Mettre en œuvre un fournisseur d’identité FranceConnect compatible FIDO 2.0 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 3](#_Toc457805561)

[A propos du kit de démarrage FranceConnect 4](#_Toc457805562)

[Licence 4](#_Toc457805563)

[Commentaires 4](#_Toc457805564)

[Prérequis 5](#_Toc457805565)

[Connaissances et attendu 5](#_Toc457805566)

[Créer un fournisseur d’identité FranceConnect avec ASP.NET Core 6](#_Toc457805567)

[Mettre en œuvre l’authentification Web du W3C (ou FIDO) 7](#_Toc457805568)

[Ajouter une base de données 7](#_Toc457805569)

[Installer les paquets NuGet 7](#_Toc457805570)

[Créer le modèle 8](#_Toc457805571)

[Créer un DbContext 8](#_Toc457805572)

[Ajouter la chaine de connexion 8](#_Toc457805573)

[Modifier la classe Startup 9](#_Toc457805574)

[Appliquer la migration 9](#_Toc457805575)

[Ajouter un service 9](#_Toc457805576)

[Modifier le processus d’authentification pour FIDO 10](#_Toc457805577)

[Modifier le contrôleur d’authentification 10](#_Toc457805578)

[Modifier et ajouter des modèles de vue 12](#_Toc457805579)

[Utiliser l’API MSCredentials 13](#_Toc457805580)

[Modifier et ajouter des vues 15](#_Toc457805581)

[Déployer le canevas de fournisseur d’identité préconfiguré du kit de démarrage 17](#_Toc457805582)

[Obtenir les sources du fournisseur d’identité 17](#_Toc457805583)

[Télécharger les sources du fournisseur d’identité 17](#_Toc457805584)

[Cloner les sources du fournisseur d’identité 18](#_Toc457805585)

[Organisation du code source du canevas de fournisseur d’identité (FI) compatible FIDO 19](#_Toc457805586)

[Comprendre l’organisation du projet 19](#_Toc457805587)

[Appréhender les éléments de configuration 19](#_Toc457805588)

[Déployer le projet dans Microsoft Azure 20](#_Toc457805589)

[Créer l’application web dans Microsoft Azure 20](#_Toc457805590)

[Publier le fournisseur d’identité dans Microsoft Azure 20](#_Toc457805591)

[Tester le fournisseur d’identité 22](#_Toc457805592)

[Configurer le compte FranceConnect 22](#_Toc457805593)

[Se connecter avec le fournisseur d’identité 22](#_Toc457805594)

[Annexe A. Références techniques 27](#_Toc457805595)

# 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 d’identité (FI) compatible FranceConnect. Il s’appuie pour cela sur la technologie [ASP.NET Core](http://www.dotnetfoundation.org/aspnet-core)[[2]](#footnote-2) ainsi que le projet de serveur d’identité [IdentityServer](http://www.dotnetfoundation.org/thinktectureidentityserver)[[3]](#footnote-3), tous deux mis à disposition sous licence libre par la [fondation .NET](http://www.dotnetfoundation.org/)[[4]](#footnote-4).

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 d’identité (FI) en partant si vous le souhaitez d’un projet générique ASP.NET Core 1.0/Identity Server. 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 d’identité ASP.NET Core/Identity Server 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 d’identité (FI) compatible FIDO nécessite un ordinateur sous Windows 10.

## Connaissances et attendu

La mise en œuvre d’un fournisseur d’identité 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)[[5]](#footnote-5),
* 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)[[6]](#footnote-6) 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/)[[7]](#footnote-7) donne par ailleurs accès à l’ensemble de la documentation disponible à date.

* 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 d’identité.

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 d’identité FranceConnect avec ASP.NET Core

Pour créer un fournisseur d’identité FranceConnnect, reportez-vous à la documentation Mettre en œuvre un FI FranceConnect avec ASP.NET Core 1.0.

# Mettre en œuvre l’authentification Web du W3C (ou FIDO)

L’objectif de cette partie et de modifier le processus d’authentification pour implémenter la spécification d’[authentification Web](http://www.w3.org/TR/webauthn/)[[8]](#footnote-8) en cours de standardisation au niveau du W3C, à savoir anciennement la spécification Web API FIDO 2.0 de l’alliance FIDO (Fast IDentity Online) qui a été soumise au W3C.

Note Pour de plus amples informations sur l’alliance FIDO, vous pouvez visiter le [site officiel de l’alliance](https://fidoalliance.org/)[[9]](#footnote-9).

Le navigateur Microsoft Edge dans Windows 10 dispose d’une pré-implémentation de cette spécification d’authentification Web. C’est cette dernière que nous vous proposons de mettre en œuvre au niveau de cette évolution du canevas de fournisseur d’identité précédent.

Note Pour de plus amples informations sur la mise en œuvre de cette spécification au sein du navigateur Microsoft Edge, reportez-vous aux articles [A world without passwords: Windows Hello in Microsoft Edge](https://blogs.windows.com/msedgedev/2016/04/12/a-world-without-passwords-windows-hello-in-microsoft-edge/)[[10]](#footnote-10) et [Web authentication and Windows Hello](https://developer.microsoft.com/en-us/microsoft-edge/platform/documentation/dev-guide/device/web-authentication/)[[11]](#footnote-11) ainsi qu’au webcast [Moving beyond passwords and credential theft with Microsoft Passport and Windows Hello](https://channel9.msdn.com/Events/Windows/Developers-Guide-to-Windows-10-Version-1511/Moving-beyond-passwords-and-credential-theft-with-Microsoft-Passport-and-Windows-Hello)[[12]](#footnote-12).

La suite de cette section décrit la mise en œuvre associée.

## Ajouter une base de données

Le fournisseur d’identité doit conserver les informations d’identification de chaque appareil de chaque utilisateur. Pour cela, vous allez ajouter une base de données au projet :

Note Pour de plus amples informations sur l’utilisation des bases de données avec ASP.NET Core, consultez la [documentation Entity Framework](https://docs.efproject.net/en/latest/platforms/aspnetcore/new-db.html)[[13]](#footnote-13).

### Installer les paquets NuGet

Tout d’abord, ajoutez les paquets suivants au projet :

|  |  |
| --- | --- |
| **Paquet NuGet** | **Version** |
| Microsoft.EntityFrameworkCore.SqlServer | 1.0.0 |
| Microsoft.EntityFrameworkCore.Tools | 1.0.0-preview2-final |

Ajoutez également la ligne suivante dans le fichier *project.json* :

"tools": {

...

"Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final"

}

### Créer le modèle

Puis, vous allez créer un nouveau modèle de donnée *Credential* :

namespace WebApi\_Identity\_Provider\_DotNet.Models

{

public class Credential

{

public string UserId { get; set; }

public string PublicKey { get; set; }

public string PublicKeyHash { get; set; }

public string DeviceName { get; set; }

public string ActiveChallenge { get; set; }

}

}

### Créer un DbContext

Ajoutez ensuite un DbContext à l’application :

namespace WebApi\_Identity\_Provider\_DotNet.Data

{

public class ApplicationDbContext : DbContext

{

public ApplicationDbContext()

: base ()

{ }

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{ }

public DbSet<Credential> Credentials { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Credential>(b =>

{

b.HasKey(c => new { c.UserId, c.PublicKeyHash });

b.Property(c => c.UserId).HasMaxLength(256);

b.Property(c => c.UserId).IsRequired();

b.Property(c => c.PublicKey).IsRequired();

b.Property(c => c.PublicKeyHash).IsRequired();

b.Property(c => c.DeviceName).IsRequired();

});

}

}

}

### Ajouter la chaine de connexion

Dans le fichier de configuration *appsettings.json*, ajoutez les lignes suivantes :

"ConnectionStrings": {

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

}

### Modifier la classe Startup

Vous allez modifiez la classe *Startup* afin d’ajouter la base de données au projet :

public void ConfigureServices(IServiceCollection services)

{

// Add framework services.

services.AddDbContext<ApplicationDbContext>(options =>

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

...

}

### Appliquer la migration

Enfin, exécutez les deux commandes suivantes :

> dotnet ef migrations add CreateDatabase

> dotnet ef database update

## Ajouter un service

Afin du mutualiser l’accès aux informations d’authentification en base de données, vous allez créer un service *CredentialService* :

namespace WebApi\_Identity\_Provider\_DotNet.Services

{

public class CredentialService

{

private ApplicationDbContext \_context;

public CredentialService(ApplicationDbContext context)

{

\_context = context;

}

public bool RegisterCredential(string userId, string publicKey, string publicKeyHash, string deviceName)

{

\_context.Credentials.Add(new Credential

{

UserId = userId,

PublicKey = publicKey,

PublicKeyHash = publicKeyHash,

DeviceName = deviceName

});

\_context.SaveChanges();

return true;

}

public bool RemoveCredential(string userId, string publicKeyHash)

{

var credential = \_context.Credentials

.SingleOrDefault(c => c.UserId == userId && c.PublicKeyHash == publicKeyHash);

\_context.Credentials.Remove(credential);

\_context.SaveChanges();

return true;

}

public JsonWebKey GetPublicKeyForUser(string userId, string publicKeyHash, out string challenge)

{

var credential = \_context.Credentials

.FirstOrDefault(c => c.UserId == userId && c.PublicKeyHash == publicKeyHash);

// Each challenge is valid only once. Prevent replay attack.

challenge = credential.ActiveChallenge;

credential.ActiveChallenge = null;

\_context.Entry(credential).State = EntityState.Modified;

\_context.SaveChanges();

return JsonConvert.DeserializeObject<JsonWebKey>(credential.PublicKey);

}

public bool SetActiveChallenge(string userId, string publicKeyHash, string challenge)

{

var credential = \_context.Credentials

.FirstOrDefault(c => c.UserId == userId && c.PublicKeyHash == publicKeyHash);

credential.ActiveChallenge = challenge;

\_context.Entry(credential).State = EntityState.Modified;

\_context.SaveChanges();

return true;

}

}

}

Comme le rapprochement d’un utilisateur avec des informations d’authentification se fait pas le *Subject* de l’utilisateur, ajoutez une méthode au service existant *SignInService* :

public InMemoryUser FindBySubject(string subject)

{

return \_users.FirstOrDefault(x => x.Subject.Equals(subject, StringComparison.Ordinal));

}

## Modifier le processus d’authentification pour FIDO

### Modifier le contrôleur d’authentification

Modifiez le contrôleur *LoginController* de la façon suivante :

public class LoginController : Controller

{

...

private readonly CredentialService \_credentialService;

public LoginController(

SignInService signInService,

SignInInteraction signInInteraction,

CredentialService credentialService)

{

\_signInService = signInService;

\_signInInteraction = signInInteraction;

\_credentialService = credentialService;

}

...

[HttpPost(Constants.RoutePaths.Login)]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Index(LoginViewModel model)

{

if (ModelState.IsValid)

{

if (\_signInService.ValidateCredentials(model.Username, model.Password))

{

var user = \_signInService.FindByUsername(model.Username);

await IssueCookie(user, "idsvr", "password");

if (!model.IsFidoAvailable || model.IsFidoEnable)

{

return new SignInResult(model.SignInId);

}

else

{

return Redirect("~/ui/add\_credential?SignInId=" + model.SignInId + "&UserId=" + user.Subject);

}

}

ModelState.AddModelError("", "Invalid username or password.");

}

return View(model);

}

...

// FIDO

[HttpGet("ui/add\_credential")]

public IActionResult AddCredential(AddCredentialViewModel model)

{

return View(model);

}

[HttpGet("ui/finalize\_credential")]

public IActionResult FinalizeCredential(string id)

{

return new SignInResult(id);

}

[HttpPost("ui/request\_challenge")]

public string RequestChallenge(UserAndKeyHintMessage message)

{

try

{

string challenge = Guid.NewGuid().ToString("N");

\_credentialService.SetActiveChallenge(message.UserId, message.PublicKeyHint, challenge);

return challenge;

}

catch (Exception)

{

return string.Empty;

}

}

[HttpPost("ui/submit\_response")]

public async Task<bool> SubmitResponse(SignatureMessage message)

{

bool retval = false;

try

{

string challenge;

JsonWebKey publicKey = \_credentialService

.GetPublicKeyForUser(message.UserId, message.PublicKeyHint, out challenge);

var decodedClientData = message.ClientData.Rfc4648Base64UrlDecode();

var decodedAuthnrData = message.AuthnrData.Rfc4648Base64UrlDecode();

var clientDataJson = Encoding.UTF8.GetString(decodedClientData);

var clientData = JsonConvert.DeserializeObject<ClientData>(clientDataJson);

if (clientData.Challenge != challenge) return false;

var sha256 = SHA256.Create();

var hashedClientData = sha256.ComputeHash(decodedClientData);

var buffer = new byte[decodedAuthnrData.Length + hashedClientData.Length];

decodedAuthnrData.CopyTo(buffer, 0);

hashedClientData.CopyTo(buffer, decodedAuthnrData.Length);

var publicKeyInfo = new RSAParameters();

publicKeyInfo.Modulus = publicKey.N.Rfc4648Base64UrlDecode();

publicKeyInfo.Exponent = publicKey.E.Rfc4648Base64UrlDecode();

var rsa = new RSACng();

rsa.ImportParameters(publicKeyInfo);

byte[] signature = message.Signature.Rfc4648Base64UrlDecode();

retval = rsa.VerifyData(buffer, signature, HashAlgorithmName.SHA256, RSASignaturePadding.Pkcs1);

if (retval)

{

var user = \_signInService.FindBySubject(message.UserId);

await IssueCookie(user, "idsvr", "fido");

}

}

catch (Exception)

{ }

return retval;

}

[HttpPost("ui/register\_credential")]

public async Task<bool> Register(RegisterMessage message)

{

bool retval = false;

var info = await HttpContext.Authentication.GetAuthenticateInfoAsync(Constants.PrimaryAuthenticationType);

var sub = info.Principal.Claims

.Where(x => x.Type == JwtClaimTypes.Subject)

.Select(x => x.Value)

.FirstOrDefault();

var user = \_signInService.FindBySubject(sub);

if (user == null || message.PublicKey == null)

{

return false;

}

try

{

SHA256 hashalg = SHA256.Create();

byte[] publicKeyHashBuffer = hashalg.ComputeHash(Encoding.UTF8.GetBytes(message.PublicKey));

string publicKeyHash = Convert.ToBase64String(publicKeyHashBuffer);

retval = \_credentialService

.RegisterCredential(message.UserId, message.PublicKey, publicKeyHash, message.DeviceName);

}

catch (Exception)

{ }

return retval;

}

[HttpPost("ui/remove\_credential")]

public bool RemoveRegisteredKey(UserAndKeyHintMessage message)

{

bool retval = false;

try

{

retval = \_credentialService.RemoveCredential(message.UserId, message.PublicKeyHint);

}

catch (Exception)

{ }

return retval;

}

}

Ajoutez également le helper suivant pour simplifier le décodage Base 64 :

namespace WebApi\_Identity\_Provider\_DotNet.Helpers

{

public static class StringExtensions

{

public static byte[] Rfc4648Base64UrlDecode(this string url)

{

url = url.Replace('-', '+');

url = url.Replace('\_', '/');

switch (url.Length % 4)

{

// Pad with trailing '='s

case 0:

// No pad chars in this case

break;

case 2:

// Two pad chars

url += "==";

break;

case 3:

// One pad char

url += "=";

break;

default:

throw new Exception("Invalid string.");

}

return Convert.FromBase64String(url);

}

}

}

### Modifier et ajouter des modèles de vue

Modifiez le modèle de vue *LoginViewModel* :

public class LoginViewModel

{

public string SignInId { get; set; }

[Required]

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

public string Username { get; set; }

[Required]

[DataType(DataType.Password)]

[Display(Name = "Mot de passe")]

public string Password { get; set; }

public bool IsFidoAvailable { get; set; }

public bool IsFidoEnable { get; set; }

public string ErrorMessage { get; set; }

}

Créez les modèles de vue suivants :

public class RegisterMessage

{

public string UserId { get; set; }

public string PublicKey { get; set; }

public string DeviceName { get; set; }

}

public class UserAndKeyHintMessage

{

public string UserId { get; set; }

public string PublicKeyHint { get; set; }

}

public class SignatureMessage

{

public string UserId { get; set; }

public string Signature { get; set; }

public string PublicKeyHint { get; set; }

public string ClientData { get; set; }

public string AuthnrData { get; set; }

}

public class ClientData

{

public string Challenge { get; set; }

public string UserPrompt { get; set; }

}

public class AddCredentialViewModel

{

public string SignInId { get; set; }

public string UserId { get; set; }

}

### Utiliser l’API MSCredentials

Ajoutez un fichier JavaScript *fido.js* :

"use strict";

if (typeof Fido == "undefined") {

var Fido = {

isAvailable: typeof window.msCredentials != "undefined",

isEnable: localStorage.getItem("fido\_isEnable") ? true : false,

makeCredential: function (signInId, userId) {

try {

var accountInfo = {

rpDisplayName: 'FranceConnect',

userDisplayName: userId

};

var cryptoParameters = [

{

type: 'FIDO\_2\_0',

algorithm: 'RSASSA-PKCS1-v1\_5'

}

];

window.msCredentials.makeCredential(accountInfo, cryptoParameters)

.then(function (cred) {

var publicKeyHash = CryptoJS.SHA256(cred.publicKey);

var publicKeyHint = publicKeyHash.toString(CryptoJS.enc.Base64);

var xhr = new XMLHttpRequest();

xhr.onreadystatechange = function () {

if (xhr.readyState == 4 && xhr.status == 200)

if (xhr.responseText == "true") {

localStorage.setItem("fido\_isEnable", true);

localStorage.setItem("fido\_publicKeyHint", publicKeyHint);

localStorage.setItem("fido\_userId", userId);

localStorage.setItem("fido\_credentialId", cred.id);

alert("Windows Hello est configuré pour FranceConnect !");

window.location = "/ui/finalize\_credential?id=" + signInId;

} else {

alert("Impossible de configurer Windows Hello pour FranceConnect.");

}

}

xhr.open("POST", "/ui/register\_credential", true);

xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

var str = "DeviceName=" + navigator.userAgent

+ "&UserId=" + userId

+ "&PublicKey=" + encodeURIComponent(cred.publicKey);

xhr.send(str);

}).catch(function (reason) {

if (reason.message === 'NotSupportedError') {

alert("Vous devez activer Windows Hello sur votre appareil pour l'utiliser sur ce site.");

}

console.log("FIDO error: " + reason.message);

});

} catch (ex) {

console.log("FIDO error: " + ex);

}

},

getAssertion: function (signInId) {

try {

var userId = localStorage.getItem("fido\_userId");

var publicKeyHint = localStorage.getItem("fido\_publicKeyHint");

var credentialId = localStorage.getItem("fido\_credentialId");

var xhrChallenge = new XMLHttpRequest();

xhrChallenge.onreadystatechange = function () {

if (xhrChallenge.readyState == 4 && xhrChallenge.status == 200) {

var challenge = xhrChallenge.responseText;

var filters = {

accept:

[

{

type: 'FIDO\_2\_0',

id: credentialId

}

]

};

window.msCredentials.getAssertion(challenge, filters)

.then(function (attestation) {

var xhrSignature = new XMLHttpRequest();

xhrSignature.onreadystatechange = function () {

if (xhrSignature.readyState == 4 && xhrSignature.status == 200)

if (xhrSignature.responseText == "true") {

window.location = "/ui/finalize\_credential?id=" + signInId;

} else {

alert("Tentative de connexion incorrecte.");

}

}

xhrSignature.open("POST", "/ui/submit\_response", true);

xhrSignature.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

var signatureParams = "UserId=" + encodeURIComponent(userId)

+ "&Signature=" + encodeURIComponent(attestation.signature.signature)

+ "&PublicKeyHint=" + encodeURIComponent(publicKeyHint)

+ "&ClientData=" + encodeURIComponent(attestation.signature.clientData)

+ "&AuthnrData=" + encodeURIComponent(attestation.signature.authnrData);

xhrSignature.send(signatureParams);

});

}

}

xhrChallenge.open("POST", "/ui/request\_challenge", true);

xhrChallenge.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

var challengeParams = "DeviceName=" + navigator.userAgent

+ "&UserId=" + encodeURIComponent(userId)

+ "&PublicKeyHint=" + encodeURIComponent(publicKeyHint);

xhrChallenge.send(challengeParams);

} catch (ex) {

console.log("FIDO error: " + ex);

}

},

removeCredential: function (callback) {

try {

var userId = localStorage.getItem("fido\_userId");

var publicKeyHint = localStorage.getItem("fido\_publicKeyHint");

var xhrDelete = new XMLHttpRequest();

xhrDelete.onreadystatechange = function () {

if (xhrDelete.readyState == 4 && xhrDelete.status == 200)

if (xhrDelete.responseText == "true") {

localStorage.removeItem("fido\_isEnable");

localStorage.removeItem("fido\_publicKeyHint");

localStorage.removeItem("fido\_userId");

localStorage.removeItem("fido\_credentialId");

callback();

alert("Vos informations d'ientification ont été supprimé.");

} else {

alert("Une erreur est survenue lors de la suppression de vos informations d'identification. Veuillez réessayer ultérieurement.");

}

}

xhrDelete.open("POST", "/ui/remove\_credential", true);

xhrDelete.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

var deleteParams = "UserId=" + encodeURIComponent(userId)

+ "&PublicKeyHint=" + encodeURIComponent(publicKeyHint);

xhrDelete.send(deleteParams);

} catch (ex) {

console.log("FIDO error: " + ex);

}

},

later: function (signInId) {

window.location = "/ui/finalize\_credential?id=" + signInId;

}

}

}

### Modifier et ajouter des vues

Modifiez la vue *Login* de la façon suivante :

@model WebApi\_Identity\_Provider\_DotNet.ViewModels.Login.LoginViewModel

<h2>Connexion</h2>

<div class="row">

<div class="col-md-8">

<section>

<form asp-route="Login" class="form-horizontal">

<h4>Utiliser l'authentification par mot de passe</h4>

<hr />

<div asp-validation-summary="All" class="text-danger"></div>

<input type="hidden" asp-for="SignInId" />

<div class="form-group">

<label asp-for="Username" class="col-md-3 control-label"></label>

<div class="col-md-9">

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

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

</div>

</div>

<div class="form-group">

<label asp-for="Password" class="col-md-3 control-label"></label>

<div class="col-md-9">

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

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

</div>

</div>

<input type="hidden" asp-for="IsFidoEnable" />

<input type="hidden" asp-for="IsFidoAvailable" />

<div class="form-group">

<div class="col-md-offset-3 col-md-9">

<button type="submit" class="btn btn-primary">Connexion</button>

</div>

</div>

</form>

</section>

</div>

<div class="col-md-4 hide" id="WindowsHello">

<section>

<h4>Utiliser Windows Hello</h4>

<hr />

<div class="form-group">

<button type="button" class="btn btn-primary" onclick="Fido.getAssertion('@Model.SignInId')">Connexion avec Windows Hello</button>

<button type="button" class="btn btn-default" onclick="Fido.removeCredential(hideWindowsHello)" title="Ne plus utiliser Windows Hello">

<span class="glyphicon glyphicon-trash"></span>

</button>

</div>

</section>

</div>

</div>

@section Scripts{

<script src="~/lib/crypto-js/crypto-js.js"></script>

<script src="~/js/fido.js"></script>

<script>

"use strict";

$(document).ready(function () {

$("[name=IsFidoAvailable]").val(Fido.isAvailable);

$("[name=IsFidoEnable]").val(Fido.isEnable);

if (Fido.isAvailable && Fido.isEnable) {

$("#WindowsHello").removeClass("hide");

}

});

function hideWindowsHello() {

$("#WindowsHello").addClass("hide");

$("[name=IsFidoEnable]").val(false);

}

</script>

}

Enfin, ajoutez la vue *AddCredential* :

@model WebApi\_Identity\_Provider\_DotNet.ViewModels.Login.AddCredentialViewModel

<h2>Connexion</h2>

<div class="row">

<div class="col-md-8">

<section>

<h4>Configurer Windows Hello</h4>

<hr />

<div class="form-group">

<p>

Votre appareil supporte Windows Hello.

Grâce à Windows Hello vous pouvez vous connecter à ce site sur ce PC, sans utiliser de mot de passe.

</p>

<p>

Voulez-vous activer Windows Hello pour la prochaine connexion ?

</p>

</div>

<div class="form-group">

<button type="button" class="btn btn-default" onclick="Fido.later('@Model.SignInId')">Plus tard</button>

<button type="button" class="btn btn-primary" onclick="Fido.makeCredential('@Model.SignInId', '@Model.UserId')">Utiliser Windows Hello</button>

</div>

</section>

</div>

</div>

@section Scripts{

<script src="~/lib/crypto-js/crypto-js.js"></script>

<script src="~/js/fido.js"></script>

}

# Déployer le canevas de fournisseur d’identité préconfiguré du kit de démarrage

## imageObtenir les sources du fournisseur d’identité

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 d’identité (FI) : [franceconnect-identity-provider-dotnet-webapi-aspnetcore](https://github.com/FranceConnectSamples/franceconnect-identity-provider-dotnet-webapi-aspnetcore)[[14]](#footnote-14).

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 d’identité

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-identity-provider-dotnet-webapi-aspnetcore](https://github.com/FranceConnectSamples/franceconnect-identity-provider-dotnet-webapi-aspnetcore)**.**
2. **Cliquez sur Clone or Download.**

****

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

**Le code source du canevas de fournisseur d’identité FranceConnect (FI) est alors situé sous le dossier *C:\Code\franceconnect-identity-provider-dotnet-webapi-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 d’identité

 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/)[[15]](#footnote-15), [GitHub](https://github.com/FranceConnectSamples)[[16]](#footnote-16) ou [Visual Studio](https://msdn.microsoft.com/vstudio/aa718325.aspx)[[17]](#footnote-17). 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/)[[18]](#footnote-18) 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 avec le browser et naviguer vers le projet** [franceconnect-identity-provider-dotnet-webapi-aspnetcore](https://github.com/FranceConnectSamples/franceconnect-identity-provider-dotnet-webapi-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-identity-provider-dotnet-webapi-aspnetcore

Le clonage démarre.

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

Cloning into 'franceconnect-identity-provider-dotnet-webapi-aspnetcore'...

Le package du canevas de fournisseur d’identité FranceConnect (FI) est disponible sous le dossier *franceconnect-identity-provider-dotnet-webapi-aspnetcore* situé sous *%UserProfile%\Documents\GitHub*.

**Le code source associé est alors situé sous le dossier** *%UserProfile%\Documents\GitHub****\franceconnect-identity-provider-dotnet-webapi-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.

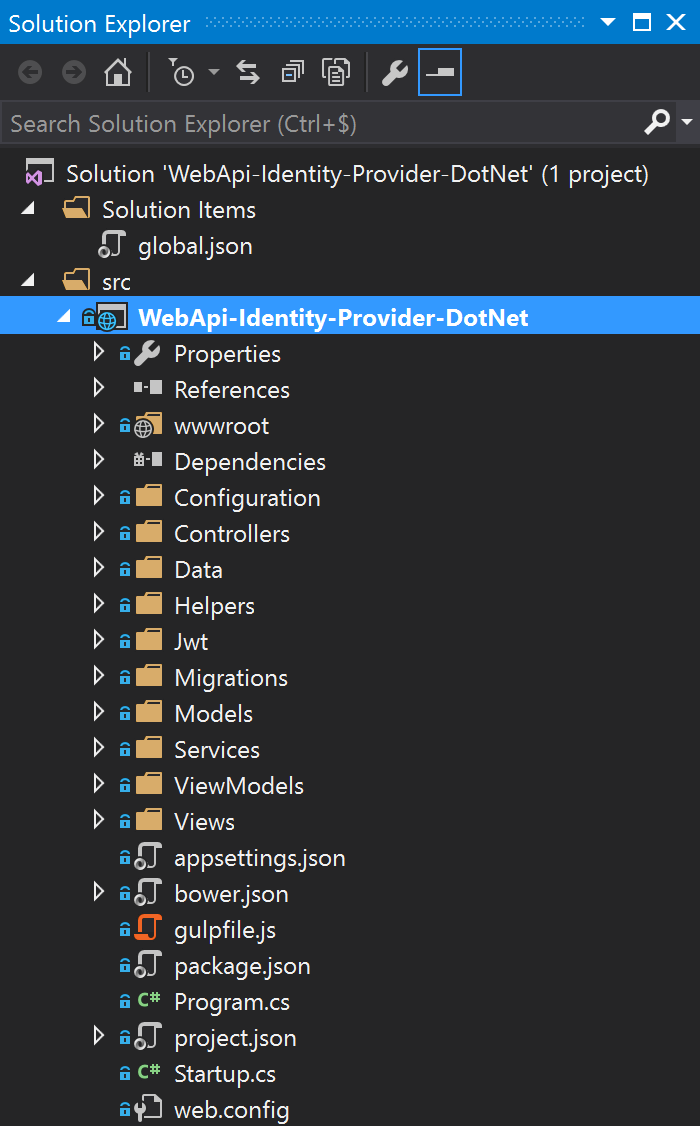
Remarque importante Nous faisons référence au dossier où se situe le code source du canevas de fournisseur d’identité (FI) comme étant le dossier Kit-Demarrage dans la suite de ce document.

## Organisation du code source du canevas de fournisseur d’identité (FI) compatible FIDO

A l’instar de ce qui est décrit dans la section § [*Créer un fournisseur d’identité FranceConnect avec ASP.NET Core*](#_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 d’identité 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"

}

}

}

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

## Déployer le projet dans Microsoft Azure

Pour déployer le fournisseur d’identité (FI) dans l’environnement Microsoft Azure, vous allez devoir créer une nouvelle application web.

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

Comme indiqué en préambule 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

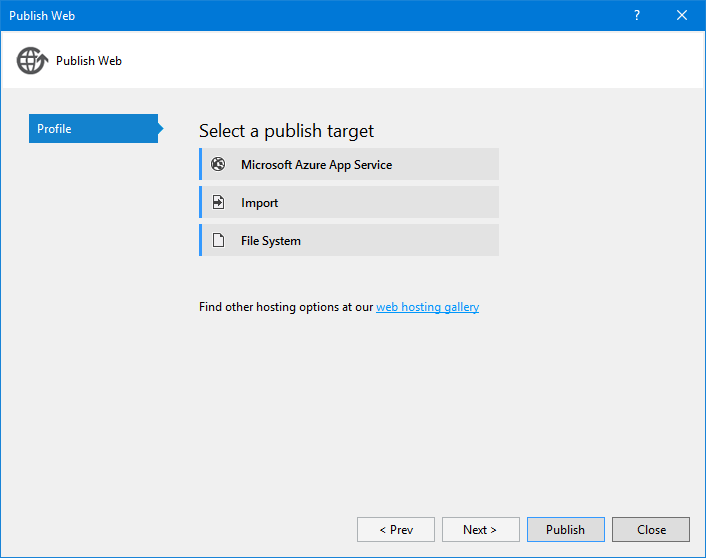
Pour créer l’application web relative à votre fournisseur d’identité dans Microsoft Azure, procédez comme suit :

1. **Ouvrez une session avec le 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**.
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 d’identité dans Microsoft Azure

Pour publier votre fournisseur d’identité 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 d’identité

## Configurer le compte FranceConnect

Vous devez inscrire votre fournisseur d’identité sur FranceConnect afin d’être référencer sur la page de connexion de FranceConnect.

1. **Ouvrez une session avec le browser et** rendez-vous sur le [formulaire d'inscription](https://docs.google.com/forms/d/15Wm0CZ4mLn9WnyGZ4fRE9PWDvJp4X0i61pv37vECUQs/viewform)
2. Ajoutez vos points de terminaison (endpoints) :
   1. « Authorization endpoint » : <*FI URL*>/connect/authorize
   2. « Token endpoint » : <FI URL>/consent/token
   3. « Userinfo endpoint » : <FI URL>/consent/userinfo

Vous pouvez retrouver ces URL sur <FI URL>/.well-known/openid-configuration

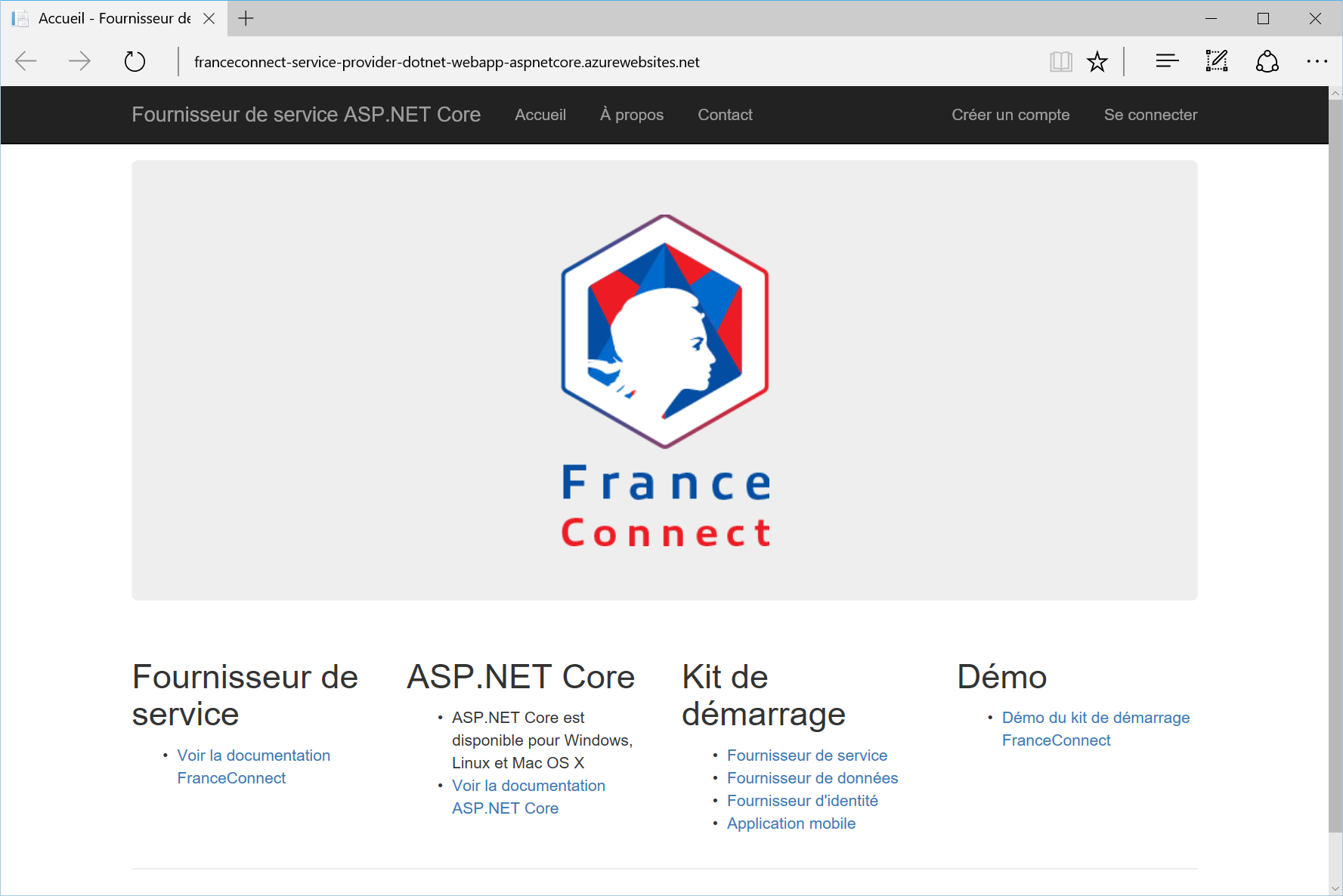
1. Générer un client ID et un client secret pour FranceConnect

Où <*FI URL*> est l’adresse à laquelle votre fournisseur d’identité (FI) est déployé dans Microsoft Azure.

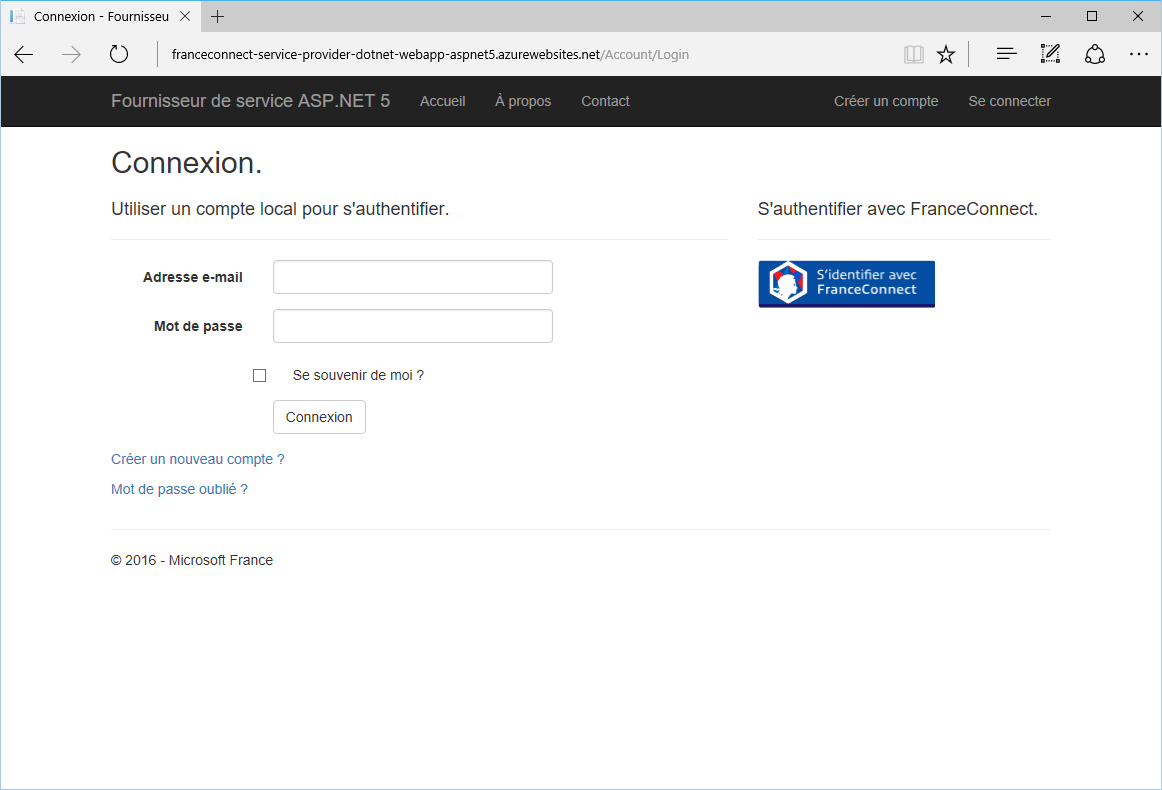
## Se connecter avec le fournisseur d’identité

Pour se connecter avec le fournisseur d’identité, procédez comme suit :

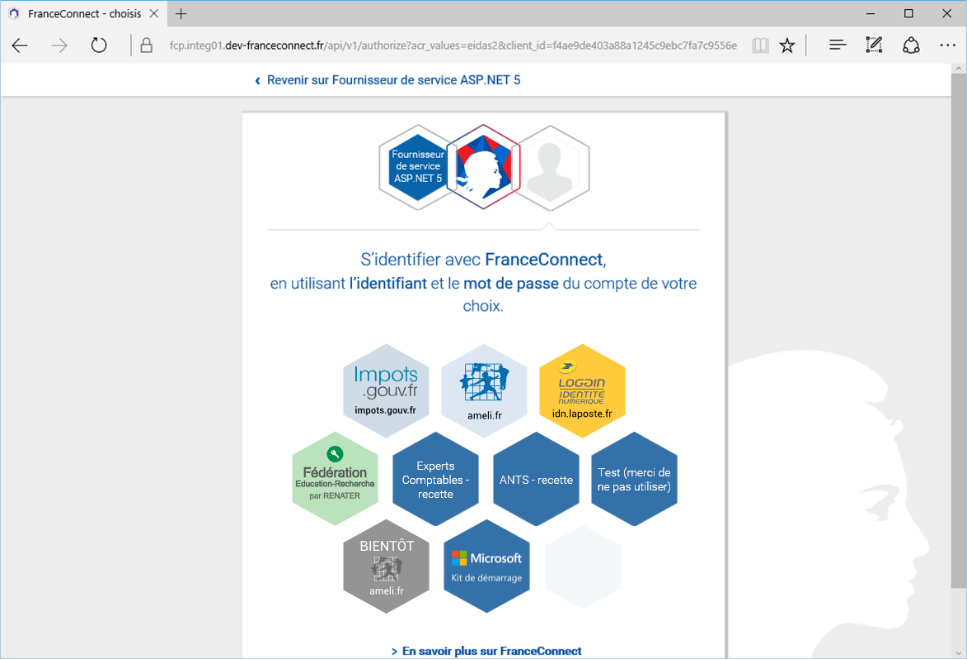
1. Ouvrez une session browser et naviguez vers l’instance de démonstration du canevas de fournisseur de service à l’adresse <http://aka.ms/FranceConnect-FS>.



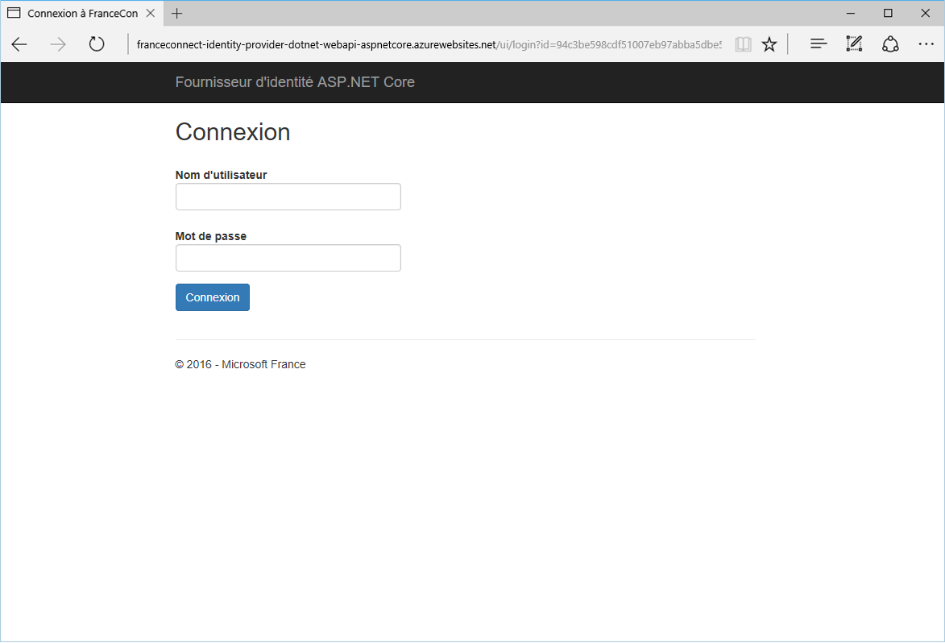
1. Cliquez sur **Se connecter** dans le coin supérieur droit de la fenêtre du browser.



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



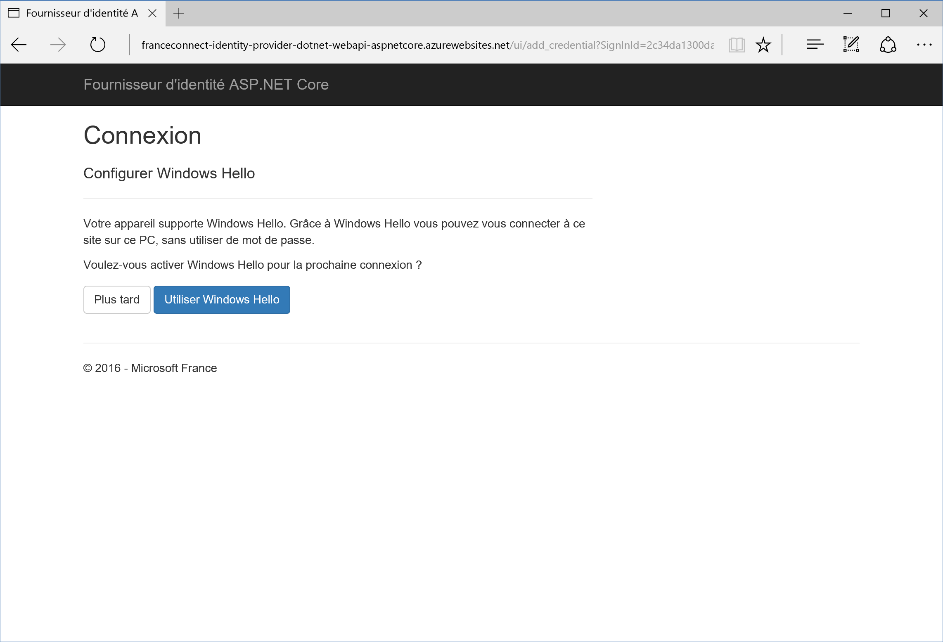
1. Sélectionnez votre canevas de fournisseur d’identité ou le fournisseur d’identité factice **Microsoft Kit de démarrage** pour l’instance de démonstration du canevas de fournisseur d’identité. Vous êtes alors redirigé(e)s vers ce fournisseur d’identité.



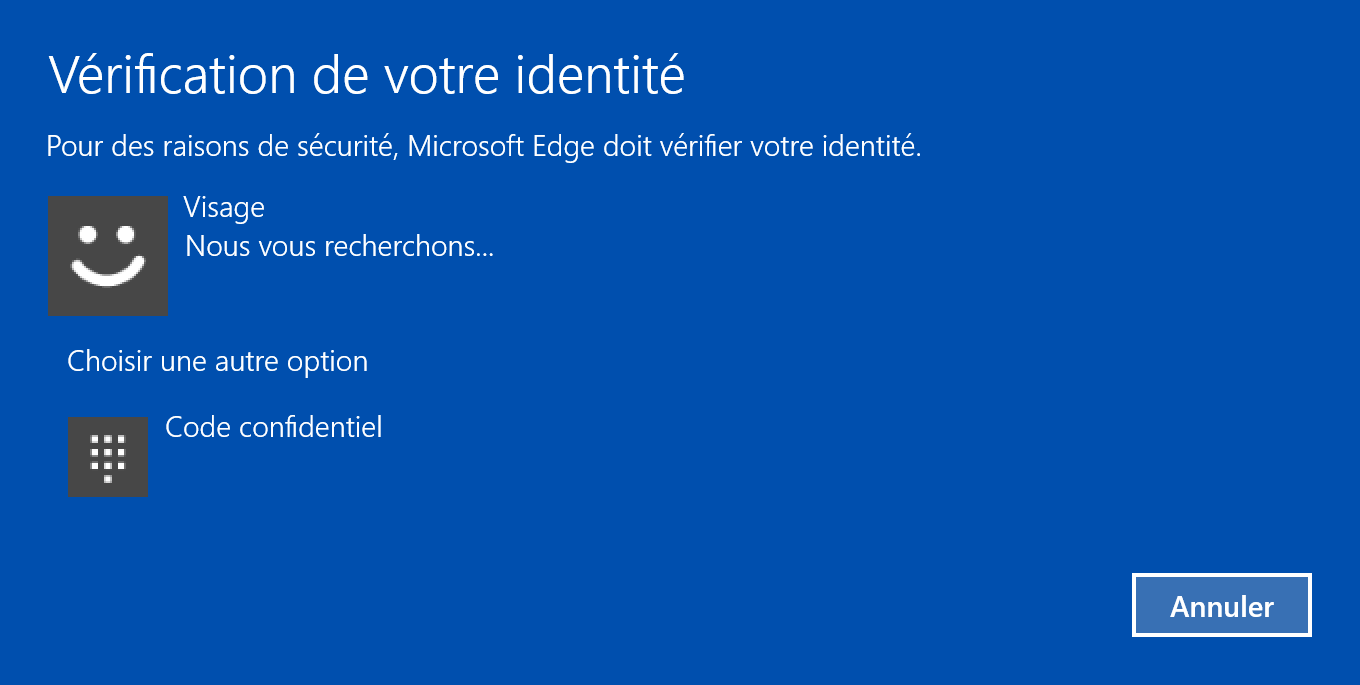
1. Au niveau du fournisseur d’identité, identifiez-vous. Si vous avez sélectionnez l’instance de démonstration du canevas de fournisseur d’identité, identifiez-vous en tant que l’utilisateur Jean Dupont. Précisez pour cela les informations suivantes dans le formulaire qui vous est proposé :

* **Nom d’utilisateur** : jean
* **Mot de passe** : password

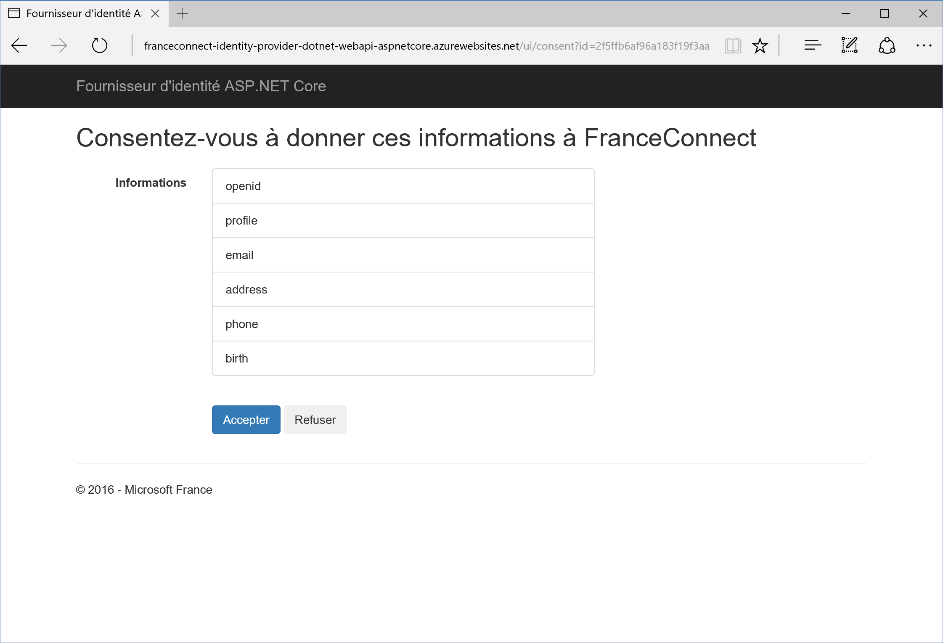
1. Cliquez sur **Utiliser Windows Hello**.



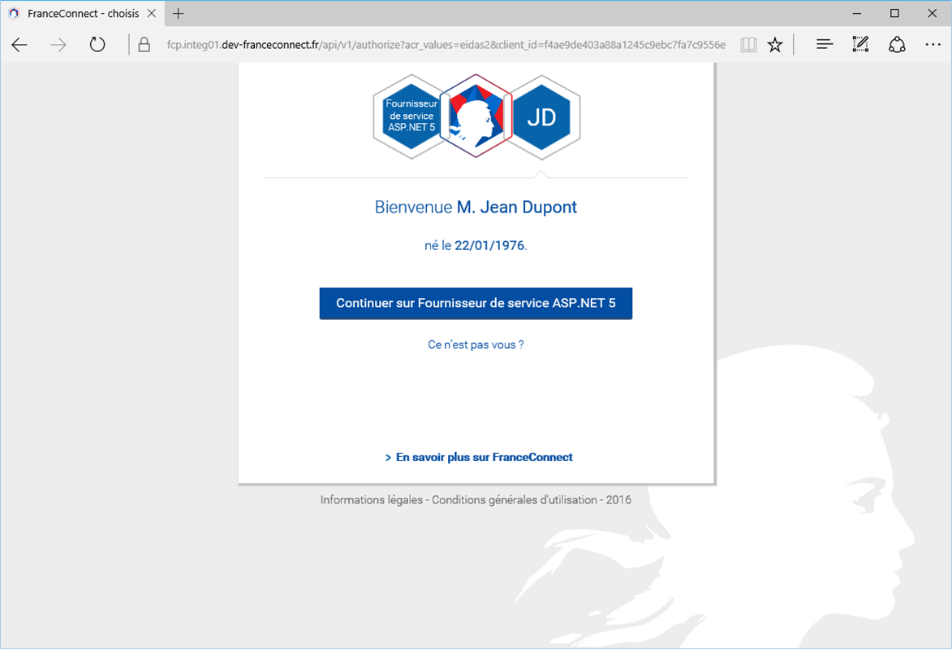
1. Une page s’affiche pour vérifier votre identité.



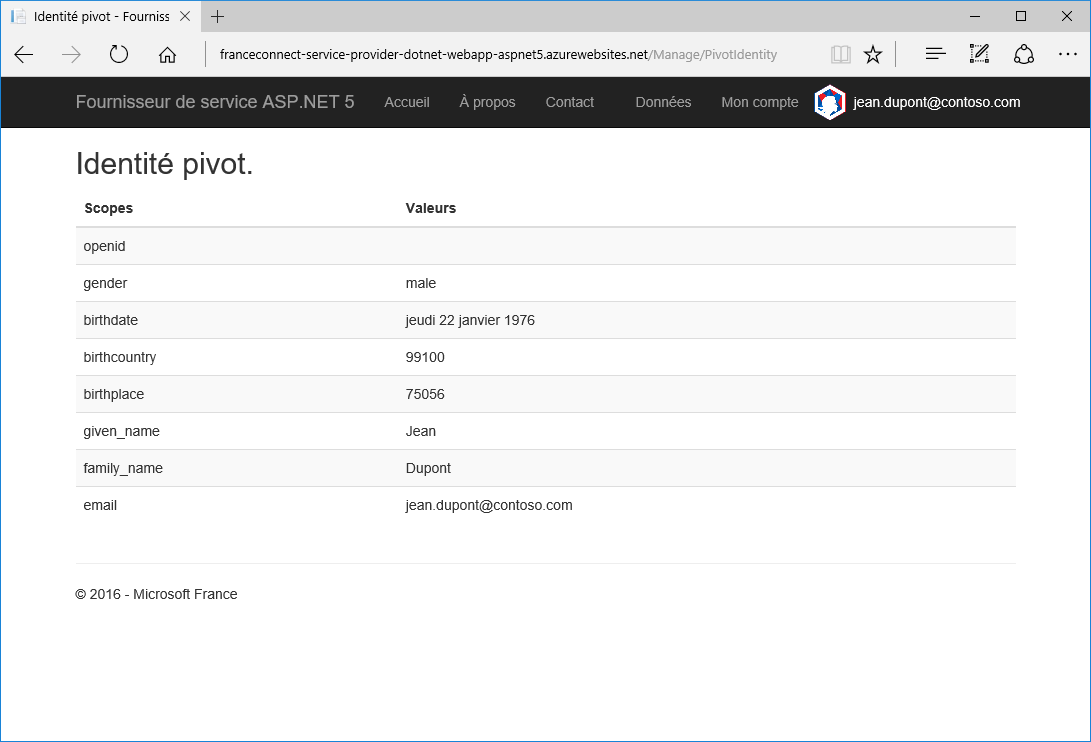
1. Cliquez sur **Connexion**. Une page de consentement s’affiche au niveau du fournisseur d’identité.



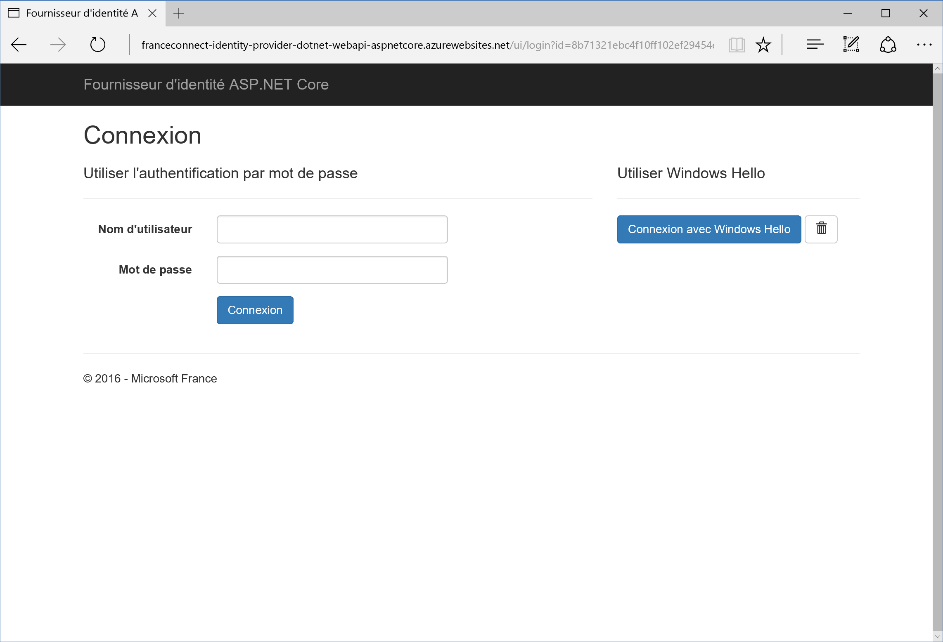
1. Les informations qui seront communiquées à la plateforme FranceConnect sont affichées. Si vous donnez votre consentement, cliquez sur **Accepter**. Vous êtes alors redirigé(e)s vers la plateforme FranceConnect qui vous invite à donner (à son tour) votre consentement.



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.



1. Lors de votre prochaine connexion, vous verrez cette page de connexion qui vous permettra de vous authentifier avec Windows Hello.



Ceci conclut ce guide de mise en œuvre d’un canevas de fournisseur d’identité (FI) 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)[[19]](#footnote-19).

Il repose plus particulièrement pour la définition des éléments de mise en œuvre du fournisseur d’identité FranceConnect ainsi proposé sur la [documentation](https://doc.integ01.dev-franceconnect.fr/fournisseur-identite)[[20]](#footnote-20).

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. IdentityServer : http://www.dotnetfoundation.org/thinktectureidentityserver [↑](#footnote-ref-3)
4. Fondation .NET : http://www.dotnetfoundation.org/ [↑](#footnote-ref-4)
5. Visual Studio Community 2015 : https://go.microsoft.com/fwlink/?LinkId=691978&clcid=0x40c [↑](#footnote-ref-5)
6. Introduction to ASP.NET Core : https://docs.asp.net/en/latest/conceptual-overview/aspnet.html [↑](#footnote-ref-6)
7. ASP.NET Core Documentation : https://docs.asp.net/en/latest/ [↑](#footnote-ref-7)
8. Web Authentication: A Web API for accessing scoped credentials : http://www.w3.org/TR/webauthn/ [↑](#footnote-ref-8)
9. FIDO Alliance : https://fidoalliance.org/ [↑](#footnote-ref-9)
10. A world without passwords: Windows Hello in Microsoft Edge : https://blogs.windows.com/msedgedev/2016/04/12/a-world-without-passwords-windows-hello-in-microsoft-edge/ [↑](#footnote-ref-10)
11. Web authentication and Windows Hello : https://developer.microsoft.com/en-us/microsoft-edge/platform/documentation/dev-guide/device/web-authentication/ [↑](#footnote-ref-11)
12. Moving beyond passwords and credential theft with Microsoft Passport and Windows Hello : https://channel9.msdn.com/Events/Windows/Developers-Guide-to-Windows-10-Version-1511/Moving-beyond-passwords-and-credential-theft-with-Microsoft-Passport-and-Windows-Hello [↑](#footnote-ref-12)
13. ASP.NET Core Application to New Database : https://docs.efproject.net/en/latest/platforms/aspnetcore/new-db.html [↑](#footnote-ref-13)
14. franceconnect-identity-provider-dotnet-webapi-aspnetcore : https://github.com/FranceConnectSamples/franceconnect-identity-provider-dotnet-webapi-aspnetcore [↑](#footnote-ref-14)
15. Git : http://git-scm.com/ [↑](#footnote-ref-15)
16. GitHub : https://github.com/FranceConnectSamples [↑](#footnote-ref-16)
17. Visual Studio : https://msdn.microsoft.com/vstudio/aa718325.aspx [↑](#footnote-ref-17)
18. Set up Git : https://help.github.com/articles/set-up-git/ [↑](#footnote-ref-18)
19. FranceConnect : https://doc.integ01.dev-franceconnect.fr/ [↑](#footnote-ref-19)
20. Documentation – Fournisseur d’identité : https://doc.integ01.dev-franceconnect.fr/fournisseur-identite [↑](#footnote-ref-20)