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

Kit de démarrage FranceConnect

Publication : Juin 2016

Version : 2016-06

Auteur : Charles de Vandière

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](#_Toc461792502)

[A propos du kit de démarrage FranceConnect 5](#_Toc461792503)

[Licence 5](#_Toc461792504)

[Commentaires 5](#_Toc461792505)

[Prérequis 6](#_Toc461792506)

[Connaissances et attendu 6](#_Toc461792507)

[Créer un fournisseur de service FranceConnect avec ASP.NET Core 7](#_Toc461792508)

[Obtenir des identifiants FranceConnect 7](#_Toc461792509)

[Créer un fournisseur de service 7](#_Toc461792510)

[Configurer l’application 8](#_Toc461792511)

[Modifier le fichier de configuration 8](#_Toc461792512)

[Comprendre la classe de configuration 8](#_Toc461792513)

[Ajouter le client secret 9](#_Toc461792514)

[Implémenter le middleware OIDC 9](#_Toc461792515)

[Modifier le schéma de donnée 10](#_Toc461792516)

[Modifier le modèle ApplicationUser 10](#_Toc461792517)

[Appliquer la migration 10](#_Toc461792518)

[Adapter le comportement de l’application 10](#_Toc461792519)

[Adapter la création d’un compte local 10](#_Toc461792520)

[Adapter la création du mot de passe 12](#_Toc461792521)

[Rapprocher les comptes 12](#_Toc461792522)

[Implémenter l’association de compte 12](#_Toc461792523)

[Implémenter la dissociation de compte 13](#_Toc461792524)

[Déconnecter l’utilisateur 14](#_Toc461792525)

[Implémenter le bouton FranceConnect 14](#_Toc461792526)

[Définir le style du bouton 14](#_Toc461792527)

[Ajouter le bouton sur la page Login 14](#_Toc461792528)

[Ajouter le bouton sur la page ManageLogins 15](#_Toc461792529)

[Mettre en œuvre le kit d’intégration FranceConnect de la DINSIC 16](#_Toc461792530)

[Récupérer des ressources auprès d’un fournisseur de données 18](#_Toc461792531)

[Déployer le canevas de fournisseur de service préconfiguré du kit de démarrage 23](#_Toc461792532)

[Obtenir les sources du fournisseur de service 23](#_Toc461792533)

[Télécharger les sources du fournisseur de service 23](#_Toc461792534)

[Cloner les sources du fournisseur de service 24](#_Toc461792535)

[Organisation du code source du canevas de fournisseur de service (FS) 25](#_Toc461792536)

[Comprendre l’organisation du projet 25](#_Toc461792537)

[Appréhender les éléments de configuration 25](#_Toc461792538)

[Déployer le projet dans Microsoft Azure 26](#_Toc461792539)

[Créer l’application web dans Microsoft Azure 26](#_Toc461792540)

[Publier le fournisseur de service dans Microsoft Azure 26](#_Toc461792541)

[Tester le fournisseur de service 28](#_Toc461792542)

[Configurer le compte FranceConnect 28](#_Toc461792543)

[Se connecter avec le système FranceConnect 28](#_Toc461792544)

[Invoquer un fournisseur de données via le système FranceConnect 30](#_Toc461792545)

[Se déconnecter du système FranceConnect 32](#_Toc461792546)

[Annexe A. Références techniques 34](#_Toc461792547)

# A propos du système FranceConnect

La plateforme FranceConnect est un nouveau système d’identification à l’initiative de la Direction interministérielle du numérique et du système d’information et de communication de l’État (DINSIC) 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.

Pour cela, 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 connaitrait pas encore l’usager, le bouton permet de sélectionner une identité compatible dont l’usager disposerait déjà (impôts, sécurité sociale, Caf, 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 environnent 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 DINSIC 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 DINSIC.

# 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](http://www.microsoft.com/france/entreprises/secteur-public/solutions-pour-secteur-public.aspx)[[1]](#footnote-1). 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](http://www.dotnetfoundation.org/aspnet-core)[[2]](#footnote-2) mise à disposition sous licence libre par la [fondation .NET](http://www.dotnetfoundation.org/)[[3]](#footnote-3).

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 1.0. 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

Le canevas du kit de démarrage FranceConnect pour réaliser un fournisseur de service (FS) nécessite un ordinateur sous Windows 7 ou ultérieur.

## Connaissances et attendu

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 Community 2015](https://go.microsoft.com/fwlink/?LinkId=691978&clcid=0x40c)[[4]](#footnote-4),
* De la programmation avec le Microsoft Framework .NET,
* Des dernières technologies Web comme ASP.NET Core MVC,

Note L’article [Introduction to ASP.NET Core](https://docs.asp.net/en/latest/conceptual-overview/aspnet.html)[[5]](#footnote-5) 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/)[[6]](#footnote-6) 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.

Avant de continuer, si vous ne possédez pas d’un compte Azure, nous vous invitons à vous rendre à l’adresse <https://azure.microsoft.com/fr-fr/pricing/free-trial/> afin d’en créer un, gratuitement.

# 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 2015 en partant de la création d’un projet générique.

Si vous le souhaitez, vous pouvez utiliser directement le projet complet terminé et préconfiguré proposé dans le cadre du kit de démarrage FranceConnect : [franceconnect-service-provider-dotnet-webapp-aspnetcore](https://github.com/FranceConnectSamples/franceconnect-service-provider-dotnet-webapp-aspnetcore)[[7]](#footnote-7).

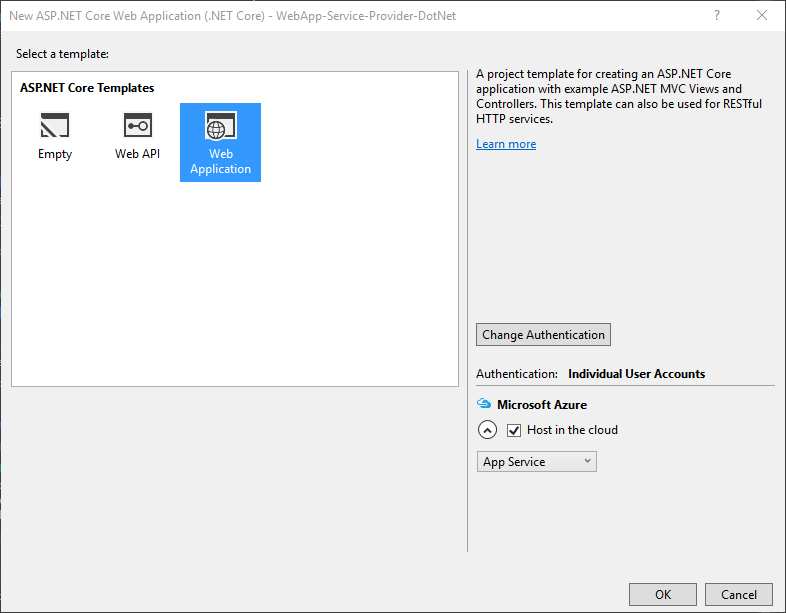
Pour cela, rendez-vous à la section § *Déployer le fournisseur de service préconfiguré du kit de démarrage*.

## Obtenir des identifiants FranceConnect

Avant de créer votre fournisseur de service, vous devez vous [inscrire](https://doc.integ01.dev-franceconnect.fr/inscription)[[8]](#footnote-8) 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.

## Créer un fournisseur de service

Une fois les identifiants obtenus, vous pouvez créer une nouvelle application. Choisissez un projet de type **ASP.NET Web Application** parmi les différents gabarits (templates) ASP.NET Core proposés :



Assistant de création de projet

L’assistant vous demande quel type d’authentification vous souhaitez pour cette application. Sélectionnez **Individual User Accounts**.

## 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.asp.net/en/latest/fundamentals/configuration.html).

Ajoutez les lignes suivantes pour configurer le middleware OpenID Connect (OIDC) qui sera utilisé pour l’authentification via FranceConnect.

"FranceConnect": {

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

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

"AuthorizationEndpoint": "https://fcp.integ01.dev-franceconnect.fr/api/v1/authorize?acr\_values={0}",

"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",

"PostLogoutRedirectUri": "<votre URL de callback>",

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

"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 Scheme

{

public const string FranceConnect = "oidc\_FranceConnect";

}

public class FranceConnectConfiguration

{

public string ClientId { get; set; }

public string ClientSecret { 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; }

public string EIdas { 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 Scheme vous permettra d’identifier plus facilement les différents middlewares.

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();

services.Configure<FranceConnectConfiguration>(Configuration.GetSection("FranceConnect"));

services.AddMvc();

### 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 [User Secrets](https://docs.asp.net/en/latest/security/app-secrets.html) :

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

### Implémenter le middleware OIDC

Pour accéder à la configuration FranceConnect depuis la méthode *Configure()*, ajoutez le paramètre IOptions<FranceConnectConfiguration> opts :

public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory, IOptions<FranceConnectConfiguration> opts)

Installez le package NuGet **Microsoft.AspNetCore.Authentication.OpenIdConnect[[9]](#footnote-9)** et copiez les lignes suivantes :

var franceConnectOptions = new OpenIdConnectOptions();

franceConnectOptions.AuthenticationScheme = Scheme.FranceConnect;

franceConnectOptions.DisplayName = "FranceConnect";

franceConnectOptions.ClientId = opts.Value.ClientId;

franceConnectOptions.ClientSecret = opts.Value.ClientSecret;

franceConnectOptions.Authority = opts.Value.Issuer;

franceConnectOptions.ResponseType = OpenIdConnectResponseType.Code;

franceConnectOptions.Scope.Clear();

franceConnectOptions.Scope.Add("openid");

franceConnectOptions.Scope.Add("profile");

franceConnectOptions.Scope.Add("birth");

franceConnectOptions.Scope.Add("email");

franceConnectOptions.GetClaimsFromUserInfoEndpoint = true;

franceConnectOptions.TokenValidationParameters.IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(opts.Value.ClientSecret));

franceConnectOptions.Configuration = new OpenIdConnectConfiguration

{

Issuer = opts.Value.Issuer,

AuthorizationEndpoint = string.Format(opts.Value.AuthorizationEndpoint, opts.Value.EIdas),

TokenEndpoint = opts.Value.TokenEndpoint,

UserInfoEndpoint = opts.Value.UserInfoEndpoint,

EndSessionEndpoint = opts.Value.EndSessionEndpoint

};

app.UseOpenIdConnectAuthentication(franceConnectOptions);

## Modifier le schéma de donnée

### Modifier le modèle ApplicationUser

Ajoutez au modèle ApplicationUser les propriétés suivantes pour qu’il corresponde à l’identité pivot :

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 PreferredUsername { get; set; }

}

### Appliquer la migration

Lancez cette commande depuis le répertoire du projet :

> dotnet ef migrations add AddUserProperties

Note Pour de plus amples informations, vous pouvez consulter la [documentation Entity Framework Core](http://docs.efproject.net/en/latest/platforms/aspnetcore/new-db.html)[[10]](#footnote-10).

## Adapter le comportement de l’application

### Adapter la création d’un compte local

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 pré-remplisse les informations de l’utilisateur à partir de l’identité pivot fournie par FranceConnect.

Modifiez le modèle de vue ExternalLoginConfirmationViewModel :

public class ExternalLoginConfirmationViewModel

{

[Required]

[EmailAddress]

[Display(Name = "Adresse e-mail")]

public string Email { get; set; }

[Required]

[Display(Name = "Sexe")]

public string Gender { get; set; }

[Required]

[Display(Name = "Date de naissance")]

public DateTimeOffset Birthdate { get; set; }

[Display(Name = "Nom d'usage")]

public string PreferredUsername { get; set; }

[Required]

[Display(Name = "Prénom")]

public string GivenName { get; set; }

[Required]

[Display(Name = "Nom")]

public string FamilyName { get; set; }

}

Ensuite, modifiez l’action AccountController.ExternalLoginCallback() :

public async Task<IActionResult> ExternalLoginCallback(string returnUrl = null)

{

var info = await \_signInManager.GetExternalLoginInfoAsync();

if (info == null)

{

return RedirectToAction(nameof(Login));

}

// Sign in the user with this external login provider if the user already has a login.

var user = await \_userManager.FindByLoginAsync(info.LoginProvider, info.ProviderKey);

if (user != null)

{

if (await \_userManager.IsLockedOutAsync(user))

{

return View("Lockout");

}

await \_signInManager.SignInAsync(user, false, info.LoginProvider);

\_logger.LogInformation(5, "User logged in with {Name} provider.", info.LoginProvider);

return RedirectToLocal(returnUrl);

}

else

{

// If the user does not have an account, then ask the user to create an account.

ViewData["ReturnUrl"] = returnUrl;

ViewData["LoginProvider"] = info.LoginProvider;

var model = new ExternalLoginConfirmationViewModel

{

Email = info.Principal.FindFirstValue("email"),

Gender = info.Principal.FindFirstValue("gender"),

Birthdate = Convert.ToDateTime(info.Principal.FindFirstValue("birthdate")),

PreferredUsername = info.Principal.FindFirstValue("preferred\_username"),

GivenName = info.Principal.FindFirstValue("given\_name"),

FamilyName = info.Principal.FindFirstValue("family\_name")

};

return View("ExternalLoginConfirmation", model);

}

}

Puis, modifiez la vue ExternalLoginConfirmation pour qu’elle corresponde au modèle de vue :

<div class="form-group">

<label asp-for="Birthdate" class="col-md-2 control-label"></label>

<div class="col-md-10">

<input asp-for="Birthdate" asp-format="{0:dd/MM/yyyy}" class="form-control" />

<span asp-validation-for="Birthdate" class="text-danger"></span>

</div>

</div>

<div class="form-group">

<label asp-for="PreferredUsername" class="col-md-2 control-label"></label>

<div class="col-md-10">

<input asp-for="PreferredUsername" class="form-control" />

<span asp-validation-for="PreferredUsername" class="text-danger"></span>

</div>

</div>

<div class="form-group">

<label asp-for="Gender" class="col-md-2 control-label"></label>

<div class="col-md-10">

<input asp-for="Gender" class="form-control" />

<span asp-validation-for="Gender" class="text-danger"></span>

</div>

</div>

<div class="form-group">

<label asp-for="GivenName" class="col-md-2 control-label"></label>

<div class="col-md-10">

<input asp-for="GivenName" class="form-control" />

<span asp-validation-for="GivenName" class="text-danger"></span>

</div>

</div>

<div class="form-group">

<label asp-for="FamilyName" class="col-md-2 control-label"></label>

<div class="col-md-10">

<input asp-for="FamilyName" class="form-control" />

<span asp-validation-for="FamilyName" class="text-danger"></span>

</div>

</div>

Enfin, récupérez les informations dans l’action AccountController.ExternalLoginConfirmation() :

public async Task<IActionResult> ExternalLoginConfirmation(

ExternalLoginConfirmationViewModel model,

string returnUrl = null)

{

if (ModelState.IsValid)

{

// Get the information about the user from the external login provider

var info = await \_signInManager.GetExternalLoginInfoAsync();

if (info == null)

{

return View("ExternalLoginFailure");

}

var user = new ApplicationUser

{

UserName = model.Email,

Email = model.Email,

Gender = model.Gender,

Birthdate = model.Birthdate,

PreferredUsername = model.PreferredUsername,

GivenName = model.GivenName,

FamilyName = model.FamilyName

};

var result = await \_userManager.CreateAsync(user);

if (result.Succeeded)

{

result = await \_userManager.AddLoginAsync(user, info);

if (result.Succeeded)

{

await \_signInManager.SignInAsync(user, false, info.LoginProvider);

\_logger.LogInformation(

6,

"User created an account using {Name} provider.",

info.LoginProvider);

return RedirectToLocal(returnUrl);

}

}

AddErrors(result);

}

ViewData["ReturnUrl"] = returnUrl;

return View(model);

}

### Adapter la création du mot de passe

Une fois que l’utilisateur a créé son mot de passe, il est reconnecté automatiquement avec son compte local. Ceci ne représente pas le comportement décrit par la documentation FranceConnect. Afin de corriger cela, supprimez la ligne suivante dans l’action ManageController.SetPassword() :

await \_signInManager.SignInAsync(user, isPersistent: false);

De cette manière, l’utilisateur reste connecté avec FranceConnect.

## Rapprocher les comptes

### Implémenter l’association de compte

Le template choisi implémente déjà l’association de compte. Néanmoins, une fois le compte associé, les claims de l’authentification externe ne sont pas sauvegardés. Pour remédier à cela, modifiez l’action ManageController.LinkLoginCallback() :

public async Task<ActionResult> LinkLoginCallback()

{

var user = await GetCurrentUserAsync();

if (user == null)

{

return View("Error");

}

var info = await \_signInManager.GetExternalLoginInfoAsync(await \_userManager.GetUserIdAsync(user));

if (info == null)

{

return RedirectToAction(

nameof(ManageLogins),

new { Message = ManageMessageId.Error });

}

var result = await \_userManager.AddLoginAsync(user, info);

ManageMessageId message;

if (result.Succeeded)

{

await \_signInManager.SignInAsync(user, false, info.LoginProvider);

message = ManageMessageId.AddLoginSuccess;

}

else

{

message = ManageMessageId.Error;

}

return RedirectToAction(nameof(ManageLogins), new { Message = message });

}

### Implémenter la dissociation de compte

De la même manière, quand l’utilisateur décide de dissocier son compte FranceConnect, il doit être rediriger vers la page de déconnection de FranceConnect (s’il s’est authentifié via FranceConnect). Pour cela, modifiez l’action ManageController.RemoveLogin() :

public async Task RemoveLogin(RemoveLoginViewModel account)

{

ManageMessageId? message = ManageMessageId.Error;

var user = await GetCurrentUserAsync();

if (user != null)

{

bool useExternalLogin = (await \_signInManager.GetExternalLoginInfoAsync()) != null;

var result = await \_userManager.RemoveLoginAsync(user, account.LoginProvider, account.ProviderKey);

if (result.Succeeded)

{

await \_signInManager.SignInAsync(user, isPersistent: false);

if (useExternalLogin)

{

string postLogoutRedirectUri = CreateUri(nameof(ManageLogins));

await HttpContext.Authentication.SignOutAsync(

Scheme.FranceConnect,

new AuthenticationProperties

{

RedirectUri = postLogoutRedirectUri

});

}

else

{

message = ManageMessageId.RemoveLoginSuccess;

Response.Redirect(Url.Action(nameof(ManageLogins), new { Message = message }));

}

}

}

}

private string CreateUri(string action)

{

string protocol = Request.IsHttps ? "https" : "http";

return string.Format("{0}://{1}{2}", protocol, Request.Host, Url.Action(action));

}

## Déconnecter l’utilisateur

Pour assurer une bonne déconnexion, il faut supprimer les cookies et demander au provider OpenID Connect de déconnecter l’utilisateur.

Pour cela, modifiez l’action AccountController.LogOff() :

public async Task LogOff()

{

await \_signInManager.SignOutAsync();

\_logger.LogInformation(4, "User logged out.");

if (await \_signInManager.GetExternalLoginInfoAsync() != null)

{

await HttpContext.Authentication.SignOutAsync(Scheme.FranceConnect);

}

else

{

Response.Redirect(Url.Action(nameof(HomeController.Index), "Home"));

}

}

Si l’utilisateur s’est authentifier avec FranceConnect, on invalide le cookie et on le redirige vers la page de déconnexion de FranceConnect, sinon on le redirige vers la page d’accueil.

## Implémenter le bouton FranceConnect

### Définir le style du bouton

Importez les images des boutons dans le dossier images du projet et créez une nouvelle class CSS dans le fichier site.css pour appliquer le style exigé par la [documentation FranceConnect](https://doc.integ01.dev-franceconnect.fr/fournisseur-service#buttons)[[11]](#footnote-11) :

.btn-fc {

background: url(../../images/FCboutons-10.png) no-repeat center;

background-image: url(../../images/FCboutons-10.svg);

background-size: contain;

height: 48px;

width: 178px;

border: none;

}

.btn-fc:hover {

background: url(../../images/FCboutons-11.png) no-repeat center;

background-image: url(../../images/FCboutons-11.svg);

}

### Ajouter le bouton sur la page Login

Dans la vue Login, modifiez la section réservée aux connexions externes :

<section>

<h4>S'authentifier avec FranceConnect.</h4>

<hr />

@{

var franceConnectProvider = SignInManager.

GetExternalAuthenticationSchemes().

FirstOrDefault(auth => auth.AuthenticationScheme == Scheme.FranceConnect);

<form asp-controller="Account"

asp-action="ExternalLogin"

asp-route-returnurl="@ViewData["ReturnUrl"]"

method="post" class="form-horizontal" role="form">

<button type="submit"

class="btn-fc"

name="provider"

value="@franceConnectProvider.AuthenticationScheme"

title="Connectez vous avec votre compte

@franceConnectProvider.DisplayName">

</button>

</form>

}

</section>

### Ajouter le bouton sur la page ManageLogins

Pour la page de gestion des connexions externes, modifiez le modèle de vue ManageLoginsViewModel :

public class ManageLoginsViewModel

{

public bool IsLinkedToFranceConnect { get; set; }

public bool CanRemoveExternalLogin { get; set; }

public UserLoginInfo FranceConnectUserAccount { get; set; }

public AuthenticationDescription FranceConnectProvider { get; set; }

}

* IsLinkedToFranceConnect définit si le compte local est lié au compte FranceConnect.
* CanRemoveExternalLogin définit si l’utilisateur peut dissocier son compte FranceConnect.
* FranceConnectUserAccount représente le compte FranceConnect lié.
* FranceConnectProvider représente le provider FranceConnect disponible pour la liaison.

Modifiez également l’action ManageController.ManageLogins() :

public async Task<IActionResult> ManageLogins(ManageMessageId? message = null)

{

ViewData["StatusMessage"] =

message == ManageMessageId.RemoveLoginSuccess ? "L'authentification externe a été supprimé."

: message == ManageMessageId.AddLoginSuccess ? "L'authentification externe a été ajouté."

: message == ManageMessageId.Error ? "Une erreur est survenue."

: "";

var user = await GetCurrentUserAsync();

if (user == null)

{

return View("Error");

}

var userLogins = await \_userManager.GetLoginsAsync(user);

var availableProviders =

\_signInManager.

GetExternalAuthenticationSchemes().

Where(auth =>

userLogins.All(ul => auth.AuthenticationScheme != ul.LoginProvider)).

ToList();

return View(new ManageLoginsViewModel

{

IsLinkedToFranceConnect = userLogins.

Any(auth => auth.LoginProvider == Scheme.FranceConnect),

CanRemoveExternalLogin = user.PasswordHash != null || userLogins.Count > 1,

FranceConnectUserAccount = userLogins.

FirstOrDefault(auth => auth.LoginProvider == Scheme.FranceConnect),

FranceConnectProvider = availableProviders.

FirstOrDefault(auth => auth.AuthenticationScheme == Scheme.FranceConnect)

});

}

Enfin, modifiez la vue ManageLoginsView :

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

<p class="text-success">@ViewData["StatusMessage"]</p>

@if (Model.IsLinkedToFranceConnect)

{

<h4>Vous êtes lié avec votre compte

@Model.FranceConnectUserAccount.LoginProvider</h4>

if (Model.CanRemoveExternalLogin)

{

RemoveLoginViewModel account = new RemoveLoginViewModel

{

LoginProvider = Model.FranceConnectUserAccount.LoginProvider,

ProviderKey = Model.FranceConnectUserAccount.ProviderKey

};

<hr />

<form asp-controller="Manage"

asp-action="RemoveLogin"

method="post" class="form-horizontal" role="form">

<div>

<input asp-for="@account.LoginProvider" type="hidden" />

<input asp-for="@account.ProviderKey" type="hidden" />

<input type="submit"

class="btn btn-default"

value="Dissocier"

title="Supprimer l'authentification par

@account.LoginProvider

pour votre compte" />

</div>

</form>

}

}

else

{

<h4>Liez votre compte local avec votre compte FranceConnect.</h4>

<hr />

<form asp-controller="Manage"

asp-action="LinkLogin"

method="post" class="form-horizontal" role="form">

<div>

<button type="submit"

class="btn-fc"

name="provider"

value="@Model.FranceConnectProvider.AuthenticationScheme"

title="Connectez vous avec votre compte

@Model.FranceConnectProvider.DisplayName"></button>

</div>

</form>

}

## Mettre en œuvre le kit d’intégration FranceConnect de la DINSIC

Le kit d’intégration FranceConnect d’intégration proposé par la DINSIC nécessite que la déconnexion se fasse par méthode GET. Pour cela, modifiez la méthode AccountController.LogOff() :

[HttpGet]

Ensuite, ajoutez le script JS de FranceConnect dans le layout principal :

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

Enfin, modifiez la vue partielle \_LoginPartial pour qu’elle affiche le kit d’intégration FranceConnect si l’utilisateur s’est authentifié avec son compte FranceConnect :

@if (User.IsSignedIn())

{

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

@if (User.HasClaim(claim => claim.Type == ClaimTypes.AuthenticationMethod && claim.Value == Scheme.FranceConnect))

{

<li>

<a asp-controller="Manage" asp-action="Index">Mon compte</a>

</li>

<li id="fconnect-profile"

data-fc-logout-url="@Url.Action("LogOff", "Account")">

<a href="#">@User.GetUserName()</a>

</li>

}

else

{

<li>

<a asp-controller="Manage"

asp-action="Index"

title="Mon compte">Bonjour @User.GetUserName() !</a>

</li>

<li>

<a asp-controller="Account" asp-action="LogOff">Se déconnecter</a>

</li>

}

</ul>

}

else

{

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

<li><a asp-controller="Account" asp-action="Register">Créer un compte</a></li>

<li><a asp-controller="Account" asp-action="Login">Se connecter</a></li>

</ul>

}

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

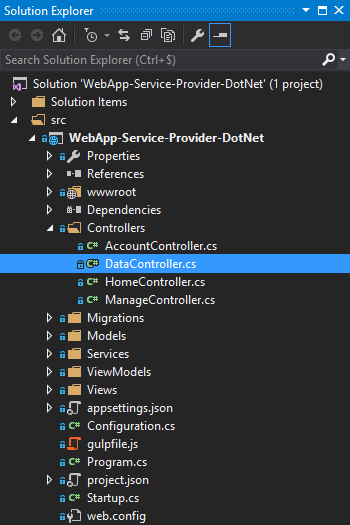
D’après la documentation FranceConnect, un fournisseur de service peut demander des ressources à un fournisseur de données avec le consentement de l’utilisateur.

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

Ajoutez donc le package suivant au projet :

|  |  |
| --- | --- |
| **Package NuGet** | **Version** |
| IdentityModel | 2.0.0-rc3 |

Afin de proposer à l’utilisateur d’accéder à ses ressources depuis un autre fournisseur, ajouter un nouveau contrôleur dédié à cela :



Ajout du contrôleur "DataController"

namespace WebApp\_Service\_Provider\_DotNet.Controllers

{

[Authorize]

public class DataController : Controller

{

private readonly UserManager<ApplicationUser> \_userManager;

private readonly SignInManager<ApplicationUser> \_signInManager;

private readonly ILogger \_logger;

private readonly FranceConnectConfiguration \_config;

public DataController(

UserManager<ApplicationUser> userManager,

SignInManager<ApplicationUser> signInManager,

ILoggerFactory loggerFactory,

IOptions<FranceConnectConfiguration> config)

{

\_userManager = userManager;

\_signInManager = signInManager;

\_logger = loggerFactory.CreateLogger<AccountController>();

\_config = config.Value;

}

//

// GET: /Data/

[HttpGet]

public IActionResult Index()

{

return View(\_config.DataProviders);

}

//

// GET: /Data/GetResource

[HttpGet]

public IActionResult GetResource(string provider)

{

var state = Guid.NewGuid().ToString("N");

var scope = new List<string> { "openid", "email" };

scope.AddRange(\_config.DataProviders.FirstOrDefault(dp => dp.Name == provider).Scopes);

var json = JsonConvert.SerializeObject(new ConsentCookie

{

Provider = provider,

State = state

});

Response.Cookies.Delete("consent");

Response.Cookies.Append("consent",

Base64Encode(json), new CookieOptions { Expires = DateTimeOffset.Now.AddMinutes(15) });

var authorizeRequest = new AuthorizeRequest(\_config.AuthorizationEndpoint);

return Redirect(authorizeRequest.CreateAuthorizeUrl(

clientId: \_config.ClientId,

responseType: "code",

scope: string.Join(" ", scope),

redirectUri: GetConsentRedirectUri(),

state: state,

nonce: Guid.NewGuid().ToString("N")));

}

//

// GET: /Data/ConsentCallback

[HttpGet]

public async Task<IActionResult> GetResourceCallback(string code, string state)

{

ConsentCookie consentCookie = null;

string json;

try

{

json = Base64Decode(Request.Cookies["consent"]);

consentCookie = JsonConvert.DeserializeObject<ConsentCookie>(json);

Response.Cookies.Delete("consent");

}

catch (Exception)

{

throw new Exception("Unable to retrieve cookie");

}

if (string.IsNullOrEmpty(code))

{

throw new ArgumentNullException("Authorization code cannot be null");

}

if (string.IsNullOrEmpty(state))

{

throw new ArgumentNullException("State cannot be null");

}

if (state != consentCookie.State)

{

throw new ArgumentException("Invalid state");

}

var tokenClient = new TokenClient(\_config.TokenEndpoint, \_config.ClientId, \_config.ClientSecret, AuthenticationStyle.PostValues);

var tokenResponse = await tokenClient.RequestAuthorizationCodeAsync(code, GetConsentRedirectUri());

if (tokenResponse.IsError || string.IsNullOrEmpty(tokenResponse.AccessToken))

{

throw new Exception("Unable to retrieve access token");

}

consentCookie.State = null;

consentCookie.Token = tokenResponse.AccessToken;

json = JsonConvert.SerializeObject(consentCookie);

Response.Cookies.Append("consent", Base64Encode(json), new CookieOptions { Expires = DateTimeOffset.Now.AddMinutes(15) });

return RedirectToAction(nameof(Resource));

}

//

// GET: /Data/GetResourceCallback

[HttpGet]

public async Task<IActionResult> Resource()

{

ConsentCookie consentCookie = null;

try

{

var json = Base64Decode(Request.Cookies["consent"]);

consentCookie = JsonConvert.DeserializeObject<ConsentCookie>(json);

}

catch (Exception)

{

ViewData["Message"] = "Impossible d'obtenir l'autorisation d'accès aux ressources, veuillez réessayer.";

return View();

}

var client = new HttpClient();

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", consentCookie.Token);

var response = await client.GetAsync(GetResourceUrl(consentCookie.Provider));

if (response.IsSuccessStatusCode)

{

var resource = await response.Content.ReadAsStringAsync();

return View(ConvertResource(resource, consentCookie.Provider));

}

else if (response.StatusCode == HttpStatusCode.NotFound)

{

ViewData["Message"] = "La ressource demandée n'a pas été trouvée.";

return View();

}

else if (response.StatusCode == HttpStatusCode.Unauthorized)

{

ViewData["Message"] = "Vous n'êtes pas autorisé à accéder cette ressource.";

return View();

}

else

{

ViewData["Message"] = "Impossible de récupérer les données auprès du fournisseur choisi.";

return View();

}

}

private class ConsentCookie

{

public string Provider { get; set; }

public string State { get; set; }

public string Token { get; set; }

}

#region Helpers

private BaseResourceViewModel ConvertResource(string json, string scheme)

{

switch (scheme)

{

case "DGFIP":

return JsonConvert.DeserializeObject<DgfipResourceViewModel>(json);

case "Custom":

return JsonConvert.DeserializeObject<CustomResourceViewModel>(json);

default:

throw new NotImplementedException();

}

}

private string GetResourceUrl(string providerName)

{

DataProvider provider = \_config.DataProviders.FirstOrDefault(dp => dp.Name == providerName);

if (provider != null)

{

return provider.Endpoint;

}

else

{

throw new InvalidOperationException(string.Format("Unable to find \"{0}\" data provider", providerName));

}

}

private string GetConsentRedirectUri()

{

return Request.Scheme + "://" + Request.Host + Url.Action(nameof(GetResourceCallback));

}

private string Base64Encode(string text)

{

var bytes = Encoding.UTF8.GetBytes(text);

return Convert.ToBase64String(bytes);

}

private string Base64Decode(string base64EncodedText)

{

var bytes = Convert.FromBase64String(base64EncodedText);

return Encoding.UTF8.GetString(bytes);

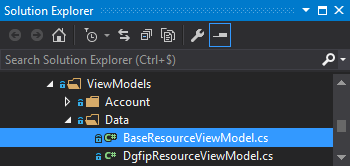
}

#endregion

}

}

Ajoutez également de nouveaux modèles de vue, un de base et tous les autres qui en hérite afin d’avoir un point commun entre les ressources obtenue de différents fournisseurs :



Modèles de vue décrivant les ressources externes

public class BaseResourceViewModel

{

}

public class DgfipResourceViewModel : BaseResourceViewModel

{

public decimal Rfr { get; set; }

public char SitFam { get; set; }

public int NbPart { get; set; }

public Pac Pac { get; set; }

}

public class Pac

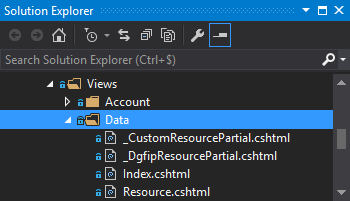
{

public int NbPac { get; set; }

}

Les noms des propriétés doivent avoir le nom correspondant au JSON retourné.

Puis, ajoutez les vues, une pour permettre à l’utilisateur de requêter le fournisseur de son choix, une autre pour afficher les données et autant de vue partielle qu’il y a de modèles de données différents :



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)

{

@Html.Partial("\_DgfipResourcePartial", 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>@string.Format("{0:C}", 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>

</tbody>

</table>

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

@if (SignInManager.IsSignedIn(User))

{

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

<li>

<a asp-controller="Data" asp-action="Index">Données</a>

</li>

@if (await SignInManager.GetExternalLoginInfoAsync() != null)

{

<li>

<a asp-controller="Manage" asp-action="Index">Mon compte</a>

</li>

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

<a href="#">@UserManager.GetUserName(User)</a>

</li>

}

else

{

<li>

<a asp-controller="Manage" asp-action="Index" title="Mon compte">Bonjour @UserManager.GetUserName(User) !</a>

</li>

<li>

<a asp-controller="Account" asp-action="LogOff">Se déconnecter</a>

</li>

}

</ul>

}

else

{

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

<li><a asp-controller="Account" asp-action="Register">Créer un compte</a></li>

<li><a asp-controller="Account" asp-action="Login">Se connecter</a></li>

</ul>

}

# 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)[[12]](#footnote-12).

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 disponibilité du projet comme repo 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/)[[13]](#footnote-13), [GitHub](https://github.com/FranceConnectSamples)[[14]](#footnote-14) ou [Visual Studio](https://msdn.microsoft.com/vstudio/aa718325.aspx)[[15]](#footnote-15). 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/)[[16]](#footnote-16) 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 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 **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 :

C:\Users\philber\Documents\GitHub>Copygit clone https://github.com/FranceConnectSamples/franceconnect-service-provider-dotnet-webapp-aspnetcore

Le clonage démarre.

C:\Users\philber\Documents\GitHub>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 sous *%UserProfile%\Documents\GitHub*.

**Le code source associé est alors situé sous le dossier** *%UserProfile%\Documents\GitHub****\franceconnect-service-provider-dotnet-webapp-aspnetcore-master\Source.***

L’environnement Visual Studio Community 2015 propose également une intégration directe avec GitHub que nous ne développons pas ici.

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

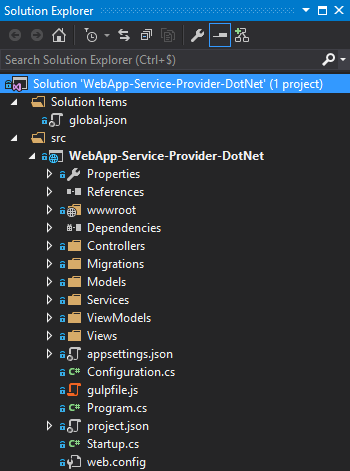
Remarque importante Nous faisons référence au dossier où se situe le code source du canevas de fournisseur de service (FS) comme étant le dossier Kit-Demarrage dans la suite de ce document.

## 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>",

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

"AuthorizationEndpoint": "https://fcp.integ01.dev-franceconnect.fr/api/v1/authorize?acr\_values={0}",

"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 de sécurité souhaité : eidas1, eidas2 ou eidas3"

}

}

|  |  |
| --- | --- |
| **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 |

## Déployer le projet dans Microsoft Azure

Pour déployer le fournisseur de service (FS) dans l’environnement Microsoft Azure, vous allez devoir créer une nouvelle application web.

Remarque importante Le fournisseur de service (FS) peut bien évidemment être déployé dans d’autres environnements mais nous n’illustrons pas ces possibilités dans la suite de ce document.

Vous devez disposer pour cela d’un abonnement Azure. Avant de continuer, si vous ne possédez pas d’un compte Azure, nous vous invitons à vous rendre à l’adresse <https://azure.microsoft.com/fr-fr/pricing/free-trial/> afin d’en créer un, gratuitement.

### Créer l’application web dans Microsoft Azure

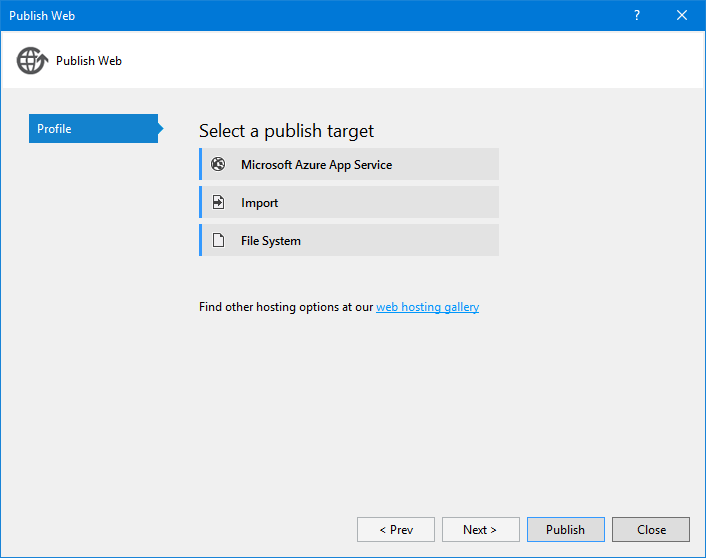
Pour créer l’application web relative à votre fournisseur de service dans Microsoft Azure, procédez comme suit :

1. **Ouvrez une session browser et** rendez-vous sur le portail Azure à l’adresse https://portal.azure.com/
2. Cliquez sur **Nouveau**.
3. Dans la catégorie **Web + mobile**, choisissez **Application web + SQL**.
4. Renseignez les champs demandés.
5. Une fois votre application crée, rendez-vous dans les paramètres de celle-ci et ajoutez le couple clé-valeur suivant : « FranceConnect:ClientSecret » : <*Votre client\_secret FranceConnect*>

### Publier le fournisseur de service dans Microsoft Azure

Pour publier votre fournisseur de service dans Microsoft Azure depuis Visual Studio Community 2015, procédez comme suit :

1. Faites un clic droit sur votre projet et sélectionnez **Publish…**
2. Sélectionnez **Microsoft Azure App Service**.



1. Sélectionnez l’application web que vous venez de créer.
2. Cliquez sur **Publish**.

# Tester le fournisseur de service

## Configurer le compte FranceConnect

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

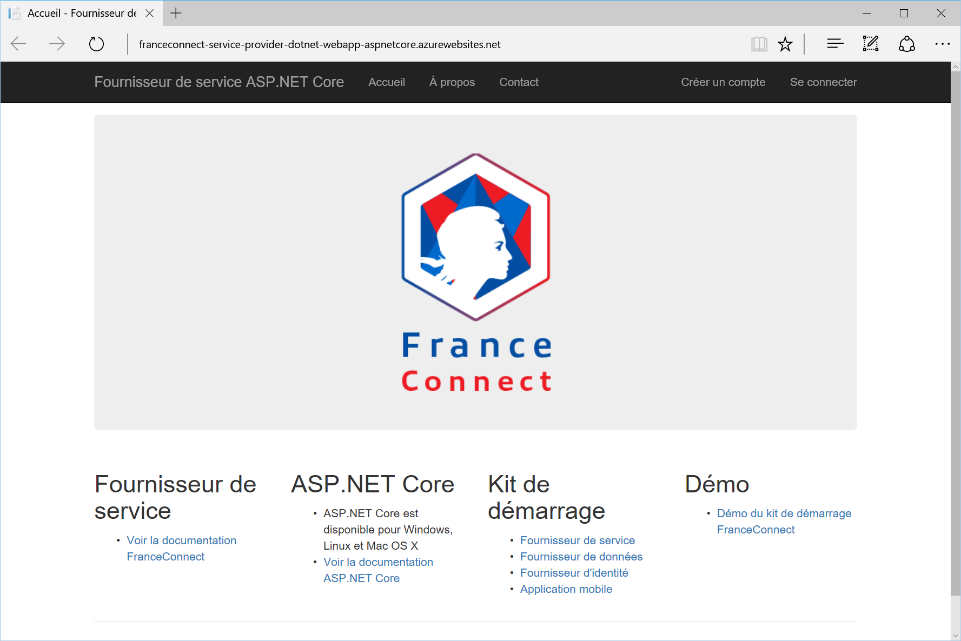
1. **Ouvrez une session browser et** rendez-vous sur [votre compte FranceConnect](https://doc.integ01.dev-franceconnect.fr/client)[[17]](#footnote-17)
2. Ajoutez les URL de callback suivantes :
   1. <*FS URL*>/signin-oidc
   2. <*FS URL*>/Data/ConsentCallback
3. Ajoutez les URL de redirections de déconnexion :
   1. <*FS URL*>/
   2. <*FS URL*>/Manage/ManageLogins

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

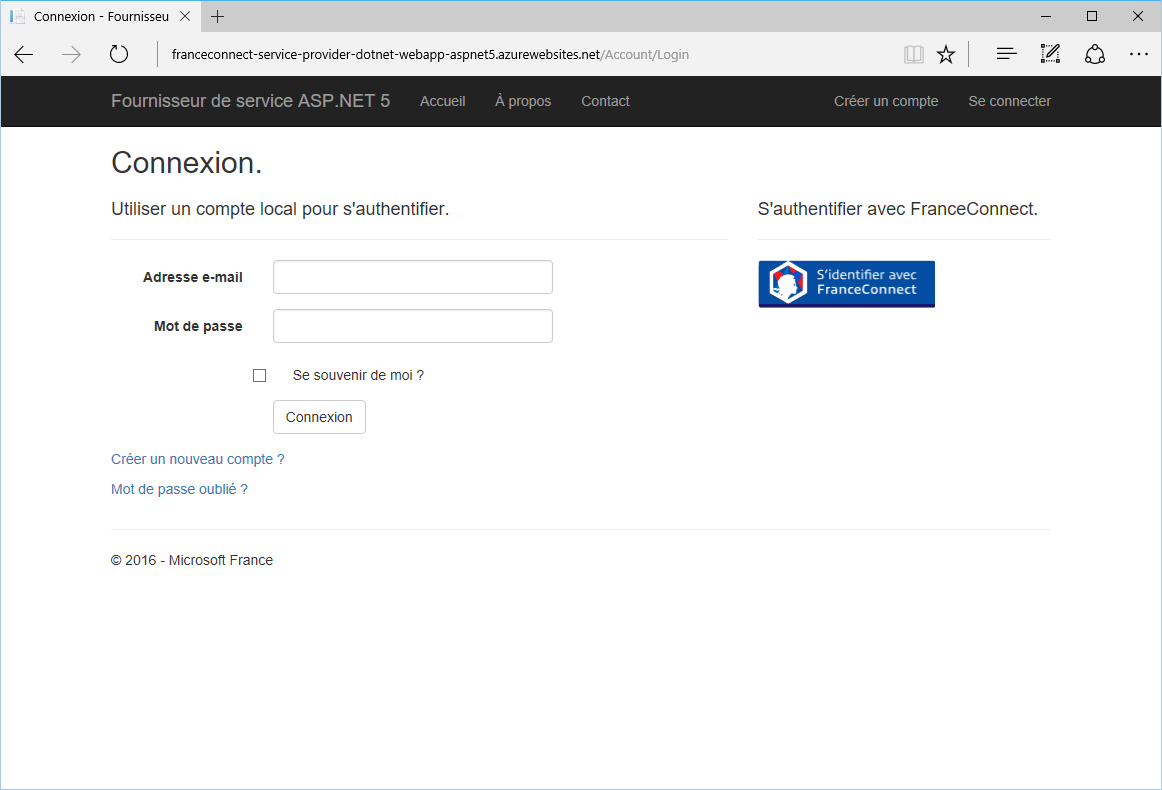
## Se connecter avec le système FranceConnect

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

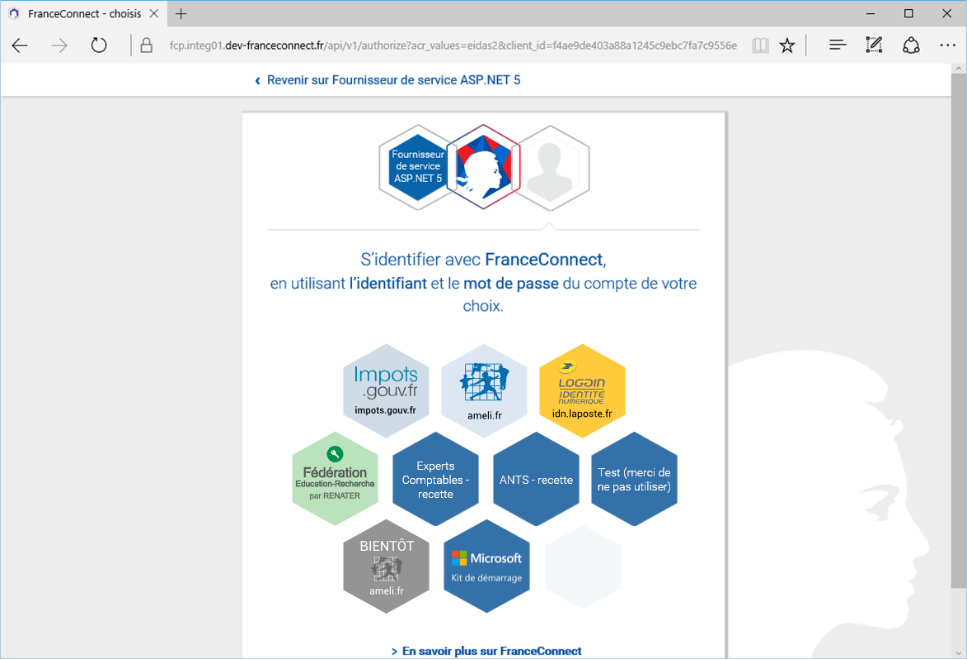
1. Ouvrez une session browser et naviguez vers le canevas de fournisseur de service FranceConnect, le vôtre, ou l’instance de démonstration à l’adresse <http://aka.ms/FranceConnect-FS>.



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.



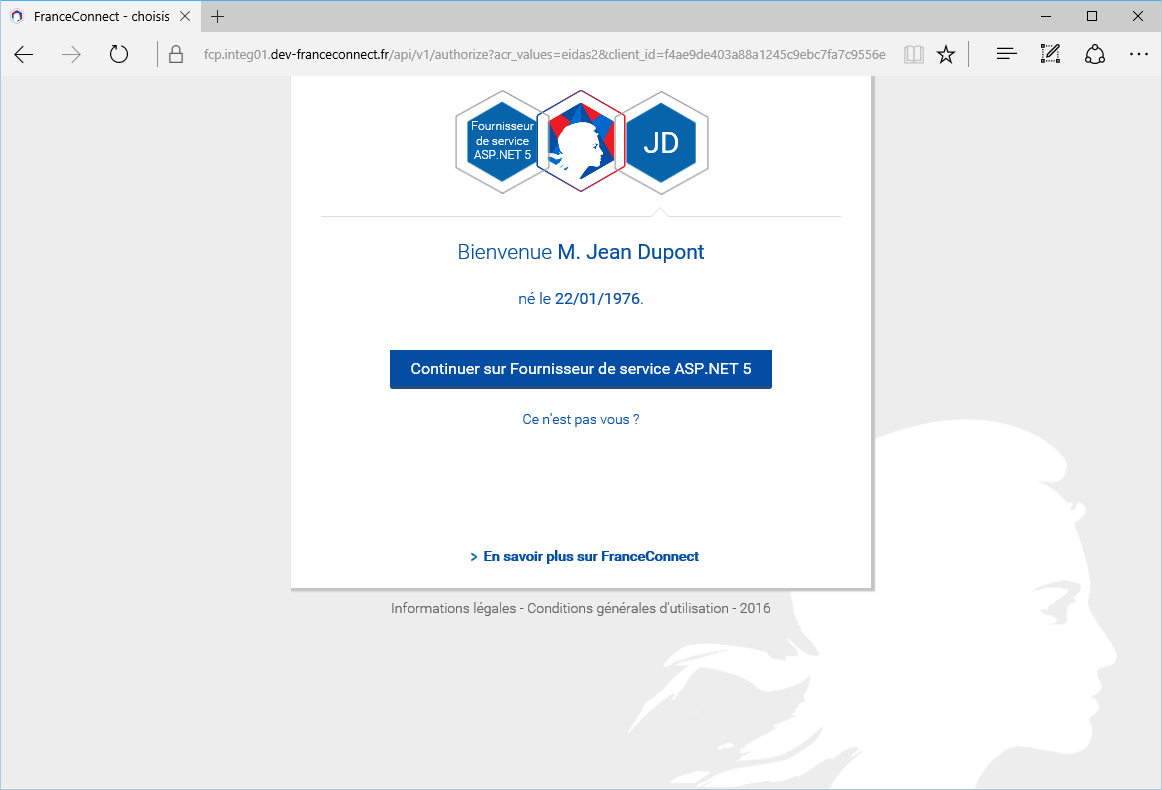
1. Sélectionnez le fournisseur d’identité compatible FranceConnect avec lequel vous souhaitez vous identifier et authentifier, par exemple le fournisseur d’identité factice Impôts. Vous êtes alors redirigé(e)s vers ce fournisseur d’identité.



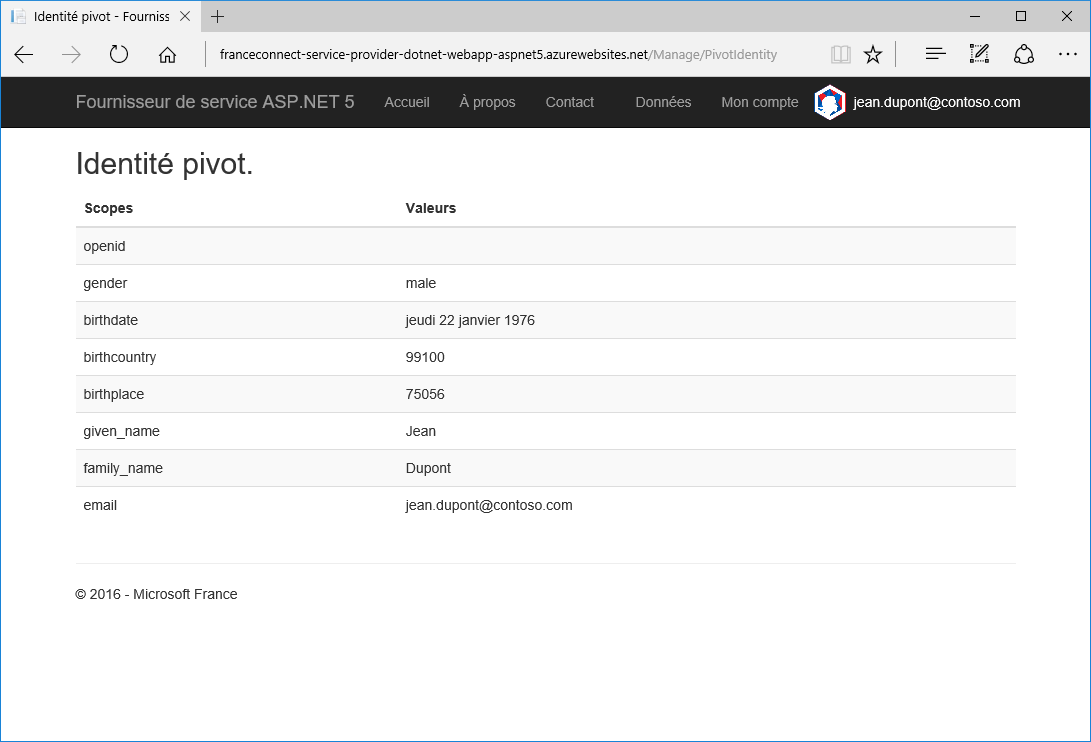
1. Au niveau du fournisseur d’identité factice Impôts, identifiez-vous en tant que l’utilisateur Jean Dupont. Précisez pour cela les informations suivantes dans le formulaire qui vous est proposé :

* **Numéro fiscal** : 248163264128
* **Mot de passe** : 123

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



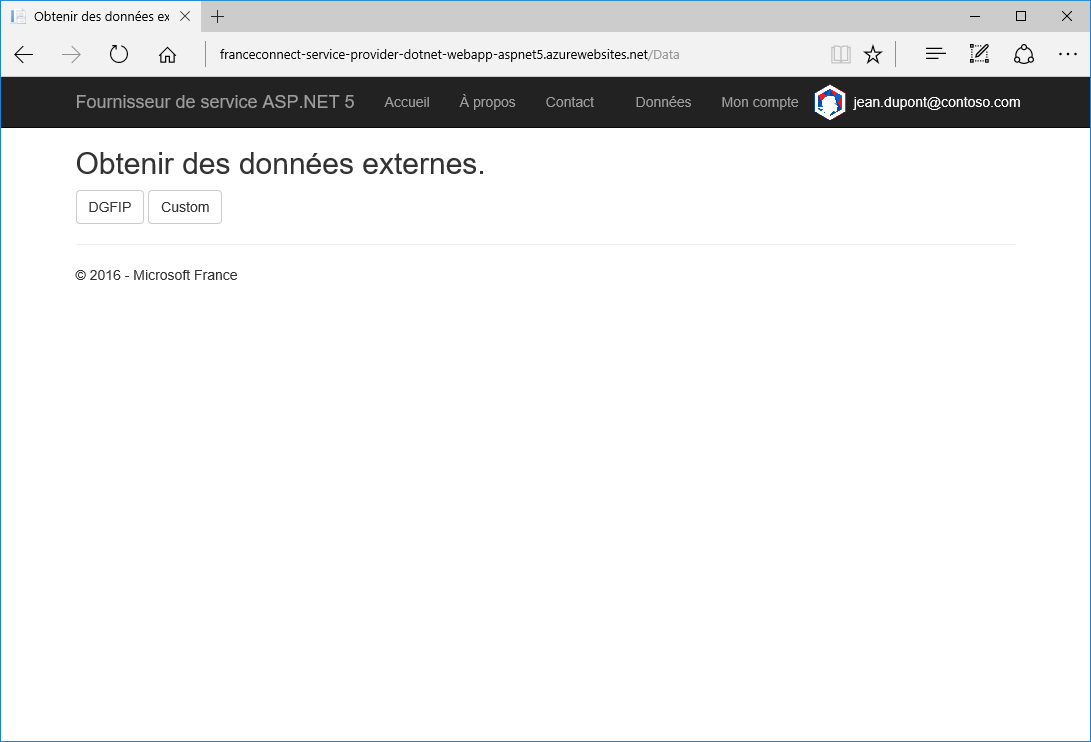
1. Si tel est le cas, cliquez sur **Continuer sur Fournisseur de service ASP.NET Core**. Vous êtes redirigé(e)s vers le canevas de fournisseur de service. Les informations sur votre identité pivot telles que transmises par FranceConnect s’affichent alors.



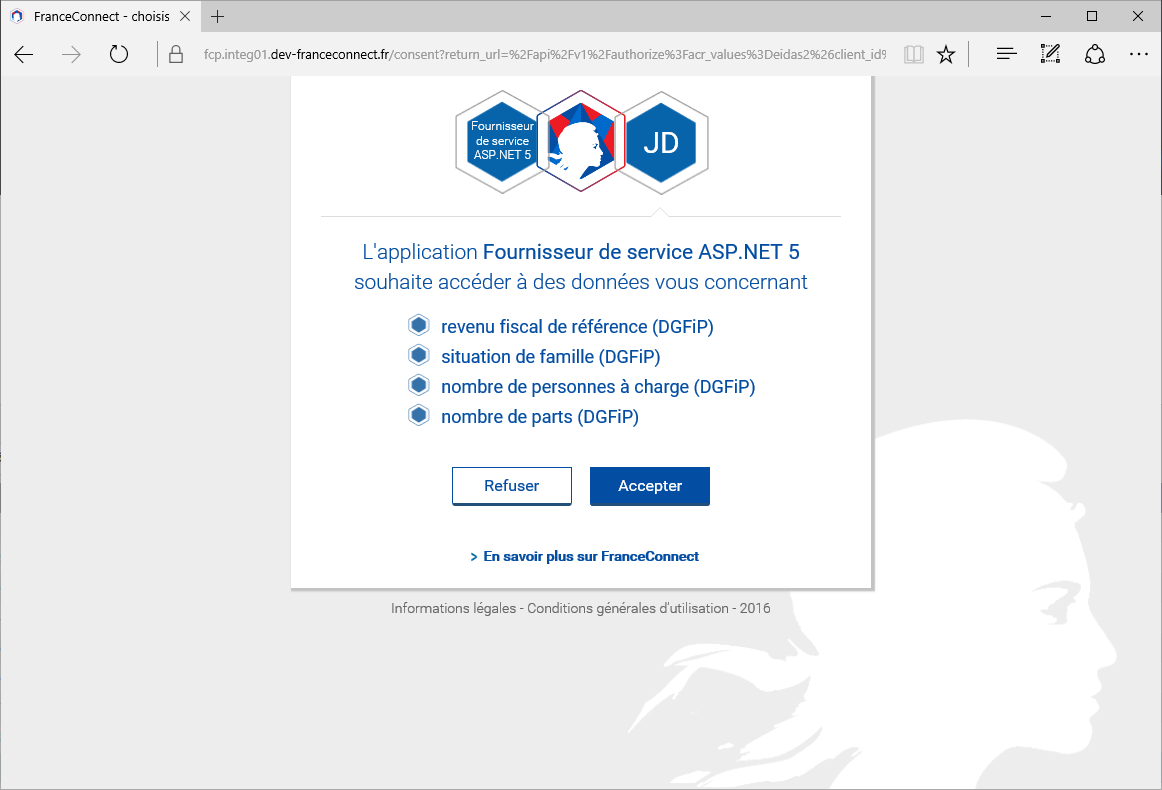
## Invoquer un fournisseur de données via le système FranceConnect

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

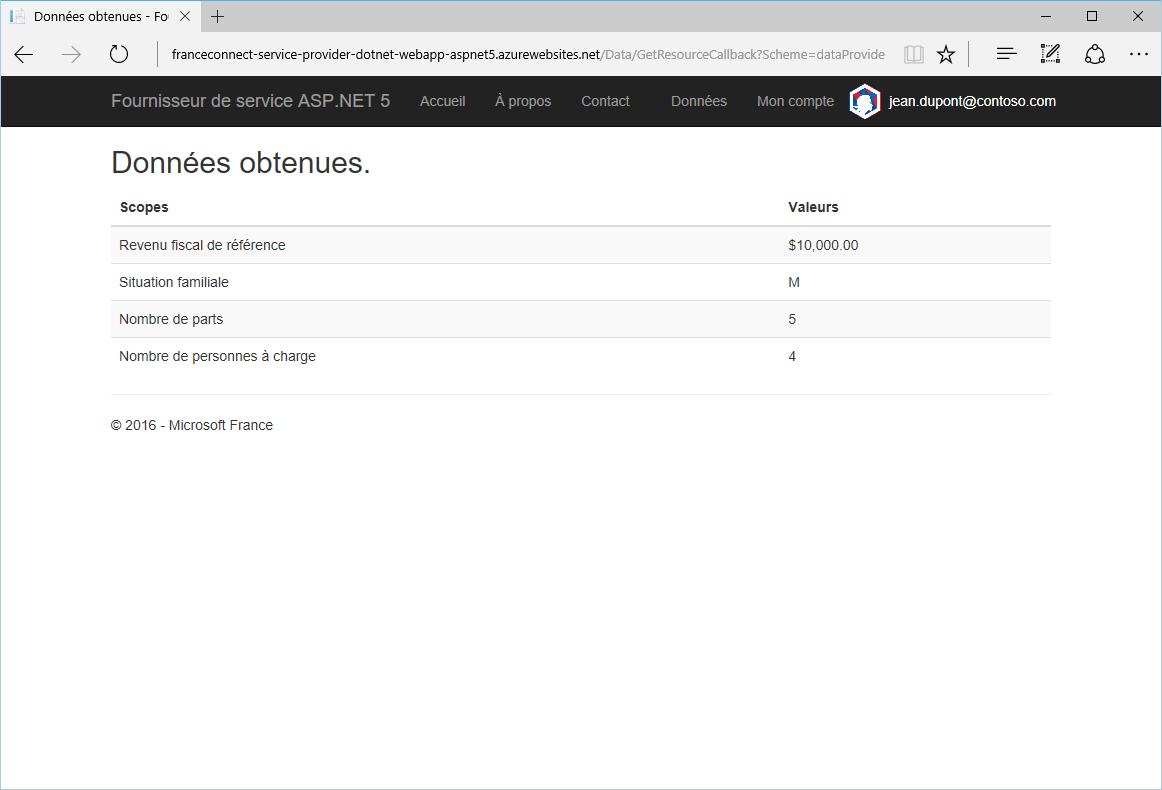
1. Depuis la fenêtre de browser précédente, cliquez sur **Données**. Les fournisseurs de données (FD) configurés au niveau du canevas de fournisseur de service s’affichent.



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.



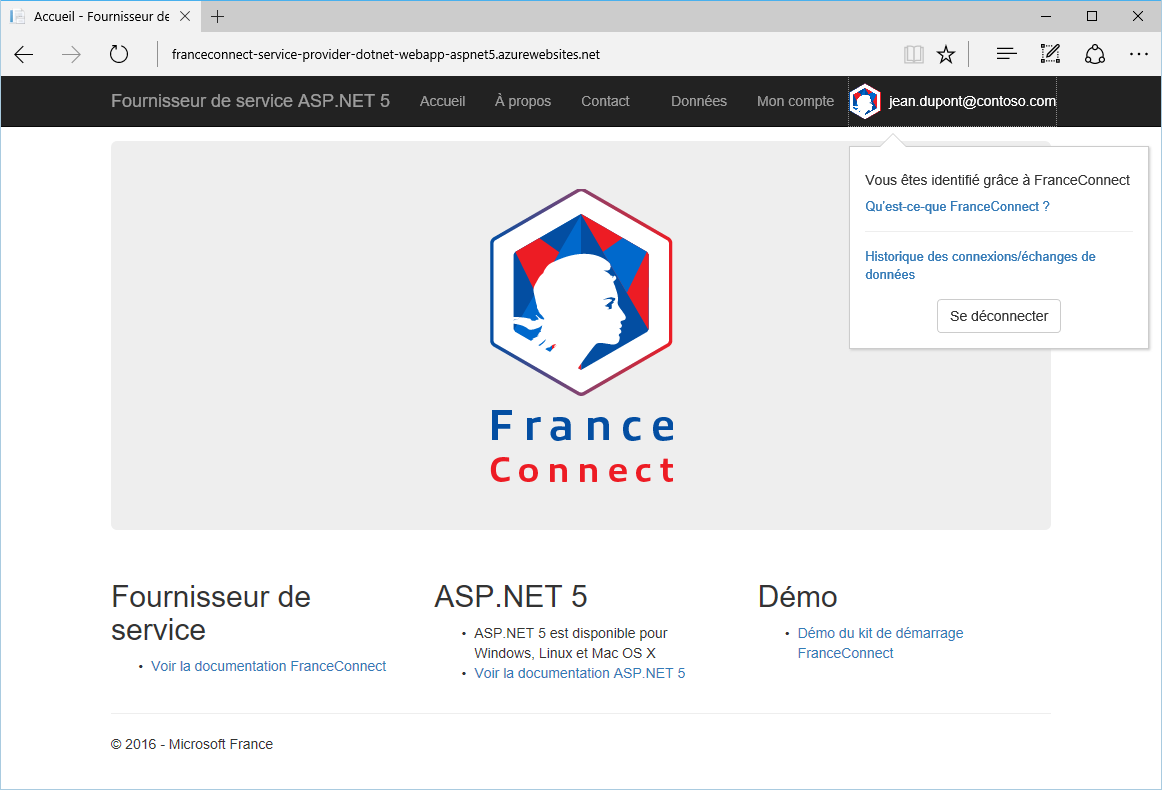
1. Si tel est le cas, cliquer sur **Accepter**. Une fois la requête traitée et de retour dans le canevas de fournisseur de service, les données ainsi obtenues sont affichées dans la fenêtre.



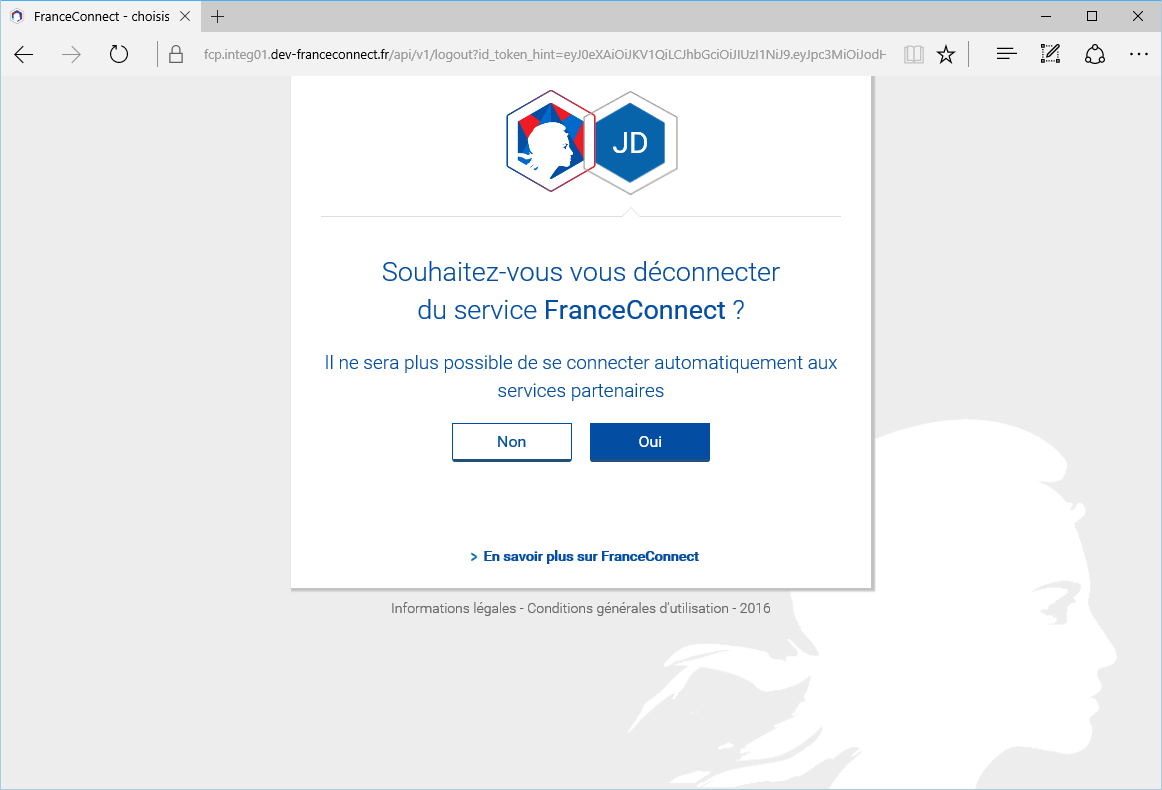
## Se déconnecter du système FranceConnect

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

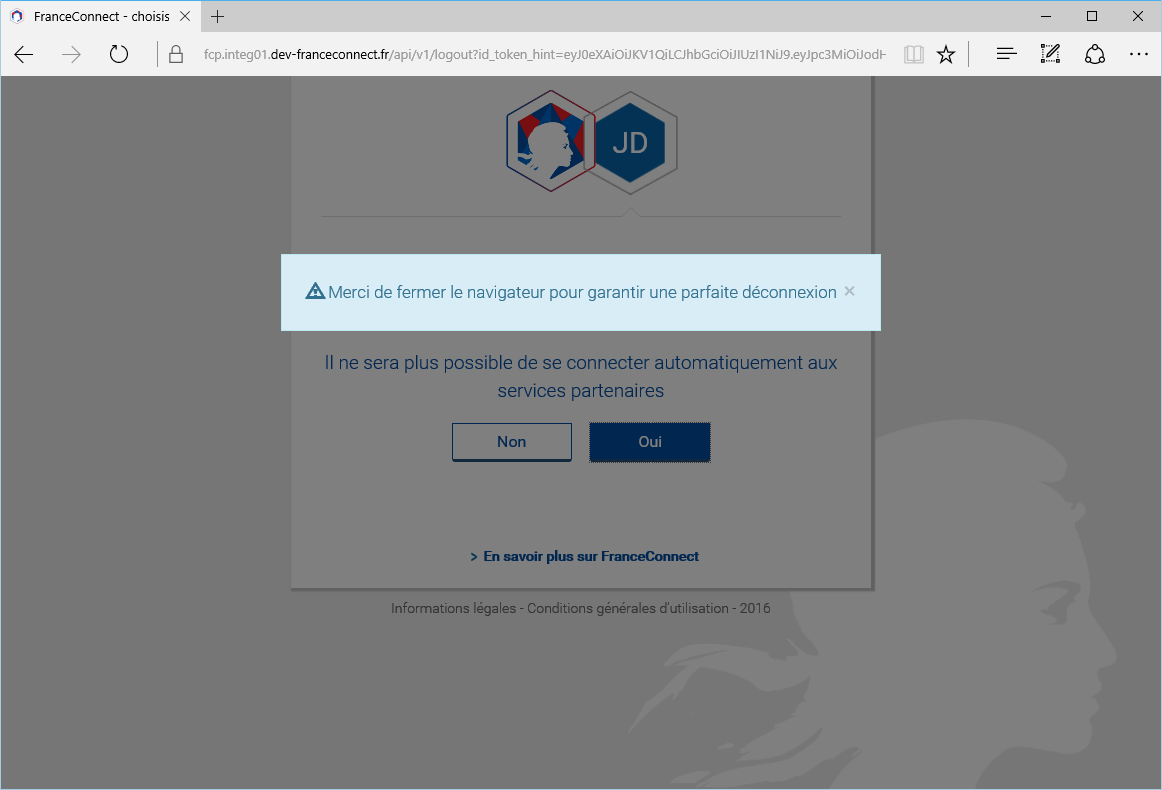
1. Depuis la fenêtre de browser précédente, cliquez sur votre nom d’utilisateur à côté du logo FranceConnect dans le coin supérieur droit de la fenêtre de browser. Un dialogue s’affiche.



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



1. Si tel est le cas, cliquez sur **Oui**. Vous êtes alors redirigé(e)s vers le canevas de fournisseur de service qui vous invite à fermer votre session browser.



1. Fermez votre session browser.

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 DINSIC et en particulier de la [documentation technique d’intégration](https://doc.integ01.dev-franceconnect.fr)[[18]](#footnote-18).

Il repose plus particulièrement pour la définition des éléments de mise en œuvre du fournisseur de service FranceConnect ainsi proposé sur la [documentation](https://doc.integ01.dev-franceconnect.fr/fournisseur-service)[[19]](#footnote-19).

1. Site portail Web Microsoft Secteur Public : http://www.microsoft.com/france/entreprises/secteur-public/solutions-pour-secteur-public.aspx [↑](#footnote-ref-1)
2. ASP.NET Core : http://www.dotnetfoundation.org/aspnet-core [↑](#footnote-ref-2)
3. Fondation .NET : http://www.dotnetfoundation.org/ [↑](#footnote-ref-3)
4. Visual Studio Community 2015 : https://go.microsoft.com/fwlink/?LinkId=691978&clcid=0x40c [↑](#footnote-ref-4)
5. Introduction to ASP.NET Core : https://docs.asp.net/en/latest/conceptual-overview/aspnet.html [↑](#footnote-ref-5)
6. ASP.NET Core Documentation : https://docs.asp.net/en/latest/ [↑](#footnote-ref-6)
7. franceconnect-service-provider-dotnet-webapp-aspnetcore : https://github.com/FranceConnectSamples/franceconnect-service-provider-dotnet-webapp-aspnetcore [↑](#footnote-ref-7)
8. Inscription FranceConnect - Fournisseur de service : https://doc.integ01.dev-franceconnect.fr/inscription [↑](#footnote-ref-8)
9. Microsoft.AspNetCore.Authentication.OpenIdConnect: https://www.nuget.org/packages/Microsoft.AspNetCore.Authentication.OpenIdConnect/ [↑](#footnote-ref-9)
10. ASP.NET Core Application to New Database : http://docs.efproject.net/en/latest/platforms/aspnetcore/new-db.html [↑](#footnote-ref-10)
11. Documentation – Fournisseur de service : https://doc.integ01.dev-franceconnect.fr/fournisseur-service#buttons [↑](#footnote-ref-11)
12. franceconnect-service-provider-dotnet-webapp-aspnetcore : https://github.com/FranceConnectSamples/franceconnect-service-provider-dotnet-webapp-aspnetcore [↑](#footnote-ref-12)
13. Git : http://git-scm.com/ [↑](#footnote-ref-13)
14. GitHub : https://github.com/FranceConnectSamples [↑](#footnote-ref-14)
15. Visual Studio : https://msdn.microsoft.com/vstudio/aa718325.aspx [↑](#footnote-ref-15)
16. Set up Git : https://help.github.com/articles/set-up-git/ [↑](#footnote-ref-16)
17. Accès aux informations d'un client provisionné – Fournisseur de service : https://doc.integ01.dev-franceconnect.fr/client [↑](#footnote-ref-17)
18. FranceConnect : https://doc.integ01.dev-franceconnect.fr/ [↑](#footnote-ref-18)
19. Documentation – Fournisseur de service : https://doc.integ01.dev-franceconnect.fr/fournisseur-service [↑](#footnote-ref-19)