Getting Started with the FranceConnect Facade (FCF)

Zzz

A developer guide to configure and test the facade

Version Beta - December 2022

This page is intentionally left blank.

Notice

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example 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.

A drawing of a face

Description automatically generated

Attribution 4.0 International (CC BY 4.0)

Microsoft and any contributors grant you a license to this document under the [Creative Commons Attribution 4.0 International Public License](https://creativecommons.org/licenses/by/4.0/legalcode), see the [LICENSE](https://github.com/microsoft/franceconnect-facade-dotnet-webapp-aspnetcore/blob/main/LICENSE-CODE) file, and grant you a license to any code in the repository under the MIT License, see the [LICENSE-CODE](https://github.com/microsoft/franceconnect-facade-dotnet-webapp-aspnetcore/blob/main/LICENSE-CODE) file.

Microsoft, Windows, Microsoft Azure and/or other Microsoft products and services referenced in the document may be either trademarks or registered trademarks of Microsoft in the United States and/or other countries. The license for this document does not grant you rights to use any Microsoft names, logos, or trademarks. Microsoft's general trademark guidelines can be found at <http://go.microsoft.com/fwlink/?LinkID=254653>.

Privacy information can be found at <https://privacy.microsoft.com/en-us/>

Microsoft and any contributors reserve all other rights, whether under their respective copyrights, patents, or trademarks, whether by implication, estoppel or otherwise.

Content

[About this guide 5](#_Toc126335200)

[Guide elements 5](#_Toc126335201)

[Guide prerequisites 5](#_Toc126335202)

[Fulfill the prerequisites for your testing environment 7](#_Toc126335203)

[Fulfill the prerequisites for your Azure testing environment 7](#_Toc126335204)

[Fulfill the prerequisites for your local environment. 9](#_Toc126335205)

[Setup the FranceConnect Facade 16](#_Toc126335206)

[Configure the FranceConnect Facade with a development certificate 16](#_Toc126335207)

[Configure the FranceConnect Facade with a key vault 21](#_Toc126335208)

[Test the FranceConnect Facade 30](#_Toc126335209)

[Run the facade code locally 30](#_Toc126335210)

[Deploy the facade on the Internet 45](#_Toc126335211)

[Appendix. Deploy the FranceConnect Facade resources with Bicep scripts 56](#_Toc126335212)

[Fulfill the prerequisites 56](#_Toc126335213)

[Deploy the resources 56](#_Toc126335214)

# About this guide

As the title of the Getting Started Guide (implicitly) suggests, this document takes you and guides you through all the steps required to prepare a suitable local development environment, to deploy and set all the needed resources in the Azure Cloud for the production environment, as well as a portal application, etc. to ultimately configure and test the FranceConnect Façade (FCF) code.

With this knowledge, you should be able to further adapt the provided code if needed to accommodate your own specific situation and/or target a specific deployment in your production environment whatever it is.

## Guide elements

For the sake of this Getting Started Guide, and considering the above, we provide the following elements:

* Hands-on instructionscontaining the most important steps and their outputs. These are meant to show you the core elements.
* FranceConnect Façade (FCF) sample codefor implementing and deploying a suitable facade so that Microsoft BizApps’ portal can interact with the FranceConnect platform. You can access this code by downloading or cloning the following GitHub repository: [microsoft/franceconnect-facade-dotnet-webapi-aspnetcore (github.com)](https://aka.ms/franceconnect).

Let’s start with the prerequisites.

## Guide prerequisites

We assume here that you have a Windows 10 local environment or above. For example, Windows 11 in our illustration.

### Install Git

Git is a [free and open source](https://git-scm.com/about/free-and-open-source) distributed version control system designed to handle everything from small to very large projects with speed and efficiency

You can download and install Git:

* Download the [Git for Windows](https://github.com/git-for-windows/git/releases/download/v2.25.0.windows.1/Git-2.25.0-64-bit.exe) and run it.
* In the pop-up window, click Install. Follow the instructions. When prompted, select Use the OpenSSL library.

Git will help us to clone the sample code project and related resources to complete this guide.

### Clone the repo using Git

To clone the FCF sample code’s project on your Windows 11 local machine, open a PowerShell console, and run the following commands:

PS C:\> cd c:\

PS C:\> git clone <https://github.com/microsoft/franceconnect-facade-dotnet-webapi-aspnetcore.git>

PS C:\> cd franceconnect-facade-dotnet-webapi-aspnetcore

The FCF sample code is provided in [.NET](https://github.com/Azure-Samples/active-directory-verifiable-credentials-dotnet) 6.

**Repository organization**

**The repo contains the FCF sample code under the *Source* folder whereas the *Scripts* folder provides you the script to help you create and configure the required resources in the Azure Cloud.**

**Une image contenant texte, moniteur, capture d’écran, noir

Description générée automatiquement**

**So, at this stage, you’re all set! It’s high time to move on to the first module.**

# Fulfill the prerequisites for your testing environment

In order to complete this walkthrough, you'll need to start fulling a set of prerequisites for both your Azure testing environment as well as your local development environment.

Let’s consider these prerequisites in order.

## Fulfill the prerequisites for your Azure testing environment

### Azure subscription

An [Azure subscription](https://azure.microsoft.com/en-gb/free/search/?OCID=AID2200274_SEM_8f466b779664190b197209460a3488a4:G:s&ef_id=8f466b779664190b197209460a3488a4:G:s&msclkid=8f466b779664190b197209460a3488a4) is required to complete this guide. If you don't have Azure subscription, [create one for free](https://azure.microsoft.com/free/?WT.mc_id=A261C142F).

See [Build in the cloud with an Azure account](https://azure.microsoft.com/en-gb/free/search/?OCID=AID2200274_SEM_9e6e31d5fb0f107e3857539f5a4b2b2b:G:s&ef_id=9e6e31d5fb0f107e3857539f5a4b2b2b:G:s&msclkid=9e6e31d5fb0f107e3857539f5a4b2b2b) to subscribe.

**All the resources of this walkthrough will be created in the FranceCentral region. The related data located and processed in Europe.** Learn more about [data residency](https://azure.microsoft.com/en-us/global-infrastructure/data-residency/#select-geography) in Azure and our commitment to protecting customer data as part of the [Microsoft EU Data Boundary](https://techcommunity.microsoft.com/t5/security-compliance-and-identity/eu-data-boundary-for-the-microsoft-cloud-frequently-asked/ba-p/2329098) program. Additional information are available in the "[transparency documentation](https://learn.microsoft.com/en-us/privacy/eudb/eu-data-boundary-learn)" on the EU Data Boundary Trust Center Page.

**You can opt to another region in Europe (or elsewhere) if needed.**

### Portal application

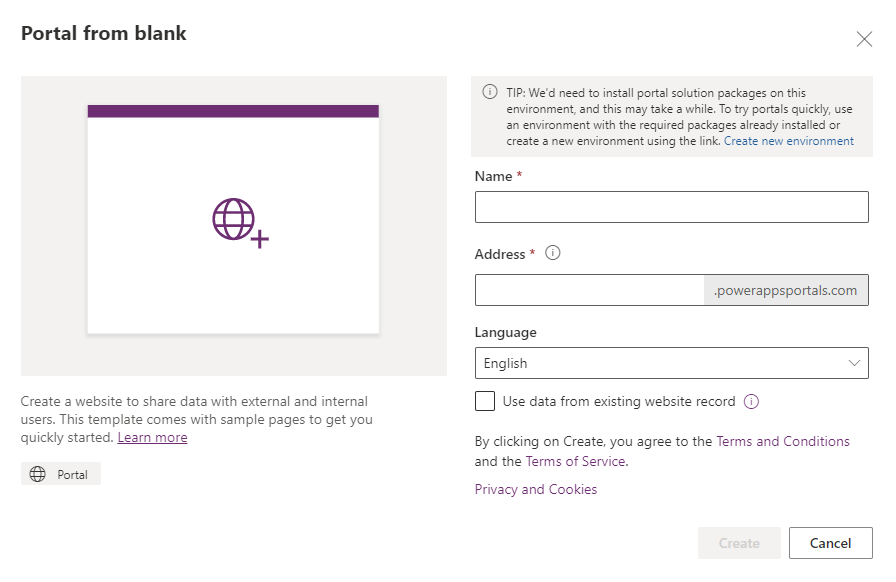
A Power Apps Portal[[1]](#footnote-2),[[2]](#footnote-3) application, is also required to test the FranceConnect facade. With such an application, referred as to a Power Pages application in the rest of this document, you can now create a powerful new type of experience: external-facing websites that allow users outside their organizations to sign in with a wide variety of identities – The purpose of the FranceConnect Facade is to activate the use of the FranceConnect platform’s identities -, create and view data in [Microsoft Dataverse](https://www.bing.com/ck/a?!&&p=f29913e84db5c23fJmltdHM9MTY2NTYxOTIwMCZpZ3VpZD0xNjFkOTk2Yi03MzFlLTZhNDEtMGE4Yy04ODg0NzcxZTY0NzMmaW5zaWQ9NTIwMQ&ptn=3&hsh=3&fclid=161d996b-731e-6a41-0a8c-8884771e6473&psq=Microsoft+Dataverse&u=a1aHR0cHM6Ly9sZWFybi5taWNyb3NvZnQuY29tL2VuLXVzL3Bvd2VyLWFwcHMvbWFrZXIvZGF0YS1wbGF0Zm9ybS9kYXRhLXBsYXRmb3JtLWludHJv&ntb=1), or even browse content anonymously. See [What are Power Apps portals?](https://learn.microsoft.com/en-us/power-apps/maker/portals/overview).

If you don't have any Power Apps Portal, [sign-up first](https://signup.microsoft.com/create-account/signup?sku=5b631642-bd26-49fe-bd20-1daaa972ef80&origin=powerappscommunity&ru=https:%2F%2Fmake.powerapps.com&products=5b631642-bd26-49fe-bd20-1daaa972ef80&ali=1&cmpid=pap-deve-hero-gst) for the [Power Apps Developer Plan](https://powerapps.microsoft.com/en-us/developerplan/).

The Power Apps Developer Plan gives you a free development environment to create, build and test apps without writing code, with full-featured Power Apps development tools. Anyone with a work or school email address backed by Azure Active Directory (Azure AD) can sign up for the Power Apps Developer Plan. If you don’t have any Azure AD account yet, see section Create your Azure AD tenant below to create one. See [Sign up for Power Apps Developer Plan](https://learn.microsoft.com/en-us/power-apps/maker/developer-plan).

Create a new Power Pages application from your (newly created) [Power Apps environment](https://make.powerapps.com/). Proceed with the following steps:

1. Sign in to [Power Apps](https://make.powerapps.com/). (To access the preview experience, sign in to [Power Pages](https://make.powerpages.microsoft.com/) instead).
2. From the right-upper corner, select the Environment drop-down to verify, or choose the Power Apps environment for your portal.
3. From the left pane, select Apps.
4. Click + New app, and select Website. A Portal from blank dialog pops up.



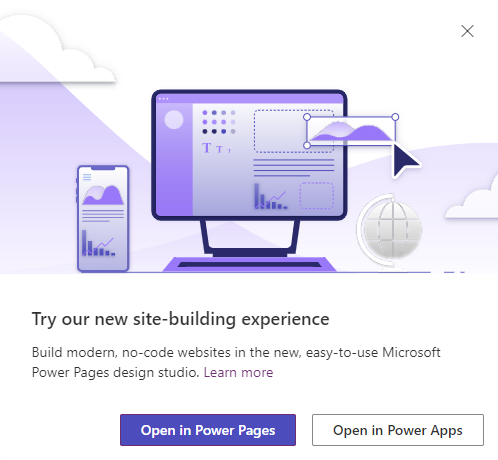
1. In Name, type a name for your application. For example, “*Litware 369*” in our illustration. Litware 369 is a fictitious FranceConnect eligible entity.
2. In Address, specify a subdomain for this portal. For example, “*litware369*” in our illustration.
3. Click Create. Your portal is being provisioned.
4. Select your portal and sign-in. When prompted, give your consent to grant the requested permissions.

Une image contenant texte

Description générée automatiquement

At this stage, your portal is up and running.

1. Back in your environment, select More Commands (…) for the portal and select Edit from the context menu to build your experience.



For more details, see [Power Apps portals documentation](https://learn.microsoft.com/en-us/power-apps/maker/portals/).

## Fulfill the prerequisites for your local environment.

In order to complete this walkthrough, you'll also need in your local development environment:

1. [Visual Studio Community](https://visualstudio.microsoft.com/vs/community/).
2. [Azure Active Directory IdentityModel Extensions for .NET](https://github.com/AzureAD/azure-activedirectory-identitymodel-extensions-for-dotnet).
3. [Azure Identity client library for .NET](https://github.com/Azure/azure-sdk-for-net/blob/main/sdk/identity/Azure.Identity).
4. [Azure Key Vault Secrets Configuration Provider](https://github.com/Azure/azure-sdk-for-net/blob/Azure.Extensions.AspNetCore.Configuration.Secrets_1.2.2/sdk/extensions/Azure.Extensions.AspNetCore.Configuration.Secrets/README.md).
5. [Azure command-line interface (Azure CLI)](https://learn.microsoft.com/en-us/cli/azure/).
6. [ngrok](https://ngrok.com/) to expose on the Internet through an instant, secure URL the above local Node server if needed.
7. [OpenSSL](http://www.openssl.org/) to issue a X.509 development certificate. (Other options on Windows for creating this certificate includes the [PowerShell New-SelfSignedCertificate cmdlet](https://learn.microsoft.com/en-us/powershell/module/pki/new-selfsignedcertificate).)
8. And optionally [Fiddler](https://www.telerik.com/fiddler) as a remote proxy to capture the traffic.

Throughout this guide, we assume you have a Windows local machine for the purpose of the illustration.

In addition, you will also (optionally) need to have a mobile device with [Microsoft Authenticator](https://www.microsoft.com/en-us/account/authenticator):

* Android version 6.2108.5654 or later installed.
* iOS version 6.5.82 or later installed.

Let’s see how to install the above prerequisites.

### Install Visual Studio Community

We will use Visual Studio Community through this guide, but you are free to use the integrated development environment (IDE) of your choice instead.

Visual Studio Community is a fully-featured, extensible, free IDE for creating modern applications for Android, iOS, Windows, as well as web applications and cloud services. Visual Studio Community runs on your desktop and is available for your Windows local machine. It comes with built-in support for .NET in C#, F# or Visual Basic, C, C++, Python, JavaScript/TypeScript and Node.js.

Proceed with the following steps:

1. [Download](https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=Community&channel=Release&version=VS2022&source=VSLandingPage&passive=false&cid=2030) Visual Studio Community on your local machine.
2. Install Visual Studio Community. You can save installation time and disk space by [selecting just the components needed](https://docs.microsoft.com/visualstudio/install/install-visual-studio). You can always incrementally add more components later at any time as needed.

### Install Azure Active Directory IdentityModel Extensions for .NET

The Azure Active Directory IdentityModel Extensions for .NET provide libraries that are in particular used part of ASP.NET security to [validate tokens](https://github.com/AzureAD/azure-activedirectory-identitymodel-extensions-for-dotnet/wiki/ValidatingTokens) in ASP.NET Web Apps and Web APIs. For that purpose, they include types that provide support for security tokens and cryptographic operations: signing, verifying signatures, and encryption.

Proceed with the following steps:

1. Open either a Windows command prompt or a PowerShell window.
2. Run the following command to install the package with [NuGet](https://www.nuget.org/):

PS C:\> dotnet add package Microsoft.IdentityModel.Tokens

### Install Azure Identity client library for .NET

The Azure Identity library provides Azure Active Directory (Azure AD) token authentication support across the Azure SDK. It provides a set of TokenCredential implementations, such as the VisualStudioCredential or the DefaultAzureCredential. See next section below.

Proceed with the following steps:

1. Open either a Windows command prompt or a PowerShell window.
2. Run the following command to install the package with [NuGet](https://www.nuget.org/):

PS C:\> dotnet add package Azure.Identity

1. This package installs a HTTPS ASP.NET Core development certificate. You now need to [trust this certificate](https://learn.microsoft.com/en-us/aspnet/core/security/enforcing-ssl?view=aspnetcore-6.0&tabs=visual-studio#trust-the-aspnet-core-https-development-certificate-on-windows-and-macos) on your local developer environment. Run the following command to approve it.

PS C:\> dotnet dev-certs https --trust

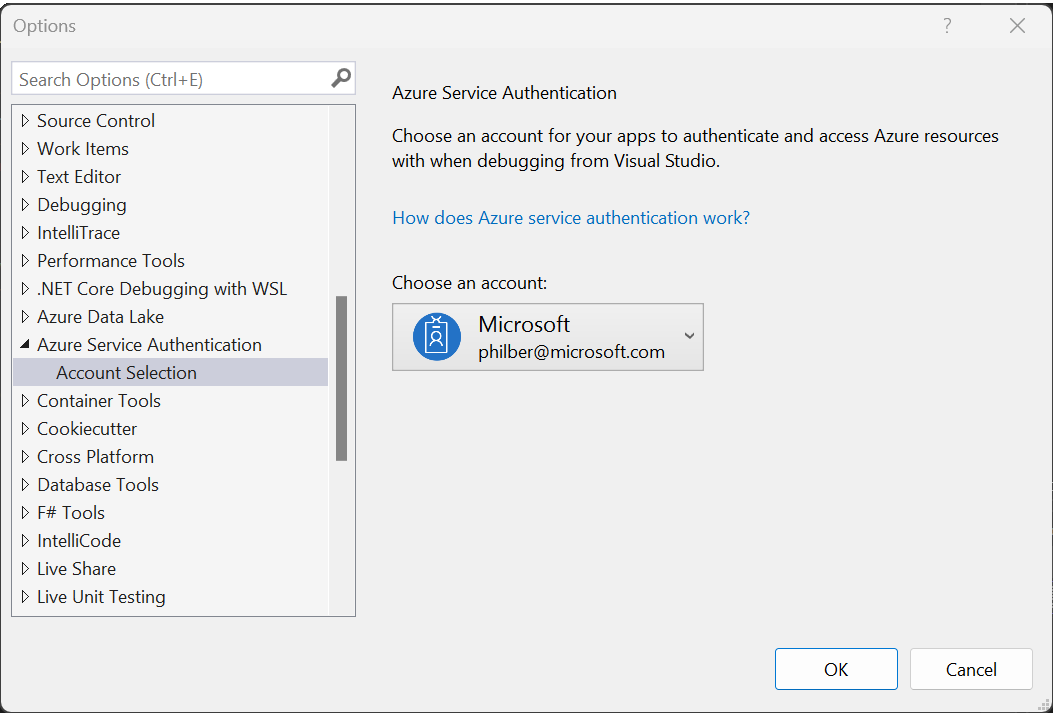
### Configure Visual Studio Community

When debugging and executing code locally it is typical for a developer to use their own Azure Active Directory account for authenticating calls to Azure services, here your key vault.

As a developers using Visual Studio Community 2022, you can authenticate an Azure AD account through the development environment. The FranceConnect Façade, that is using either the VisualStudioCredential or the DefaultAzureCredential (see below) can then in turn this account to authenticate calls in its code when running locally vs. elsewhere once deployed in the production environment, e.g., in the Azure Cloud for production.

To authenticate in Visual Studio Community, proceed with the following steps:

1. Launch Visual Studio Community.
2. Select TOOLS > Options… An Options dialog opens up.
3. Navigate to Azure Service Authentication and select Account Selection underneath to sign in with your Azure AD account.
4. Under Choose an account, click Sign In… and sign in with your credentials.



1. Click OK.

When running locally in debug mode from the IDE, the FranceConnect Façade leverages the VisualStudioCredential as illustrated in the code snippet hereafter to use this account to authenticate calls in the code. For more details, see [VisualStudioCredential Class (Azure.Identity)](https://learn.microsoft.com/en-us/dotnet/api/azure.identity.visualstudiocredential?view=azure-dotnet).

#if (DEBUG)

[…]

VisualStudioCredentialOptions options = new VisualStudioCredentialOptions

{

// Paramètre TenantId Azure Active Directory de l'abonnement

// Azure qui contient le Key Vault

TenantId = configuration["TenantId"],

};

[…]

// Le compte utilisé est celui qui est connecté à Visual Studio et

// utilisé dans le paramètre "Azure Service Authentification" des options

// Visual Studio.

// D'autre part il faut que ce compte ai les droits d'accès au Key vault.

// https://learn.microsoft.com/en-us/azure/key-vault/general/assign-access-policy?tabs=azure-portal

VisualStudioCredential VisualStudioCredential = new VisualStudioCredential(options);

#else

Conversely, you are no longer in the debug mode, i.e., when the production environment is targeted, the DefaultAzureCredential is used instead. See below.

As such, the DefaultAzureCredential is appropriate for most scenarios where the code is intended to ultimately be run in the Azure Cloud for production. This is because the DefaultAzureCredential determines the appropriate credential type based of the environment it is executing in. It supports authenticating both as a service principal or managed identity, and can be configured so that it will work both in a local development environment or when deployed to the cloud. For more details, see [DefaultAzureCredential Class (Azure.Identity)](https://learn.microsoft.com/en-us/dotnet/api/azure.identity.defaultazurecredential?view=azure-dotnet) and [DefaultAzureIdentity And Its Various Credential Types](https://www.c-sharpcorner.com/article/defaultazureidentity-and-its-various-credential-types3/).

The DefaultAzureCredential will first attempt to authenticate using credentials provided in the environment. (In a development environment you can authenticate as a service principal with by providing configuration in environment variables.)

However, if the environment configuration is not present or incomplete, the DefaultAzureCredential will then determine if a managed identity is available in the current environment. Authenticating as a managed identity requires no configuration, but does require platform support. For more details, see [Azure Services with managed identities support](https://learn.microsoft.com/en-us/azure/active-directory/managed-identities-azure-resources/managed-identities-status). This is the route we follow here thanks to with Azure Key Vault support.

Please note that DefaultAzureCredential is intended to simplify getting started with the Azure development by handling common scenarios with reasonable default behaviors. Developers who want more control or whose scenario isn't served by the default settings should use other credential types.

### Install Azure Key Vault Secrets Configuration Provider

[Azure Key Vault](https://azure.microsoft.com/en-us/products/key-vault/#product-overview) is a cloud service that provides secure storage and automated management of secrets, keys, and X.509 certificates.

Azure Key Vault offers a tight integration with ASP.NET Core by way of a Configuration Provider that plugs into the IConfiguration system in ASP.NET Core. You can find more details in the article [Azure Key Vault configuration provider in ASP.NET Core](https://learn.microsoft.com/en-us/aspnet/core/security/key-vault-configuration?view=aspnetcore-6.0), but in a nut shell the provider package offers a single method called AddAzureKeyVault that allows you to specify your key vault name and how to authenticate, here with the DefaultAzureCredential, see above section.

#else

if (builder.Environment.IsProduction())

{

configuration.AddAzureKeyVault(new Uri(AzureKeyVaultEndpoint),

new DefaultAzureCredential());

}

#endif

The Azure Key Vault Secrets Configuration Provider allows storing configuration values using Azure Key Vault secrets.

Proceed with the following steps:

1. Open either a Windows command prompt or a PowerShell window.
2. Run the following command to install the package with NuGet:

PS C:\> dotnet add package Azure.Extensions.AspNetCore.Configuration.Secrets

### Install Azure CLI

The Azure command-line interface (Azure CLI) is a cross-platform command-line tool used to create and manage Azure resources. The Azure CLI is available across Azure services and is designed to get you working quickly with Azure, with an emphasis on automation. This will allow you to both create and configure a key vault and X.509 certificates in Azure Key Vault.

The Azure CLI can also be useful for authenticating in a development environment, creating accounts, and managing account roles. It will be indeed used later to create a user-managed identity in your Azure AD tenant.

Proceed with the following steps:

1. Download and install the [latest release of the Azure CLI](https://aka.ms/installazurecliwindows) .mis file, i.e., *azure-cli-2.40.0.msi* as this writing - The current version of the Azure CLI is 2.40.0 -. Follow the installer’s steps. When the installer asks if it can make changes to your computer, click the Yes box.
2. When you first use Azure CLI, you might need to install Azure CLI extensions. Azure CLI indeed offers the capability to load extensions. CLI extensions are characterized as Python wheels that aren't shipped as part of the CLI but run as CLI commands. See [Use extensions with the Azure CLI](https://learn.microsoft.com/en-us/cli/azure/azure-cli-extensions-overview).

Run the following command to enable dynamic install without a prompt.

PS C:\> az config set extension.use\_dynamic\_install=yes\_without\_prompt

When you run an extension command that is not installed, Azure CLI can recognize the command you run, and automatically install the extension for you.

1. After installing the CLI for the first time, open either a Windows command prompt or PowerShell window and check that it's installed and you've got the correct version by running:

PS C:\> az version

To connect to Azure with an authenticated account for use with commands from Azure CLI, perform the following steps:

1. Open either a Windows command prompt or PowerShell window, and run the login command.

PS C:\> az login

If the CLI can open your default browser, it will initiate an [authorization code flow](https://learn.microsoft.com/en-us/azure/active-directory/develop/v2-oauth2-auth-code-flow) and open the default browser to load an Azure sign-in page.

Otherwise, it will initiate a [device code flow](https://learn.microsoft.com/en-us/azure/active-directory/develop/v2-oauth2-device-code) and tell you to open a browser page at <https://aka.ms/devicelogin> and enter the code displayed in your terminal.

If no web browser is available or the web browser fails to open, you may force device code flow with az login --use-device-code.

See [Sign in with Azure CLI — Login and Authentication](https://learn.microsoft.com/en-us/cli/azure/authenticate-azure-cli).

1. Sign in with your account credentials in the browser.
2. After logging in, you see a list of subscriptions associated with your Azure account. The subscription information with isDefault: true is the currently activated subscription after logging in.

If needed, set the other subscription ID that you’d like to switch to for the sake of this walkthrough. See [How to manage Azure subscriptions – Azure CLI](https://learn.microsoft.com/en-us/cli/azure/manage-azure-subscriptions-azure-cli).

PS C:\> az account set <*your\_subscription\_id*>

Where <*your\_subscription\_id*> is the ID of your subscription retrieved from the Azure portal. For example in our illustration :

PS C:\> az account set "8848a529-9d69-4049-8469-8218547a61e2"

For more details, see [Get started with Azure Command-Line Interface (CLI)](https://learn.microsoft.com/en-us/cli/azure/get-started-with-azure-cli) .

### Install ngrok

ngrok is a cross-platform application that can create a tunnelling or forwarding URL, so that internet requests reach your local machine for local development.

When needed, you will use this tool to expose our localhost on a public URL.

See [ngrok - secure introspectable tunnels to localhost](https://ngrok.com/download) to download and install ngrok. If you have [Chocolatey](https://chocolatey.org) installed, you can also use here the [Chocolaty repository manager](https://chocolatey.org/packages?q=ngrok). Open either a Windows command prompt or PowerShell window, and run the following command.

PS C:\> choco install ngrok

Please note that you need to sign up for an account to get and install an Authtoken to use the tool. So, regardless of the above approach, proceed with the following steps:

1. Open a browser session and navigate to the ngrok web site at <https://ngrok.com/>.
2. Click **Sign up for free** and follow the instructions.
3. Back to the home page, now click **Login** and enter your related credentials.
4. On the left pane, select Authtoken to get your personal Authtoken. It is used to authenticate the ngrok agent that you downloaded. Take a note of it.
5. Now open either a Windows command prompt or a PowerShell window and run the following command:

PS C:\> ngrok authtoken <*your\_auth\_token*>

Replace the placeholder by your own value. The Authtoken is saved in the default configuration file.

### Install OpenSSL

OpenSSL is s an open-source command-line tool intended to provide a full-featured toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols. It allows users to perform various TLS/SSL -related tasks.

An easy way to install OpenSSL is by directly using the *openssl.exe* file that comes inside your prior Git for Windows installation as your default OpenSSL, and include that into your path in Windows environmental variables. In our illustration, you will found this file in the following location of your Git for Windows Installation: *C:\Program Files\Git\usr\bin\openssl.exe*.

One should note that the related version of OpenSSL is outdated: OpenSSL 1.1.1d 10 Sep 2019.

So, you can instead decide to install OpenSSL with the latest OpenSSL Windows installer file: OpenSSL 3.0.5 as of this writing.

To do so, proceed with the following steps:

1. Click the link hereafter to visit OpenSSL download page: <http://slproweb.com/products/Win32OpenSSL.html>.
2. Download the system-specific Win64 OpenSSL v3.0.5 (Light) setup file. If you get a warning from Microsoft Defender SmartScreen, keep the *Win64OpenSSL-3\_0\_5.exe* file.
3. Execute the *Win64OpenSSL-3\_0\_5.exe* file. Follow the installation steps. Follow the installation steps and make a note of where this is installed, e.g., by default *C:\Program Files\OpenSSL-Win64\bin*.
4. After installing, you now need to set the environment variables to function OpenSSL properly on your local environment. More specifically, you are required to set the OPENSSL\_CONF and PATH environment variables.
   1. Open Settings from the Windows menu and search for “*environment*”.
   2. Select Edit the system environment variables. A System Properties dialog pops up.
   3. At the bottom of this window, click Environment Variables…
   4. Under System variables, set the OPENSSL\_CONF variable. click to New to add a new variable. Specify the variable name and click on Browse File… to select the *openssl.cfg* file in the OpenSSL directory. Click OK.
   5. Now change the PATH variable (double-click on it or select it and click Edit). The next dialog shows the PATH variable. Select New, then add a new line pointing to your OpenSSL installation folder, e.g. *C:\Program Files\OpenSSL-Win64\bin*.
   6. Click OK twice.
5. Open a Win64 OpenSSL Command Prompt window and run the following command to verify your installation:

C:\> openssl version

OpenSSL 3.0.5 5 Jul 2022 (Library: OpenSSL 3.0.5 5 Jul 2022)

This confirms that OpenSSL is installed and configured in your PATH.

You can also use Chocolatey and the [Chocolaty repository manager](https://chocolatey.org/packages?q=openssl).

Open either a Windows command prompt or PowerShell window, and run the following command.

PS C:\> choco install openssl

This will install OpenSSL 1.1.1.1700 as of this writing.

# Setup the FranceConnect Facade

You can setup the FranceConnect Facade to either locally use a development certificate or to use instead Azure Key Vault to get the certificate. The X.509 certificate will be exposed as part of the FCF facade metadata and be used for signing the security tokens to issue.

The first option while being the simplest one is NOT suitable for a production environment. The guide covers the two options:

1. Configure the FranceConnect Facade with a development certificate.
2. Configure the FranceConnect Facade with a key vault.

## Configure the FranceConnect Facade with a development certificate

### Create a X.509 self-signed certificate

To later configure the FranceConnect Facade, you will need a X.509 signing certificate.

For a production environment, you will typically request such a certificate from a public Certificate Authority (CA) or from your own PKI if any via a certificate signing request (CSR). See section Configure the FranceConnect Facade with a key vault below.

For the sake of this specific configuration, you will use instead OpenSSL to issue a self-signed certificate instead, but with the same CSR so that you can accommodate the related instructions with you own situation. A key length of at least 2048 bits is recommended for the certificate, 1024 bits being the minimum for the length of the RSA key.

#### Create a private key

Let’s start by creating a private key for the certificate. Such a private key helps to enable the signature, and is the most important component that relates to the certificate.

Proceed with the following steps:

1. Open a command prompt where you can execute OpenSSL commands. For example, an Open a Win64 OpenSSL Command Prompt window in our illustration.
2. Execute the following command to create a password-protected, 2048-bit RSA private key named *<your\_certname>*.key:

C:\> openssl genrsa -des3 -out <*your\_certname*>.key 2048

For example, *fcfsigningcert* in our illustration.

When prompted, enter a passphrase. If you want the private key to be unencrypted, simply remove the -des3 parameter from the above command.

#### Create a Certificate Signing Request (CSR)

**You need at this stage a certificate signing request (CSR) to have your certificate signed**. The CSR includes the public key and some additional information (such as your organization and the country where you operate).

Proceed with the following steps:

1. From the previous window, create a CSR named <your\_certname>.csr from the above existing private key:

C:\> openssl req -key <*your\_certname*>.key -new -out <*your\_certname*>.csr

1. When prompted, enter the priorly chosen password to protect the private key password, as well as, when invited, some CSR information to complete the process. The output will look like the following:

Enter PEM pass phrase:

Verifying - Enter PEM pass phrase:

C:\Users\philber\fcf>openssl req -key fcfsigningcert.key -new -out fcfsigningcert.csr

Enter pass phrase for fcfsigningcert.key:

You are about to be asked to enter information that will be incorporated

into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value,

If you enter '.', the field will be left blank.

-----

Country Name (2 letter code) [AU]:FR

State or Province Name (full name) [Some-State]:IDF

Locality Name (eg, city) []:Issy-Les-Moulineaux

Organization Name (eg, company) [Internet Widgits Pty Ltd]:Litware369

Organizational Unit Name (eg, section) []:DSI

Common Name (e.g. server FQDN or YOUR name) []:www.litware369.com

Email Address []:contact@litware369.com

Please enter the following 'extra' attributes

to be sent with your certificate request

A challenge password []:

An optional company name []:Microsoft France

As notice above, an important field is Common Name, which could be the (exact) Fully Qualified Domain Name (FQDN) of our custom domain for the FranceConnect Facade, see section Create a custom domain below. For example www.litware369.com in our illustration.

One should note that, as far as a FranceConnect Facade’s relying party is concerned, here your portal application, and from a security standpoint, this CN for the certificate that will be published in the public metadata of the FranceConnect Facade should be later checked against the audience of the issued id\_token tokens by the facade. See section Configure the public metadata endpoint for the OpenID Connect Discovery Document below.

Both A challenge password and *An optional company name* can be left empty.

**One should note that you can also create both the private key and CSR with a single command:**

C:\> openssl req -newkey rsa:2048 -keyout <*your\_certname*>.key -out <*your\_certname*>.csr

If you want the private key to be unencrypted, you can add the -nodes parameter:

C:\> openssl req -newkey rsa:2048 -nodes -keyout <*your\_certname*>.key -out <*your\_certname*>.csr

#### Create a self-signed certificate

A self-signed certificate is **a certificate that's signed with its own private key**. It can be used to sign data like a security token just as well as CA-signed certificates, but your users will be shown a warning that says the certificate isn't trusted if they’d like to verify the signature.

With that, let's create a self-signed certificate, i.e., the <your\_certname>.crt file, with our existing private key and CSR. Proceed with the following steps. From the previous window, run this command:

C:\> openssl x509 -signkey <*your\_certname*>.key -in <*your\_certname*>.csr -req -sha256 -days 365 -out <*your\_certname*>.crt

Enter pass phrase for fcfsigningcert.key:

Certificate request self-signature ok

subject=C = FR, ST = IDF, L = Issy-Les-Moulineaux, O = Litware369, OU = DSI, CN = "CN=fcf,OU=useraccounts,DC=corp,DC=litware369,DC=com", emailAddress = contact@litware369.com

The -days parameter specifies the number of days that the certificate will be valid. Let’s say one year.

Please note that you can create a self-signed certificate with just a private key:

C:\> openssl req -key <*your\_certname*>.key -new -x509 -sha256 -days 365 -out <*your\_certname*>.crt

**In this case, a temporary CSR will be created and you still will** have the above illustrated CSR information prompt, of course.

**You can even create a private key and a self-signed certificate with just the following single command:**

C:\> openssl req -newkey rsa:2048 -keyout <*your\_certname*>.key -x509 -sha256 -days 365 -out <*your\_certname*>.crt

Regardless of the approach you have opted to, at this stage, you should have:

1. A <your\_certname>.crt file, which is your development certificate. For example, the *fcfsigningcert.crt* file in our illustration. This is**an X.509 certificate that's A ASCII PEM-encoded**.
2. A <*your\_certname*>.key file, which contains the associated private key. For example, the *fcfsigningcert.key* file in our illustration.

At this stage, you can use OpenSSL to convert your development to other formats for multi-purpose use.

#### Convert the development certificate to a PKCS12 file

For a primer on how to convert X.509 certificates’ formats and encoding from one to another, see [PEM, DER, CRT, and CER: X.509 Encodings and Conversions](https://www.ssl.com/guide/pem-der-crt-and-cer-x-509-encodings-and-conversions/).

A PKCS12 file, also known as .pfx file, is an archive file format commonly used to directly store X.509 certificates and private keys, which conforms with the PKCS#12 standard. PKCS#12 is one of the families of standards called Public-Key Cryptography Standards (PKCS) published by RSA Laboratories. (This extension could also be .pkcs12 or .p12.)

Such a file is usually used for importing and exporting certificate chains in a Microsoft environment.

Proceed as follows. From the previous window, combine your development certificate and its private key into a PKCS12 file:

C:\> openssl pkcs12 -export -in <*your\_certname*>.crt -inkey <*your\_certname*>.key -CSP "Microsoft Enhanced RSA and AES Cryptographic Provider" -out <*your\_certname*>.pfx

Enter pass phrase for fcfsigningcert.key:

Enter Export Password:

Verifying - Enter Export Password:

#### Install the certificate in the Windows certificate store

Install the certificate into your current user's personal certificate store. Marking the key as exportable is optional. Make a note the certificate's thumbprint, which is used later.

**With that, let’s now configure the FranceConnect Facade accordingly.**

### Set up the FranceConnect Facade

#### Declare the X.509 signing certificate

Proceed with the following steps:

1. Open the *FranceConnectFacade.sln* solution with Visual Studio Community.
2. Open the *appsettings.json* file.
3. In the UseDevelopmentCertificate section:
   1. Ensure that UseDevelopmentCertificate is set to true. This settings is tested in the code to determine where to get the certificate:

bool UseDevelopmentCertificate = builder.Configuration.IsSettingEnabled("X509:UseDevelopmentCertificate");

* 1. In DevelopmentCertificatePfx, specify the .pfx file path. For example, "*fcfsigningcert.pfx*" in our illustration.

// Si false alors utilise le certificat provenant du Service

// Azure Key vault

// Pour utiliser le certificat de développement vous devez

// au préalablement le créer (voir fichier CreateASelfCert.txt dans ce repo)

"UseDevelopmentCertificate": true,

// Utilisation du certificat de développement auto signé

// Le mot de passe doit être inclus dans le fichier secrets.json

// https://learn.microsoft.com/en-us/aspnet/core/security/app-secrets?view=aspnetcore-7.0&tabs=windows

"DevelopmentCertificatePfx": "<*your\_certname*>.pfx",

// Obligatoire ne pas supprimer

"DevCertificateName": "devcert",

// Utilisation du Service key vault Azure

"CertificateNameKeyVault": "<*your\_certname*>",

"AzureKeyVaultEndpoint": "https://<*your\_key\_vault\_name*>.vault.azure.net/"

1. Save the *appsettings.json* file.
2. Now, right-click the FranceConnectFacade.Identity.WebApi project in the Solution Explorer blade, and select Manage User Secrets. This creates and opens up a new empty JSON file named *secrets.json*.
   1. Copy the following text and paste it.

{

DevelopmentCertificatePassword": "<*your\_PEM\_passphrase*>"

}

* 1. Replace <*your\_PEM\_passphrase*> by the one you specify when creating the certificate if any. For example, “*Your\_Own\_PassPhrase\_Here*” Leave the field empty otherwise.

1. Save the *secrets.json* file.

The *secrets.json* file is located in the folder:

*%UserProfile%\AppData\Roaming\Microsoft\UserSecrets\<projectSecretGuid>*

Where <*projectSecretGuid*> is a randomly generated GUID. For example, in our illustration:

*C:\Users\philber\AppData\Roaming\Microsoft\UserSecrets\22f6cb47-1e01-4674-895b-5234998d9dc0*

The important thing to notice here is that the file is located outside your FCF project directory. In order to "bind" the *secrets.json*file location to your project, Visual Studio added a bit of markup to the *FranceConnectFacade.Identity.WebApi.csproj* file:

<PropertyGroup>

[…]

   <UserSecretsId>22f6cb47-1e01-4674-895b-5234998d9dc0</UserSecretsId>

</PropertyGroup>

#### Target the FranceConnect Integration environment

##### Set the FranceConnect credentials

We will use here the FranceConnect dev account through the available so-called “integration key for public use”, which allows in this environment to test the FranceConnect Facade:

* Client id: '211286433e39cce01db448d80181bdfd005554b19cd51b3fe7943f6b3b86ab6e'
* Client secret: '2791a731e6a59f56b6b4dd0d08c9b1f593b5f3658b9fd731cb24248e2669af4b'

Please note that this free of charge integration key is limited use and has NO support.

Proceed with the following steps:

1. Open the previous *secrets.json* file.
2. Add the following code snippet corresponding to the above-mentioned values at the end preceded by a coma.

"FranceConnect": {

"ClientId": "211286433e39cce01db448d80181bdfd005554b19cd51b3fe7943f6b3b86ab6e",

"ClientSecret": "2791a731e6a59f56b6b4dd0d08c9b1f593b5f3658b9fd731cb24248e2669af4b"

}

The *secrets.json* file should looks like the following one:

{

"DevelopmentCertificatePassword": "Your\_Own\_PassPhrase\_Here",

"FranceConnect": {

"ClientId": "211286433e39cce01db448d80181bdfd005554b19cd51b3fe7943f6b3b86ab6e",

"ClientSecret": "2791a731e6a59f56b6b4dd0d08c9b1f593b5f3658b9fd731cb24248e2669af4b"

}

}

1. Save the file.

You can also specify the above secrets via the Secret Manager tool. For more details, see [Safe storage of app secrets in development in ASP.NET Core](https://learn.microsoft.com/en-us/aspnet/core/security/app-secrets?view=aspnetcore-7.0&tabs=windows).

To do so, proceed with the following command:

1. Open either a Windows command prompt or a PowerShell window and run the following command.
2. Run the following commands:

PS C:\> dotnet user-secrets set "FranceConnect:ClientId" "211286433e39cce01db448d80181bdfd005554b19cd51b3fe7943f6b3b86ab6e"

PS C:\> dotnet user-secrets set "FranceConnect:ClientSecret" "2791a731e6a59f56b6b4dd0d08c9b1f593b5f3658b9fd731cb24248e2669af4b"

##### Target the FranceConnect Production environment

To test the FranceConnect facade in the FranceConnect production environment, it is necessary to obtain prior authorization. The request must be made at the following address: <https://franceconnect.gouv.fr/partenaires>.

**The related configuration in not further cover in this Getting Started guide excepted from an App Service perspective, see section Deploy the facade on the Internet below.**

## Configure the FranceConnect Facade with a key vault

### Create your Azure AD tenant

**To get started, you first need an Azure AD tenant. This tenant will allow you to both setup a developer account and a user-managed identity to later used it with your key vault.**

[Create a new tenant in Azure AD](https://learn.microsoft.com/en-us/azure/active-directory/fundamentals/active-directory-access-create-new-tenant) if you do not already have one or if you want to create a new one for ant development tasks that related or pertain to the FCF facade. You can use the [directory creation experience](https://portal.azure.com/#create/Microsoft.AzureActiveDirectory) in the Aure portal.

You'll provide the following information to create your new tenant:

* Tenant type. select Azure AD.
* Organization name.
* Initial domain: the initial domain <*your\_domainname*>.onmicrosoft.com can't be edited or deleted. You can add a customized domain name later.
* Country or region. Specify France.

When you create a new Azure AD tenant, you become the first user of that tenant. As the first user, you're automatically assigned the Global Admin role.

Your Azure subscription must have the above Azure AD tenant as its directory. If it doesn't, you need to [switch directory](https://docs.microsoft.com/en-us/azure/role-based-access-control/transfer-subscription) for the considered subscription. You also need to ensure that you have the [global administrator](https://docs.microsoft.com/en-us/azure/active-directory/roles/permissions-reference#global-administrator) permission for this directory.

At this stage, we assume that:

* You have an Azure AD tenant with an active Azure subscription and that your local machine fulfills the prerequisites outlined in the previous section Fulfill the prerequisites for your testing environment.
* You have the [global administrator](https://docs.microsoft.com/en-us/azure/active-directory/roles/permissions-reference#global-administrator) permission for the directory.

With that in place, let’s see how to (further) configure your Azure AD tenant and other related Azure resources to be created.

Specifically, you will need to complete the following steps:

* Create a user-assigned managed identity for the FranceConnect Facade.
* Create a Azure Key Vault instance to store the X.509 signing certificate used by the FranceConnect Facade, and thus set the appropriate access policies.
* Create a certificate in your key vault.

Once the above Azure components are in place, you will need to configure the code for the FranceConnect Facade.

**The above steps are covered below in order for setting up the FranceConnect Facade step by step, and whenever necessary, additional considerations and/explanations are provided.**

While Azure CLI cmdlets are widely used in these steps to ease the setup and the configuration of your environment, one should note the availability of an Azure Resource Manager (ARM) template along with a series of [Bicep scripts](https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/overview?tabs=bicep) to completely automate both the setup and the configuration. See Appendix. Deploy the FranceConnect Facade resources with Bicep scripts below.

You will also be able to leverage in a near future instead an Azure Blueprint. Please note that the resources are provided as-is as part the FranceConnect Facade repo.

The objectives we pursue here is to help you understand the setup and the configuration of the needed components for the FranceConnect Facade, so we do not leverage that ARM template/Azure blueprint.

### Configure your Azure AD tenant

#### Take a note of your Tenant ID

The Tenant ID will be used to later connect to your Azure AD tenant.

Proceed with the following steps:

1. Open a browser session and navigate to Azure portal at <https://portal.azure.com>.
2. Sign-in with your administrative credentials.
3. Go to **Azure Active Directory** for the subscription you will use for your Microsoft Entra Verified ID deployment.
4. Under **Manage**, select **Properties**.
5. Take a note of the Tenant ID. For example, 164f6854-a553-4084-9dc8-499819f6fceb in our illustration.

#### Specify your Tenant ID in your secrets.json file

Proceed with the following steps:

1. Open the *secrets.json* file.
2. Add the following line at the end preceded by a coma.

"TenantId": "164f6854-a553-4084-9dc8-499819f6fceb"

The *secrets.json* file should looks like the following one:

{

"DevelopmentCertificatePassword": "Your\_Own\_PassPhrase\_Here",

"FranceConnect": {

"ClientId": "211286433e39cce01db448d80181bdfd005554b19cd51b3fe7943f6b3b86ab6e",

"ClientSecret": "2791a731e6a59f56b6b4dd0d08c9b1f593b5f3658b9fd731cb24248e2669af4b"

},

"TenantId": "164f6854-a553-4084-9dc8-499819f6fceb"

}

1. Save the file.

You can use instead the Secret Manager tool with the following command:

C:\> dotnet user-secrets set "TenantId" "164f6854-a553-4084-9dc8-499819f6fceb"

#### Set up the account for Microsoft Authenticator (optional)

We do recommend to have Microsoft Authenticator setup for your user account created in your tenant as you were assigned the Global Admin role.

To enable Microsoft Authenticator, proceed with the following steps:

1. On your mobile device (see section Fulfill the prerequisites for your local environmentabove), open Microsoft Authenticator, go to the Authenticator tab at the bottom and tap **+** sign to add a new account. Select **Work or school account**.
2. At the prompt, select **Sign in**. Don't select **Scan QR code**.
3. Click **+**, and sign in with your user’s credentials in your Azure AD tenant.
4. Microsoft Authenticator will launch <https://aka.ms/mfasetup> in the browser on your mobile device. You will need to sign in again with your test user’s credentials.
5. In the **Set up your account** in the app, select **Pair your account to the app** **by clicking this link**. The Microsoft Authenticator app and opens and you see your test user as an added account.

Note that if <https://aka.ms/mfasetup> launches without prompting you to sign in, this means you have already set up authenticator for another user on this device. When already configured with a user, Microsoft Authenticator signs you in automatically. Sign out the browser's currently logged in user and then repeat the steps above. If you zoom in on the page, you find the **Sign out** button at the top right corner.

### Create a user-assigned managed identity for the FranceConnect Facade

For more details, see [Managed identities for Azure resources](https://learn.microsoft.com/en-us/azure/active-directory/managed-identities-azure-resources/overview).

A common challenge for developers is the management of certificates, keys, secrets, etc. Managed identities eliminate the need to manage these credentials.

More specifically, while you can securely store your certificate in Azure Key Vault as you will do, a service like the FranceConnect Facade needs a way to access Azure Key Vault. Managed identities provide an automatically managed identity in your Azure AD tenant for services and applications to use when connecting to resources that support Azure AD authentication. As such, a service can use managed identities to obtain Azure AD tokens without having to manage any credentials. This is the path you take here.

To create a user-assigned managed identity, your account needs the [Managed Identity Contributor](https://learn.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#managed-identity-contributor) role assignment.

Proceed with the following steps:

1. Open either a Windows command prompt or a PowerShell window, and run the az login command to authenticate, and switch to the target Azure subscription if needed as per section Install Azure CLIabove.

PS C:\> az login

1. Create a resource group for all the FranceConnect Facade resources.

PS C:\> az group create --name <*your\_resource\_group\_name*> --location francecentral

Replace the placeholder value in brackets with your own, unique name. For example, in our illustration:

PS C:\> az group create --name FranceConnectFacadeTest --location "France Central"

1. Now create a user-assigned managed identity.

PS C:\> az identity create --name <*your\_user\_managed\_identity\_name*> --resource-group <*your\_resource\_group\_name*> --location "France Central"

Replace the placeholder values in brackets with your own, unique names. When you create user-assigned managed identities, only alphanumeric characters (0-9, a-z, and A-Z) and the hyphen (-) are supported. For the assignment to a virtual machine or virtual machine scale set to work properly, the name is limited to 24 characters. For more information, see FAQs and known issues.

For example, in our illustration:

PS C:\> az identity create --name FranceConnectFacadeId --resource-group FranceConnectFacadeTest --location "France Central"

The JSON response contains details for the user-assigned managed identity created. The resource ID value assigned to the user-assigned managed identity is used in the following steps. For example, here 3fedf722-7c5d-426f-9d35-d985d3eb59bc in our configuration.

Make of note of it. We will later referred to it as of <*your\_managed\_identity\_client\_Id*>.

You can also see that your user-assigned managed identity has the Microsoft.ManagedIdentity/userAssignedIdentities value returned for the key type.

You can list user-assigned managed identities in your resource group with this command:

PS C:\> az identity list --resource-group <*your\_resource\_group\_name*>

Replace the placeholder value in brackets with your own value.

This user-assigned managed identity can be in turn assigned to the DefaultAzureCredential.

// When deployed to an azure host, the default azure credential will authenticate the specified user assigned managed identity.

string userAssignedClientId = "<*your managed identity client Id*>";

var credential = new DefaultAzureCredential(new DefaultAzureCredentialOptions {

ManagedIdentityClientId = userAssignedClientId });

// Create a secret client using the DefaultAzureCredential

var client = new SecretClient(new Uri("https://yourvault.vault.azure.net/"), new DefaultAzureCredential());

In addition to configuring <*your\_managed\_identity\_client\_Id*> via code, it can also be set using the AZURE\_CLIENT\_ID environment variable. These two approaches are equivalent when using the DefaultAzureCredential.

### Create a Azure Key Vault instance

[Azure Key Vault](https://azure.microsoft.com/en-us/products/key-vault/#product-overview) is a cloud service that provides secure storage and automated management of certificates used throughout a (cloud) application or service, such, as the FranceConnect Facade.

Multiple X.509 (signing vs. encryption) certificates, and multiple versions of the same certificate, can be kept in the Azure Key Vault. Each certificate in the key vault has a policy associated with it which controls the issuance and lifetime of the certificate, along with actions to be taken as certificates near expiry.

#### Create a key vault

If you don’t have any existing key vault, you need to create one at this stage.

Proceed with the following steps:

1. When you try to create a new key vault, you may see the error "*The subscription is not registered to use namespace 'Microsoft.KeyVault'*".

So, from the previous Windows command prompt or a PowerShell window, start by making sure that the Key Vault resource provider is registered in your subscription. This is a one-time operation for each subscription.

PS C:\> az provider register -n Microsoft.KeyVault

1. Create a key vault. When doing this step, you need some information, and we recommend the following resource details:

|  |  |
| --- | --- |
| Detail | Recommended |
| Region | *FranceCentral* |
| Pricing Tier | *Standard* |
| Key Vault name | *FranceConnectFacade-Vault\** |
| Resource group name | *FranceConnectFacadeTest* |

\*Although we use “*VCIssuer-Vault*” as the name for the Key Vault throughout this illustration, you must use a unique name.

Run the following [az keyvault](https://learn.microsoft.com/en-us/cli/azure/keyvault?view=azure-cli-latest) command:

PS C:\> az keyvault create --name <*your\_key\_vault\_name*> --resource-group *<your\_resource\_group\_name*> --location "France Central"

Because the command does not specify a value for the SKU parameter, it creates a Standard key vault.

Replace the placeholder value in brackets with the above value. For example, in our illustration:

PS C:\> az keyvault create --name FranceConnectFacade-Vault --resource-group FranceConnectFacadeTest --location "France Central"

The output of the az keyvault create command shows properties of the newly created key vault. Take note of the two properties listed hereafter:

1. **name**: For example, "*FranceConnectFacade-Vault* " in our illustration. You will use this name later to obtain an access token.
2. **vaultURI**: For example, <https://franceconnectfacade-vault.vault.azure.net/> in our illustration. The code of the FranceConnect Facade and scripts that use your key vault through its REST API must use this URI.

When you create a key vault in an Azure subscription, it is automatically associated with the Azure AD tenant of the subscription. Anyone trying to manage or retrieve content from a Key Vault must be authenticated by Azure AD.

**At this point, your Azure account is now authorized to perform any operations on this new key vault. As of yet, nobody else is authorized.**

#### Set an access policy for the user-assigned managed identity

For more details, see [Azure Key Vault security](https://learn.microsoft.com/en-us/azure/key-vault/general/security-features#identity-and-access-management) and [Assign an Azure Key Vault access policy (CLI)](https://learn.microsoft.com/en-us/azure/key-vault/general/assign-access-policy?tabs=azure-portal) .

You can control who has access to the contents of your key vault by using access policies. Key Vault [access policies](https://docs.microsoft.com/en-us/azure/key-vault/general/assign-access-policy) grant permissions separately to certificates, keys, and secrets. You can grant a user(-managed identity) access only to certificates and not to keys, and secrets. Access permissions for certificates, keys, and secrets at the vault level.

**At this time, do not remove any permissions!**

From the previous Windows command prompt or a PowerShell window, run the [az keyvault set-policy](https://learn.microsoft.com/en-us/cli/azure/keyvault?view=azure-cli-latest#az-keyvault-set-policy) command to set the access policy:

PS C:\> az keyvault set-policy --name <*your\_key\_vault\_name*> --object-id <*your\_managed\_identity\_client\_Id*> --certificate-permission get list –key-permission sign

Replace the placeholder values in brackets with the your own values. <*your\_managed\_identity\_client\_Id*> is the object ID of the user-assigned managed identity you made a note of, and to which you want to assign the access policy, here the Azure identity with ID 3fedf722-7c5d-426f-9d35-d985d3eb59bc. See section Create a user-assigned managed identity for the FranceConnect Facade above.

For example, in our illustration:

PS C:\> az keyvault set-policy --name FranceConnectFacade-Vault --object-id 3fedf722-7c5d-426f-9d35-d985d3eb59b --certificate-permission get list –key-permission sign

### Create a certificate in your key vault

Let’s now create a X.509 self-signed certificate for testing purposes. You can alternatively:

* Create a new certificate manually with a Certificate Authority (CA) that is not partnered with Key Vault. This will create a public-private key pair and generate an X.509 certificate signing request (CSR) for that CA.
* Create a certificate with a CA that is partnered with Key Vault, e.g., [DigiCert](https://learn.microsoft.com/en-us/azure/key-vault/certificates/how-to-integrate-certificate-authority) and [GlobalSign](https://support.globalsign.com/digital-certificates/digital-certificate-installation/generating-and-importing-certificate-microsoft-azure-key-vault).
* Import an existing certificate. Both PKCS12 and PEM formats are supported.

For more details, see [Certificate creation methods](https://learn.microsoft.com/en-us/azure/key-vault/certificates/create-certificate).

Proceed with the following steps:

1. A policy is required to create certificates in Azure Key Vault. From the previous Windows command prompt or PowerShell windows, get the default policy from your Azure subscription:

PS C:\> az keyvault certificate get-default-policy | Out-File -Encoding utf8 defaultpolicy.json

The policy could look like this:

{

  "issuerParameters": {

    "certificateTransparency": null,

    "name": "Self"

  },

  "keyProperties": {

    "curve": null,

    "exportable": true,

    "keySize": 2048,

    "keyType": "RSA",

    "reuseKey": true

  },

  "lifetimeActions": [

    {

      "action": {

        "actionType": "AutoRenew"

      },

      "trigger": {

        "daysBeforeExpiry": 90

      }

    }

  ],

  "secretProperties": {

    "contentType": "application/x-pkcs12"

  },

  "x509CertificateProperties": {

    "keyUsage": [

      "cRLSign",

      "dataEncipherment",

      "digitalSignature",

      "keyEncipherment",

      "keyAgreement",

      "keyCertSign"

    ],

    "subject": "CN=CLIGetDefaultPolicy",

    "validityInMonths": 12

  }

}

1. Run one of the following two commands to create a X.509 self-signed certificate using the default policy above:

PS C:\> az keyvault certificate create --name <*your\_certname*> --vault-name <*your\_key\_vault\_name*> --policy `@defaultpolicy.json

-or-

PS C:\> az keyvault certificate create --name <*your\_certname*> --vault-name <*your\_key\_vault\_name*> --policy "$(az keyvault certificate get-default-policy)"

Replace the placeholder values in brackets with the your own values. For example, in our illustration:

PS C:\> az keyvault certificate create --name fcfsigningcert --vault-name FranceConnectFacade-Vault --policy "$(az keyvault certificate get-default-policy)"

You can now reference this certificate added to your key vault by using its URI. Use the following URL to get the current version:

https://< *your\_key\_vault\_name*>.vault.azure.net/certificates/<*your\_certname*>.

For example, in our illustration:

<https://franceconnectfacade-vault.vault.azure.net/certificates/fcfsigningcert>

1. Let's view the certificate that you created:

PS C:\> az keyvault certificate list --vault-name <*your\_key\_vault\_name*>

Replace the placeholder value in brackets with the your own value. For example, in our illustration:

PS C:\> az keyvault certificate list --vault-name FranceConnectFacade-Vault

### Set up the FranceConnect Facade

All the FranceConnect Facade's settings are contained in:

* A file named *appsettings.json*.
* A file named *secrets.json*.

#### Declare the X.509 signing certificate

Proceed with the following steps:

1. Open the *FranceConnectFacade.sln* solution with Visual Studio Community.
2. Open the *appsettings.json* file.
3. In the UseDevelopmentCertificate section:
   1. Set UseDevelopmentCertificate to false.
   2. In CertificateNameKeyVault, replace the <*your\_certname*> placeholder by the name of the certificate in your key vault.
   3. In AzureKeyVaultEndpoint, specify the endpoint of your key vault. Replace the <*your\_key\_vault\_name*> placeholder by the name of your key vault.

For example, in our illustration:

// Si false alors utilise le certificat provenant du Service

// Azure Key vault

// Pour utiliser le certificat de développement vous devez

// au préalablement le créer (voir fichier CreateASelfCert.txt dans ce repo)

"UseDevelopmentCertificate": false,

// Utilisation du certificat de développement auto signé

// Le mot de passe doit être inclus dans le fichier secrets.json

// https://learn.microsoft.com/en-us/aspnet/core/security/app-secrets?view=aspnetcore-7.0&tabs=windows

"DevelopmentCertificatePfx": "<*your\_certname*>.pfx",

// Obligatoire ne pas supprimer

"DevCertificateName": "devcert",

// Utilisation du Service key vault Azure

"CertificateNameKeyVault": "fcfsigningcert",

"AzureKeyVaultEndpoint": "https://FranceConnectFacade-Vault.vault.azure.net/"

1. Save the file.

#### Declare the FranceConnect Platform endpoints

See eponym section Target the FranceConnect Integration environment above.

##### Target the FranceConnect Production environment

See eponym section Target the FranceConnect Production environment above.

# Test the FranceConnect Facade

Now you are ready to verify and then issue security tokens as part of the identity dance with tree actors, namely:

1. The Portal application, referred as to the D365 BizApps' Portal[[3]](#footnote-4).
2. The Identity provider, i.e., the FranceConnect Platform (FCP).
3. The FranceConnect Facade (FCF), i.e., the Man in the Middle, which will aim to satisfy the requirements of the other two actors, a.k.a. the FranceConnect connector.

## Run the facade code locally

The FranceConnect dev account’s integration key (client id, client secret) being used in your configuration is configured with local callback URLs (localhost) for both the sign-in and the logout as described hereafter.

In this specific environment, only the following login callback URLs are accepted by the FranceConnect platform with the development account as per section:

* <http://localhost:1337/callback>
* <http://localhost:3000/callback>
* <http://localhost:4242/callback>
* <http://localhost:8080/callback>
* <http://localhost:1337/login-callback>
* <http://localhost:3000/login-callback>
* <http://localhost:4242/login-callback>
* <http://localhost:8080/login-callback>

Same applies to the logout callback URLs that are as follows - Logout is not (yet) implemented in the currently available version of the FranceConnect Facade - :

* <http://localhost:1337/logout>
* <http://localhost:3000/logout>
* <http://localhost:4242/logout>
* <http://localhost:8080/logout>
* <http://localhost:1337/logout-callback>
* <http://localhost:3000/logout-callback>
* <http://localhost:4242/logout-callback>
* <http://localhost:8080/logout-callback>

So:

1. The facade must listen locally on one of the above port, e.g., 4242, to intercept[[4]](#footnote-5) the call of the calling application and change the redirect URL accordingly, e.g., <http://localhost:4242/login-callback>:
   1. In the request, during the authentication phase
   2. In the message body, when requesting the FranceConnect token.
2. A tcp tunnel needs to be started with ngrok so that the calling application can call the above-mentioned endpoints of the FranceConnect Facade (FCF) running on your local development environment.

### Start a tcp tunnel with ngrok

**if you use**[ngrok](https://ngrok.com/)**, a tip to leverage here is to start it and leave it running so that your publicly available hostname for the FranceConnect Facade will NOT change for the rest of this document.**

**This will allow you to later configure a custom OpenID Connect identity provider for your Power Pages app, and this configuration will remain valid for the time of your testing.**

#### Get an Internet address

Proceed with the following steps:

1. Open either a Windows command prompt or a PowerShell window.
2. Run the following command:

PS C:\> ngrok.exe http 4242 --host-header=localhost:4242

More specifically, this command runs ngrok to set up a URL on 4242 where the FranceConnect Facade’s endpoints will be later listening on, and make it publicly available on the Internet. In other words, it will give the FranceConnect Facade and Internet address like:

<https://7516-2a01-110-8012-1013-7925-1440-7ba6-d086.ngrok.io>

1. Leave the prompt or the window opened for the rest of this document.

#### Update the configuration of the FranceConnect Facade

The settings in the *appsettings.json* file must be modified accordingly.

Proceed with the following steps:

1. Open the *FranceConnectFacade.sln* solution with Visual Studio Community.
2. Open the *appsettings.json* file.
   1. In the ngrok section, add the above obtained address. For example:

"ngrok": https://7516-2a01-110-8012-1013-7925-1440-7ba6-d086.ngrok.io

* 1. In the FranceConnect section, set the redirect URL for the FranceConnect platform:

"FranceConnect": {

…

"fcdevredirecturi": "http://localhost:4242/login-callback",

…

}

1. Save the file.
2. **Also ensure that the TEST\_FC\_IN\_PORTAL compile constant is active**.

This constant affects the code being build.

#if TEST\_FC\_IN\_PORTAL

issuerEndPoint = \_configuration["ngrok"];

#else

issuerEndPoint = context.Request.FormatBaseAddress();

#endif

* 1. Right-click the FranceConnectFacade.Identity.WebApi project and select Properties.
  2. Select **Build**. You should the constant listed under the Conditional compilation symbols as illustrated here. Otherwise add it to the list.

Une image contenant texte

Description générée automatiquement

### Declare the FranceConnect Platform endpoints

Regardless of the execution environment, i.e., integration vs. production, the FranceConnect platform exposes the following endpoints:

|  |  |
| --- | --- |
| Endpoints | URL |
| Authorization | FCP\_URL/api/v1/authorize |
| Token | FCP\_URL/api/v1/token |
| UserInfo | FCP\_URL/api/v1/userinfo |
| Logout | FCP\_URL/api/v1/logout |

The base address of this platform is <https://fcp.integ01.dev-franceconnect.fr> in the integration environment; this corresponds to the following endpoints:

|  |  |
| --- | --- |
| Endpoint | URL |
| Authorization | <https://fcp.integ01.dev-franceconnect.fr/api/v1/authorize> |
| Token | <https://fcp.integ01.dev-franceconnect.fr/api/v1/token> |
| UserInfo | <https://fcp.integ01.dev-franceconnect.fr/api/v1/userinfo> |
| Logout | <https://fcp.integ01.dev-franceconnect.fr/api/v1/logout> |

Ensure the above endpoints are correctly declared in the settings. Proceed with the following steps:

1. From the *FranceConnectFacade.sln* solution in Visual Studio Community, open if needed the *appsettings.json* file.
2. In the FranceConnect section, verify or set if needed the above endpoints:
   1. In Issuer, specify the base address of this platform.
   2. In AuthorizationEndpoint, specify the above Authorization endpoint.
   3. In TokenEndpoint, specify the above Token endpoint.
   4. In UserInfoEndpoint, specify the above UserInfo endpoint.
   5. In EndSessionEndpoint, specify the above Logout endpoint.

"FranceConnect": {

"Issuer": "https://fcp.integ01.dev-franceconnect.fr",

"AuthorizationEndpoint": "https://fcp.integ01.dev-franceconnect.fr/api/v1/authorize",

"TokenEndpoint": "https://fcp.integ01.dev-franceconnect.fr/api/v1/token",

"UserInfoEndpoint": "https://fcp.integ01.dev-franceconnect.fr/api/v1/userinfo",

"EndSessionEndpoint": "https://fcp.integ01.dev-franceconnect.fr/api/v1/logout",

…

}

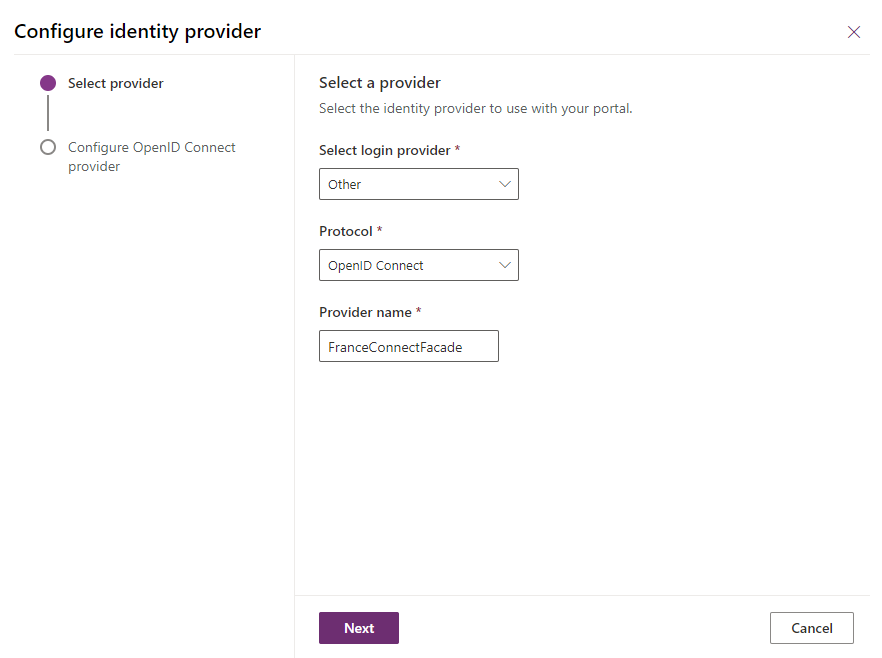
1. Save the file.

### Configure the Facade as an OIDC identity provider on your Portal application

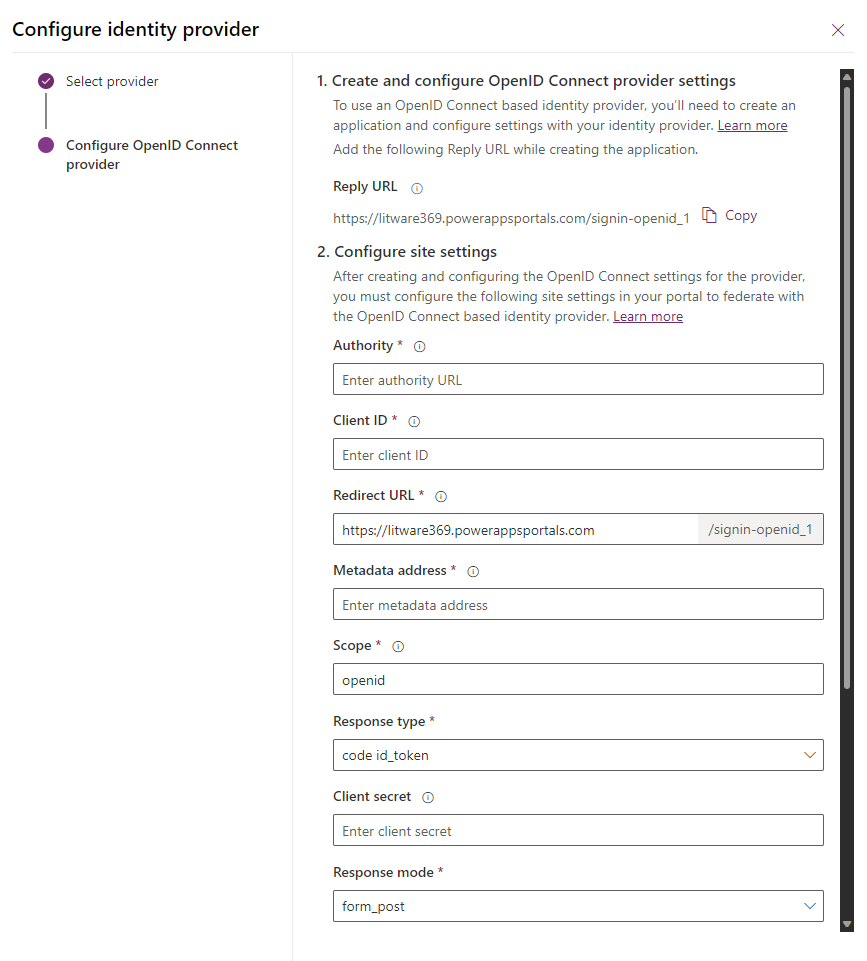
For more details, see [Overview of authentication in Power Apps portals](https://docs.microsoft.com/en-us/power-apps/maker/portals/configure/configure-portal-authentication) and [Configure an OpenID Connect provider for portals](https://learn.microsoft.com/en-us/power-apps/maker/portals/configure/configure-openid-provider).

Proceed with the following steps:

1. Open a browser session, navigate to [Power Apps](https://make.powerapps.com/) and sign-in with your credentials.
2. From the right-upper corner, select the Environment drop-down to verify, or choose the Power Apps environment for your portal.
3. From the left pane, select Apps.
4. Select your portal, click More Commands (…) and select Settings from the context menu. In the Portal settings blade that pops up, click Authentication settings.
5. Click + Add provider. The Configure identity provider page opens up.
6. In Select a provider, set the following values:
   1. In Select login provider: select Other.
   2. In Protocol: select OpenID Connect.
   3. In Provider name, type a name. For example, “*FranceConnectFacade*” in our illustration.



* 1. And click Next.



1. Under Reply URL, make a note of the Redirect URL that will be used by your Power Pages app to redirect users to the portal after the authentication succeeds with the FranceConnect Platform through the FranceConnect Facade. For example, in our illustration: <https://litware369.powerappsportals.com/signin-openid_1>

Please note that if your Power Pages app uses a custom domain name for a production environment, you might have a different URL than the one provided here.

1. Under 2. Configuration site settings, enter the following site settings for portal configuration:

|  |  |
| --- | --- |
| Field | Value |
| Authorization | https://7516-2a01-110-8012-1013-7925-1440-7ba6-d086.ngrok.io |
| Client ID | 211286433e39cce01db448d80181bdfd005554b19cd51b3fe7943f6b3b86ab6e |
| Redirect URL | https://<*your\_AppName*>.powerappsportals.com/signin-openid\_1 |
| Metadata address | https://7516-2a01-110-8012-1013-7925-1440-7ba6-d086.ngrok.io/common/.well-known/openid-configuration |
| Scope | profile birth email |
| Response type | id\_token |
| Client secret | 2791a731e6a59f56b6b4dd0d08c9b1f593b5f3658b9fd731cb24248e2669af4b |
| Response mode | query |

1. Do NOT configure the settings for signing users out. This capability is not (yet) available in current bits of the FranceConnect Facade. Under Logout settings, ensure that External logout is set to Off.

Une image contenant texte

Description générée automatiquement

1. Click Confirm, and then Close.

It’s now time to reflect the above Redirect URL value in the FranceConnect Facade configuration.

#### Update the configuration of the FranceConnect Facade

The settings in the *appsettings.json* file must be modified accordingly.

Proceed with the following steps:

1. Open the .sln solution with Visual Studio Community.
2. Open the *appsettings.json* file
3. In the FranceConnect section, add the reply/redirect URL of your Power Pages application you took a note of:

"FranceConnect": {

…

portalredirecturi:"https://<your\_AppName>.powerappsportals.com/signin-openid\_1"

…

}

Where <*your\_AppName*> is the name of your Power Pages application.

1. Save the file.

### Build and run the FranceConnect Facade locally

From Visual Studio Community, either press F5 to debug or SHIF+F5 to run the facade without debugging capabilities.

### Test the FranceConnect Facade from your Portal application

Proceed with the following steps:

1. Open a new browser session and navigate to your Power Pages application at https://<*your\_AppName*>.powerappsportals.com/signin-openid\_1 where <*your\_AppName*> is the name of your app.
2. Click Sign-in.
3. Under External SignIn, select FacadeFranceConnectDev. The FranceConnect button is currently not yet integrated as you can see. Stay tuned!

Une image contenant texte

Description générée automatiquement

You are redirected to the FranceConnect Facade, which in turn redirects you to the FranceConnect platform login page.

Graphical user interface, application

Description automatically generated

1. Interestingly enough, the FranceConnect integration platform allows you to use a "Demo" identity provider for testing purposes. Select Démonstration eIDAS faible.
2. Enter you credentials. The different "user" accounts available are listed here <https://github.com/france-connect/identity-provider-example/blob/master/database.csv>.

Graphical user interface, application, Teams

Description automatically generated

1. Click Valider.

Graphical user interface, application, Teams

Description automatically generated

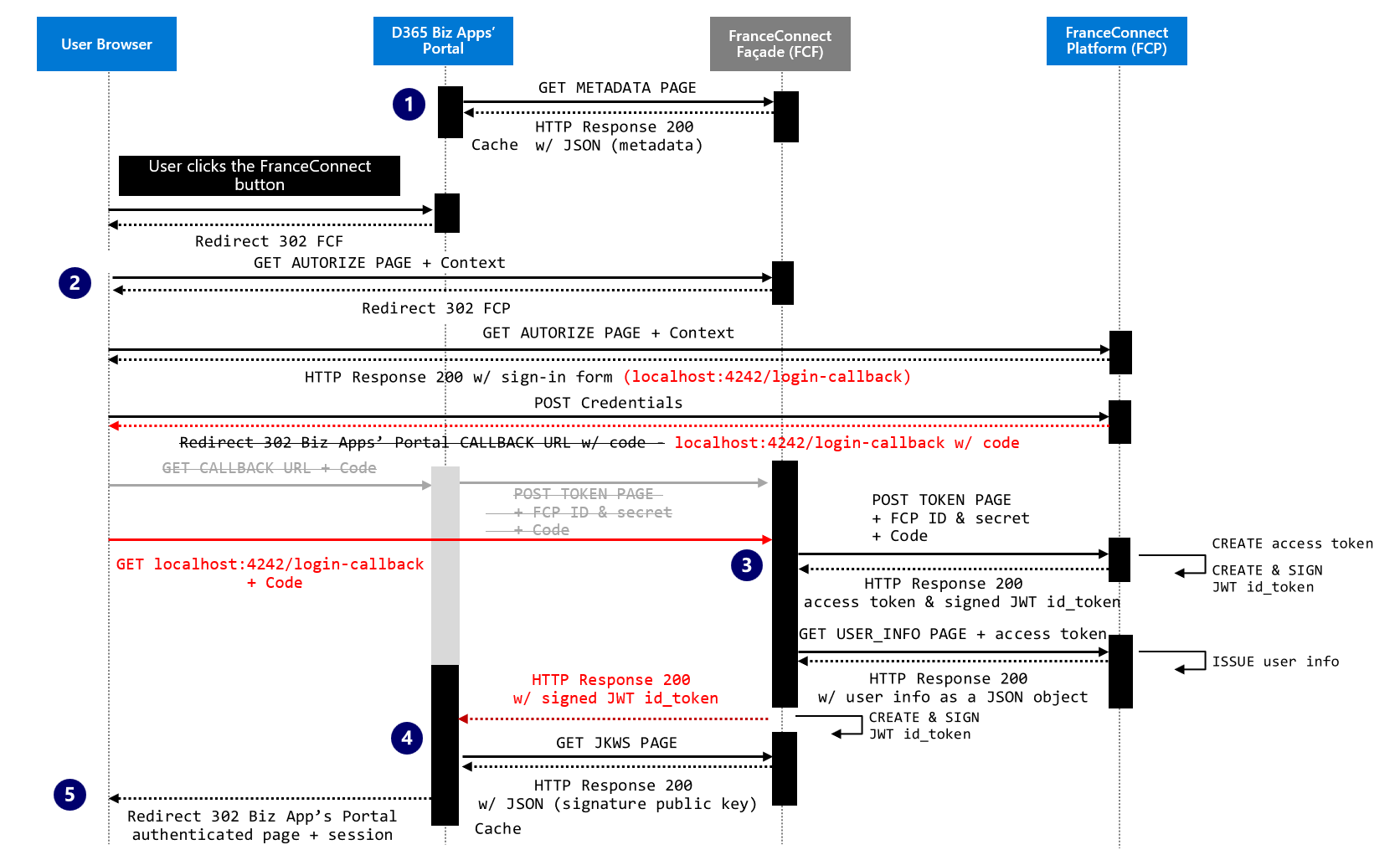
1. Give your consent by clicking Continuer sur FSPublic.
2. Once authenticated, the user's profile page is displayed.

Graphical user interface, application

Description automatically generated

### About the Identity dance and the related code in the facade

The above user experience results from a, identity dance between three actors as follows:



#### Discovery phase

The first time the user authenticates, the Power Pages application calls in order the following endpoints exposed by the FranceConnect Facade:

1. ./api/<*version*>/.well-known/openid-configuration

{

"token\_endpoint":"https://7516-2a01-110-8012-1013-7925-1440-7ba6-d086.ngrok.io/api/beta/token",

"jwks\_uri":"https://7516-2a01-110-8012-1013-7925-1440-7ba6-d086.ngrok.io/common/discovery/keys",

"response\_modes\_supported": [

"query"

],

"acr\_values\_supported": [

"eidas1",

"eidas2",

"eidas3"

],

"subject\_types\_supported": [

"pairwise"

],

"id\_token\_signing\_alg\_values\_supported": [

"RS256"

],

"response\_types\_supported": [

"code"

],

"scopes\_supported": [

"openid",

"profile",

"birth",

"email"

],

"issuer":"https://7516-2a01-110-8012-1013-7925-1440-7ba6-d086.ngrok.io",

"userinfo\_endpoint":"https://7516-2a01-110-8012-1013-7925-1440-7ba6-d086.ngrok.io/api/beta/userinfo",

"authorization\_endpoint":"https://7516-2a01-110-8012-1013-7925-1440-7ba6-d086.ngrok.io/api/beta/authorize",

"http\_logout\_supported":true,

"end\_session\_endpoint":"https://7516-2a01-110-8012-1013-7925-1440-7ba6-d086.ngrok.io/api/beta/logout",

"claims\_supported":null

}

1. ./api/<*version*>/discovery/key

{

keys: [

{

kty: "RSA",

alg: "RSA256",

use: "sig",

kid: "",

"x5c": [

""

]

}

]

}

To retrieve the authorization/authentication endpoints + the token's signing public key in accordance to the [Final: OpenID Connect Discovery 1.0 incorporating errata set 1](https://openid.net/specs/openid-connect-discovery-1_0.html) standard specification.

The facade builds the above endpoints and other information for this discovery phase on the fly by code. In the Solution Explorer, see *FranceConnectFacadeOpenIdConfiguration.cs* file under Controllers section.

1. ./common/.well-known/openid-configuration. Based on the OpenIdConfiguration section in the *appsettings.json* file:

"OpenIdConfiguration": {

"JwksUri": "common/discovery/keys",

"AuthorizationEndPoint": "api/beta/authorize",

"EndSessionEndpoint": "api/beta/logout",

"TokenEndpoint": "api/beta/token",

"UserInfoEndpoint": "api/beta/userinfo",

"SubjectTypesSupported": [ "pairwise" ],

"ScopesSupported": [ "openid", "profile", "birth", "email" ],

"AcrValuesSupported": [ "eidas1", "eidas2", "eidas3" ],

"ResponseModesSupported": [ "query" ],

"ResponseTypesSupported": [ "code" ],

"HttpLogoutSupported": true,

"IdTokenSigningAlgValuesSupported": [ "RS256" ],

"ClaimsSupported": []

}

The disco document is build and exposed with the following code:

/// </summary>

/// <see href="https://openid.net/specs/openid-connect-discovery-1\_0.html"/>

/// <returns></returns>

[HttpGet]

[Route(".well-known/openid-configuration")]

public IActionResult OpenIdConfiguration()

{

\_logger.LogInformation($"Controller : .well-know/OpenIdConfiguration");

var baseAddress = Request.FormatBaseAddress();

#if TEST\_FC\_IN\_PORTAL

//Changer adresse dans AppSettings.json du

//paramètre ngrok pour test avec Portal

baseAddress = \_configuration["ngrok"];

#endif

var openIdConfiguration = \_configuration

.GetSection("OpenIdConfiguration")

.Get<OpenIdConfiguration>();

if (openIdConfiguration == null)

{

return StatusCode(500);

}

openIdConfiguration.AuthorizationEndpoint=

$"{baseAddress}/{openIdConfiguration.AuthorizationEndpoint}";

openIdConfiguration.JwksUri= $"{baseAddress}/{openIdConfiguration.JwksUri}";

openIdConfiguration.EndSessionEndpoint= $"{baseAddress}/{openIdConfiguration.EndSessionEndpoint}";

openIdConfiguration.UserInfoEndpoint= $"{baseAddress}/{openIdConfiguration.UserInfoEndpoint}";

openIdConfiguration.TokenEndpoint = $"{baseAddress}/{openIdConfiguration.TokenEndpoint}";

openIdConfiguration.Issuer = baseAddress;

return Ok(openIdConfiguration);

}

1. ./api/<*version*>/discovery/key. Based on the OpenIdDiscoveryKeys section in the *appsettings.json* file:

"OpenIdDiscoveryKeys": {

"Keys": [

{

"alg": "RSA256",

"kty": "RSA",

"use": "sig",

"kid": "",

//"e": "AQAB",

//"n": "",

//"x5t": "",

"x5c": [

""

]

}

]

}

the key material information is build and exposed with the following code:

/// </summary>

/// <see href="https://datatracker.ietf.org/doc/html/rfc7515"/>

/// <returns></returns>

[HttpGet]

[Route("discovery/keys")]

[Produces("application/json")]

public IActionResult Keys()

{

\_logger.LogInformation($"Controller : Discovery/Keys");

OpenIdDiscoveryKeys? discovery = \_configuration

.GetSection("OpenIdDiscoveryKeys")

.Get<OpenIdDiscoveryKeys>();

if (discovery == null)

{

return StatusCode(500);

}

string CertificateNameKeyVault = \_configuration.GetCertificateName();

string rawCertificate = \_configuration[CertificateNameKeyVault];

if (string.IsNullOrEmpty(rawCertificate))

{

return StatusCode(500);

}

X509Certificate2 x509 = new X509Certificate2(Convert.FromBase64String(rawCertificate));

if (discovery.keys == null)

{

return StatusCode(500);

}

// Récupère le certificat avec la clé publique qui

// permettra de signer le jeton

discovery.keys[0].x5c = new string[] { Convert.ToBase64String(x509.GetRawCertData()) };

discovery.keys[0].kid = x509.Thumbprint;

return Ok(discovery);

}

#### User authentication phase

The /authorize endpoint is then called by the Power Pages application and does two things:

1. Reconstruct the Query String by:

* Adding the acr\_values field which is mandatory for the FranceConnect platform (FCP),
* Removing the elements not necessary for the FranceConnect platform (FCP),
* Replacing the Power Pages redirect URL for the FCP by <http://localhost:4242/login-callback> (if in test/dev mode)

Following is an example of the Query String:

?client\_id=211286433e39cce01db448d80181bdfd005554b19cd51b3fe7943f6b3b86ab6e&redirect\_uri=http://localhost:4242/login-callback&response\_type=code&scope=profile%20birth%20email&response\_mode=query&nonce=637937309212325972.NDU3NDFmM2QtNWU0MS00NWRiLTk5MTItMTE2OGVhMzQ2ZGI2Zjc4OWNlN2QtMDc3Ny00ODJkLTg5ZmUtYTZkODMwMTUwZThi&acr\_values=eidas1&state=OpenIdConnect.AuthenticationProperties%3DcTOYl1A3LKWHKdDWfNUAPQbev0ljomFqAOA0Mw7VLw1EK8ZzHmEoxXlo2hHlQzirVuaIuDqr-5mEvRRY\_coHWdYlPTxA8olvn8Hu1JQ3Yf3iANM\_Xl-N17OphDFXStWLJ8l46w2wMpGrfvwGHCKG0uHJzpBChVuz7G-MdmHHq8fkPG\_\_knmzUJp6joo54y64VxHnEgOtBwGUWVXDPfMoIgrxrVnVOrUr9qge-V7gXCpyGYAp7L9LxdsiDWLO7NtNkaenm2hMUlAkICwyut9x2IOtrVx36oCcTffMZ0M4D7WuBFFqidLlkj3GAFdb4Gh-uuqv7ZEEmjnL1kP9sb6QqQ

1. Redirect to FCP login page.

As far as the former is concerned, in the Solution Explorer, see *FranceConnectFacadeOpenIdMiddleware.cs* file under Middleware section.

/// </summary>

/// <param name="context"></param>

/// <returns></returns>

public void CreateQueryForAuthorizeEndPoint(HttpContext context)

{

\_logger.LogInformation($"Middle : CreateQueryForAuthorizeEndPoint");

QueryString qs = context.Request.QueryString;

string? fromQuery = qs.Value;

if (string.IsNullOrEmpty(fromQuery))

{

context.Response.StatusCode = 401;

return;

}

// Construction d'une nouvelle QueryString avec ajout

// de acr\_values obligatoire pour FranceConnect

string query = $"?client\_id={Common.GetValue("client\_id", fromQuery)}" +

$"&redirect\_uri={Common.GetValue("redirect\_uri", fromQuery)}" +

$"&response\_type={Common.GetValue("response\_type", fromQuery)}" +

$"&scope={Common.GetValue("scope", fromQuery)}" +

$"&response\_mode={Common.GetValue("response\_mode", fromQuery)}" +

$"&nonce={Common.GetValue("nonce", fromQuery)}" +

$"&acr\_values={\_configuration["OpenIdConfiguration:AcrValuesSupported:0"]}" +

$"&state={Common.GetValue("state", fromQuery)}";

// Sauvegarde de la QueryString dans le contexte http

// pour réutilisation avec le EndPoint api/authorize

/// <see cref="FranceConnectFacadeOpenIdConnectController"/>

context.Items["query"] = query;

}

As far as the latter is concerned, in the Solution Explorer, see *FranceConnectFacadeOpenIdController.cs* file under Controllers section.

[HttpGet()]

[Route("authorize")]

#if TEST\_FC\_IN\_PORTAL

[FranceConnectFacadeEndPoint(EndPoint = "authorize:testinportal")]

#else

[FranceConnectFacadeEndPoint(EndPoint = "authorize")]

#endif

public RedirectResult Authorize()

{

string baseAddress = \_configuration["FranceConnect:AuthorizationEndpoint"];

string? query = HttpContext.Items["query"] as string;

string redirectUri = $"{baseAddress}/{query}";

var redirectReponse = this.Redirect(redirectUri);

\_logger.LogInformation($"Controller : authorize");

return redirectReponse;

}

#### Authorization phase for obtaining the id\_token

Once authenticated, the FCP platform invokes the FranceConnect Facade on the URL <https://localhost:4242/login-callback>. Following is an example of the message body:

client\_id=211286433e39cce01db448d80181bdfd005554b19cd51b3fe7943f6b3b86ab6e&client\_secret=2791a731e6a59f56b6b4dd0d08c9b1f593b5f3658b9fd731cb24248e2669af4b&code=a65ade01-cbad-4133-9af5-a85fbe1def01&grant\_type=authorization\_code&redirect\_uri=http://localhost:4242/login-callback

The FranceConnect Facade then only redirects the call to the redirect URL provided by the Power Pages application[[5]](#footnote-6).

The Power Pages application calls the /token endpoint of the FranceConnect Facade front-end which does 3 things:

1. Remove unnecessary information for the FCP platform from the message body before requesting the token
2. Request and validate the FranceConnect (FC) id\_token token, call the FCP /user\_info endpoint to complement the set of claims for the user.
3. Forge on that basis in turn a new JWT id\_token token compatible with the Portal app and signed it with the private key corresponding to the declared X.509 signing certificate. Depending on your own configuration, and as declared in the *appsettings.json* file, this certificate can be either a self-signed developer certificate or a certificated securely stored in your key vault in Azure.

In the Solution Explorer, see *FranceConnectFacadeOpenIdMiddleware.cs* file under Middleware section, and more specifically the InvokeTokenEndPointAsync method:

public async Task InvokeTokenEndPointAsync(HttpContext context,

FranceConnectFacadeConfigurationOptions options)

{

\_logger.LogInformation($"Middle : InvokeTokenEndPointAsync");

[…region VERIFICATION DES PARAMETRES omitted…]

//Récupère le corps du message car nous allons le transformer

//pour être compatbile

string fromBody = await context.Request.GetBodyAsync();

// France Connect ne supporte pas le flux PKCE

fromBody = Helpers.OAuth.DisablePKCE(fromBody);

// Obtenir le jeton FranceConnect (endpoint api/v1/token)

\_logger.LogInformation("Middle : Obtient le jeton FranceConnect");

var franceConnectResult = await \_httpFcClient.GetFranceConnectToken(fromBody);

if (franceConnectResult == null)

{

context.Response.StatusCode = 401;

return;

}

if (string.IsNullOrEmpty(franceConnectResult.IdToken))

{

context.Response.StatusCode = 401;

return;

}

// Besoin de valider et récupèrer les claims du Jeton FranceConnect

// afin de créer le nouveau id\_token compatible Portal

ClaimsPrincipal? claimsPrincipal = Helpers.Token.ValidateFranceConnectToken(options.FranceConnectOptions, franceConnectResult.IdToken);

if (claimsPrincipal == null)

{

context.Response.StatusCode = 401;

return;

}

[…omitted…]

//Obtient les informations utilisateur à l'aide du jeton d'accès

string authorization = $"Bearer {franceConnectResult.AccessToken}";

var UserInfo = await \_httpFcClient.GetFranceConnectUserInfo(authorization);

if (UserInfo == null)

{

context.Response.StatusCode = 401;

return;

}

ClaimsIdentity claimsIdentity = new ClaimsIdentity(new[]

{

new Claim("family\_name", UserInfo.FamilyName != null ? UserInfo.FamilyName : ""),

new Claim("given\_name", UserInfo.GivenName != null ? UserInfo.GivenName : ""),

new Claim("email", UserInfo.Email != null ? UserInfo.Email : ""),

new Claim("gender", UserInfo.Gender != null ? UserInfo.Gender : ""),

new Claim("birthcountry",UserInfo.BirthCountry != null ? UserInfo.BirthCountry : ""),

new Claim("birthdate",UserInfo.BirthDate != null ? UserInfo.BirthDate : ""),

new Claim("birthplace",UserInfo.BirthPlace != null ? UserInfo.BirthPlace : "" ),

new Claim("preferred\_username",UserInfo.PreferredUsername !=null ?

UserInfo.PreferredUsername : ""),

new Claim("sub", UserInfo.Sub != null ? UserInfo.Sub : "" ),

new Claim("nonce", nonce.Value)

});

context.User.AddIdentity(claimsIdentity);

// Crée un nouveau jeton et signe le avec la clé

// privée contenue dans le certificat X509

string franceConnectFacadeIdToken = Helpers.Token.CreateTokenAndSignWithX509Cert(options.X509Cert,

options.FranceConnectOptions.ClientId,

issuerEndPoint,

claimsIdentity);

if (string.IsNullOrEmpty(franceConnectFacadeIdToken))

{

context.Response.StatusCode = 401;

return;

}

// Sauvegarde du nouveau jeton

// pour réutilisation avec le EndPoint /api/token

/// <see cref="FranceConnectFacadeOpenIdConnectController"/>

/// <remarks>Voir si c'est judicieux de faire comme cela

/// en terme de sécurité</remarks>

franceConnectResult.IdToken = franceConnectFacadeIdToken;

context.Items.Add("token", franceConnectResult);

}

As well as the token method in the *FranceConnectFacadeOpenIdConnectController.cs* file under Controllers section:

[HttpPost]

[Route("token")]

[Produces("application/json")]

#if TEST\_FC\_IN\_PORTAL

[FranceConnectFacadeEndPoint(EndPoint = "token:testinportal")]

#else

[FranceConnectFacadeEndPoint(EndPoint = "token")]

#endif

public IActionResult Token()

{

var fcfResult = HttpContext.Items["token"];

if (fcfResult == null)

{

return new UnauthorizedResult();

}

\_logger.LogInformation(fcfResult.ToString());

\_logger.LogInformation("Controller : token");

return new OkObjectResult(fcfResult);

}

Enough code? Following is a resulting example of a “compatible” id\_token returned to the Power Pages app:

{"access\_token":"c8f3504d-b9dd-4521-add6-29c8f13e1fa0","token\_type":"Bearer","expires\_in":60,"ext\_expires\_in":0,"id\_token":"eyJhbGciOiJSUzI1NiIsImtpZCI6IjY4QjE2N0U4RjQ4NkYzOTk0NTkwN0VDMkM4Qzg0RkZBOUREQkU3QTEiLCJ0eXAiOiJKV1QifQ..PEuhQ43gZFiMbsDvITlAY7qUMlq1-5mlFKxXLsL9Rt2VpgemRNMzgqCtDTl19KklHebgDjR8hyr2R2W8fu4tdqgLrc-exJFsUtgxSbpzgDxh9C5TO9gXVQiVmZtrcl241IWaQCZECyWAaMiedeqKkc3S7K4P18GH8oV0jqfDfGbYIsMHzCQnvPYnGcWZxdznz4HPtTlmMXJ7r4iSEJqTfP3Y3gc8aP4fa9nc9M4QWLC4Rb84oJMg7uxOvICcqrCjGR8Fd7QiH5t67RuXIXqGoFXT1YzqVe9bpr8p71ZCM86c81UHKrzvIVE682\_44g\_TMhEpIsALCux0Y7v55TMN5g","scope":null,"state":null}

To see the content of the id\_token, proceed with the following steps:

1. Copy the value of the id\_token field.
2. Open a browser session and navigate to <https://jwt.ms>. The id\_token is formatted as a JSON Web Token (JWT) token.
3. Paste the value into Enter token below. The JWT token is decoded.

Une image contenant texte

Description générée automatiquement

## Deploy the facade on the Internet

**The ability to fully test the FranceConnect Facade in an environment other than local, here Microsoft Azure, requires a prior authorization. See section Target the FranceConnect Production environment above.**

**The next sections only highlight here all the steps required to deploy the FranceConnect Facade code in Microsoft Azure.**

At this stage, you will need to [create an App Service app](https://docs.microsoft.com/en-us/azure/app-service/), or use an app that you created for another purpose. Please note that the web app's [App Service plan](https://docs.microsoft.com/en-us/azure/app-service/overview-hosting-plans) must be a paid tier and not Free (F1). See [Scale up an app](https://docs.microsoft.com/en-us/azure/app-service/manage-scale-up#scale-up-your-pricing-tier) to update the tier if needed.

More specifically, as covered in the next sections, you will need to:

* Create a custom domain your FranceConnect instance.
* Create an App Service app tor this instance to run.
* Map your custom domain to your App Service app.
* Create a TLS/SSL certificate binding for your App Service app.
* Deploy the code sample application to Azure using Visual Studio code.
* Maximize the availability of the FranceConnect Facade.

### Create a custom domain

A custom domain in a public DNS zone is also required to both resolve the host name of the FranceConnect Facade, i.e., a web site on the Internet, and verify the FranceConnect eligible organization that operates it. For example the fictitious entity Litware369 in our illustration.

**For illustration purposes, we’ve opted to purchase and configure the domain litware369.com. You will have to choose in lieu of a domain name of yours.**

For checking purpose, you can for instance use the domain search capability provided by several popular domain providers. For example, <https://www.godaddy.com/domains> with GoDaddy.

Make sure you can edit the DNS records for your custom domain. To edit DNS records, you need access to the DNS registry for your domain provider, such as GoDaddy. For example, to add DNS entries for litware369.com and www.litware369.com, you must be able to configure the DNS settings for the litware369.com root domain.

**Whenever a reference to litware369.com is made in a procedure, it has to be replaced by the DNS domain name of your choice to reflect accordingly the change in naming.**

See [Buy a custom domain name](https://docs.microsoft.com/en-us/azure/app-service/manage-custom-dns-buy-domain).

### Create an App Service app

If you already have a suitable App Service app for the FranceConnect Facade, you can skip this section.

Otherwise, proceed with the following steps:

1. Open a browser session, navigate to the Azure portal at <https://portal.azure.com>, and sign in with your Azure administrative credentials.
2. Search for “*app services”,* then select **App Services**. An eponym **App Services** page opens up.
3. Click Create. A **Create Web App** blade pops up with the **Basics** tab selected.

Une image contenant texte

Description générée automatiquement

1. Under **Project Details**,
2. **Subscription**: ensure the correct subscription is selected.
3. **Resource Group**: select the resource group you created previously for the sake of this walkthrough. For example, FranceConnectTest in our illustration.
4. Under**Instance Details**:
5. **Name**: type a globally unique name for your FranceConnect Facade instance. For example “franceconnectfacade” in our illustration.
6. Publish: leave **Code** selected.
7. **Runtime stack**: select **Node 14 LTS**.
8. **Operating System**: leave **Linux** selected. A Linux container is used by default.
9. **Region**: pick the region you want to serve your app from. For example, **France Central** in our illustration.
10. Under **App Service Plan**:
11. **Linux Plan (<yourRegion>)**: click **Create new**. In the **New App Service Plan** popup, type a name. For example, “*MyAppServicePlanforVC*” in our illustration. Click **OK**.
12. **Sku and size**: click **Change size**. A **Spec Picker** blade pops up. Select any of the non-free tiers (B1, B2, B3, or any tier in the Production category)
13. Select the plan of your choice that includes the **Custom domains / SSL** feature under the **Dev/Test** tab or the **Production** tab.

Please note that:

* The **Free (F1)** plansdo not support this feature and cannot be thus selected.
* The first Basic (B1) core for Linux is free for the first 30 days.

Once selected, click Apply.

1. Under **Zone redundancy**, leave **Disabled** selected.
2. Click **Review + create**. The deployment process may take few minutes.

### Map your custom domain to your App Service app

See [Tutorial: Map existing custom DNS name](https://docs.microsoft.com/en-us/azure/app-service/app-service-web-tutorial-custom-domain?tabs=a%2Cazurecli).

Proceed with the following steps:

1. In the Azure portal, from the **App Services** page, select the name of the App Service app you previously created. You see the management page of the App Service app.
2. To add a custom domain to your app, you need to verify your ownership of the domain by adding a verification ID as a TXT record with your domain provider.

In the left pane of your app page, select **Custom domains**. A **Custom Domains** page opens up.

Une image contenant texte

Description générée automatiquement

1. **Custom Domain Verification ID**: copy the ID for the next step.
2. IP address: copy the value of IP address. To map an [A record](https://en.wikipedia.org/wiki/List_of_DNS_record_types#A), for your custom domain (top-level domain (TLD) or root domain), you indeed need the app's external IP address.
3. Create the DNS A and TXT records in your domain provider. For example, GoDaddy.com in our illustration for the TLD custom domain litware369.com.
4. Sign in to the website of your domain provider. For example, <https://www.godaddy.com> in our illustration.

You can use Azure DNS to manage DNS records for your domain and configure a custom DNS name for Azure App Service. See [Tutorial: Host your domain in Azure DNS](https://docs.microsoft.com/en-us/azure/dns/dns-delegate-domain-azure-dns).

1. Find the page for managing DNS records.

Every domain provider has its own DNS records interface, so consult the provider's documentation. Look for areas of the site labeled **Domain Name**, **DNS**, or **Name Server Management**.

Often, you can find the DNS records page by viewing your account information and then looking for a link such as **My domains**. Go to that page, and then look for a link that's named something like **Zone file**, **DNS Records**, or **Advanced configuration**.

The following screenshot is the DNS records page for litware369.com at GoDaddy:

Une image contenant texte

Description générée automatiquement

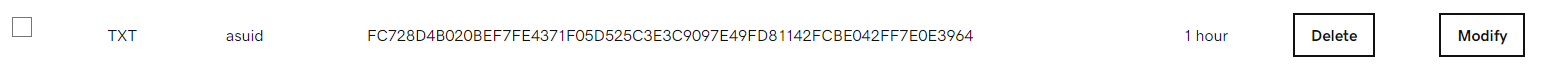
1. Select **Add** or the appropriate widget to create a record.
2. Select the type of record to create and follow the instructions.
3. For a top level domain like litware369.com, create two records according to the following table:

|  |  |  |  |
| --- | --- | --- | --- |
| Record type | Host | Value | Comments |
| A | *@* | IP address you got earlier | The domain mapping itself (@ typically represents the top level domain). |
| TXT | *asuid* | [The verification ID you got earlier](https://docs.microsoft.com/en-us/azure/app-service/app-service-web-tutorial-custom-domain?tabs=a%2Cazurecli#2-get-a-domain-verification-id) | For a top-level domain, App Service accesses asuid TXT record to verify your ownership of the custom domain |

A record:



TXT record:



For certain providers, such as GoDaddy, changes to DNS records don't become effective until you select a separate **Save Changes** link.

1. In your domain provider, also create a DNS CNAME record for the www subdomain. For example www.litware369.com in our illustration.

CNAME record:



1. After you [create DNS records](https://docs.microsoft.com/en-us/azure/app-service/app-service-web-tutorial-custom-domain?tabs=a%2Cazurecli#3-create-the-dns-records), you enable the mapping in your App Service app. In the Azure portal, from the previous **Custom Domains** page,
2. Select **Add custom domain**. An eponym **Add custom domain** blade pops up.

Une image contenant texte

Description générée automatiquement

1. In **Custom domain**, type the fully qualified domain name that you configured the A record for, such as litware369.com.
2. Click **Validate**. The **Add custom domain** blade refreshes.

Une image contenant texte

Description générée automatiquement

1. Make sure that **Hostname record type** is set to **A record (example.com)**. Select **Add custom domain**.

It might take some time for the new custom domain to be reflected in the app's **Custom Domains** page. Refresh the browser to update the data.

Check if the DNS records are properly configured using an [online DNS lookup](https://www.nslookup.io/) tool.

A warning label for your custom domain means that it's not yet bound to a TLS/SSL certificate. Any HTTPS request from a browser to your custom domain will receive an error or warning, depending on the browser. So let’s add a TLS binding for the sake of our configuration!

### Create a TLS/SSL certificate binding for your App Service app

See [Add and manage TLS/SSL certificates](https://docs.microsoft.com/en-us/azure/app-service/configure-ssl-certificate?tabs=apex%2Cportal) and [Secure a custom DNS name with a TLS/SSL binding in Azure App Service](https://docs.microsoft.com/en-us/azure/app-service/configure-ssl-bindings).

You will now secure the custom domain in your App Service app by creating a TLS/SSL certificate binding. When you're finished, you can access your App Service app at the https:// endpoint for your custom DNS name. For example, <https://www.litware369.com> in our illustration.

Securing a custom domain with a certificate is a two steps process:

1. Add a private certificate to App Service app that satisfies [all the private certificate requirements](https://docs.microsoft.com/en-us/azure/app-service/configure-ssl-certificate?tabs=apex%2Cportal#private-certificate-requirements).
2. Create a TLS/SSL binding to the corresponding custom domain.

As far as the former is concerned, you will create a free App Service managed certificate. This is a turn-key solution for securing your custom DNS name in App Service. It's a TLS/SSL server certificate that's fully managed by App Service and renewed continuously and automatically in six-month increments, 45 days before expiration, as long as the prerequisites set-up remain the same without any action required from you.

Other options are available to add certificates in App Service: purchase an App Service certificate, import a certificate from your existing key vault (see section Create a Azure Key Vault instance above), upload a private certificate, and upload a public certificate. For the sake of brevity, this walkthrough do not cover them.

For a root domain, (like litware369.com), make sure your App Service app doesn't have any [IP restrictions](https://docs.microsoft.com/en-us/azure/app-service/app-service-ip-restrictions) configured. Both certificate creation and its periodic renewal for a root domain depends on your app being reachable from the Internet.

Proceed with the following steps:

1. Create a free managed certificate:
2. In the Azure portal, from the **App Services** page, select the name of the App Service app you previously created. You see the management page of the App Service app.
3. From the left pane, select **TLS/SSL settings**, and select the **Private Key Certificates (.pfx)** tab.
4. Click **Create App Service Managed Certificate**. An eponym **Create a free managed certificate** blade pops up.
5. Select the custom domain you created earlier to create a free certificate. For example, litware369.com in our configuration.
6. Click **Create**. Note that you can create only one certificate for each supported custom domain.

Une image contenant texte

Description générée automatiquement

At this stage, you have fulfilled all prerequisites and are ready to bind the TLS/SSL certificate.

1. Create a TLS/SSL binding to the corresponding custom domain.
2. From the same the page, now select the **Binding** tab, under **TLS/SSL bindings**, click **+ TLS/SSL Binding**. An eponym **TLS/SSL Binding** blade pops up.

Une image contenant texte

Description générée automatiquement

1. Fill **the following information:**

* **Custom domain**: select your custom domain.
* **Private certificate Thumbprint**: choose the certificate you created previously.
* **TLS/SSL Type**: select **SNI SSL**.

1. Click **Add binding**.

Une image contenant texte

Description générée automatiquement

As you can see, your App Service app allows TLS 1.2 by default, which is the recommended TLS level by industry standards.

1. Once the **operation** is complete, the custom domain's TLS/SSL state is changed to Secure.

Une image contenant texte

Description générée automatiquement

**At this stage, you have successfully deployed your App Service app, mapped it to your custom domain name, and secured by using TLS/SSL certificate binding.**

**Please note that, in App Service,**[TLS termination](https://wikipedia.org/wiki/TLS_termination_proxy) **happens at the network load balancers, so all HTTPS requests reach your App Service app as unencrypted HTTP requests. If your app logic needs to check if the user requests are encrypted or not, inspect the X-Forwarded-Proto header.**

**Language specific configuration guides, such as the** [Linux Node.js configuration](https://docs.microsoft.com/en-us/azure/app-service/configure-language-nodejs#detect-https-session) **guide, shows you how to detect an HTTPS session in your application code logic.**

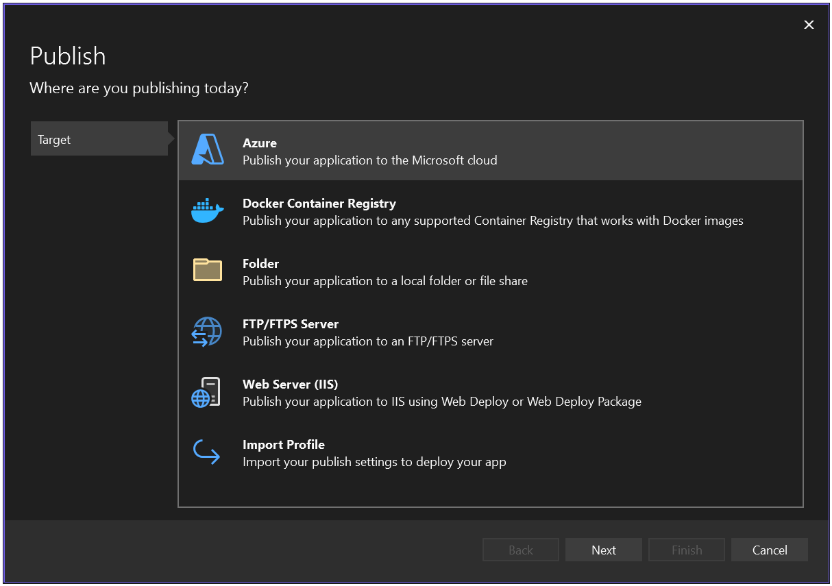
### Deploy the FranceConnect Facade code to Azure using Visual Studio Community

Azure offers several ways to deploy your FranceConnect Facade to App Service. In this section, you will deploy the instance on Azure using Visual Studio Community. See [Publish an ASP.NET Core app to Azure with Visual Studio](https://learn.microsoft.com/en-us/aspnet/core/tutorials/publish-to-azure-webapp-using-vs?view=aspnetcore-7.0).

At this stage, and as the prerequisites for this end-to-end walkthrough as per chapter Fulfill the prerequisites for your testing environment above, we assume that you have an Azure account and active subscription in place, [Visual Studio Community](https://visualstudio.microsoft.com/vs/community/), etc.

Proceed with the following steps:

1. Right-click on the FranceConnectFacade.Identity.WebApi project in Solution Explorer and select Publish. A Publish dialog opens up.



1. In the Publish dialog:
   1. Select Azure.
   2. Select Next.
   3. Select either Azure App Service (Windows) or Azure App Service (Linux).
   4. Select Next.
   5. In App Service, select your previously created instance.
   6. Select Finish.
   7. Next you see the Publish Profile summary page.
   8. Click Publish. Visual Studio publishes your FranceConnect Facade instance to Azure.

**Congratulations, you have successfully deployed your FranceConnect Facade code on Azure. To troubleshoot a deployment issue if any, see** [Troubleshoot ASP.NET Core on Azure App Service and IIS](https://learn.microsoft.com/en-us/aspnet/core/test/troubleshoot-azure-iis?view=aspnetcore-7.0)**.**

**You now require an accreditation** to continue with your testing.

As already stated, the request must be made to the following address: <https://franceconnect.gouv.fr/partenaires>.

### Maximize the availability of the FranceConnect Facade

For maximum availability and minimal disruption to your newly deployed facade, we do advise following our list of best practices outlined in [The Ultimate Guide to Running Healthy Apps in the Cloud](https://nam06.safelinks.protection.outlook.com/?url=https%3A%2F%2Fazure.github.io%2FAppService%2F2020%2F05%2F15%2FRobust-Apps-for-the-cloud.html&data=05%7C01%7Cphilippe.beraud%40microsoft.com%7C47f18b02c8d14ba61dfe08dad59b5c68%7C72f988bf86f141af91ab2d7cd011db47%7C1%7C0%7C638057160518422384%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=%2Bbm0P%2Bq1SWH6H04dYbsQT75UQA4Bsa0Ng1VNqITiUHU%3D&reserved=0).

Modern-day datacenters like the ones for Microsoft Azure are extremely complex and have many moving parts. VMs can restart or move, systems are upgraded, and file servers are scaled up and down, etc. All these events are to be expected in a cloud environment. However, you can make your FranceConnect Façade in the cloud resilient to these events.

As such, this document outlines thirteen crucial steps that you can take to ensure that your facade is cloud ready. By taking these steps, you will ensure that any events in the data center will have negligible effects on your facade and that your facade will be more resilient and future proof to serve the interest of your users.

As mentioned above, your App Services instances for the façade are expected to and will restart. They will be upgraded and will sometimes suffer from file server movements. However you can make your façade resilient to all these incidents. In order to guarantee the maximum uptime for your facade, please ensure that you follow all practices:

1. Use multiple instances using the Scale out (App Service Plan) blade. See [Get started with Autoscale in Azure](https://docs.microsoft.com/azure/azure-monitor/platform/autoscale-get-started?toc=/azure/app-service/toc.json) and [App Service Warm-Up Demystified](https://michaelcandido.com/app-service-warm-up-demystified/).
2. Update your default settings to set Always on to On and ARR Affinity to Off. See [Configure an App Service app in the Azure portal](https://docs.microsoft.com/azure/app-service/configure-common#configure-general-settings) and [Disable Session affinity cookie (ARR cookie) for Azure web apps](https://azure.github.io/AppService/2016/05/16/Disable-Session-affinity-cookie-(ARR-cookie)-for-Azure-web-apps.html).
3. Use production hardware to your plan. See [Scale up an app in Azure App Service](https://docs.microsoft.com/azure/app-service/manage-scale-up).
4. Leverage the deployment slots features in App Services to test your future changes. See [Set up staging environments in Azure App Service](https://docs.microsoft.com/azure/app-service/deploy-staging-slots), [Azure Web App Deployment Slot Swap with Preview](https://ruslany.net/2015/10/azure-web-app-deployment-slot-swap-with-preview/), and [Deployment best practices](https://docs.microsoft.com/azure/app-service/deploy-best-practices#use-deployment-slots).
5. Set your health check path to pool the critical components of your facade, and thus ensure an accurate picture of the overall health of your facade. Please note that the Health Check feature works only when you have two or more instances. See above. Also see [Health Check (Preview)](https://github.com/projectkudu/kudu/wiki/Health-Check-(Preview)).
6. Use application initialization to ensure that your facade instances have fully started before they are added to they start serving requests and be part of the identity dance with the FranceConnect platform. See [How to warm up Azure Web App during deployment slots swap](https://ruslany.net/2015/09/how-to-warm-up-azure-web-app-during-deployment-slots-swap/).
7. Enable local cache for certain capabilities, e.g. the (not yet implemented) logout support. See [Azure App Service Local Cache overview](https://docs.microsoft.com/en-us/azure/app-service/overview-local-cache).
8. Auto-heal to possibly recover from unexpected behaviors by a simple restart. See [Azure App Service Auto-Heal](https://stack247.wordpress.com/2019/05/20/azure-app-service-auto-healing/) and [Announcing the New Auto-Heal Experience in App Service Diagnostics](https://azure.github.io/AppService/2018/09/10/Announcing-the-New-Auto-Healing-Experience-in-App-Service-Diagnostics.html).
9. Minimize App Service plan density to avoid a negative impact performance. See [App Service Plan Density Check](https://azure.github.io/AppService/2019/05/21/App-Service-Plan-Density-Check.html).
10. Monitor disk space usage to ensure that the disk space used by www folder should be less than 1GB.
11. Enable Application Insights to troubleshoot incidents, i.e., debug code errors, diagnose performance degradations caused by dependencies and more, that happen on your FranceConnect facade. See [Profile production applications in Azure with Application Insights](https://docs.microsoft.com/azure/azure-monitor/app/profiler-overview), [Diagnose exceptions in your web apps with Application Insights](https://docs.microsoft.com/azure/azure-monitor/app/asp-net-exceptions), [Dependency Tracking in Azure Application Insights](https://docs.microsoft.com/azure/azure-monitor/app/asp-net-dependencies#diagnosis), as well as [App Insights integration with App Service Diagnostics](https://azure.github.io/AppService/2020/04/21/Announcing-Application-Insights-Integration-with-App-Service-Diagnostics.html).
12. Deploy in multiple regions, i.e., France Central and France South, with Azure Front Door or Azure Traffic Manager. See [Controlling Azure App Service traffic with Azure Traffic Manager](https://docs.microsoft.com/en-us/azure/app-service/web-sites-traffic-manager) and [Quickstart: Create a Front Door for a highly available global web application](https://docs.microsoft.com/en-us/azure/frontdoor/quickstart-create-front-door).
13. Check App Service diagnostics.

Finally, we also recommend that you take a look at the [Cloud Design Patterns](https://docs.microsoft.com/en-us/azure/architecture/patterns/) document to minimize the FranceConnect Facade start time and follow more resiliency recommendations.

You can also refer to your Service Health portal at <https://aka.ms/servicehealthpm> for more information.

# Appendix. Deploy the FranceConnect Facade resources with Bicep scripts

## Fulfill the prerequisites

When you use Azure CLI with Bicep, you already have everything in place to deploy and decompile Bicep files as per section Install Azure CLI above. Indeed, Azure CLI automatically installs the Bicep Command Line Interface (CLI) when a command is executed that needs it. For information on the Bicep available commands, see [Bicep CLI commands and overview](https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/bicep-cli).

To validate your Bicep CLI install, run the following command from either a Windows command prompt or a PowerShell window:

PS C:\> az bicep version

Bicep CLI version 0.11.1 (030248df55)

Azure CLI installs a self-contained instance of the Bicep CLI. This instance doesn't conflict with any versions you may have manually installed. Azure CLI doesn't add Bicep CLI to the PATH environment variable.

## Deploy the resources

To deploy the resources required by the FranceConnect Facade (FCF), proceed with the following steps:

1. First, gather the following information from the Azure portal:

* Your Azure Active Directory (Azure AD) tenant ID, referred as to <*your\_tenant\_id*>. see section Create your Azure AD tenant above to create one.
* Your Azure subscription ID, referred as to <*your\_subscription\_id*> below. See Azure subscription in section Fulfill the prerequisites for your Azure testing environment above.

1. Now open a PowerShell console and set the following parameters accordingly:

PS C:\> $AzureActiveDirectoryTenantId="<*your\_tenant\_id*>"

PS C:\> $AzureSubscriptionId="<*your\_subscription\_id*>"

Replace the values in brackets by your own values to reflect your specific configuration in the Azure cloud.

1. Specify the resource group (to create) and the location to which to deploy the FranceConnect façade (FCF):

PS C:\> $ResourceGroupName="<*your\_resourcegroup\_name*>"

PS C:\> $location="<*your\_location*>"

1. Set your FranceConnect (dev) account, i.e., the corresponding Client id and Client Secret.

PS C:\> $FranceConnectClientId="<*your\_franceconnect\_clientid*>"

PS C:\> $FranceConnectClientSecret="<*your\_franceconnect\_clientsecret*>"

For example, the FranceConnect dev account through the available so-called “integration key for public use”, in this illustration:

PS C:\> $FranceConnectClientId = "211286433e39cce01db448d80181bdfd005554b19cd51b3fe7943f6b3b86ab6e"

PS C:\> $FranceConnectClientSecret = "2791a731e6a59f56b6b4dd0d08c9b1f593b5f3658b9fd731cb24248e2669af4b"

1. Specify whether or not you’d like to deploy a Linux ($true), vs. Windows ($false), Web application :

PS C:\> $IsLinuxWebApp=[$true|$false]

1. Eventually execute the provided PowerShell launcher script *Deployer.ps1* with the following command parameters. This PowerShell script is located in the folder *bicep* located under *Scripts*:

PS C:\> .\Deployer.ps1 -FranceConnectClientId $FranceConnectClientId `

-FranceConnectClientSecret $FranceConnectClientSecret `

-ResourceGroupName $ResourceGroupName `

-SubscriptionId $AzureSubscriptionId `

-TenantId $AzureActiveDirectoryTenantId `

-Location $location `

-IsLinuxWebApp $IsLinuxWebApp

This script executes the *main.bicep* script, which runs in sequence the *service.managed.identity.bicep* file for the managed identity, the *service.keyvault.bicep* file for the keyvault, the *service.keyvault.setsecrets.bicep* file for the certificate, etc.

1. After the launch of Power Apps portals on October 1, 2019, the full capabilities of Dynamics 365 Portals, previously offered only as an add-on to customer engagement apps (Dynamics 365 Sales, Dynamics 365 Customer Service, Dynamics 365 Field Service, Dynamics 365 Marketing, and Dynamics 365 Project Service Automation), are now available standalone in Power Apps. In other word, all Dynamics 365 Portals are now referred to as Power Apps portals. [↑](#footnote-ref-2)
2. Effective October 12, 2022, Power Apps portals are Power Pages. See [Microsoft Power Pages is now generally available](https://powerpages.microsoft.com/en-us/blog/microsoft-power-pages-is-now-generally-available/). [↑](#footnote-ref-3)
3. See document "FranceConnect (Serverless) Façade (FCF) Technical-Functional Specifications", as well as the Addendum, and eponym deck [↑](#footnote-ref-4)
4. When the façade is not deployed locally, the interception still occurs in the same manner but with one slight difference: the façade is known by both the calling application and the FranceConnect Facade (FCF) under the same DNS name. Here, the name is one hand, the URL resulting from the tcp tunnel started with ngrok (see below) for the calling application, and on the other hand, localhost on port 4242 for the FranceConnect platform (FCP). [↑](#footnote-ref-5)
5. With a production FranceConnect account, this interception/redirection will not be necessary. [↑](#footnote-ref-6)