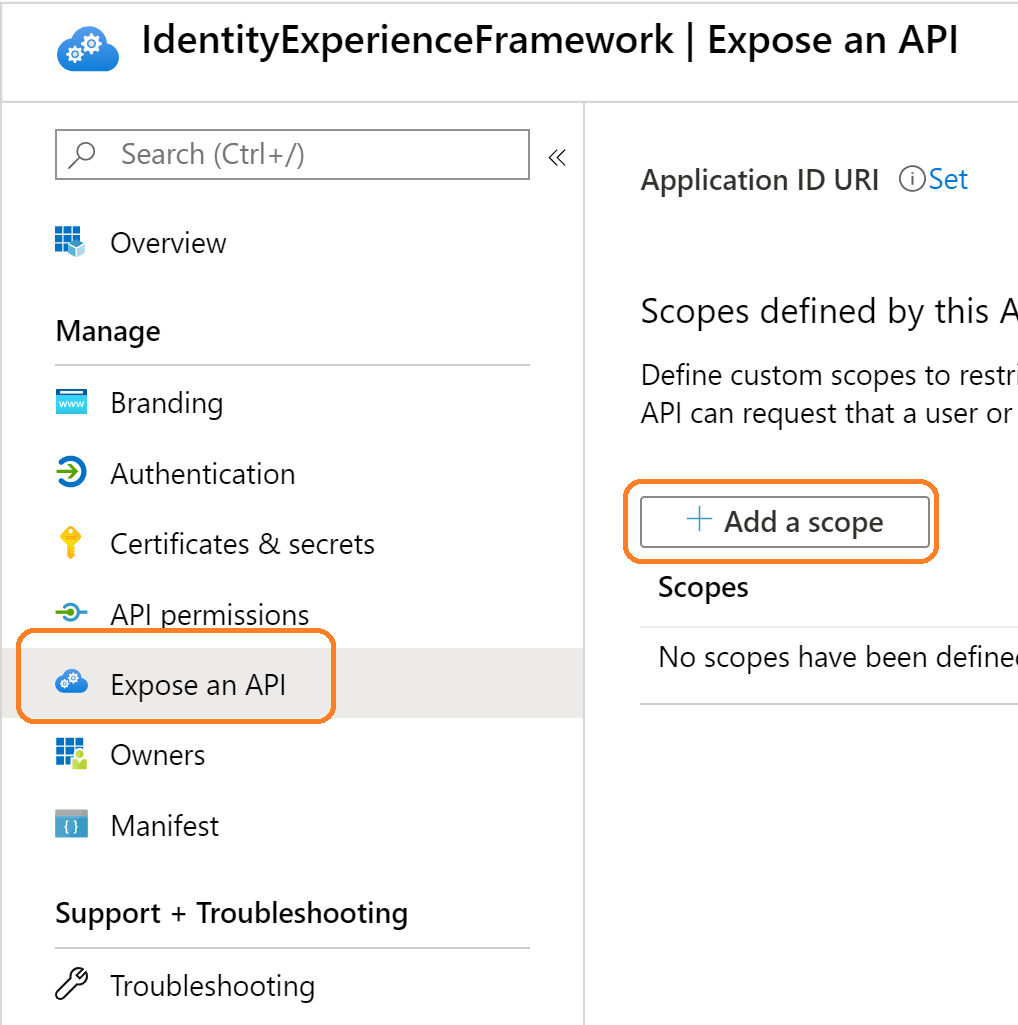
1. Under ***Permissions***, select the **Grant admin consent to openid and offline\_access permissions** check box.
2. Select **Register**.



1. **Record the Application (client) ID** for use in a later step.

Next, expose the API by adding a scope.

1. Under **Manage**, select **Expose an API**.
2. Select **Add a scope**, then select **Save** and continue to accept the default application ID URI.

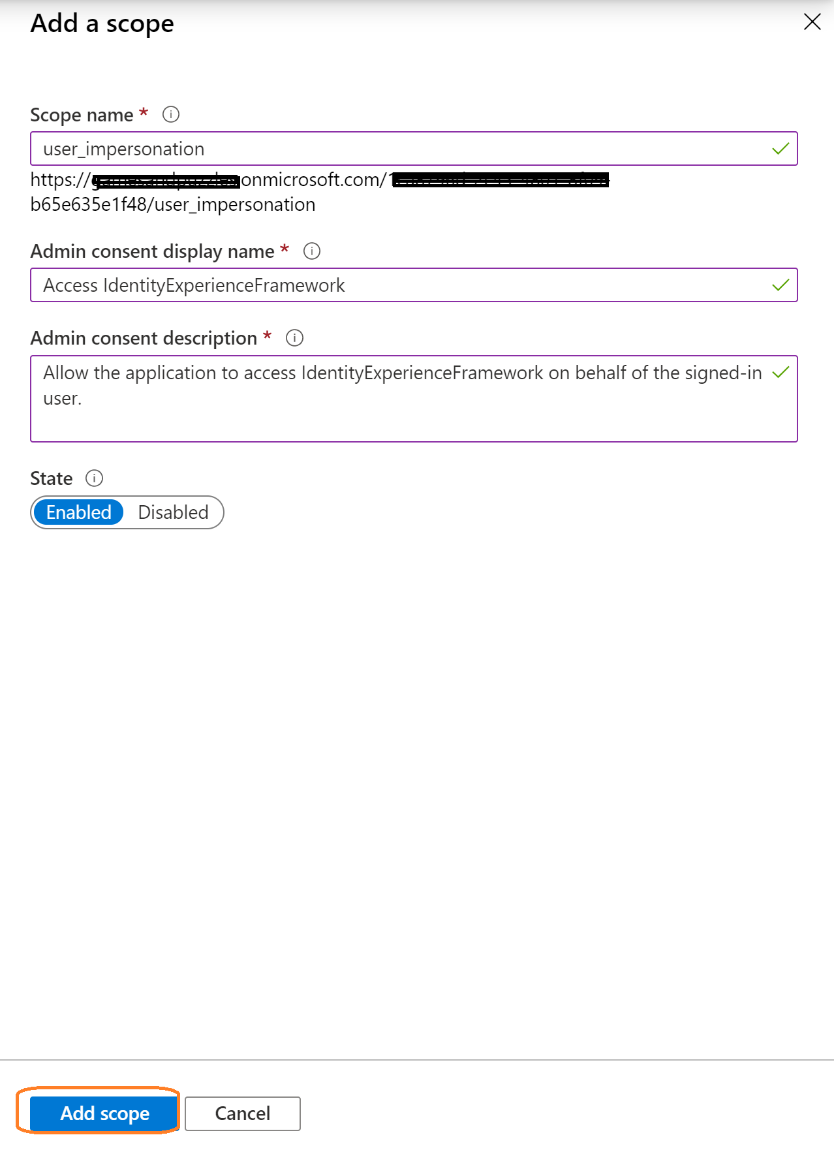


|  |
| --- |
| **IMPORTANT:** Note that the GUID value for the default scope is the **Application (client) ID** of the application that you just created. You can change this but for now we will not for this **Lab**. |

1. Enter the following values to create a scope that allows custom policy execution in your Azure AD B2C tenant:

|  |  |
| --- | --- |
| Scope name | **user\_impersonation** |
| Admin consent display name | **Access IdentityExperienceFramework** |
| Admin consent description | **Allow the application to access IdentityExperienceFramework on behalf of the signed-in user.** |
| State | **Enabled** |

1. Select **Add scope**



### Step 2: Register the ProxyIdentityExperienceFramework application

1. Open the **Azure AD B2C** menu by searching for it under **More Services**
2. Select **App registrations (Preview)** tab and then select ![A close up of a logo

   Description automatically generated](data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAEYAAAA2CAYAAAB6DO9FAAAAAXNSR0IArs4c6QAAAARnQU1BAACxjwv8YQUAAAAJcEhZcwAADsMAAA7DAcdvqGQAAAD4SURBVGhD7duxDsFAAIDhY2RmZ/UQJLyUSU1eiqQewlo7c6245ib8SUtSHf4vae7SrX+uHa5t7/4U9KafRr0wDDAMMAwwDDAMMAwwDDAMMAwwDOh8mGx/Cb31qRrb5IoBhgGGAYYBhgGGAYYBhgGGAYYBhgGGAYYBhgGtvKLdHq5p1lxelNWxmA6r41ub5SjN6mklTNxP+bf7bpZm9bQS5pdNpry4heO5DPNJXDGDdLa5bDVOs3o6/7VDjBpvxXgrNL24X/jwBYYBhgGGAYYBhgGGAYYBhgGGAYYBhgGGAYYB/mQBXDHAMMAwwDDAMMAwwDDAMB+F8AA9Njjq6/DL/gAAAABJRU5ErkJggg==) **New registration**

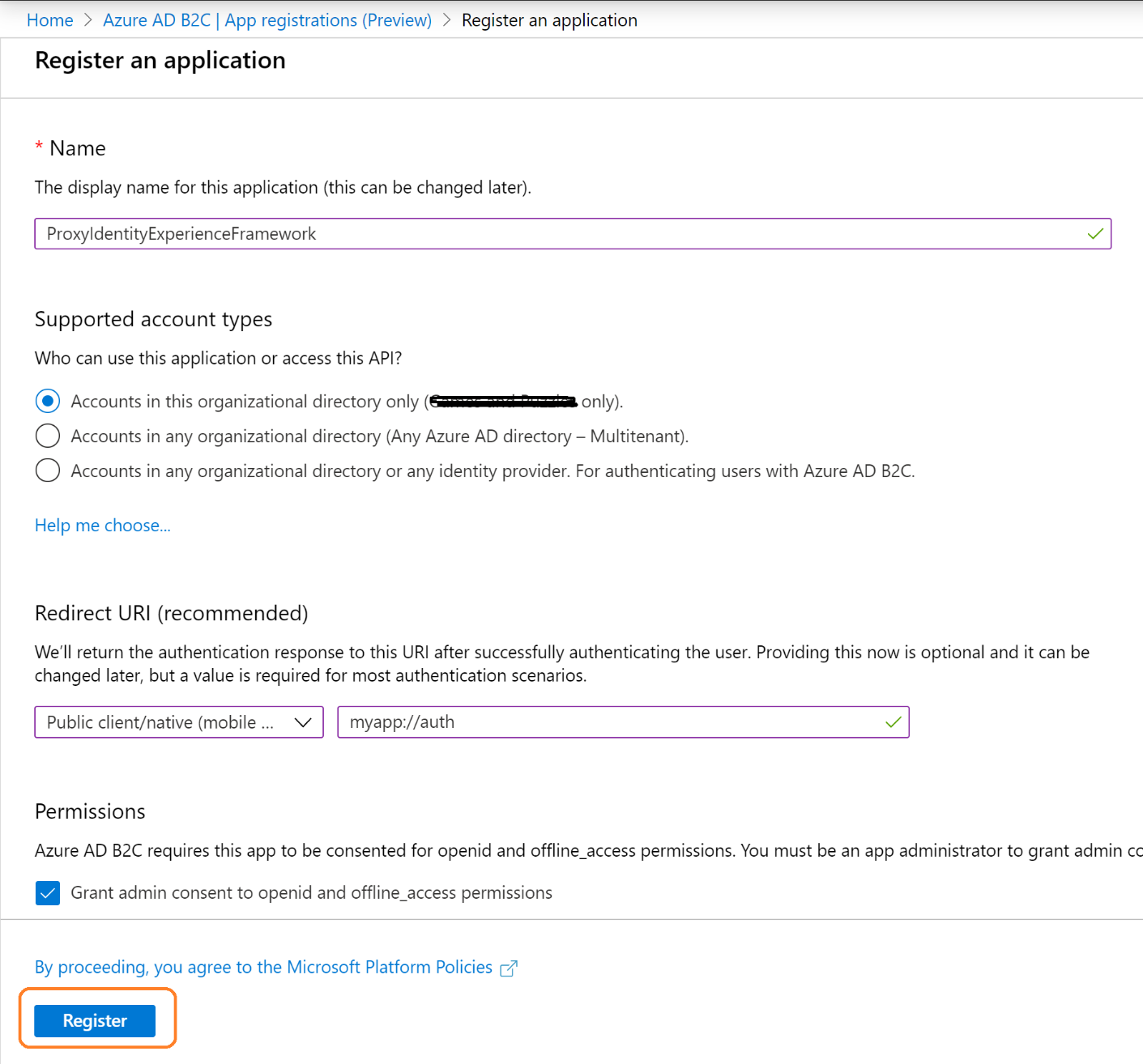
A screenshot of a cell phone

Description automatically generated

1. Enter **“ProxyIdentityExperienceFramework”** as the ***Name*** of the application.
2. Under ***Supported account types***, select **Accounts in this organizational directory only**.
3. Under ***Redirect URI***, use the drop-down to select **Public client/native (mobile & desktop)**.
4. For ***Redirect URI***, enter “**myapp://auth**”.

|  |
| --- |
| **IMPORTANT:** Note that native applications and web applications use different type of tokens and sessions when authenticating. Explicitly defining this in the app/service allows Azure AD to process the request correctly. |

1. Under ***Permissions***, select the **Grant admin consent to openid and offline\_access permissions** check box.
2. Select **Register**.
3. **Record the Application (client) ID** for use in a later step.



Next, specify that the application should be treated as a public client:

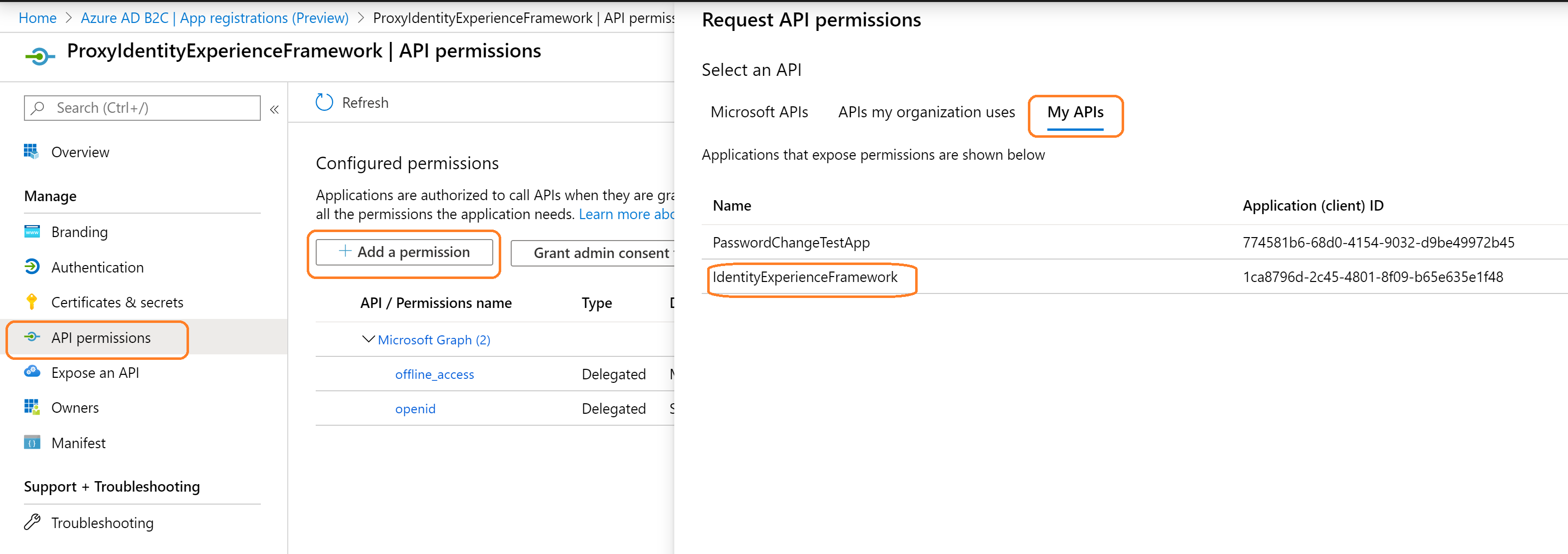
1. Under ***Manage***, select **Authentication**.
2. Select **Try out the new Experience** (if shown).
3. Under ***Advanced settings***, enable **Treat application as a public client** (select **Yes**).

Ensure that **“allowPublicClient”: true** is set in the application manifest.

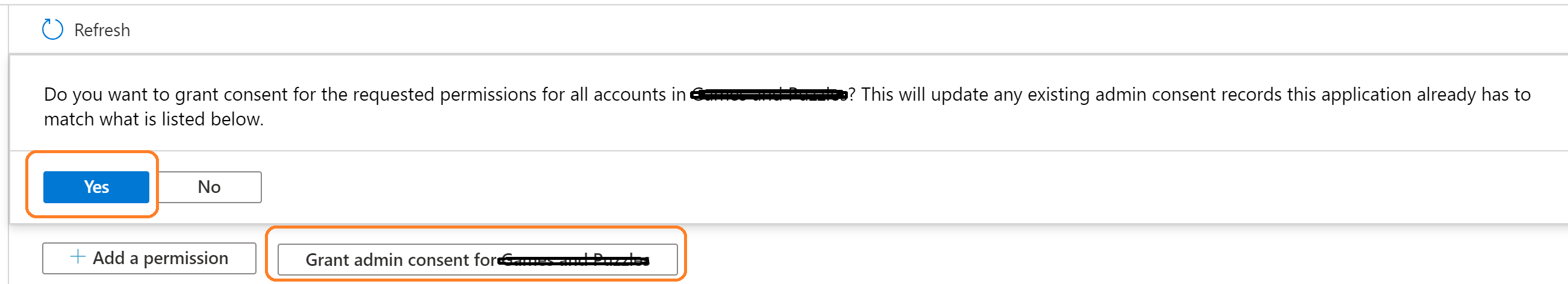
1. Select **Save**.

Now, grant permissions to the API scope you exposed earlier in the *IdentityExperienceFramework* registration.

1. Under ***Manage***, select **API permissions**.
2. Under ***Configured permissions***, select **Add a permission**.
3. Select the ***My APIs*** tab, then select the **IdentityExperienceFramework** application.



1. Under ***Permission***, select the **user\_impersonation** scope that you defined earlier.
2. Select **Add permissions**. *As directed, wait a few minutes before proceeding to the next step.*
3. Select **Grant admin consent for (your tenant name)**.
4. Select your currently signed-in administrator account, or sign in with an account in your Azure AD B2C tenant that's been assigned at least the *Cloud application administrator role*.
5. Select **Yes**.



1. Select A picture containing drawing

   Description automatically generated **Refresh**, and then verify that "Granted for ..." appears under **Status** for both scopes*. It might take a few minutes for the permissions to propagate.*

|  |
| --- |
| **IMPORTANT:** Note that if you do not accept “Grant admin consent for…” you will put your application in a broken state for this **Lab**. This is a common mistake to overlook this **step** but it is required and explicit – this requires **admin credentials**. |

# Exercise 6: Download starter pack and modify policies

### Scenario

In this scenario you will download the .zip file and extract the required files for the Azure AD custom policies. While not required, we recommend using an XML editor, try [**Visual Studio Code**](https://code.visualstudio.com/download) to make it easier to read and navigate through the policies.

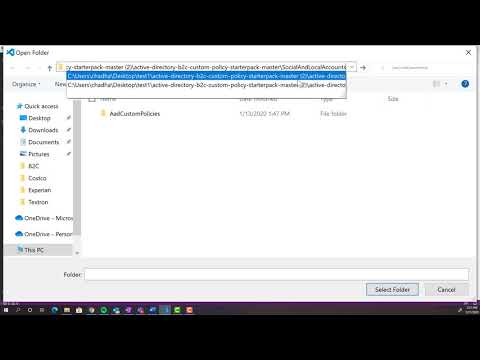
Custom policies are a set of XML files you upload to your Azure AD B2C tenant to define technical profiles and user journeys. We provide starter packs with several pre-built policies to get you going quickly. Each of these starter packs contains the smallest number of technical profiles and user journeys needed to achieve the scenarios described:

* **Base file** - Few modifications are required to the base. Example: *TrustFrameworkBase.xml*
* **Extension file** - This file is where most configuration changes are made. Example: *TrustFrameworkExtensions.xml*
* **Relying party files** - Task-specific files called by your application. Examples: *SignUpOrSignin.xml*, *ProfileEdit.xml*, *PasswordReset.xml*

|  |
| --- |
| **IMPORTANT:** Note that in this lab you edit the XML custom policy files in the **SocialAndLocalAccounts** starter pack. If you need an XML editor, try [Visual Studio Code](https://code.visualstudio.com/download), a lightweight cross-platform editor. |

### Step 1: Download and configure policy

Get the custom policy starter packs from GitHub, then update the XML files in the SocialAndLocalAccounts starter pack with your Azure AD B2C tenant name.

[](https://www.youtube.com/embed/399geRV43vA)Like many policy engines, the Identity Experience Framework requires a strongly typed configuration file to render the correct behavior. Therefore XML was adopted and chosen over JSON format. Within the XML file (custom policy), you will configure tenant specific configuration to include add references to application ID’s and secrets. Microsoft will provide the base information that is required in most common deployments and requires you to make minor changes to the XML file to get started.

1. [Download the .zip file](https://github.com/Azure-Samples/active-directory-b2c-custom-policy-starterpack/archive/master.zip) or clone the repository:

|  |
| --- |
| git clone https://github.com/Azure-Samples/active-directory-b2c-custom-policy-starterpack |

1. In all of the files in the **SocialAndLocalAccounts** directory, replace the string yourtenant with the name of your Azure AD B2C tenant.

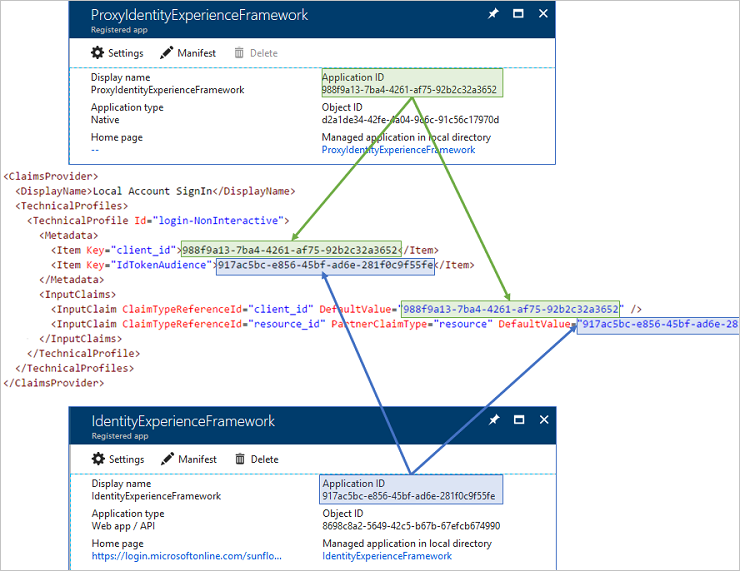
For example, if the name of your B2C tenant is *contosotenant*, all instances of yourtenant.onmicrosoft.com become contosotenant.onmicrosoft.com.

|  |
| --- |
| **IMPORTANT**: You can easily find all instance of “**yourtenant**” by using the “Find in Folder” option under right clicking the high-level folder of **SocialAndLocalAccounts** (find and replace) |

### Step 2: Add application IDs to the custom policy

Add the application IDs to the extensions file TrustFrameworkExtensions.xml.

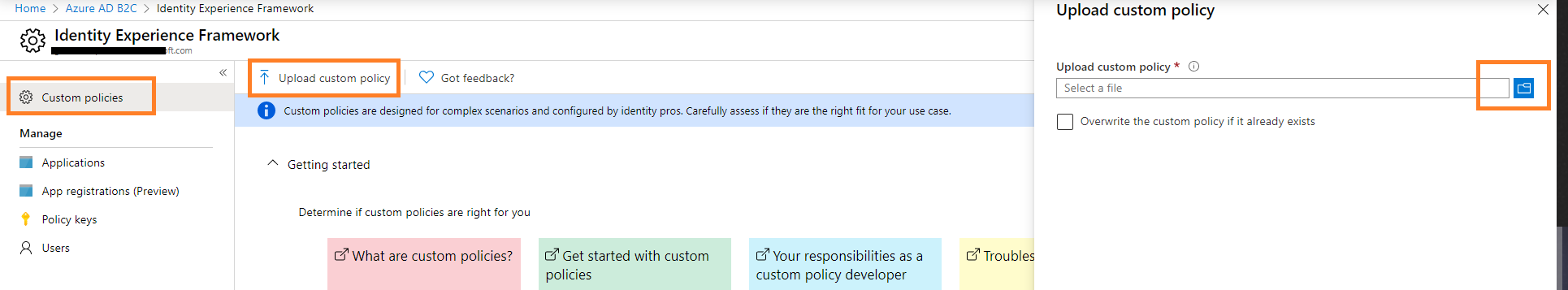
1. Open **SocialAndLocalAccounts/TrustFrameworkExtensions.xml** and find the element **<TechnicalProfile Id="login-NonInteractive">.**
2. Replace both instances of **IdentityExperienceFrameworkAppId** with the application ID of the **IdentityExperienceFramework** application that you created earlier.
3. Replace both instances of **ProxyIdentityExperienceFrameworkAppId** with the application ID of the **ProxyIdentityExperienceFramework** application that you created earlier.
4. Save the file.



### Step 3: Upload the policies

1. Select the **Identity Experience Framework** menu item in your B2C tenant in the Azure portal.
2. Select **Upload custom policy**.
3. In this order, upload the policy files:
   1. *TrustFrameworkBase.xml*
   2. *TrustFrameworkExtensions.xml*
   3. *SignUpOrSignin.xml*
   4. *ProfileEdit.xml*
   5. *PasswordReset.xml*

As you upload the files, Azure adds the prefix **B2C\_1A\_** to each.



|  |
| --- |
| **IMPORTANT:** Note that you must upload the files in order as defined in this step. If you do this out of order, you will receive an error. The Identity Experience Framework (IEF) was designed in a hierarchical fashion – where one file points to the previous layer of files. If you look at the TrustFrameworkBase.xml, you will notice it is the largest with most of the business logic – hence why it is uploaded first. |

### Step 4: Test your policy

To continue with the lab, we want to confirm that the foundation configuration has been completed correctly. While this is an extra step (and not necessarily needed), it will allow us to stop and fix any misconfiguration in the XML policies before adding additional logic to our policies. This is a great practice to follow when making multiple scenarios within a single implementation. We will test both Sign-in and Sign-up.

*Test Sign-up*

1. Under **Custom policies**, select **B2C\_1A\_signup\_signin**.
2. For **Select application** on the overview page of the custom policy, select the web application named *MyWebAppJWT* that you previously registered.
3. Make sure that the **Reply URL** is *https://jwt.ms*.
4. Select **Run now**.
5. Sign up using an email address.

*Test Sign-in*

1. Select **Run now** again on previous tab.
2. Sign in with the same account to confirm that you have the correct configuration.

A screenshot of a cell phone

Description automatically generated

# Exercise 7: Facebook - Set up sign-up and sign-in using Azure Active Directory B2C

### Scenario

### As mentioned in [Prerequisites](#_Prerequisites), Facebook is not required for using custom policies, but is used here to demonstrate how you can enable federated social login in a custom policy.

### Step 1: Add Facebook’s claim provider information to your TrustFrameworkExtensions.XML file

You must define Facebook’s **clientid** in the TrustFrameworkExtension file – this will be used during the OIDC flow between Azure AD B2C and Facebook. Attributes on your Facebook’s user object will be sent in the form of claims from Facebook to Azure AD B2C in a JWT token which will enable login and creation of accounts.

1. Locate & Open the **TrustFrameworkExtension.xml** file in the **SocialAndLocalAccounts** folder
2. In the **TrustFrameworkExtensions.mxl** file, replace the value of **client\_id** with the Facebook application ID collected in the prerequisites.
   * *Note: Replace* ***facebook\_clientid*** *with your application ID value with the brackets > <.*
3. <ClaimsProvider>
4. <DisplayName>Facebook</DisplayName>
5. <TechnicalProfiles>
6. <TechnicalProfile Id="Facebook-OAUTH">
7. <Metadata>
8. <Item Key="client\_id">facebook\_clientid</Item>
9. <Item Key="scope">email public\_profile</Item>
10. <Item Key="ClaimsEndpoint">https://graph.facebook.com/me?fields=id,first\_name,last\_name,name,email</Item>
11. </Metadata>
12. </TechnicalProfile>
13. </TechnicalProfiles>
14. </ClaimsProvider>
15. **Save** all changes in Visual Studio Code.

### Step 2: Upload the policies

#### Scenario

In this exercise, you will be uploading the custom policies (XML files) to Azure AD B2C admin console. In the lab, we made changes to the Extension XML file by adding the Facebook ClientID information. We need to upload these changes to the Identity Experience Framework. This is done by uploading the policies to Azure AD B2C.

1. Select the **Identity Experience Framework** menu item in your B2C tenant in the Azure portal.
2. Select **Upload custom policy**.
3. Enable **Overwrite the policy if it exists**; click the folder icon.
4. In this order, upload the policy files:
   1. *TrustFrameworkExtensions.xml*
5. Select **Upload Policy** after each selection of files

A screenshot of a social media post

Description automatically generated

|  |
| --- |
| **IMPORTANT:** Note that we previously uploaded the other XML files (Base, Extension & RelyingParty). In this step, we are only overriding the TrustFrameworkExtension.xml file. The policyID is **unique** and is how the Identity Experience Framework maps your uploaded file to the one that already exists in Azure AD B2C. This is also the same value that is used to represent the name in the Azure AD B2C admin console. We recommend naming it something easily represented. |

### Step 3: Test the custom policy

##### Scenario

Follow the same steps as [described above located here](#_Step_4:_Test).

# Configuring RESTful Provider

###### Abstract

In this portion of the lab you will configure a RESTful API Provider to perform an API call to a third-party service during a sign-up or sign-in flow. This scenario will retrieve a Membership Number/Loyalty Number of a customer. Once you understand how to do this, you can easily integrate this into any of your flows called by your application/service.

## Prerequisites

Have a local account or social identity that can successfully authenticate using Azure AD custom policies while authenticating to an application. If you do not have a social identity provider, you can still configure this with a local account and test this successfully.

# Exercise 1: Add client secret and client id for REST API call

### Scenario

In this exercise, you will need to create the Client ID and Client Secret in order to perform basic authentication with the REST API when it attempts to execute a REST call to gain the **loyaltynumber** of the user that is stored in another service. The keys below will be referenced in the B2C policies in a later step.

**START HERE:** While on the Azure AD B2C blade, select **Identity Experience Framework** then, select **Policy keys**

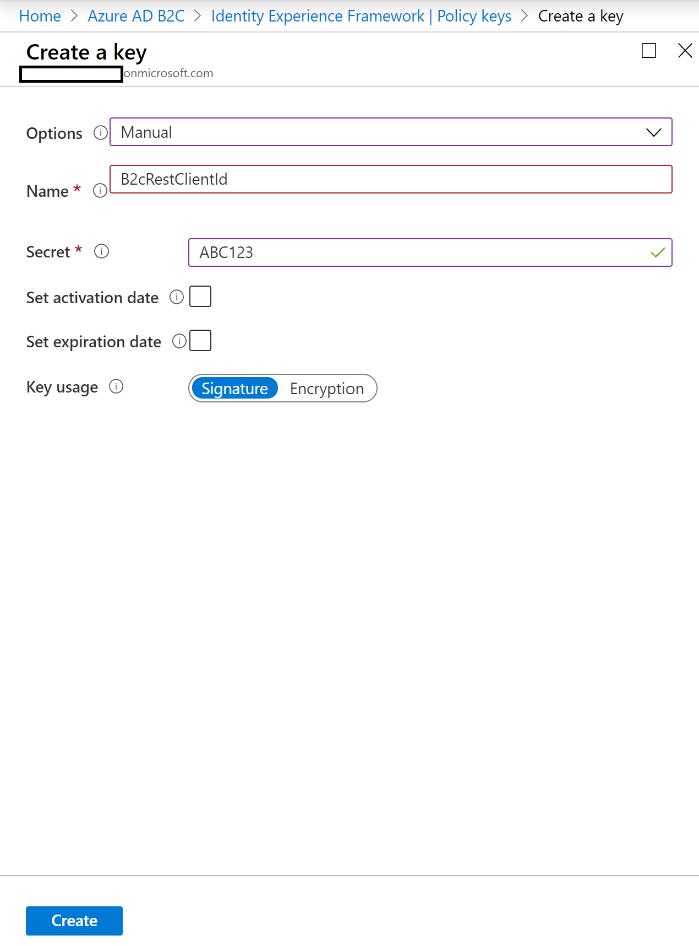
A screenshot of a cell phone

Description automatically generated

##### Step 1: Create the Rest Client Id

1. While in the ***Policy keys*** tab, select ![A close up of a logo

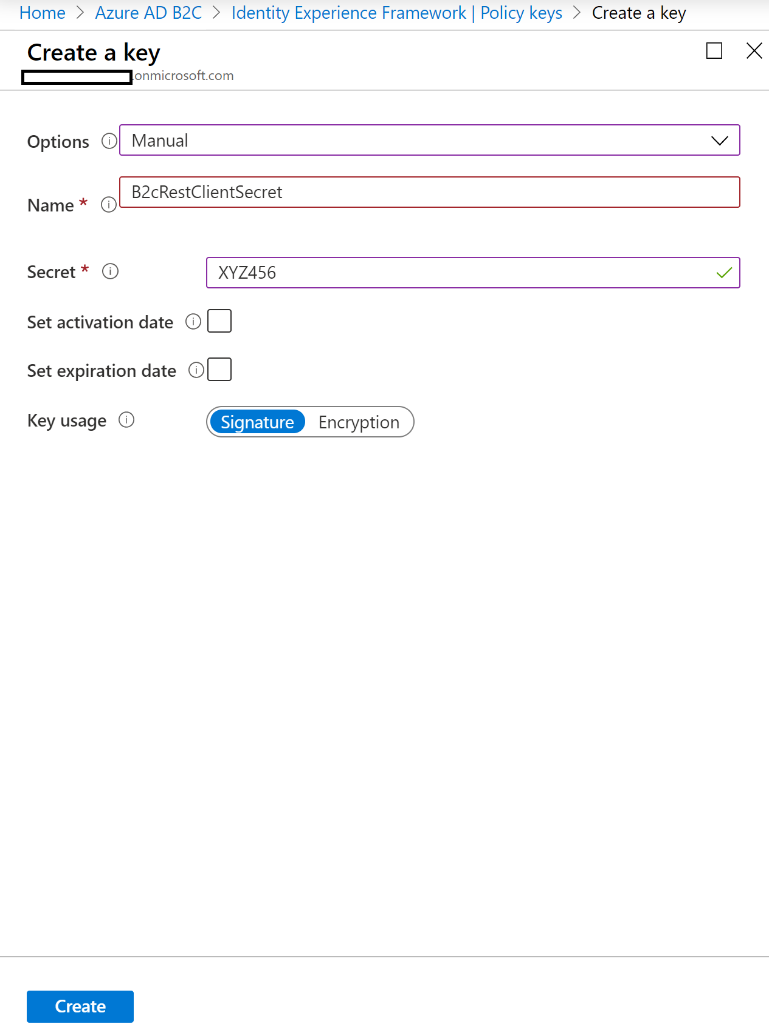
   Description automatically generated](data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAEYAAAA2CAYAAAB6DO9FAAAAAXNSR0IArs4c6QAAAARnQU1BAACxjwv8YQUAAAAJcEhZcwAADsMAAA7DAcdvqGQAAAD4SURBVGhD7duxDsFAAIDhY2RmZ/UQJLyUSU1eiqQewlo7c6245ib8SUtSHf4vae7SrX+uHa5t7/4U9KafRr0wDDAMMAwwDDAMMAwwDDAMMAwwDOh8mGx/Cb31qRrb5IoBhgGGAYYBhgGGAYYBhgGGAYYBhgGGAYYBhgGtvKLdHq5p1lxelNWxmA6r41ub5SjN6mklTNxP+bf7bpZm9bQS5pdNpry4heO5DPNJXDGDdLa5bDVOs3o6/7VDjBpvxXgrNL24X/jwBYYBhgGGAYYBhgGGAYYBhgGGAYYBhgGGAYYB/mQBXDHAMMAwwDDAMMAwwDDAMB+F8AA9Njjq6/DL/gAAAABJRU5ErkJggg==)**Add**.
2. For ***Options***, choose **Manual**.
3. In ***Name***, enter **B2cRestClientId**
4. For ***Secret***, enter **[insert your clientid secret taken from your REST service – this is usually marked as the ClientID/applicationID]**
5. No configuration required for **Set activation date** or **Set expiration date**
6. For ***Key usage***, select **Signature**.
7. Click **Create**.



##### Step 2: Create the Rest Client Secret

1. While in the ***Policy keys*** tab, select ![A close up of a logo

   Description automatically generated](data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAEYAAAA2CAYAAAB6DO9FAAAAAXNSR0IArs4c6QAAAARnQU1BAACxjwv8YQUAAAAJcEhZcwAADsMAAA7DAcdvqGQAAAD4SURBVGhD7duxDsFAAIDhY2RmZ/UQJLyUSU1eiqQewlo7c6245ib8SUtSHf4vae7SrX+uHa5t7/4U9KafRr0wDDAMMAwwDDAMMAwwDDAMMAwwDOh8mGx/Cb31qRrb5IoBhgGGAYYBhgGGAYYBhgGGAYYBhgGGAYYBhgGtvKLdHq5p1lxelNWxmA6r41ub5SjN6mklTNxP+bf7bpZm9bQS5pdNpry4heO5DPNJXDGDdLa5bDVOs3o6/7VDjBpvxXgrNL24X/jwBYYBhgGGAYYBhgGGAYYBhgGGAYYBhgGGAYYB/mQBXDHAMMAwwDDAMMAwwDDAMB+F8AA9Njjq6/DL/gAAAABJRU5ErkJggg==)**Add**.
2. For ***Options***, choose **Manual**.
3. In ***Name***, enter **B2cRestClientSecret**
4. For ***Secret***, enter **[insert your client secret taken from your REST service – this is known as the “password’ for basic authentication]**
5. No configuration required for **Set activation date** or **Set expiration date**
6. For ***Key usage***, select **Signature**.
7. Click **Create**.



# Exercise 2: Move UserJourneys from Base file to Extension file

### Scenario

In this exercise, you will need to cut the UserJourneys from the **TrustFrameworkBase.xml** and paste them to the **TrustFrameworkExtension.xml** file. The TrustFrameworkBase.xml file will inherent what is in the TrustFrameworkExtension.xml file therefore, by following best practices we will not modify the base file and instead add the additional Orchestrated Steps to the extension file. We recommend to avoid modifying the base file at all cost. In certain cases such as this, we require to move the logic higher level in the hierarchy so we may be consistent to this practice.

This practice should be done if you require to make any changes to the IEF such as adding additional claim schemas or additional orchestration steps to your user journey.

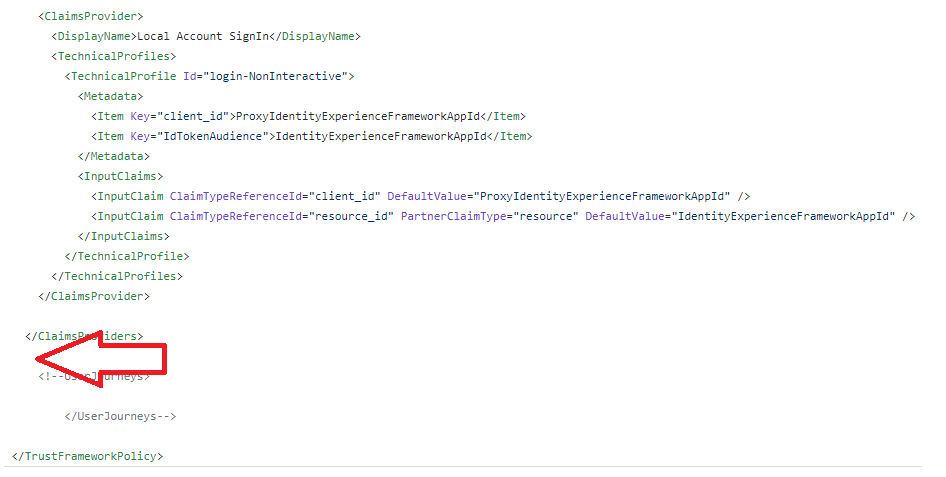
#### Step 1: **Cut** UserJourneys section in Base Extension

1. Navigate to the TrustFrameworkBase.xml file within Visual Studio Code
2. Press Ctrl + “F” to pull up the find bar and type “UserJourneys”
3. Highlight all UserJourneys and Cut between the following brackets:
   1. <UserJourneys>……….</UserJourneys>
4. Save your **TrustFrameworkBase.xml** file.

|  |
| --- |
| **IMPORTANT:** Note there are several UserJourneys that are within this section such as Sign-up, Sign-in, Password Reset, and Profile Edit. We will be moving all UserJourney(s) to the TrustFrameworkExtension.xml file. XML requires open and closing tags – make sure you do not miss a tag or you will receive an XML formatting error during upload. |

#### Step 2: **Paste** to UserJourneys section to Extension file

1. Navigate to the TrustFrameworkExtensions.xml file within Visual Studio Code
2. Scroll at the bottom of the page
3. Press Ctrl + “F” to pull up the find bar and type “UserJourneys”
4. After the **ClaimsProviders**, right click and select “Paste”
   1. This will paste the entire **UserJourneys**
      1. Note that you can delete or replace the commented out <!—UserJourneys> </UserJourneys --> section with the pasted XML snippet.

[](https://www.youtube.com/embed/ZBQ-4_uc9To?feature=oembed)

1. Save your **TrustFrameworkExtension.xml** file.

# Exercise 3: Create a REST API ClaimsProvider

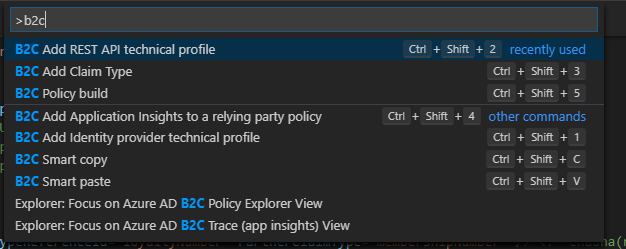
### Scenario

In this exercise, you will add the ClaimProvider Technical Profile for the REST API. This Technical Profile will contain the configuration for your REST API such as Client ID, Client Secret and endpoints. This is the connection between your REST API service and Azure AD B2C.

1. Navigate to the **TrustFrameworkExtension.xml** file within Visual Studio Code
2. In the ClaimsProviders section, click after </ClaimsProvider> of Facebook’s ClaimsProvider to add a new ClaimsProvider for the REST API .



1. Press Ctrl +Shift + P on the keyboard
2. Type **B2C** and choose B2C Add REST API technical profile



1. In **Provide a name**, enter **REST-API-SignUp**
2. In **Service URL**, enter your service URL endpoint for your rest service. Example: [**https://contoso.azurewebsites.net/api/identity/membership**](https://contoso.azurewebsites.net/api/identity/membership)
3. In **Select Authentication Type**, select Basic
   1. This should automatically create the REST API technical profile
4. In the OutputClaim section, replace **PartnerClaimType = ”loyaltyNumber”** to equal **membershipNumber** instead; example:

**EXAMPLE**

<OutputClaims>

<OutputClaim ClaimTypeReferenceId="loyaltyNumber" PartnerClaimType="membershipNumber" />

</OutputClaims>

*This remapping allows to send a claim from the REST-API called “membershipNumber” to be mapped to a claim in Azure AD B2C called “loyaltyNumber”. OutputClaim in this scenario means, what is the output of calling this REST-API.*

|  |
| --- |
| **IMPORTANT:** Note that using the B2C extension for Visual Studio automatically populates the key name. If you had to do this without the B2C extension, you would have to match the name created in the Azure AD B2C Keys  The above OutputClaims assumes you are sending the claim from your REST API as “membershipNumber”. If you name it something else, update that value in the PartnerClaimType value. |

# Exercise 4: Add the claim “loyaltyNumber” to the schema

### Scenario

In this exercise, you will be required to add the claim called loyaltyNumber to your schema inside the TrustFrameworkExtension.xml file. When calling a claim inside a technical profile, there must be a correlated claim that is instantiated within the schema. That is defined in either the base or extension file. In this example, we will be defining this within the TrustFrameworkExtension.xml file.

1. **Navigate** to the TrustFrameworkExtensions.xml file.
2. Within the **<BuildingBlocks></BuildingBlocks>** brackets, Enter the following **ClaimsSchema** value:

<ClaimsSchema>

<ClaimType Id="loyaltyNumber">

<DisplayName>Your loyalty number from your membership</DisplayName>

<DataType>string</DataType>

  <AdminHelpText>This value is taken from the REST API</AdminHelpText>

</ClaimType>

</ClaimsSchema>

**EXAMPLE**

<BuildingBlocks>

<ClaimsSchema>

<ClaimType Id="loyaltyNumber">

<DisplayName>Your loyalty number from your membership</DisplayName>

<DataType>string</DataType>

  <AdminHelpText>This value is taken from the REST API</AdminHelpText>

</ClaimType>

</ClaimsSchema>

</BuildingBlocks>

# Exercise 5: Add the Orchestration Step to call the REST API

### Scenario

In this exercise, you will add the orchestrated step to trigger the REST API after a successful sign-in or sign-up flow. This is done within the UserJourney that is contained within the TrustFrameworkExtension.xml file. This step is required in order for the REST API ClaimsProvider configured earlier to be triggered.

1. **Navigate** to the TrustFrameworkExtensions.xml file.
2. Scroll at the bottom of the file until you see the OrchestrationSteps listed within the **<UserJourneys><UserJourneys/>**
3. We will be adding a new OrchestrationStep the REST-API-SignUp between the existing Order “6” and ”7”, which will leave us with a *total of 8 Orchestration Steps*.
   1. Input the following:

<OrchestrationStep Order="6" Type="ClaimsExchange">

<Preconditions>

<Precondition Type="ClaimsExist" ExecuteActionsIf="true">

<Value>objectId</Value>

<Action>SkipThisOrchestrationStep</Action>

</Precondition>

</Preconditions>

<ClaimsExchanges>

<ClaimsExchange Id="AADUserWrite" TechnicalProfileReferenceId="AAD-UserWriteUsingAlternativeSecurityId"/>

</ClaimsExchanges>

</OrchestrationStep>

<OrchestrationStep Order="7" Type="ClaimsExchange">

<ClaimsExchanges>

<ClaimsExchange Id="AnyValue" TechnicalProfileReferenceId="REST-API-SignUp" />

</ClaimsExchanges>

</OrchestrationStep>

<OrchestrationStep Order="8" Type="SendClaims" CpimIssuerTechnicalProfileReferenceId="JwtIssuer"/>

* 1. Remember to update the last OrchestrationStep to be in sequence with the ordering, which is changing “7” to “8” as highlighted above.

# Exercise 6: Add the loyaltyNumber claim to be added in the JWT token

### Scenario

In this exercise, you will be adding the outbound **loyaltyNumber** claim to be added to the JWT token. This configuration tells Azure AD B2C explicitly to send the specific claim to your application. This is done by adding an **OutputClaim** to the **SignUpOrSignIn policy**. This allows Azure AD B2C to send the loyalty Number to your application/service.

1. **Navigate** to the SignUpOrSignin.xml file.
2. **Add** the text <OutputClaim ClaimTypeReferenceId=“loyaltyNumber” /> within the **<OutputClaims></OutputClaims>** brackets

**EXAMPLE**

<OutputClaims>

<OutputClaim ClaimTypeReferenceId="displayName" />

<OutputClaim ClaimTypeReferenceId="givenName" />

<OutputClaim ClaimTypeReferenceId="surname" />

<OutputClaim ClaimTypeReferenceId="email" />

<OutputClaim ClaimTypeReferenceId="objectId" PartnerClaimType="sub"/>

<OutputClaim ClaimTypeReferenceId="identityProvider" />

<OutputClaim ClaimTypeReferenceId="loyaltyNumber" />

</OutputClaims>

# Exercise 7: Upload the policies

### Scenario

In this exercise, you will be uploading the custom policies (XML files) to Azure AD B2C admin console

1. Select the **Identity Experience Framework** menu item in your B2C tenant in the Azure portal.
2. Select **Upload custom policy**.
3. Select **Overwrite the custom policy if it already exists. (Important step, do not skip)**
4. In this order, upload the policy files:
   1. *TrustFrameworkBase.xml*
   2. *TrustFrameworkExtensions.xml*
   3. *SignUpOrSignin.xml*
   4. *ProfileEdit.xml*
   5. *PasswordReset.xml*

As you upload the files, Azure adds the prefix **B2C\_1A\_** to each.

A screenshot of a social media post

Description automatically generated

# Exercise 8: Test the custom policy

### Scenario

Follow the same steps as [described above located here](#_Step_4:_Test).

# Reference Documentation

* [Set up sign-up and sign-in with a Facebook account using Azure Active Directory B2C](https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-setup-fb-app)