Mettre en œuvre un fournisseur de service FranceConnect avec ASP.NET Core 3.1

Kit de démarrage FranceConnect

Publication : Juin 2016 (Révision Octobre 2021)

Version : 2021-10

Auteurs : Charles de Vandière, Amine Agoussal

Contributeurs/Relecteurs : Philippe Beraud

Pour les dernières informations sur le kit de démarrage FranceConnect, consultez

aka.ms/FranceConnect

Sommaire

[A propos du système FranceConnect 4](#_Toc82424735)

[A propos du kit de démarrage FranceConnect 5](#_Toc82424736)

[Licence 5](#_Toc82424737)

[Commentaires 5](#_Toc82424738)

[Prérequis, connaissances et attendu 6](#_Toc82424739)

[Créer un fournisseur de service FranceConnect avec ASP.NET Core 7](#_Toc82424740)

[Obtenir des identifiants FranceConnect 7](#_Toc82424741)

[Créer un fournisseur de service 7](#_Toc82424742)

[Configurer l’application 8](#_Toc82424743)

[Modifier le fichier de configuration 8](#_Toc82424744)

[Comprendre la classe de configuration 8](#_Toc82424745)

[Ajouter le client secret 9](#_Toc82424746)

[Créer un schéma de données 10](#_Toc82424747)

[Créez le modèle d’utilisateurs et de base de données. 10](#_Toc82424748)

[Configurez la connexion 11](#_Toc82424749)

[Créez et appliquez la migration 11](#_Toc82424750)

[Implémenter le middleware OIDC 11](#_Toc82424751)

[Adapter le comportement de l’application 13](#_Toc82424752)

[Ajouter les fonctions de connexions 13](#_Toc82424753)

[Ajout des boutons d’inscription et connexion 14](#_Toc82424754)

[Ajout de liens à la barre de navigation 14](#_Toc82424755)

[Définir le style du bouton FranceConnect des pages de connexion/inscription 15](#_Toc82424756)

[Récupérer des ressources auprès d’un fournisseur de données 16](#_Toc82424757)

[Obtenir et déployer le canevas de fournisseur de service préconfiguré du kit de démarrage 19](#_Toc82424758)

[Obtenir les sources du fournisseur de service 19](#_Toc82424759)

[Télécharger les sources du fournisseur de service 19](#_Toc82424760)

[Cloner les sources du fournisseur de service 20](#_Toc82424761)

[Organisation du code source du canevas de fournisseur de service (FS) 21](#_Toc82424762)

[Comprendre l’organisation du projet 21](#_Toc82424763)

[Appréhender les éléments de configuration 21](#_Toc82424764)

[Déployer le projet 22](#_Toc82424765)

[Tester le fournisseur de service 23](#_Toc82424766)

[Configurer le compte FranceConnect 23](#_Toc82424767)

[Se connecter avec le système FranceConnect 23](#_Toc82424768)

[Invoquer un fournisseur de données via le système FranceConnect 26](#_Toc82424769)

[Se déconnecter du système FranceConnect 27](#_Toc82424770)

[Annexe A. Références techniques 29](#_Toc82424771)

# A propos du système FranceConnect

La plateforme FranceConnect est un système d’identification visant à faciliter l’accès des usagers aux services numériques de l’administration en ligne, en évitant à tout un chacun de devoir créer un nouveau compte lors de l’accès à un nouveau service et donc à se remémorer différents mots de passe pour l’ensemble des services accédés.

Cette plateforme est portée par la direction interministérielle du numérique, ou DINUM, qui accompagne les ministères dans leur transformation numérique, conseille le gouvernement et développe des services et ressources partagées comme ici ce système d’identification et d’authentification en ligne de l’État, ou encore le réseau interministériel de l’État, data.gouv.fr ou api.gouv.fr.

Le système FranceConnect permet à chaque usager de disposer d’un mécanisme d’identification reconnu par les téléservices de l’administration au travers du bouton FranceConnect. Lors de l’accès à un nouveau service, et au-delà de la possibilité toujours proposée de s’inscrire vis-à-vis d’une autorité administrative que ne connaîtrait pas encore l’usager, le bouton permet de sélectionner une identité compatible dont l’usager disposerait déjà (impôts, assurance maladie, identité numérique la poste, etc.) et de l’utiliser dans ce contexte.

Le système FranceConnect agit alors en tant que tiers de confiance mettant en relation usagers et autorités administratives auprès de fournisseurs d'identité compatibles et reconnus. Le système FranceConnect permet donc sous le contrôle de l’usager et de son consentement la fédération de comptes. L’expérience utilisateur (UX) ainsi proposée s’apparente à celle communément proposée par une application dont les comptes auraient été fédérées avec un réseau Facebook. Cela suppose toutefois que le niveau d’authentification (faible, substantiel ou fort) pris en charge par le fournisseur d’identité pour le compte sélectionné soit au moins équivalent à celui imposé au niveau du service numérique accédé.

Le système d’identification FranceConnect couvre les rôles suivants :

* **Fournisseur de service** (FS). Un site web qui s’appuie sur FranceConnect pour identifier les internautes qui le souhaitent (ex. Ameli.fr, Laposte.fr).
* **Fournisseur de données** (FD). Un site/API web qui propose des ressources à un fournisseur de service, et qui utilise FranceConnect pour sécuriser cet échange (ex. EDF, impôts).
* **Fournisseur d’identité** (FI). Un environnement Web qui permet à FranceConnect d'identifier et d'authentifier un internaute (ex. Ameli.fr, Laposte.fr)

Enfin, le système FranceConnect est conforme à la directive européenne eIDAS (Electronic Identification and Signature) proposant ainsi une interopérabilité des systèmes d’identification utilisés par les Etats membres pour accéder à leurs services en ligne. Ainsi, un autre pays de l’Union sera ainsi de facto reconnu sur ledit service numérique de l’Administration en ligne française si le système d’identification d’origine est à la fois compatible eIDAS et fédéré avec FranceConnect.

Compte tenu des éléments précédents, le système FranceConnect – vous l’aurez compris - constitue une composante essentielle de la stratégie d’[Etat plateforme](http://etatplateforme.modernisation.gouv.fr/) poussée par la DINUM en facilitant l’accès des usagers aux différents services numériques disponibles tout en renforçant la confiance de ces mêmes usagers dans ces services au travers de la prise en charge reproductible et contrôlée des identités existantes compatibles de leur choix. Ceci représente à n’en point douter un élément central dans les échanges de l’Administration électronique.

L’article [L'administration change avec le numérique](http://www.modernisation.gouv.fr/ladministration-change-avec-le-numerique/par-son-systeme-dinformation/france-connect-un-acces-universel-aux-administrations-en-ligne) présente plus en détail cette initiative de la DINUM.

# A propos du kit de démarrage FranceConnect

Le kit de démarrage FranceConnect sous licence libre est une initiative menée par les équipes Microsoft en charge du [Secteur Public](https://www.microsoft.com/fr-fr/industry/government). Il propose un ensemble d’exemples/accélérateurs facilitant la mise en œuvre des différents rôles prévus par le système d’identification FranceConnect avec les services, produits et technologies Microsoft.



Plus précisément, ce document détaille la mise en œuvre d’un canevas de fournisseur de service (FS) compatible FranceConnect. Il s’appuie pour cela sur la technologie [ASP.NET Core](https://dotnetfoundation.org/projects/aspnet-core) mise à disposition sous licence libre par la [fondation .NET](https://www.dotnetfoundation.org/).

Note Les très nombreuses contributions de Microsoft dans le domaine de l’Open Source se traduisent en particulier par la création de cette fondation indépendante afin de favoriser le développement ouvert et la collaboration autour du nombre toujours croissant de technologies Open Source pour .NET.

Cette fondation sert de forum pour tous développeurs issus de communautés et d’entreprises commerciales afin de renforcer l’avenir de l’écosystème .NET en favorisant la transparence, la participation communautaire et l’innovation rapide. La fondation .NET préside à la destinée de plus de 32 projets en mode Open Source sous son égide dont le projet ASP.NET Core dont il est question ici.

Un accompagnement pas-à-pas vous est proposé pour toutes les étapes du processus de mise en œuvre d’un canevas de fournisseur de service (FS) en partant si vous le souhaitez d’un projet générique ASP.NET Core 3.1. Une solution complète prête à être configurée et déployée vous est également proposée.

Ainsi, vous abordez directement si vous le souhaitez comment configurer et publier le canevas de fournisseur de service ASP.NET Core mise à disposition.

## Licence

Le kit de démarrage FranceConnect est publié sous le contrat de licence de logiciel libre MIT tel que décrit à l’adresse <https://opensource.org/licenses/MIT>.

## Commentaires

Votre opinion nous intéresse. N’hésitez pas à nous transmettre vos commentaires via la boîte aux lettres FranceConnect (<mailto:franceconnect@microsoft.com>) afin que nous puissions améliorer la solution communautaire ainsi mise à disposition.

## Prérequis, connaissances et attendu

Le canevas du kit de démarrage FranceConnect pour réaliser un fournisseur de service (FS) nécessite un ordinateur sous un système Windows, Linux, ou MacOS [supporté par .NET Core](https://devblogs.microsoft.com/dotnet/announcing-net-core-3-1/#platform-support). Vous devrez également installer le dernier [SDK .NET Core 3.1](https://dotnet.microsoft.com/download/dotnet/3.1).

L’environnement ici conseillé est un système Windows 7 ou ultérieur supportant les [prérequis](https://docs.microsoft.com/visualstudio/releases/2019/system-requirements) de [Microsoft Visual Studio 201](https://visualstudio.microsoft.com/vs/community/)9, lui-même installé avec la charge de travail de développement ASP.NET et Web.

La mise en œuvre d’un fournisseur de service avec le kit de démarrage FranceConnect requiert une expérience préalable :

* De l’environnement de développement [Microsoft Visual Studio 201](https://visualstudio.microsoft.com/vs/community/)9, ou d’un environnement ligne de commandes tel que PowerShell ou Bash.
* De la programmation avec le framework .NET,
* Des technologies Web comme ASP.NET Core MVC,

Note L’article [Introduction to ASP.NET Core](https://docs.microsoft.com/aspnet/core/introduction-to-aspnet-core) propose comme son nom l’indique une première introduction à ASP.NET Core (anciennement ASP.NET 5). La page [ASP.NET Core Documentation](https://docs.asp.net/en/latest/) donne par ailleurs accès à l’ensemble de la documentation disponible à date.

* Du développement et de l’utilisation d’interfaces Web de style REST (Representational State Transfer) pour l’invocation de fournisseurs de données (FD),
* De l’environnement d’exécution et d’hébergement Microsoft Azure si vous souhaitez bénéficier de cet environnement d’exécution pour facilement tester une instance avec la configuration associée du fournisseur de service.

# Créer un fournisseur de service FranceConnect avec ASP.NET Core

Cette section décrit comment créer un canevas de fournisseur de service FranceConnect depuis Visual Studio 2019 ou le CLI (Interface en Ligne de Commande) du SDK .NET Core en partant de la création d’un projet générique.

Si vous le souhaitez, vous pouvez utiliser directement le projet complet terminé et préconfiguré proposé sur GitHub dans le cadre du kit de démarrage FranceConnect : [franceconnect-service-provider-dotnet-webapp-aspnetcore](https://github.com/FranceConnectSamples/franceconnect-service-provider-dotnet-webapp-aspnetcore).

Pour cela, rendez-vous à la section § *Déployer le fournisseur de service préconfiguré du kit de démarrage*. sur le document situé dans le répertoire GitHub, sous [Documentation/GitHub Actions & Déploiement sur Azure.md](https://github.com/FranceConnectSamples/franceconnect-service-provider-dotnet-webapp-aspnetcore/blob/v2-RC3-DotNetCore-3.1/Documentation/GitHub%20Actions%20%26%20D%C3%A9ploiement%20sur%20Azure.md)

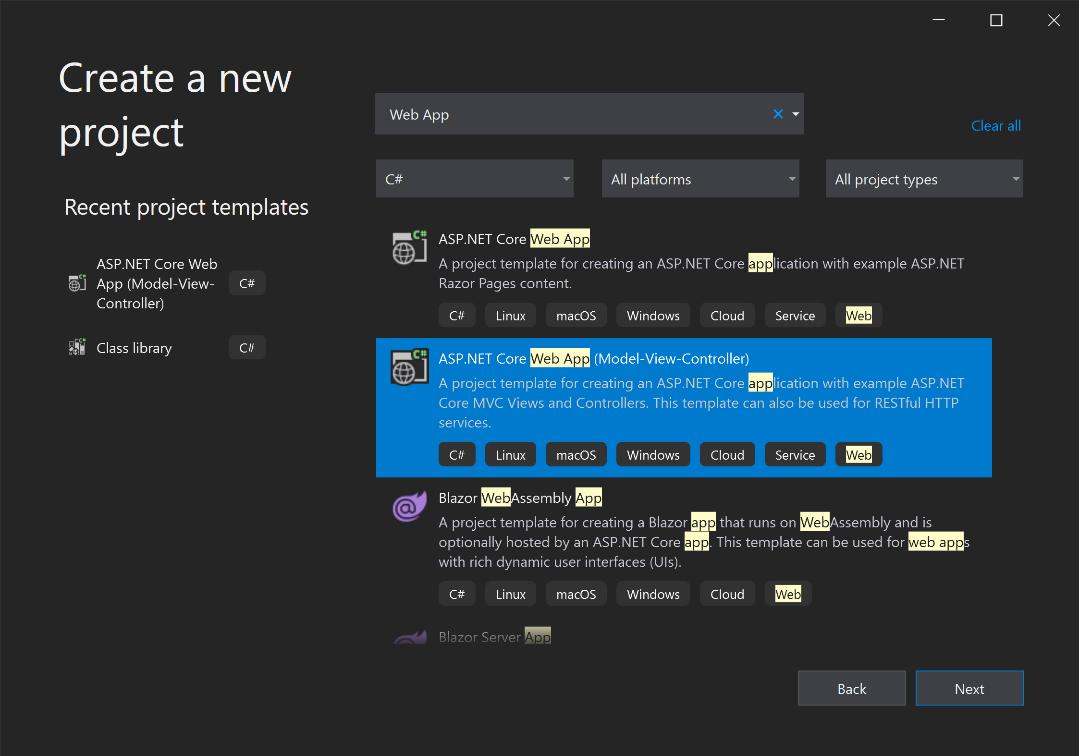
## Obtenir des identifiants FranceConnect

Avant de créer votre fournisseur de service, vous devez vous [inscrire](https://partenaires.franceconnect.gouv.fr/monprojet/inscription) sur FranceConnect pour obtenir des identifiants (***client\_id***et ***client\_secret***) pour votre application. Notez bien ces valeurs ; ces dernières vous serviront dans la configuration de votre solution.

En attendant que l’inscription soit réalisée, vous pouvez utiliser un jeu d’identifiants dédiés au développement, [documenté ici](https://partenaires.franceconnect.gouv.fr/fcp/fournisseur-service#sign_in)

## Créer un fournisseur de service

Une fois les identifiants obtenus, vous pouvez créer une nouvelle application. Sous Visual Studio 2019, creez un nouveau projet de type **ASP.NET Core Web App (Model-View-Controller)** parmi les différents gabarits (templates) ASP.NET Core proposés :



Assistant de création de projet

L’assistant vous demande quelle version de .NET Core cibler. Sélectionnez **.NET Core 3.1**, et conservez les autres paramètres par défaut.

Note Vous pouvez alternativement réaliser cette opération à l’aide du CLI (Interface en Ligne de Commande) du SDK .NET Core. Pour cela, créez un dossier pour votre projet, ouvrez le dans l’environnement ligne de commandes de votre choix, et exécutez la commande > dotnet new mvc -f netcoreapp3.1

## Configurer l’application

### Modifier le fichier de configuration

Les applications Web ASP.NET Core possèdent un fichier appsettings.json qui contient toute la configuration.

Note Pour de plus amples informations, vous pouvez consulter la documentation [ici](https://docs.microsoft.com/aspnet/core/fundamentals/configuration/?view=aspnetcore-3.1).

Ajoutez les lignes suivantes pour configurer le middleware OpenID Connect (OIDC) qui sera utilisé pour l’authentification via FranceConnect. Les points de terminaisons indiqués ci-dessous sont ceux correspondant à un environnement d’intégration. Ceux-ci, ainsi que ceux d’un environnement de production, sont indiqués dans la [documentation FranceConnect pour un fournisseur de service](https://partenaires.franceconnect.gouv.fr/fcp/fournisseur-service#sign_in)

"FranceConnect": {

"ClientId": "<votre client\_id FranceConnect>",

"CallbackPath": "/login-callback",

"SignedOutCallbackPath": "/logout-callback",

"DataCallbackPath": "/data-callback",

"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",

"EIdasLevel": "<niveau eIDAS souhaité : 1, 2 ou 3>",

"Scopes": [ "profile","birth","email"],

"DataProviders": [

{

"Name": "<nom du fournisseur de données>",

"Scopes": [ "<scope>" ],

"Endpoint": "<URL d’accès aux données>"

}

]

}

FranceConnect vous permet de filtrer les fournisseurs d’identité en fonction de leur niveau de sécurité : eIDAS 1, eIDAS 2 et eIDAS 3. Précisez le niveau eIDAS que vous souhaitez pour votre application dans la configuration.

### Comprendre la classe de configuration

La configuration de l’application peut être convertie en objet pour être accessible depuis un contrôleur. Pour cela, créer un fichier de classe Configuration.cs à la racine du projet :

namespace WebApp\_Service\_Provider\_DotNet

{

public class FranceConnectConfiguration

{

public string ClientId { get; set; }

public const string ProviderScheme = "oidc\_FranceConnect";

public const string ProviderDisplayName = "FranceConnect";

public string ClientSecret { get; set;

public string CallbackPath { get; set; }

public string SignedOutCallbackPath { get; set; }

public string DataCallbackPath { get; set; }

public string Issuer { get; set; }

public string AuthorizationEndpoint { get; set; }

public string TokenEndpoint { get; set; }

public string UserInfoEndpoint { get; set; }

public string EndSessionEndpoint { get; set; }

private int \_EIdasLevel;

public int EIdasLevel

{

get => \_EIdasLevel;

//Valid if between 1 & 3, invalid (set to 1 instead) otherwise

set => \_EIdasLevel = value > 0 && value < 4 ? value : 1;

}

public List<string> Scopes { get; set; }

public List<DataProvider> DataProviders { get; set; }

}

public class DataProvider

{

public string Name { get; set; }

public List<string> Scopes { get; set; }

public string Endpoint { get; set; }

}

}

La classe FranceConnectConfiguration décrit la configuration du middleware OIDC pour FranceConnect.

Enfin, la classe DataProvider décrit les fournisseurs de données utilisés par l’application.

Pour accéder à la configuration depuis un contrôleur, ajouter les lignes suivantes au fichier Startup.cs dans la méthode ConfigureServices() :

// Add configuration

services.AddOptions();

IConfiguration franceConnectConfig = Configuration.GetSection("FranceConnect");

services.Configure<FranceConnectConfiguration>(franceConnectConfig);

services.AddControllersWithViews();

### Ajouter le client secret

Afin de ne pas publier votre secret client (valeur ***client\_secret*** copiée précédemment) dans votre gestionnaire de code source, Visual Studio propose de le conserver sur votre ordinateur et de l’ajouter automatiquement à la configuration quand vous êtes en environnement de développement.

Pour cela, utilisez la fonctionnalité [User Secrets](https://docs.asp.net/en/latest/security/app-secrets.html) :

Cliquez droit sur votre projet, et sélectionnez « Manage User Secrets »

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

Description générée automatiquement

Dans le document qui s’ouvre alors, indiquez le couple clé valeur

{

"FranceConnect:ClientSecret": "<votre valeur client\_secret FranceConnect>"

}

Puis sauvegardez.

Note Vous pouvez également réaliser cette opération à l’aide du CLI .NET Core, lancé depuis le dossier source de votre projet (contenant le fichier .csproj ), comme suit

> dotnet user-secrets init

> dotnet user-secrets set FranceConnect:ClientSecret <votre valeur client\_secret FranceConnect> -p .\src\WebApp-Service-Provider-DotNet

## Créer un schéma de données

### Créez le modèle d’utilisateurs et de base de données.

Installez tout d’abord les packages

Microsoft.AspNetCore.Identity.EntityFrameworkCore, Microsoft.EntityFrameworkCore.Tools, et Microsoft.AspNetCore.Identity.EntityFrameworkCore ; en version 3.1.x

Cela peut être réalisé depuis Visual Studio, ou depuis le CLI avec la commande

**> dotnet add package *NomDuPackage* --version 3.1.18**

Creez

Créez une classe modèle PivotIdentity, possédant les propriétés suivantes pour correspondre à l’identité pivot :

public class PivotIdentity

{

public string Gender { get; set; }

public DateTimeOffset Birthdate { get; set; }

public string Birthcountry { get; set; }

public string Birthplace { get; set; }

public string GivenName { get; set; }

public string FamilyName { get; set; }

public string Email { get; set; }

}

Créez de même une classe ApplicationUser dérivant de la classe IdentityUser, contenant les champs que vous souhaitez sauvegarder pour un utilisateur. Notez que la propriété Email est déjà héritée de la classe IdentityUser.

public class ApplicationUser : IdentityUser

{

public string Gender { get; set; }

public DateTimeOffset? Birthdate { get; set; }

public string GivenName { get; set; }

public string FamilyName { get; set; }

public string PreferredName { get; set; }

}

Créez ensuite la classe *ApplicationDbContext*.

public class ApplicationDbContext : IdentityDbContext<ApplicationUser>

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

}

protected override void OnModelCreating(ModelBuilder builder)

{

base.OnModelCreating(builder);

}

}

### Configurez la connexion

Installez le package Microsoft.EntityFrameworkCore.SqlServer, ou l’une de ses alternatives selon la base de données que vous utilisez, en version 3.1.x.

Cela peut être réalisé depuis Visual Studio, ou depuis le CLI avec la commande

> dotnet add package *NomDuPackage* --version 3.1.18

Copiez ensuite les lignes en gras suivantes dans la fonction *ConfigureServices* du fichier *Startup.cs* afin de déclarer la base de données utilisée pour les connexions. Si vous avez choisi un autre type de base de données, faites varier la ligne options.UseSqlServer en conséquence.

public void ConfigureServices(IServiceCollection services)

{

**// Add framework services.**

**services.AddDbContext<ApplicationDbContext>(options =>**

**options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"))**

**);**

**services.AddIdentity<ApplicationUser, IdentityRole>()**

**.AddEntityFrameworkStores<ApplicationDbContext>()**

**.AddDefaultTokenProviders();**

// Add configuration

services.AddOptions();

Ajoutez par la suite la chaine de connexion à votre base de données dans vos paramètres d’application, sous le couple "ConnectionStrings:DefaultConnection" :"chaine de connexion". Cette information est sensible, il convient donc de le faire par le biais des User Secrets comme précédemment.

Avec Visual Studio sous Windows, vous pouvez également définir l’utilisation d’une base de données locale de développement avec la chaine de connexion suivante a: "Server=(localdb)\\mssqllocaldb;Database=aspnetcore-WebApp\_Service\_Provider;Trusted\_Connection=True;MultipleActiveResultSets=true"

### Créez et appliquez la migration

Lancez ces commandes depuis le répertoire du projet :

> dotnet ef migrations add AddUserProperties

> dotnet ef database update

Cette dernière commande met à jour la base de données dont la chaine de connexion a été précisée, afin de créer les différentes tables et valeurs de départs définies.

Note Pour de plus amples informations, vous pouvez consulter la [documentation Entity Framework Core](https://docs.microsoft.com/aspnet/core/data/ef-mvc/?view=aspnetcore-3.1).

### Implémenter le middleware OIDC

Installez le package NuGet Microsoft.AspNetCore.Authentication.OpenIdConnect, en version 3.1.x.

Cela peut être réalisé depuis Visual Studio, ou depuis le CLI avec la commande

**> dotnet add package Microsoft.AspNetCore.Authentication.OpenIdConnect --version 3.1.18**

Copiez ensuite les lignes en gras suivantes dans la fonction *ConfigureServices* du fichier *Startup.cs*

IConfiguration franceConnectConfig = Configuration.GetSection("FranceConnect");

services.Configure<FranceConnectConfiguration>(franceConnectConfig);

**services.AddAuthentication(**

**options =>**

**{**

**options.DefaultChallengeScheme = Scheme.FranceConnect;**

**})**

**.AddOpenIdConnect(FranceConnectConfiguration.ProviderScheme, FranceConnectConfiguration.ProviderDisplayName, options => ConfigureFranceConnect(options, franceConnectConfig.Get<FranceConnectConfiguration>()));**

**// Since updates to SameSite cookie policies, this must be used for the authentication cookies to avoid a correlation error when running without HTTPS**

**services.Configure<CookiePolicyOptions>(**

**options =>**

**{**

**options.MinimumSameSitePolicy = SameSiteMode.Lax;**

**});**

services.AddControllersWithViews();

Ainsi que la fonction ConfigureFranceConnect comme suit :

private void ConfigureFranceConnect(OpenIdConnectOptions oidc\_options, FranceConnectConfiguration fcConfig)

{

// FC refuses unknown parameters in the requests, so the two following options are needed

oidc\_options.DisableTelemetry = true; // This is false by default on .NET Core 3.1, and sends additional parameters such as "x-client-ver" in the requests to FC.

oidc\_options.UsePkce = false; / / This is true by default on .NET Core 3.1, and enables the PKCE mechanism which is not supported by FC.

// FC has restrictions in the nonce (max 128 alphanumeric characters) and errors out in the logout flow otherwise. We use this option so that the nonce does not contain a dot.

oidc\_options.ProtocolValidator.RequireTimeStampInNonce = false;

oidc\_options.SaveTokens = true; // This is needed to keep the id\_token obtained for authentication : we have to send it back to FC to logout.

oidc\_options.ClientId = fcConfig.ClientId;

oidc\_options.ClientSecret = fcConfig.ClientSecret;

oidc\_options.CallbackPath = fcConfig.CallbackPath;

oidc\_options.SignedOutCallbackPath = fcConfig.SignedOutCallbackPath;

oidc\_options.Authority = fcConfig.Issuer;

oidc\_options.ResponseType = OpenIdConnectResponseType.Code;

oidc\_options.Scope.Clear();

oidc\_options.Scope.Add("openid");

foreach (string scope in fcConfig.Scopes)

{

oidc\_options.Scope.Add(scope);

}

oidc\_options.GetClaimsFromUserInfoEndpoint = true;

oidc\_options.TokenValidationParameters.IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(fcConfig.ClientSecret));

oidc\_options.Configuration = new OpenIdConnectConfiguration

{

Issuer = fcConfig.Issuer,

AuthorizationEndpoint = fcConfig.AuthorizationEndpoint,

TokenEndpoint = fcConfig.TokenEndpoint,

UserInfoEndpoint = fcConfig.UserInfoEndpoint,

EndSessionEndpoint = fcConfig.EndSessionEndpoint,

};

oidc\_options.Events = new OpenIdConnectEvents

{

OnRedirectToIdentityProvider = (context) =>

{

context.ProtocolMessage.AcrValues = "eidas" + fcConfig.EIdasLevel;

return Task.FromResult(0);

}

};

// We specify claims to be kept, as .NET Core 2.0+ doesn't keep claims it does not expect.

oidc\_options.ClaimActions.MapUniqueJsonKey("birthcountry", "birthcountry");

oidc\_options.ClaimActions.MapUniqueJsonKey("preferred\_username", "preferred\_username");

oidc\_options.ClaimActions.MapUniqueJsonKey("birthdate", "birthdate");

oidc\_options.ClaimActions.MapUniqueJsonKey("birthplace", "birthplace");

oidc\_options.ClaimActions.MapUniqueJsonKey("gender", "gender");

}

Enfin, ajoutez une définition pour les services d’e-mail à utiliser, par exemple lors de l’envoi d’un e-mail de confirmation de création de compte.

Créez une interface *IEmailSender*

public interface IEmailSender

{

Task SendEmailAsync(string email, string subject, string message);

}

Et son implementation générique.

// This class is used by the application to send Email and SMS

// when you turn on two-factor authentication in ASP.NET Identity.

// For more details see this link http://go.microsoft.com/fwlink/?LinkID=532713

public class AuthMessageSender : IEmailSender

{

public Task SendEmailAsync(string email, string subject, string message)

{

// Plug in your email service here to send an email.

return Task.FromResult(0);

}

}

Avant de les déclarer dans la fonction *ConfigureServices* du fichier *Startup.cs*

// Add application services.

services.AddTransient<IEmailSender, AuthMessageSender>();

Enfin, ajoutez dans la fonction *Configure* du fichier *Startup.cs* l’instruction suivante, entre app.UseRouting(); et app.UseAuthorization();

app.UseRouting();

app.UseAuthentication();

app.UseAuthorization();

ainsi que

app.UseCookiePolicy();

app.UseStaticFiles();

app.UseRouting();

### Validation et Sécurité

Afin de respecter les cahiers de recette indiqués par FranceConnect, il est nécessaire d’afficher le nom et prénom de l’utilisateur une fois connecté. Pour réaliser cela sans requêter la base de données de nombreuses fois, il faut ajouter ces informations au cookie de session de l’utilisateur connecté. Nous créons une classe pour ce faire.

public class ApplicationUserClaimsPrincipalFactory : UserClaimsPrincipalFactory<ApplicationUser, IdentityRole>

{

public ApplicationUserClaimsPrincipalFactory(<ApplicationUser> userManager, <IdentityRole> roleManager, <IdentityOptions> optionsAccessor) : base(userManager, roleManager, optionsAccessor)

{

}

protected override async Task<ClaimsIdentity> GenerateClaimsAsync(ApplicationUser user)

{

var identity = await base.GenerateClaimsAsync(user);

identity.AddClaim(new Claim("GivenName", user.GivenName ?? ""));

//We use the preferred name of the user if one is specified

identity.AddClaim(new Claim("PreferredName", user.PreferredName ?? user.FamilyName));

return identity;

}

}

}

Nous pourrons par la suite faire référence à ces données dans nos pages, après avoir déclaré notre classe lors de la configuration dans la classe Startup, comme suit

services.AddIdentity<ApplicationUser, IdentityRole>()

.AddEntityFrameworkStores<ApplicationDbContext>()

.AddDefaultTokenProviders()

.AddClaimsPrincipalFactory<ApplicationUserClaimsPrincipalFactory>();

.AddClaimsPrincipalFactory<ApplicationUserClaimsPrincipalFactory>();

Sur recommandation des équipes FranceConnect, il convient de hasher le sub (identifiant technique) recu par FranceConnect lors de la connexion. Cela se reflète dans le code par l’implémentation d’une méthode de hashage telle que

class Hashing

{

public static string HashString(string stringToHash)

{

using (SHA512 sha512 = new SHA512Managed())

{

var hash = sha512.ComputeHash(Encoding.UTF8.GetBytes(stringToHash));

StringBuilder stringHashBuilder = new StringBuilder(hash.Length \* 2);

foreach (byte b in hash)

{

stringHashBuilder.Append(b.ToString("X2").ToLower());

}

return stringHashBuilder.ToString();

}

}

}

Utilisée par la suite dans une classe SignInManager telle que celle-ci, afin de hasher le sub, libellé « ProviderKey ».

class FCSignInManager : SignInManager<ApplicationUser>

{

public FCSignInManager(UserManager<ApplicationUser> userManager, IHttpContextAccessor contextAccessor, IUserClaimsPrincipalFactory<ApplicationUser> claimsFactory, IOptions<IdentityOptions> optionsAccessor, ILogger<SignInManager<ApplicationUser>> logger, IAuthenticationSchemeProvider schemes, IUserConfirmation<ApplicationUser> confirmation)

: base(userManager, contextAccessor, claimsFactory, optionsAccessor, logger, schemes, confirmation)

{

}

public override async Task<ExternalLoginInfo> GetExternalLoginInfoAsync(string expectedXsrf = null)

{

ExternalLoginInfo info = await base.GetExternalLoginInfoAsync(expectedXsrf);

if (info != null)

{

// The sub is the ProviderKey obtained from this method.

info.ProviderKey = Hashing.HashString(info.ProviderKey);

}

return info;

}

}

La aussi nous déclarons ce service dans la configuration de la classe Startup

services.AddIdentity<ApplicationUser, IdentityRole>()

.AddEntityFrameworkStores<ApplicationDbContext>()

.AddDefaultTokenProviders()

.AddClaimsPrincipalFactory<ApplicationUserClaimsPrincipalFactory>()

.AddSignInManager<FCSignInManager>();

La capacité de vérifier que le niveau Eidas de l’utilisateur identifié est au moins supérieur à celui exigé a également son importance, et est implémentée à l’aide d’une expression régulière comme suit.

public static class Validation

{

public static bool IsEIdasLevelMet(string acrValues, int minimumEIdasLevel)

{

if (acrValues != null)

{

Regex myRegex = new Regex(@"eidas(\d)");

Match match = myRegex.Match(acrValues);

if (match.Success)

{

int eIdasLevel = int.Parse(match.Groups[1].Value);

if (eIdasLevel >= minimumEIdasLevel)

return true;

}

}

return false;

}

}

Que nous réutiliserons dans les contrôleurs lorsque nécessaire.

## Adapter le comportement de l’application

### Ajouter les fonctions de connexions

Quand un nouvel utilisateur s’authentifie via FranceConnect, l’application va lui créer un compte local en base de données. L’objectif est que l’application préremplisse les informations de l’utilisateur à partir de l’identité pivot fournie par FranceConnect. Nous ajoutons ainsi les contrôleurs, modèles de vue, et vues relatives aux fonctions de création d’un compte local ou externe, d’un mot de passe de compte externe, et l’association/dissociation de compte.

Le code source modèle est disponible sur le répertoire GitHub du starter kit de fournisseur de service, soit les dossiers Account et Manage disponibles respectivement ici. Vous pouvez reproduire ces éléments dans votre dossier source. Veillez à bien utiliser l’espace de noms correspondant à votre application.

[Source/WebApp-Service-Provider-DotNet/Controllers](https://github.com/FranceConnectSamples/franceconnect-service-provider-dotnet-webapp-aspnetcore/tree/v2-RC3-DotNetCore-3.1/Source/WebApp-Service-Provider-DotNet/Controllers), *AccountController*  et *ManageController*

[Source/WebApp-Service-Provider-DotNet/ViewModels](https://github.com/FranceConnectSamples/franceconnect-service-provider-dotnet-webapp-aspnetcore/tree/v2-RC3-DotNetCore-3.1/Source/WebApp-Service-Provider-DotNet/ViewModels), les éléments des dossiers *Account* et *Manage*.

Ainsi que pour les vues, [Source/WebApp-Service-Provider-DotNet/Views](https://github.com/FranceConnectSamples/franceconnect-service-provider-dotnet-webapp-aspnetcore/tree/v2-RC3-DotNetCore-3.1/Source/WebApp-Service-Provider-DotNet/Views), les éléments des dossiers *Account* et *Manage*.

Ajoutez également par la suite dans le fichier *Source/Views/\_ViewImports.cshtml* la directive

@using Microsoft.AspNetCore.Identity

## Ajout des boutons d’inscription et connexion

### Ajout de liens à la barre de navigation

Il est pertinent d’ajouter à la barre de navigation du site les liens d’inscription et de connexion.

Pour cela, créez sous le dossier *Views/Shared* un fichier *\_LoginPartial.cshtml*, contenant le code suivant.

@using System.Security.Claims

@inject SignInManager<ApplicationUser> SignInManager

@inject UserManager<ApplicationUser> UserManager

<ul class="navbar-nav">

@if (SignInManager.IsSignedIn(User))

{

@if (User.HasClaim(ClaimTypes.AuthenticationMethod, FranceConnectConfiguration.ProviderScheme))

{

<li class="nav-item">

<a class="nav-link text-dark" asp-controller="Manage" asp-action="Index">Mon compte</a>

</li>

<li class="nav-item" id="fconnect-profile" data-fc-logout-url="@Url.Action(" LogOff", "Account" )">

<a class="nav-link text-dark" href="#">

@User.FindFirstValue("GivenName") @User.FindFirstValue("PreferredName")

</a>

</li>

}

else

{

<li class="nav-item">

<a class="nav-link text-dark" asp-controller="Manage" asp-action="Index" title="Mon compte">

Bonjour @User.FindFirstValue("GivenName") @User.FindFirstValue("PreferredName") !

</a>

</li>

<li class="nav-item">

<a class="nav-link text-dark" asp-controller="Account" asp-action="LogOff">Se déconnecter</a>

</li>

}

}

else

{

<li class="nav-item">

<a class="nav-link text-dark" asp-controller="Account" asp-action="Register">Créer un compte</a>

</li>

<li class="nav-item">

<a class="nav-link text-dark" asp-controller="Account" asp-action="Login">Se connecter</a>

</li>

}

</ul>

Cette vue tire parti du kit d’intégration FranceConnect proposé par la DINUM. En effet, l’élément d’id fconnect-profile est reconnu par le kit d’intégration, qui vient y ajouter les éléments de profil FranceConnect de l’utilisateur.

Importez la vue partielle précédemment crée, dans le fichier *\_Layout.cshtml*, sous l’element div de bar barre de navigation,

19 <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">

+ <partial name="\_LoginPartial"/>

20 <ul class="navbar-nav flex-grow-1">

Ainsi que le javascript du kit d’intégration FranceConnect de la DINUM en fin de document.

<script src="https://app.franceconnect.gouv.fr/js/franceconnect.js"></script>

//Ou, dans le cas d’un environnement de developpement

// script src="https://fcp.integ01.dev-franceconnect.fr/js/franceconnect.js"></script>

@RenderSection("Scripts", required: false)

</body>

### Définir le style du bouton FranceConnect des pages de connexion/inscription

Importez dans le dossier wwwroot/images du projet les images des boutons, fournies dans les [packs d’images de la documentation FranceConnect](https://partenaires.franceconnect.gouv.fr/fcp/fournisseur-service#buttons) , ou directement dans le [dossier wwwroot/images du répertoire GitHub](https://github.com/FranceConnectSamples/franceconnect-service-provider-dotnet-webapp-aspnetcore/tree/v2-RC3-DotNetCore-3.1/Source/WebApp-Service-Provider-DotNet/wwwroot/images) du starter kit. Créez ensuite une nouvelle classe CSS dans le fichier site.css pour appliquer le style exigé par la [documentation FranceConnect](https://partenaires.franceconnect.gouv.fr/fcp/fournisseur-service#buttons) :

.btn-fc {

background: url(/images/franceconnect-bouton.png) no-repeat center;

background-image: url(/images/franceconnect-bouton.svg);

background-size: contain;

height: 48px;

width: 178px;

border: none;

}

.btn-fc:hover {

background: url(/images/franceconnect-bouton-hover.png) no-repeat center;

background-image: url(/images/franceconnect-bouton-hover.svg);

}

#fconnect-profile > a {

padding-left: 40px !important;

font-size: inherit !important;

background-size: 32px !important;

}

## Récupérer des ressources auprès d’un fournisseur de données

D’après la documentation FranceConnect, un fournisseur de service peut demander des ressources à un fournisseur de données avec le consentement de l’utilisateur.

Le protocole utilisé pour la récupération de données est OAuth 2. Comme OpenID Connect est basé sur OAuth 2, il est possible d’utiliser un client OpenID Connect pour la récupération de données.

Ajoutez donc le package NuGet IdentityModel, en version 5.1.0.

Cela peut être réalisé depuis Visual Studio, ou depuis le CLI avec la commande

> dotnet add package IdentityModel --version 5.1.0

Afin de proposer à l’utilisateur d’accéder à ses ressources depuis un autre fournisseur, ajouter un nouveau contrôleur dédié à cela, DataController. Vous pouvez récupérer celui proposé dans ce starter kit sous  [Source/WebApp-Service-Provider/Controllers/DataController.cs](https://github.com/FranceConnectSamples/franceconnect-service-provider-dotnet-webapp-aspnetcore/blob/v2-RC3-DotNetCore-3.1/Source/WebApp-Service-Provider-DotNet/Controllers/DataController.cs) . Veillez à l’inclure dans votre espace de noms existant.

Il est important de vérifier que l’utilisateur dont on aura obtenu le consentement pour la récupération des données est le même que l’utilisateur actuellement connecté.

Pour cela nous validons la signature du jeton d’identité obtenu lors du consentement, avant de lire son sub (identifiant technique) grâce à une méthode définie dans notre classe de Validation, comme suit :

static public JwtSecurityToken ReadAndValidateToken(string token, SecurityKey securityKey)

{

var tokenHandler = new JwtSecurityTokenHandler();

var validationParameters = new TokenValidationParameters

{

ValidateAudience = false,

ValidateIssuer = false,

ValidateActor = false,

ValidateIssuerSigningKey = true,

IssuerSigningKey = securityKey

};

SecurityToken validatedToken;

try

{

tokenHandler.ValidateToken(token, validationParameters, out validatedToken);

}

catch (Exception)

{

return null;

}

return (JwtSecurityToken)validatedToken;

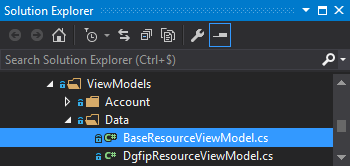
}

}

Cette classe est par la suite utilisée dans le DataController avant d’appeler le fournisseur de données.

De la même façon, vous pouvez récupérer les modèles de vues sous [Source/WebApp-Service-Provider-DotNet/ViewModels/Data](https://github.com/FranceConnectSamples/franceconnect-service-provider-dotnet-webapp-aspnetcore/tree/v2-RC3-DotNetCore-3.1/Source/WebApp-Service-Provider-DotNet/ViewModels/Data), ainsi que les vues sous [Source/WebApp-Service-Provider-DotNet/Views/Data](https://github.com/FranceConnectSamples/franceconnect-service-provider-dotnet-webapp-aspnetcore/tree/v2-RC3-DotNetCore-3.1/Source/WebApp-Service-Provider-DotNet/Views/Data)

Les modèles de vues sont ici modélisés sous la forme d’un modèle de base, et d’autres qui en héritent afin d’avoir un point de rattachement entre les ressources obtenues de différents fournisseurs :



Modèles de vue décrivant les ressources externes

public class BaseResourceViewModel

{

}

public class DgfipResourceViewModel : BaseResourceViewModel

{

public string Rfr { get; set; }

public char SitFam { get; set; }

public string NbPart { get; set; }

public Pac Pac { get; set; }

}

public class Pac

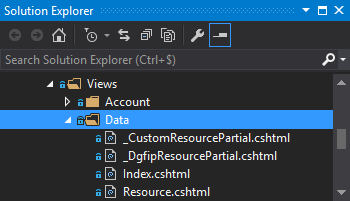
{

public string NbPac { get; set; }

}

Les noms des propriétés doivent correspondrent à ceux du JSON retourné par le fournisseur de données.

Par la suite, les différentes vues se présentent de la façon suivante la vue principale, Index, permet à l’utilisateur de requêter le fournisseur de son choix. Une vue aura ensuite pour objectif d’afficher les données, et celle-ci invoquera la vue partielle correspondant au modèle de données requêté, parmi celles définies :



Vue Index :

@model IEnumerable<DataProvider>

@{

ViewData["Title"] = "Obtenir des données externes";

}

<h2>@ViewData["Title"].</h2>

@foreach (DataProvider provider in Model)

{

<a asp-controller="Data" asp-action="GetResource" asp-route-provider="@provider.Name" class="btn btn-default">@provider.Name</a>

}

Vue Resource :

@using WebApp\_Service\_Provider\_DotNet.ViewModels.Data

@model BaseResourceViewModel

@{

ViewData["Title"] = "Données obtenues";

}

<h2>@ViewData["Title"].</h2>

@if (Model is DgfipResourceViewModel)

{

<partial name="\_DgfipResourcePartial" model="Model"/>

}

Vue Partielle pour les données de la Direction Générale des Finances Publiques :

@using WebApp\_Service\_Provider\_DotNet.ViewModels.Data

@model DgfipResourceViewModel

<table class="table table-striped">

<thead>

<tr>

<th>Scopes</th>

<th>Valeurs</th>

</tr>

</thead>

<tbody>

<tr>

<td>Revenu fiscal de référence</td>

<td>@Model.Rfr</td>

</tr>

<tr>

<td>Situation familiale</td>

<td>@Model.SitFam</td>

</tr>

<tr>

<td>Nombre de parts</td>

<td>@Model.NbPart</td>

</tr>

<tr>

<td>Nombre de personnes à charge</td>

<td>@(Model.Pac != null ? Model.Pac.NbPac : 0)</td>

</tr>

<tr>

<td>Adresse fiscale de taxation</td>

<td>@Model.aft</td>

</tr>

</tbody>

</table>

Enfin, modifiez la vue partielle \_LoginPartial pour ajouter un lien vers le contrôleur DataController :

<ul class="navbar-nav">

@if (SignInManager.IsSignedIn(User))

{

<li class="nav-item">

<a class="nav-link text-dark" asp-controller="Data" asp-action="Index">Données</a>

</li>

@if (User.HasClaim(ClaimTypes.AuthenticationMethod, FranceConnectConfiguration.ProviderScheme))

Avant d’ajouter la route suivante dans l’instruction UseEndpoints() de la fonction *Configure* du fichier *Startup.cs*

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute("default", "{controller=Home}/{action=Index}/{id?}");

endpoints.MapControllerRoute("data-consent-callback", franceConnectConfig.Value.DataCallbackPath, new { controller = "Data", action = "GetResourceCallback" });

});

# Obtenir et déployer le canevas de fournisseur de service préconfiguré du kit de démarrage

## imageObtenir les sources du fournisseur de service

Le kit de démarrage FranceConnect est disponible avec le code source des différents exemples/accélérateurs ainsi proposés sur le repo/la forge communautaire GitHub à l’adresse <https://github.com/FranceConnectSamples>.

Vous trouverez à cette adresse notamment le projet relatif à l’exemple/accélérateur de fournisseur de service (FS) : [franceconnect-service-provider-dotnet-webapp-aspnetcore](https://github.com/FranceConnectSamples/franceconnect-service-provider-dotnet-webapp-aspnetcore).

Les solutions communautaires évoluant avec l’investissement de la communauté, il est souhaitable de vous assurer que vous disposez de la dernière version, ne serait-ce que pour bénéficier, le cas échéant, de l’ensemble des correctifs. La version correspondant à ce document est celle labelisée *v2-RC3-DotNetCore-3.1*

La disponibilité du projet comme répertoire GitHub signifie que vous pouvez non seulement obtenir le package relatif au projet avec le code source correspondant à compiler mais également cloner le repo Git, accéder au code source, le compiler le code mais aussi soumettre des demandes de type Pull (pull request) à l’instar de n’importe quel autre package open source vous pourriez trouver sur GitHub et, ce afin de proposer et/ou de collaborer sur des évolutions au niveau du repo.

Les deux prochaines sections explorent les deux options possibles.

### Télécharger les sources du fournisseur de service

Pour télécharger les sources depuis la forge GitHub, le moyen le plus simple et rapide est de télécharger le package complet sous forme de fichier archive .zip.

**Pour cela, procédez comme suit :**

1. **Ouvrez une session avec le browser et naviguer vers le projet** [franceconnect-service-provider-dotnet-webapp-aspnetcore](https://github.com/FranceConnectSamples/franceconnect-service-provider-dotnet-webapp-aspnetcore)**.**
2. **Cliquez sur Clone or Download.**



1. **Cliquez sur Download ZIP.**
2. **Enregistrez le fichier *franceconnect-service-provider-dotnet-webapp-aspnetcore-master.zip* en local sur votre machine.**
3. **Extrayez le contenu du fichier *franceconnect-service-provider-dotnet-webapp-aspnetcore-master.zip* sur un disque local, par exemple sous le dossier *C:\Code*.**

**Le code source du canevas de fournisseur de service FranceConnect (FS) est alors situé sous le dossier *C:\Code\franceconnect-service-provider-dotnet-webapp-aspnetcore-master\Source.***

Si cette approche constitue le moyen le plus simple, la seconde option abordée ci-après se révèle plus adaptée pour mettre en place une gestion de versions.

### Cloner les sources du fournisseur de service

 Comme décrit précédemment, le kit de démarrage FranceConnect est disponible sur le repo **GitHub**, qui repose sur **Git** comme système de contrôle de code source.

Pour accéder au package du projet et au code source correspondant, il est préférable dans cette seconde approche de disposer d’une connaissance préalable de [Git](http://git-scm.com/), [GitHub](https://github.com/FranceConnectSamples) ou [Visual Studio](https://visualstudio.microsoft.com/). Ceci étant, les étapes suivantes fournissent des informations et des liens pour vous aider à démarrer.

Note Pour plus d’informations sur la façon de configurer Git et GitHub, consultez l’article [Set up Git](https://help.github.com/articles/set-up-git/) sur le site de GitHub.

Pour accéder au package source du projet et contribuer davantage à celui, vous devez i) créer une branche (fork) sur le repo Git qui le contient et ii) cloner cette branche sur votre machine de développement.

Pour créer un fork sur le repo Git, procédez comme suit :

1. **Ouvrez une session de navigation et naviguez vers le projet** [franceconnect-service-provider-dotnet-webapp-aspnetcore](https://github.com/FranceConnectSamples/franceconnect-service-provider-dotnet-webapp-aspnetcore).
2. Cliquez sur **Fork** dans le coin supérieur droit la fenêtre de votre browser.

Vous pouvez dès lors cloner la branche à l’aide de l’application GitHub ou via la ligne de commande dans le Shell Git.

Pour cloner le repo avec le Shell Git, procédez comme suit :

1. Ouvrez un Shell Git Shell en double-cliquant sur l’icône éponyme sur votre bureau Windows.
2. Depuis l’invite de commande, tapez la commande suivante :

> Copygit clone https://github.com/FranceConnectSamples/franceconnect-service-provider-dotnet-webapp-aspnetcore

Le clonage démarre.

> Copygit clone https://github.com/FranceConnectSamples/franceconnect-service-provider-dotnet-webapp-aspnetcore

Cloning into 'franceconnect-service-provider-dotnet-webapp-aspnetcore'...

Le package du canevas de fournisseur de service FranceConnect (FS) est disponible sous le dossier *franceconnect-service-provider-dotnet-webapp-aspnetcore* situé dans le répertoire ou vous avez lancé la commande git clone.

**Le code source associé est alors situé sous le dossier *franceconnect-service-provider-dotnet-webapp-aspnetcore/Source.***

Les environnements Visual Studio 2019 ou Visual Studio Code proposent également une intégration directe avec GitHub que nous ne développons pas ici.

Quelle que soit l’approche retenue, et une fois le code source téléchargé, nous vous invitons à passer à la section suivante.

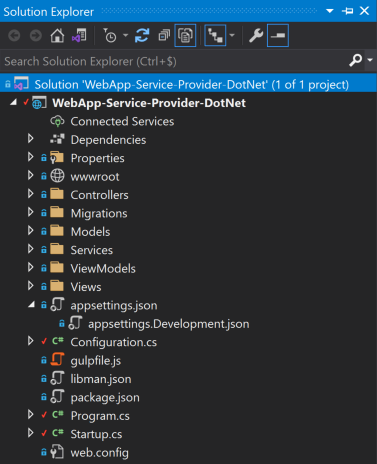
Remarque importante Nous faisons référence au dossier où se situe le code source du canevas de fournisseur de service (FS) comme étant le dossier Kit-Demarrage dans la suite de ce document.

## Organisation du code source du canevas de fournisseur de service (FS)

A l’instar de ce qui est décrit dans la section § [*Créer un fournisseur de service FranceConnect avec ASP.NET Core*](https://microsoft-my.sharepoint.com/personal/philber_microsoft_com/Documents/Drafts/Kit%20de%20démarrage%20FranceConnect/Versions/2016-06/franceconnect-service-provider-dotnet-webapp-aspnetcore/Documentation/Mettre%20en%20oeuvre%20un%20fournisseur%20de%20service%20FranceConnect%20avec%20ASP.NET%20Core-PHILBER002.docx#_Créer_un_fournisseur), la solution préconfigurée prête à l’emploi est constituée d’un unique projet ASP.NET Core MVC de type **Web Application**. Ouvrez le fichier solution Visual Studio .sln depuis le dossier **Kit-Demarrage***.*

### Comprendre l’organisation du projet

Le canevas de fournisseur de service proposé par la solution Visual Studio se compose du projet suivant :



Architecture du projet

### Appréhender les éléments de configuration

La configuration de l’application se trouve dans le fichier *Appsettings.json*. Il est possible de créer des extensions de ce fichier (par exemple : *Appsettings.Production.json*) pour chaque environnement.

{

"ConnectionStrings": {

"DefaultConnection": "<chaine de connexion à votre base de données>"

},

"Logging": {

"IncludeScopes": false,

"LogLevel": {

"Default": "Debug",

"System": "Information",

"Microsoft": "Information"

}

},

"FranceConnect": {

**"ClientId": "<votre client\_id FranceConnect>",**

"CallbackPath": "/login-callback",

"SignedOutCallbackPath": "/logout-callback",

"DataCallbackPath": "/data-callback",

"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",

**"EIdas": "<niveau eIDAS souhaité : eidas1, eidas2 ou eidas3>",**

"DataProviders": [

{

"Name": "<nom du fournisseur de données>",

"Scopes": [ "<scope>" ],

"Endpoint": "<URL d’accès aux données>"

}

]

}

}

|  |  |
| --- | --- |
| **Section de configuration** | **Description** |
| ConnectionStrings | Contient la chaine de connexion à la base de données |
| Logging | Contient les paramètres du Logger |
| FranceConnect | Contient les paramètres de connexion à FranceConnect |

Les points de terminaisons indiqués ci-dessous sont ceux correspondant à un environnement d’intégration. Ceux-ci, ainsi que ceux d’un environnement de production, sont indiqués dans la [documentation FranceConnect pour un fournisseur de service](https://partenaires.franceconnect.gouv.fr/fcp/fournisseur-service#sign_in)

## Déployer le projet

Pour déployer le fournisseur de service (FS) dans l’environnement Microsoft Azure, de façon manuelle comme automatisée à l’aide des outils DevOps, la procédure est documentée sur le répertoire du starter kit, sous [Documentation/GitHub Actions & Déploiement sur Azure.md](https://github.com/FranceConnectSamples/franceconnect-service-provider-dotnet-webapp-aspnetcore/blob/v2-RC3-DotNetCore-3.1/Documentation/GitHub%20Actions%20%26%20D%C3%A9ploiement%20sur%20Azure.md)

# Tester le fournisseur de service

## Configurer le compte FranceConnect

Vous êtes obligé de renseigner vos URL de callback à FranceConnect pour que l’authentification via FranceConnect fonctionne.

1. **Ouvrez une session browser et** rendez-vous sur [votre compte partenaires FranceConnect](https://partenaires.franceconnect.gouv.fr/login)
2. Ajoutez les URL de callback que vous avez précisées sur votre fichier *appsettings.json*, initialisées dans notre répertoire de démarrage à :
   1. <*FS URL*>/login-callback
   2. <*FS URL*>/data-callback
   3. <*FS URL*>/logout-callback

Où <*FS URL*> est l’adresse à laquelle votre fournisseur de service (FS) est déployé dans Microsoft Azure.

Vous pouvez également utiliser une instance locale de développement, ne vous nécessitant pas de compte si vous utilisez le set de crédentités client\_id et client\_secret fourni sur la [documentation FranceConnect](https://partenaires.franceconnect.gouv.fr/fcp/fournisseur-service#sign_in), respectivement

Client\_id : '211286433e39cce01db448d80181bdfd005554b19cd51b3fe7943f6b3b86ab6e'

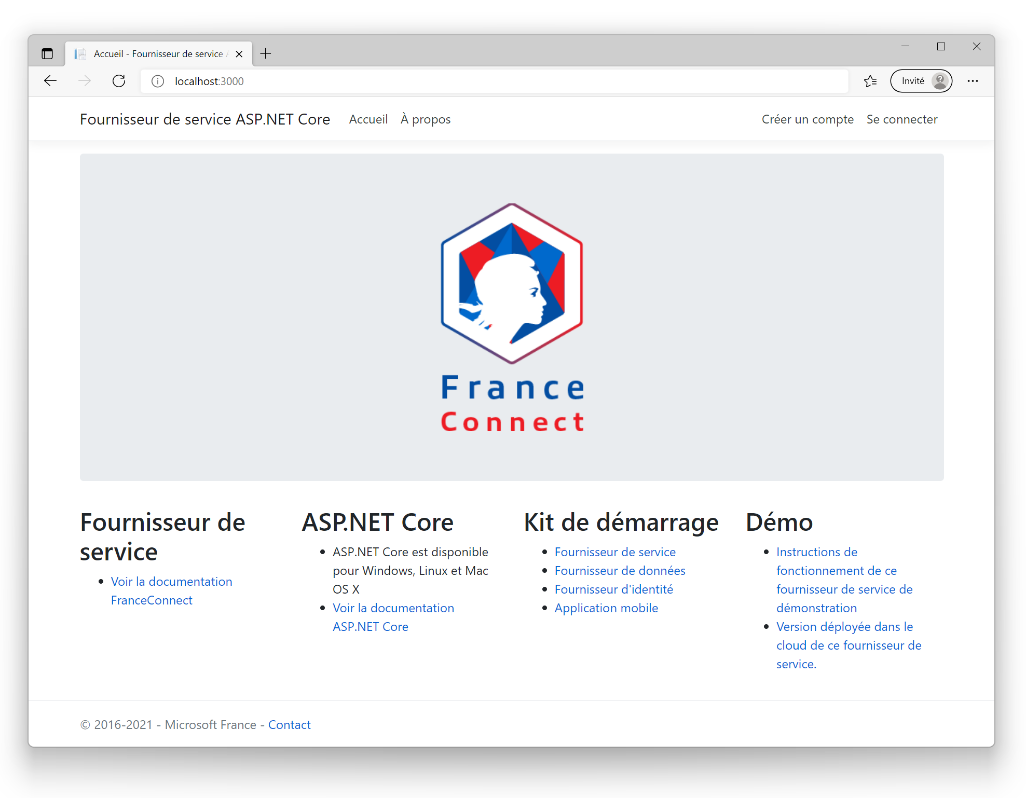
Client\_secret : '2791a731e6a59f56b6b4dd0d08c9b1f593b5f3658b9fd731cb24248e2669af4b'

Ceux-ci sont conçu pour fonctionner dans le cas d’un environnement de développement localhost, hébérgé en http *(et non https, auques cas vos requêtes se verront refuser par FranceConnect*) sous certains ports définis, et autorisant des URLs de callback spécifiées. Vous pouvez retrouver toutes ces informations [ici](https://partenaires.franceconnect.gouv.fr/fcp/fournisseur-service#sign_in).

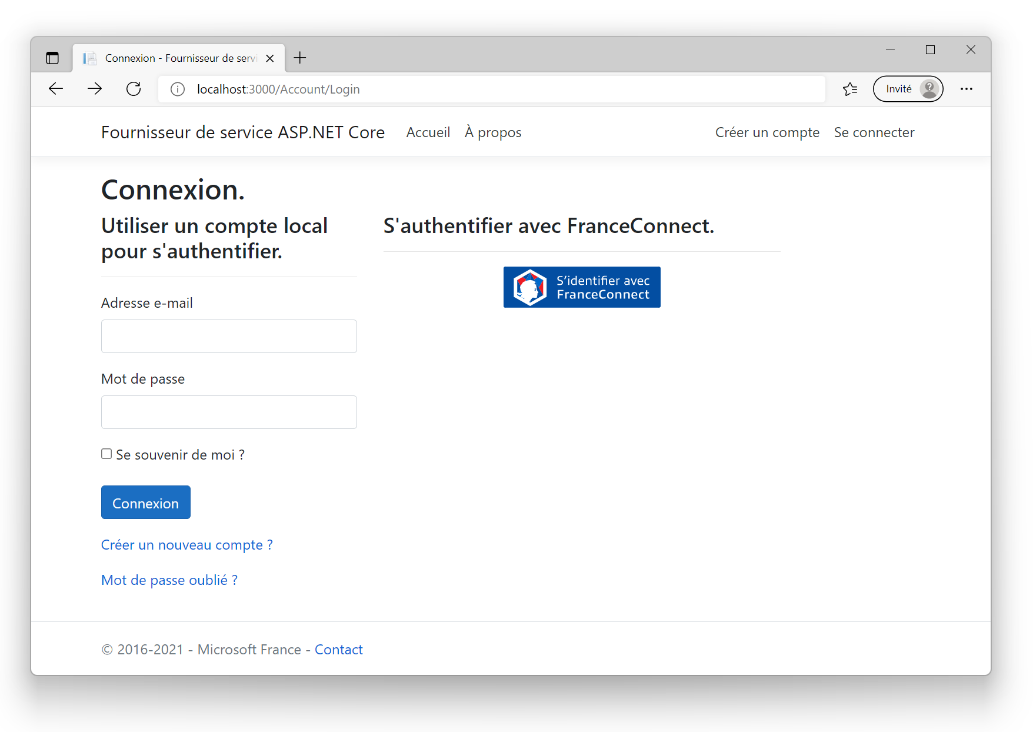
## Se connecter avec le système FranceConnect

Pour se connecter avec le système FranceConnect, procédez comme suit :

1. Ouvrez une session browser et naviguez vers le canevas de fournisseur de service FranceConnect, le vôtre, ou l’instance de démonstration à l’adresse <https://aka.ms/FranceConnect-FS>.



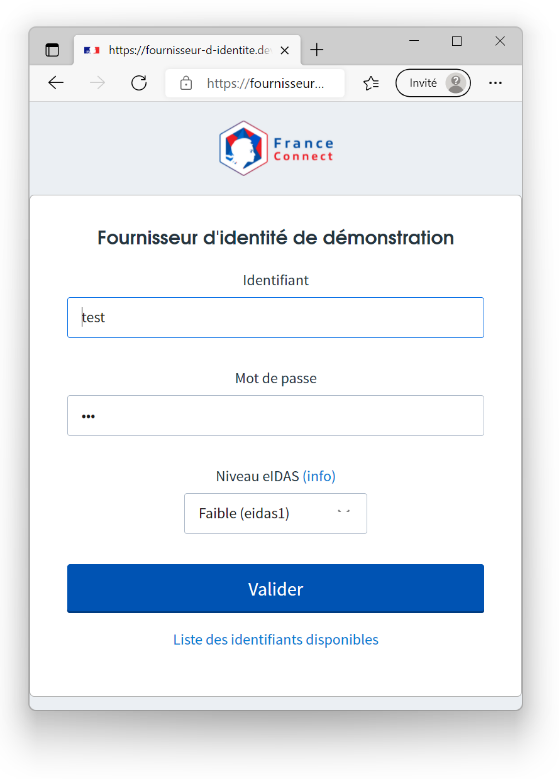
1. Cliquez sur **Se connecter** dans le coin supérieur droit de la fenêtre du browser.



1. Cliquez sur le bouton FranceConnect à droite. Vous êtes alors redirigé(e)s vers la plateforme FranceConnect.



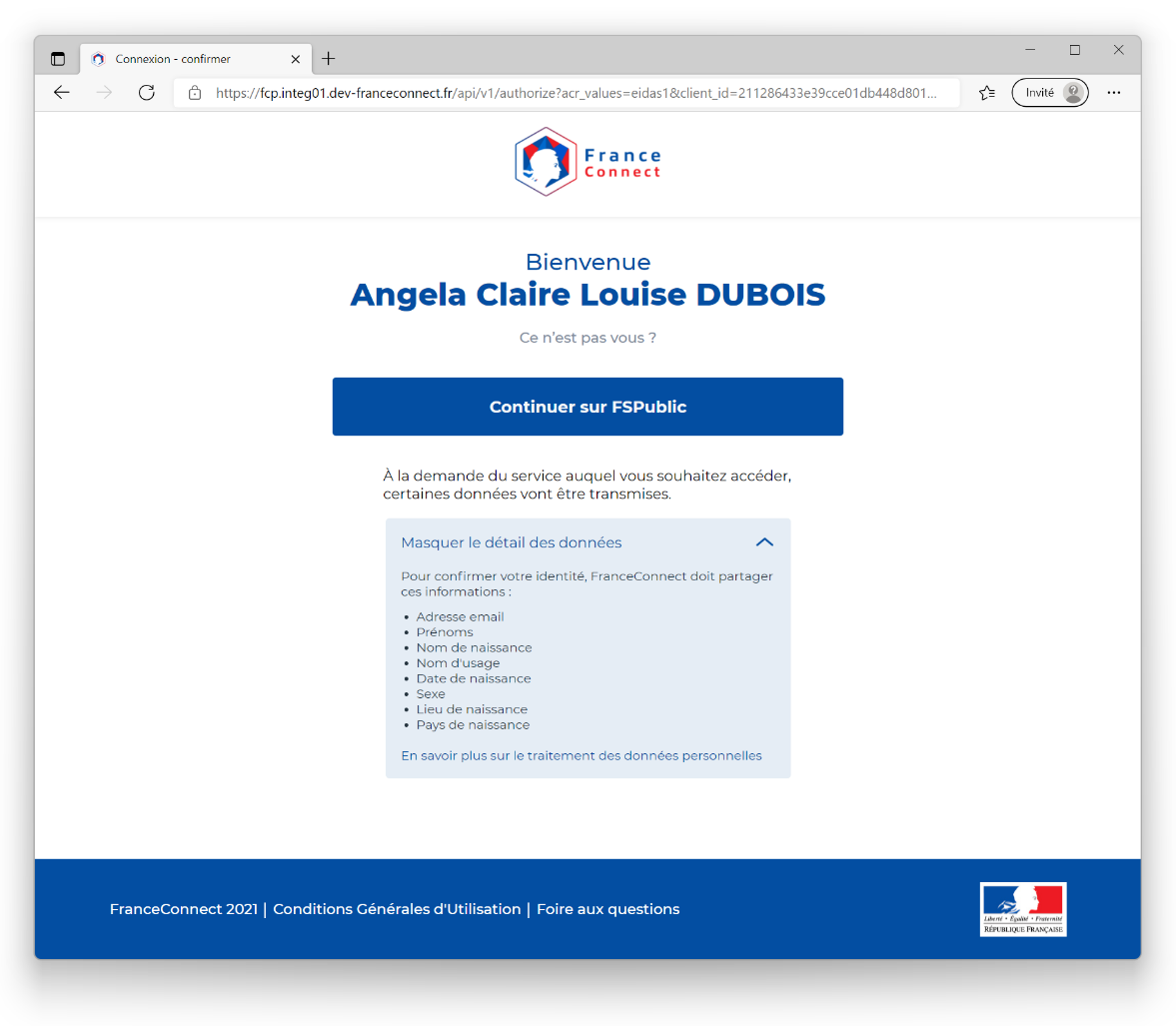
1. Sélectionnez le fournisseur d’identité compatible FranceConnect avec lequel vous souhaitez vous identifier et authentifier, par exemple le fournisseur d’identité factice de démonstration eidas élevé. Vous êtes alors redirigé(e)s vers ce fournisseur d’identité.



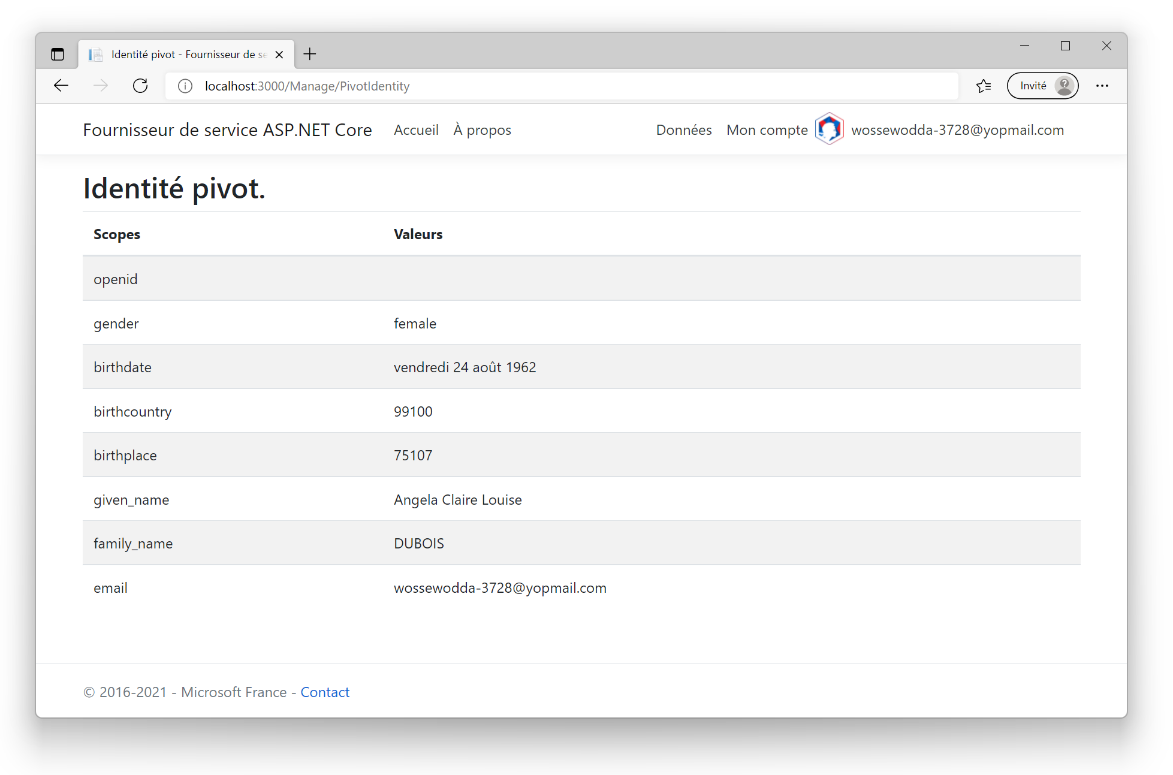
1. Au niveau du fournisseur d’identité factice de démonstration, identifiez-vous à l’aide de crédentités de test. Précisez pour cela les informations suivantes dans le formulaire qui vous est proposé :

* **Numéro fiscal** : test
* **Mot de passe** : 123

1. Cliquez sur **Valider**. Vous êtes redirigé(e)s vers la plateforme FranceConnect qui vous invite à donner votre consentement.



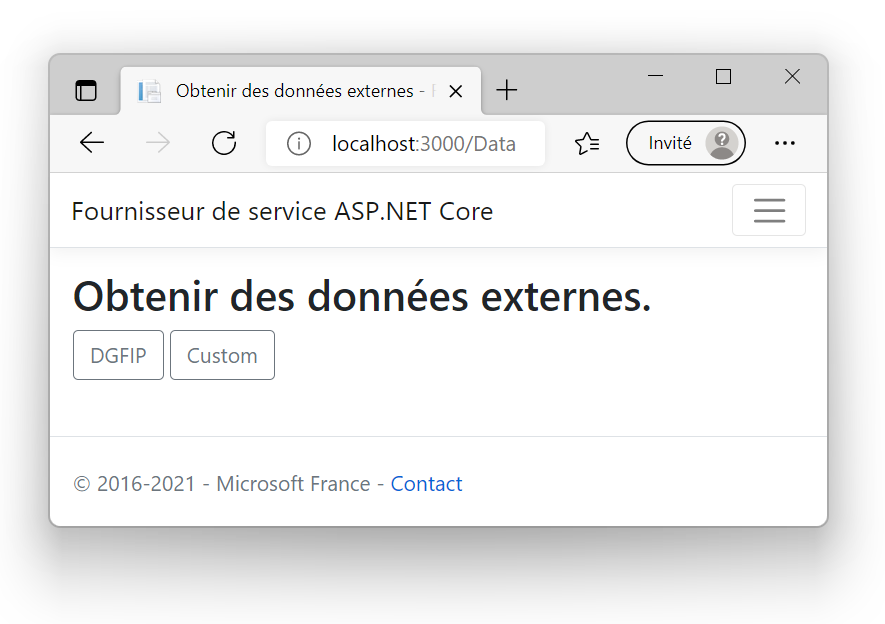
1. Si tel est le cas, cliquez sur **Continuer sur FSPublic**. Vous êtes redirigé(e)s vers le canevas de fournisseur de service, qui vous préremplit les champs d’inscription. Les informations sur votre identité pivot telles que transmises par FranceConnect s’affichent alors.



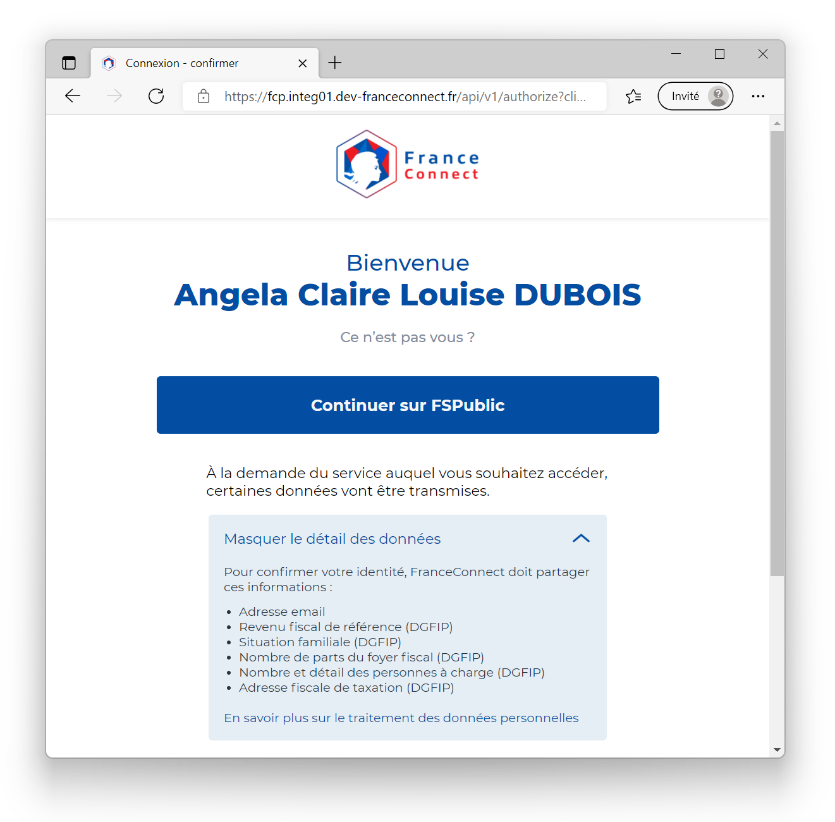
## Invoquer un fournisseur de données via le système FranceConnect

Pour invoquer un service de données depuis le canevas de fournisseur de service, procédez comme suit :

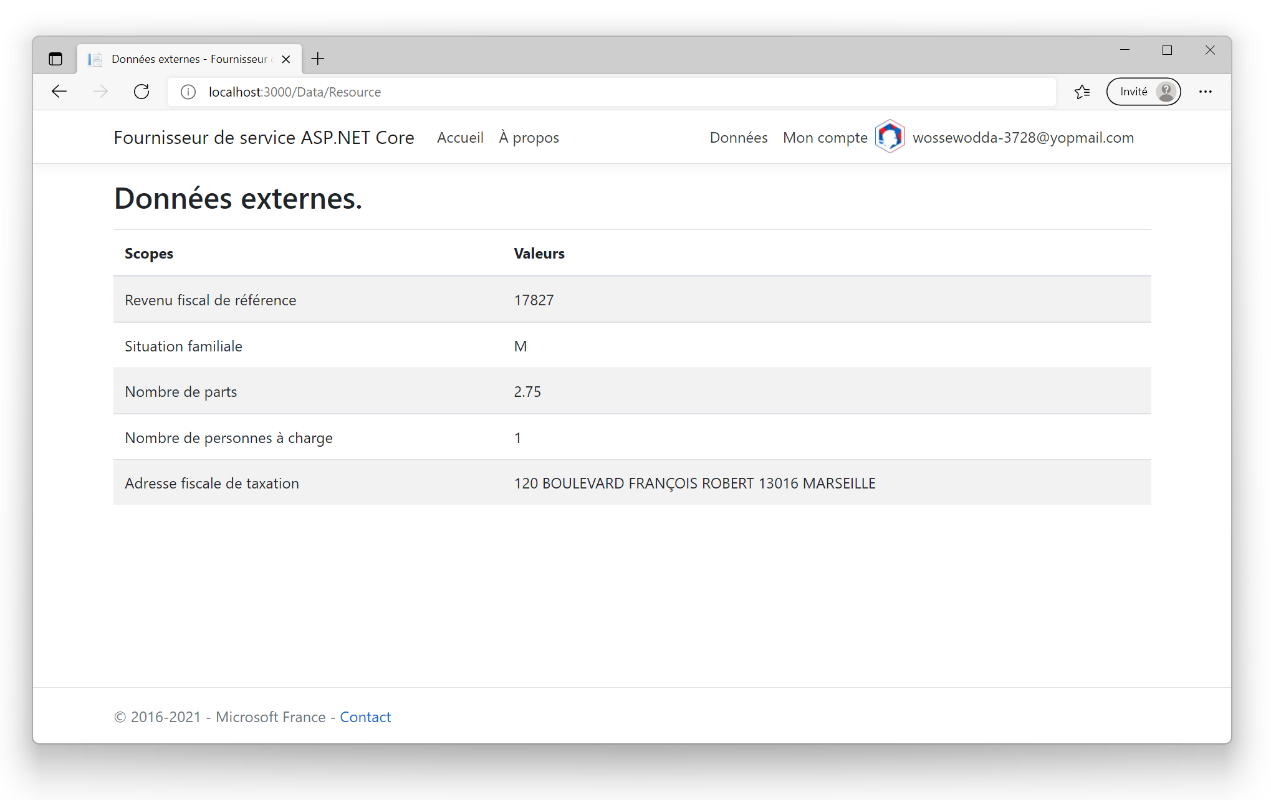
1. Depuis la fenêtre de browser précédente, cliquez sur **Données**. Les fournisseurs de données (FD) configurés au niveau du canevas de fournisseur de service s’affichent.



1. Cliquer sur **DGFIP**. Vous êtes redirigé(e)s sur la plateforme FranceConnect qui vous invite à donner votre consentement avant de transmettre la requête au service de données sélectionné.



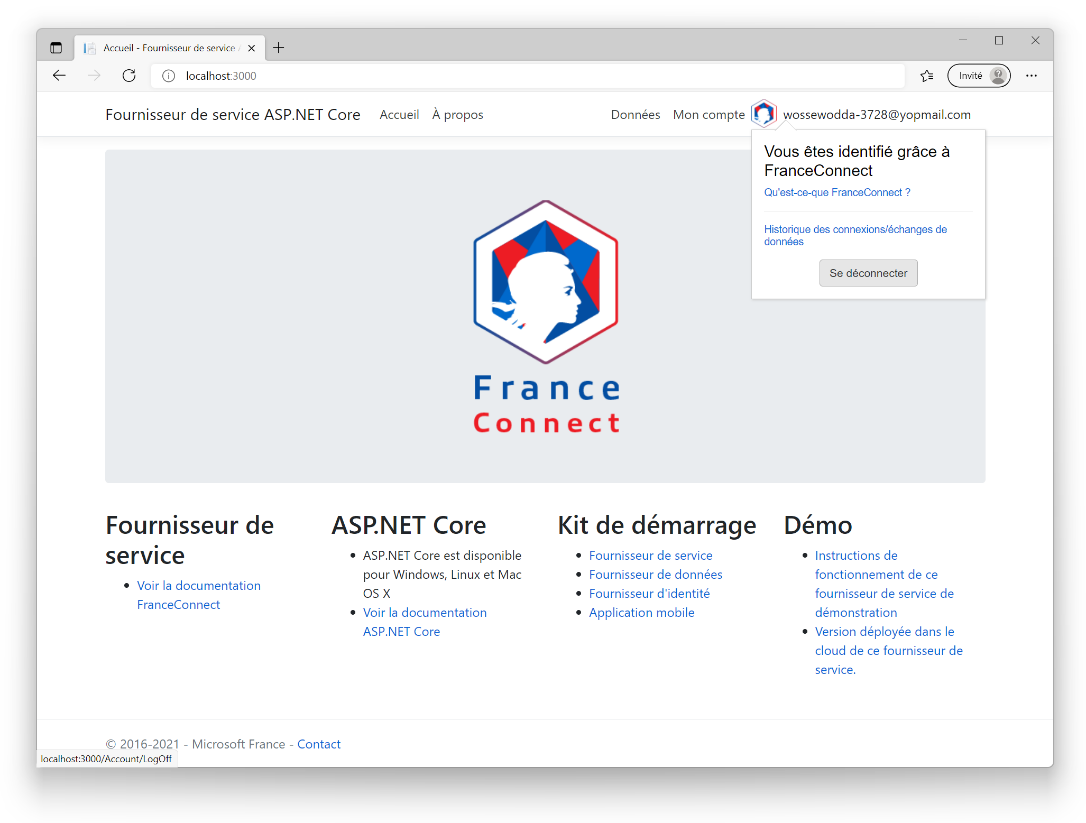
1. Si tel est le cas, cliquer sur **Continuer sur FSPublic**. Une fois la requête traitée et de retour dans le canevas de fournisseur de service, les données ainsi obtenues sont affichées dans la fenêtre.



## Se déconnecter du système FranceConnect

Pour se déconnecter depuis le canevas de fournisseur de service, procédez comme suit :

1. Depuis la fenêtre de navigation précédente, cliquez sur votre nom d’utilisateur à côté du logo FranceConnect dans le coin supérieur droit de la fenêtre de browser. Une boite de dialogue s’affiche.



1. Cliquez sur **Se déconnecter**. Vous êtes alors redirigé(e)s vers le canevas de fournisseur de service ;

Ceci conclut ce guide de mise en œuvre d’un canevas de fournisseur de service (FS) FranceConnect.

# Annexe A. Références techniques

Le document présent s’appuie sur les diverses ressources d’intégration mises à disposition par la DINUM et en particulier de la [documentation technique d’intégration](https://partenaires.franceconnect.gouv.fr/documentation).

Il repose plus particulièrement pour la définition des éléments de mise en œuvre du fournisseur de service FranceConnect ainsi proposé sur la [documentation](https://partenaires.franceconnect.gouv.fr/fcp/fournisseur-service).