Mettre en œuvre un fournisseur d’identité 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 3](#_Toc455038285)

[A propos du kit de démarrage FranceConnect 4](#_Toc455038286)

[Licence 4](#_Toc455038287)

[Commentaires 4](#_Toc455038288)

[Prérequis 5](#_Toc455038289)

[Connaissances et attendu 5](#_Toc455038290)

[Créer un fournisseur d’identité FranceConnect avec ASP.NET Core 6](#_Toc455038291)

[Créer un fournisseur d’identité 6](#_Toc455038292)

[Ajouter IdentityServer 4 7](#_Toc455038293)

[Installer le package NuGet 7](#_Toc455038294)

[Configurer IdentityServer 7](#_Toc455038295)

[Modifier l’algorithme de signature du JWT 8](#_Toc455038296)

[Configurer l’application pour IdentityServer 13](#_Toc455038297)

[Créer l’UI 14](#_Toc455038298)

[Configurer l’application pour les vues 14](#_Toc455038299)

[Ajouter la vue Login 17](#_Toc455038300)

[Ajouter la vue Consentement 19](#_Toc455038301)

[Déployer le canevas de fournisseur d’identité préconfiguré du kit de démarrage 22](#_Toc455038302)

[Obtenir les sources du fournisseur d’identité 22](#_Toc455038303)

[Télécharger les sources du fournisseur d’identité 22](#_Toc455038304)

[Cloner les sources du fournisseur d’identité 23](#_Toc455038305)

[Organisation du code source du canevas de fournisseur d’identité (FI) 24](#_Toc455038306)

[Comprendre l’organisation du projet 24](#_Toc455038307)

[Appréhender les éléments de configuration 24](#_Toc455038308)

[Déployer le projet dans Microsoft Azure 25](#_Toc455038309)

[Créer l’application web dans Microsoft Azure 25](#_Toc455038310)

[Publier le fournisseur d’identité dans Microsoft Azure 25](#_Toc455038311)

[Tester le fournisseur d’identité 27](#_Toc455038312)

[Configurer le compte FranceConnect 27](#_Toc455038313)

[Se connecter avec le fournisseur d’identité 27](#_Toc455038314)

[Annexe A. Références techniques 31](#_Toc455038315)

# 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 (RC2)/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) nécessite un ordinateur sous Windows 7 ou ultérieur.

## 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

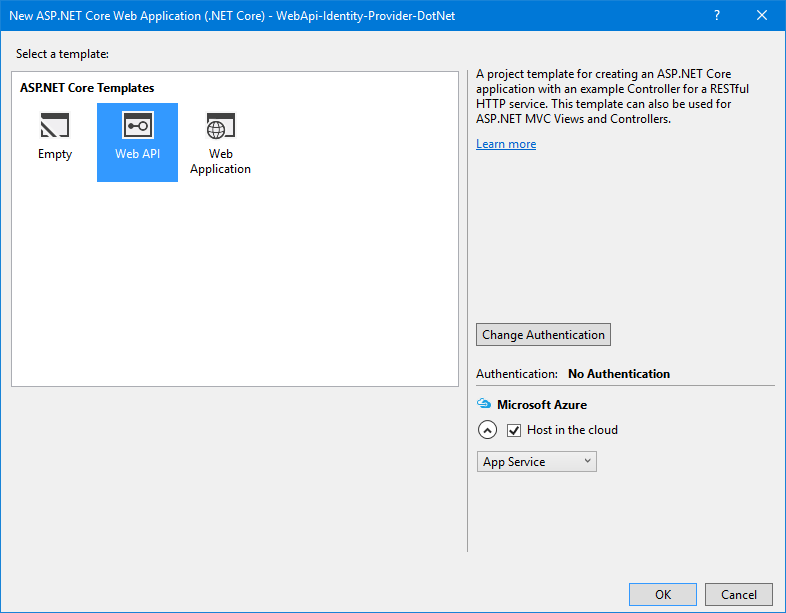
Cette section décrit comment créer un canevas de fournisseur d’identité FranceConnect depuis Visual Studio 2015 en partant de la création d’un projet générique.

Si vous le souhaitez, vous pouvez utiliser directement le projet complet terminé et préconfiguré proposé dans le cadre du kit de démarrage FranceConnect : [franceconnect-identity-provider-dotnet-webapi-aspnetcore](https://github.com/FranceConnectSamples/franceconnect-identity-provider-dotnet-webapi-aspnetcore)[[8]](#footnote-8).

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

## Créer un fournisseur d’identité

Dans Visual Studio, créez une nouvelle application. Choisissez un projet de type **ASP.NET Core Web Application** parmi les différents gabarits (templates) ASP.NET Core proposés :



Assistant de création de projet

## Ajouter IdentityServer 4

IdentityServer 4 est un fournisseur OpenID Connect sous licence open source compatible avec ASP.NET Core.

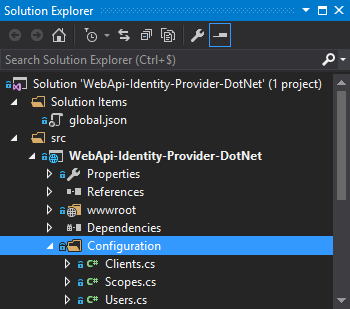
Note Pour de plus amples informations, vous pouvez consulter le projet GitHub [ici](https://github.com/IdentityServer/IdentityServer4).

### Installer le package NuGet

Ajoutez le package NuGet **IdentityServer4** à votre projet.

## Configurer IdentityServer

Ajoutez un dossier **Configuration** au projet :



Créez trois classes :

1. **Clients**. Cette classe va référencer FranceConnect en tant que client OpenID Connect.

public class Clients

{

public const string FranceConnectId = "9f2334a7f81648d3ac850c774450c420";

public const string FranceConnectSecret = "cef97158682942fa98033c6dc90d4984";

public static IEnumerable<Client> Get()

{

return new List<Client>

{

new Client

{

ClientId = FranceConnectId,

ClientName = "FranceConnect",

AllowedGrantTypes = GrantTypes.Code,

RedirectUris = new List<string>

{

"https://fcp.integ01.dev-franceconnect.fr/oidc\_callback"

},

ClientSecrets = new List<Secret>

{

new Secret(FranceConnectSecret.Sha256())

},

AllowedScopes = new List<string>

{

StandardScopes.OpenId.Name,

StandardScopes.Profile.Name,

StandardScopes.Email.Name,

StandardScopes.Address.Name,

StandardScopes.Phone.Name,

"birth",

"preferred\_username"

}

}

};

}

}

1. **Scopes**. Cette classe va référencer tous les scopes disponibles.

public class Scopes

{

public static IEnumerable<Scope> Get()

{

return new List<Scope>

{

StandardScopes.OpenId,

StandardScopes.Profile,

StandardScopes.Email,

StandardScopes.Address,

StandardScopes.Phone,

new Scope

{

Name = "birth",

DisplayName = "Birth",

Description = "Birth",

Type = ScopeType.Identity,

Claims = new List<ScopeClaim> { new ScopeClaim("birthplace"), new ScopeClaim("birthcountry") }

},

new Scope

{

Name = "preferred\_username",

DisplayName = "Preferred username",

Description = "Preferred username",

Type = ScopeType.Identity,

Claims = new List<ScopeClaim> { new ScopeClaim("preferred\_username") }

}

};

}

}

1. **Users**. Cette classe va référencer tous les utilisateurs ayant une identité sur votre fournisseur.

public class Users

{

public static List<InMemoryUser> Get()

{

return new List<InMemoryUser>

{

new InMemoryUser

{

Subject = "6867085672678036750625",

Username = "jean",

Password = "password",

Claims = new Claim[]

{

new Claim(JwtClaimTypes.GivenName, "Jean"),

new Claim(JwtClaimTypes.FamilyName, "Dupont"),

new Claim(JwtClaimTypes.Email, "jean.dupont@contoso.com"),

new Claim(JwtClaimTypes.BirthDate, "1993-06-16"),

new Claim(JwtClaimTypes.Gender, "male"),

new Claim("birthplace", "75056"),

new Claim("birthcountry", "99100"),

new Claim(JwtClaimTypes.PreferredUserName, "Jean Dupont")

}

}

};

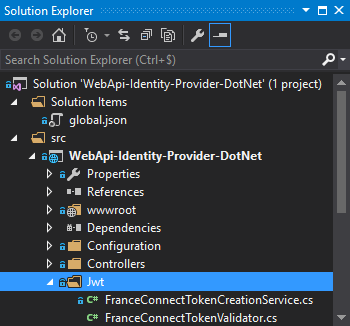
}

}

### Modifier l’algorithme de signature du JWT

IdentityServer signe par défaut le jeton au format JWT (JSON Web Token) avec l’algorithme RS256. Or, pour l’instant, comme cela est indiqué dans la documentation, la plateforme FranceConnect ne supporte que l’algorithme HS256 de type HMAC. Pour remédier à cela, procédez comme suit.

Ajoutez un dossier Jwt :



Créez deux classes :

1. **FranceConnectTokenCreationService** :

public class FranceConnectTokenCreationService : ITokenCreationService

{

public async Task<string> CreateTokenAsync(Token token)

{

var header = await CreateHeaderAsync(token);

var payload = await CreatePayloadAsync(token);

return await SignAsync(new JwtSecurityToken(header, payload));

}

protected virtual Task<JwtHeader> CreateHeaderAsync(Token token)

{

JwtHeader header = new JwtHeader(new SigningCredentials(new SymmetricSecurityKey(Encoding.UTF8.GetBytes(Clients.FranceConnectSecret)), "HS256"));

return Task.FromResult(header);

}

protected virtual Task<JwtPayload> CreatePayloadAsync(Token token)

{

var payload = new JwtPayload(

token.Issuer,

token.Audience,

null,

DateTime.UtcNow,

DateTime.UtcNow.AddSeconds(token.Lifetime));

var amrClaims = token.Claims.Where(x => x.Type == JwtClaimTypes.AuthenticationMethod);

var jsonClaims = token.Claims.Where(x => x.ValueType == Constants.ClaimValueTypes.Json);

var normalClaims = token.Claims.Except(amrClaims).Except(jsonClaims);

payload.AddClaims(normalClaims);

// deal with amr

var amrValues = amrClaims.Select(x => x.Value).Distinct().ToArray();

if (amrValues.Any())

{

payload.Add(JwtClaimTypes.AuthenticationMethod, amrValues);

}

// deal with json types

// calling ToArray() to trigger JSON parsing once and so later

// collection identity comparisons work for the anonymous type

var jsonTokens = jsonClaims.Select(x => new { x.Type, JsonValue = JRaw.Parse(x.Value) }).ToArray();

var jsonObjects = jsonTokens.Where(x => x.JsonValue.Type == JTokenType.Object).ToArray();

var jsonObjectGroups = jsonObjects.GroupBy(x => x.Type).ToArray();

foreach (var group in jsonObjectGroups)

{

if (payload.ContainsKey(group.Key))

{

throw new Exception(String.Format("Can't add two claims where one is a JSON object and the other is not

a JSON object ({0})", group.Key));

}

if (group.Skip(1).Any())

{

// add as array

payload.Add(group.Key, group.Select(x => x.JsonValue).ToArray());

}

else

{

// add just one

payload.Add(group.Key, group.First().JsonValue);

}

}

var jsonArrays = jsonTokens.Where(x => x.JsonValue.Type == JTokenType.Array).ToArray();

var jsonArrayGroups = jsonArrays.GroupBy(x => x.Type).ToArray();

foreach (var group in jsonArrayGroups)

{

if (payload.ContainsKey(group.Key))

{

throw new Exception(String.Format("Can't add two claims where one is a JSON array and the other is not a JSON array ({0})", group.Key));

}

List<JToken> newArr = new List<JToken>();

foreach (var arrays in group)

{

var arr = (JArray)arrays.JsonValue;

newArr.AddRange(arr);

}

// add just one array for the group/key/claim type

payload.Add(group.Key, newArr.ToArray());

}

var unsupportedJsonTokens = jsonTokens.Except(jsonObjects).Except(jsonArrays);

var unsupportedJsonClaimTypes = unsupportedJsonTokens.Select(x => x.Type).Distinct();

if (unsupportedJsonClaimTypes.Any())

{

throw new Exception(String.Format("Unsupported JSON type for claim types: {0}", unsupportedJsonClaimTypes.Aggregate((x, y) => x + ", " + y)));

}

return Task.FromResult(payload);

}

protected virtual Task<string> SignAsync(JwtSecurityToken jwt)

{

var handler = new JwtSecurityTokenHandler();

return Task.FromResult(handler.WriteToken(jwt));

}

}

1. **FranceConnectTokenValidator** :

[EditorBrowsable(EditorBrowsableState.Never)]

public class FranceConnectTokenValidator : ITokenValidator

{

private readonly ILogger \_logger;

private readonly IdentityServerOptions \_options;

private readonly IdentityServerContext \_context;

private readonly ITokenHandleStore \_tokenHandles;

private readonly ICustomTokenValidator \_customValidator;

private readonly IClientStore \_clients;

private readonly IEnumerable<IValidationKeysStore> \_keys;

public FranceConnectTokenValidator(IdentityServerOptions options, IdentityServerContext context, IClientStore clients, ITokenHandleStore tokenHandles, ICustomTokenValidator customValidator, IEnumerable<IValidationKeysStore> keys, ILogger<TokenValidator> logger)

{

\_options = options;

\_context = context;

\_clients = clients;

\_tokenHandles = tokenHandles;

\_customValidator = customValidator;

\_keys = keys;

\_logger = logger;

}

public virtual async Task<TokenValidationResult> ValidateIdentityTokenAsync(string token, string clientId = null, bool validateLifetime = true)

{

\_logger.LogTrace("Start identity token validation");

if (token.Length > \_options.InputLengthRestrictions.Jwt)

{

\_logger.LogError("JWT too long");

return Invalid(OidcConstants.ProtectedResourceErrors.InvalidToken);

}

if (string.IsNullOrWhiteSpace(clientId))

{

clientId = GetClientIdFromJwt(token);

if (string.IsNullOrWhiteSpace(clientId))

{

\_logger.LogError("No clientId supplied, can't find id in identity token.");

return Invalid(OidcConstants.ProtectedResourceErrors.InvalidToken);

}

}

var client = await \_clients.FindClientByIdAsync(clientId);

if (client == null)

{

\_logger.LogError("Unknown or diabled client.");

return Invalid(OidcConstants.ProtectedResourceErrors.InvalidToken);

}

var keys = new List<SymmetricSecurityKey> { new SymmetricSecurityKey(Encoding.UTF8.GetBytes(Clients.FranceConnectSecret)) };

var result = await ValidateJwtAsync(token, clientId, keys, validateLifetime);

result.Client = client;

if (result.IsError)

{

\_logger.LogError("Error validating JWT");

return result;

}

var customResult = await \_customValidator.ValidateIdentityTokenAsync(result);

if (customResult.IsError)

{

\_logger.LogError("Custom validator failed: " + (customResult.Error ?? "unknown"));

return customResult;

}

\_logger.LogInformation("Token validation succeded");

return customResult;

}

public virtual async Task<TokenValidationResult> ValidateAccessTokenAsync(string token, string expectedScope = null)

{

\_logger.LogTrace("Start access token validation");

TokenValidationResult result;

if (token.Contains("."))

{

if (token.Length > \_options.InputLengthRestrictions.Jwt)

{

\_logger.LogError("JWT too long");

return new TokenValidationResult

{

IsError = true,

Error = OidcConstants.ProtectedResourceErrors.InvalidToken,

ErrorDescription = "Token too long"

};

}

result = await ValidateJwtAsync(

token,

string.Format(Constants.AccessTokenAudience, \_context.GetIssuerUri().EnsureTrailingSlash()),

new List<SymmetricSecurityKey> { new SymmetricSecurityKey(Encoding.UTF8.GetBytes(Clients.FranceConnectSecret)) });

}

else

{

if (token.Length > \_options.InputLengthRestrictions.TokenHandle)

{

\_logger.LogError("token handle too long");

return new TokenValidationResult

{

IsError = true,

Error = OidcConstants.ProtectedResourceErrors.InvalidToken,

ErrorDescription = "Token too long"

};

}

result = await ValidateReferenceAccessTokenAsync(token);

}

if (result.IsError)

{

return result;

}

if (!string.IsNullOrWhiteSpace(expectedScope))

{

var scope = result.Claims.FirstOrDefault(c => c.Type == JwtClaimTypes.Scope && c.Value == expectedScope);

if (scope == null)

{

\_logger.LogError(string.Format("Checking for expected scope {0} failed", expectedScope));

return Invalid(OidcConstants.ProtectedResourceErrors.InsufficientScope);

}

}

var customResult = await \_customValidator.ValidateAccessTokenAsync(result);

if (customResult.IsError)

{

\_logger.LogError("Custom validator failed: " + (customResult.Error ?? "unknown"));

return customResult;

}

\_logger.LogInformation("Token validation Succeded");

return customResult;

}

private async Task<TokenValidationResult> ValidateJwtAsync(string jwt, string audience, IEnumerable<SymmetricSecurityKey> symmetricKeys, bool validateLifetime = true)

{

var handler = new JwtSecurityTokenHandler();

handler.InboundClaimTypeMap.Clear();

var parameters = new TokenValidationParameters

{

ValidIssuer = \_context.GetIssuerUri(),

IssuerSigningKeys = symmetricKeys,

ValidateLifetime = validateLifetime,

ValidAudience = audience

};

try

{

SecurityToken jwtToken;

var id = handler.ValidateToken(jwt, parameters, out jwtToken);

// load the client that belongs to the client\_id claim

Client client = null;

var clientId = id.FindFirst(JwtClaimTypes.ClientId);

if (clientId != null)

{

client = await \_clients.FindClientByIdAsync(clientId.Value);

if (client == null)

{

throw new InvalidOperationException("Client does not exist anymore.");

}

}

return new TokenValidationResult

{

IsError = false,

Claims = id.Claims,

Client = client,

Jwt = jwt

};

}

catch (Exception ex)

{

\_logger.LogError("JWT token validation error", ex);

return Invalid(OidcConstants.ProtectedResourceErrors.InvalidToken);

}

}

private async Task<TokenValidationResult> ValidateReferenceAccessTokenAsync(string tokenHandle)

{

var token = await \_tokenHandles.GetAsync(tokenHandle);

if (token == null)

{

\_logger.LogError("Token handle not found");

return Invalid(OidcConstants.ProtectedResourceErrors.InvalidToken);

}

if (token.Type != OidcConstants.TokenTypes.AccessToken)

{

\_logger.LogError("Token handle does not resolve to an access token - but instead to: " + token.Type);

await \_tokenHandles.RemoveAsync(tokenHandle);

return Invalid(OidcConstants.ProtectedResourceErrors.InvalidToken);

}

if (DateTimeOffset.UtcNow >= token.CreationTime.AddSeconds(token.Lifetime))

{

\_logger.LogError("Token expired.");

await \_tokenHandles.RemoveAsync(tokenHandle);

return Invalid(OidcConstants.ProtectedResourceErrors.ExpiredToken);

}

return new TokenValidationResult

{

IsError = false,

Client = token.Client,

Claims = ReferenceTokenToClaims(token),

ReferenceToken = token,

ReferenceTokenId = tokenHandle

};

}

private IEnumerable<Claim> ReferenceTokenToClaims(Token token)

{

var claims = new List<Claim>

{

new Claim(JwtClaimTypes.Audience, token.Audience),

new Claim(JwtClaimTypes.Issuer, token.Issuer),

new Claim(JwtClaimTypes.NotBefore, token.CreationTime.ToEpochTime().ToString()),

new Claim(JwtClaimTypes.Expiration, token.CreationTime.AddSeconds(token.Lifetime).ToEpochTime().ToString())

};

claims.AddRange(token.Claims);

return claims;

}

private string GetClientIdFromJwt(string token)

{

try

{

var jwt = new JwtSecurityToken(token);

var clientId = jwt.Audiences.FirstOrDefault();

return clientId;

}

catch (Exception ex)

{

\_logger.LogError("Malformed JWT token", ex);

return null;

}

}

private TokenValidationResult Invalid(string error)

{

return new TokenValidationResult

{

IsError = true,

Error = error

};

}

}

public static class StringExtensions

{

public static string EnsureTrailingSlash(this string url)

{

if (!url.EndsWith("/"))

{

return url + "/";

}

return url;

}

}

### Configurer l’application pour IdentityServer

Modifiez la classe *Startup* pour utiliser IdentityServer :

public class Startup

{

private readonly IHostingEnvironment \_environment;

public Startup(IHostingEnvironment env)

{

\_environment = env;

var builder = new ConfigurationBuilder()

.SetBasePath(env.ContentRootPath)

.AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)

.AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true)

.AddEnvironmentVariables();

Configuration = builder.Build();

}

public IConfigurationRoot Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();

var builder = services.AddIdentityServer();

// remove default services

builder.Services.Remove(builder.Services.SingleOrDefault(s => s.ImplementationType == typeof(DefaultTokenSigningService)));

builder.Services.Remove(builder.Services.SingleOrDefault(s => s.ImplementationType == typeof(TokenValidator)));

// add FranceConnect services

builder.Services.TryAddTransient<ITokenCreationService, FranceConnectTokenCreationService>();

builder.Services.TryAddTransient<ITokenValidator, FranceConnectTokenValidator>();

builder.AddInMemoryClients(Clients.Get());

builder.AddInMemoryScopes(Scopes.Get());

builder.AddInMemoryUsers(Users.Get());

// Add framework services.

services.AddMvc();

services.AddTransient<SignInService>();

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)

{

loggerFactory.AddConsole(Configuration.GetSection("Logging"));

loggerFactory.AddDebug();

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseIdentityServer();

app.UseStaticFiles();

app.UseMvcWithDefaultRoute();

}

}

## Créer l’UI

Ajoutez les 4 dossiers suivants dans l’application :

* **Controllers**.
* **Services**.
* **ViewModels**.
* **Views**.

### Configurer l’application pour les vues

Tout d’abord, il faut ajouter des composant au projet pour qu’il puisse retourner des vues. Commencez par ajoutez les paquets :

|  |  |
| --- | --- |
| **Package** | **Version** |
| *Microsoft.AspNetCore.Diagnostics* | 1.0.0 |
| *Microsoft.AspNetCore.Mvc.TagHelpers* | 1.0.0 |

Ajoutez un fichier de configuration *Bower* :

*.bowerrc* :

{

"directory": "wwwroot/lib"

}

*bower.json* :

{

"name": "ASP.NET",

"private": true,

"dependencies": {

"bootstrap": "3.3.6",

"jquery": "2.1.4",

"jquery-validation": "1.14.0",

"jquery-validation-unobtrusive": "3.2.4"

}

}

Note Pour de plus amples informations, vous pouvez consulter [la documentation](https://docs.asp.net/en/latest/client-side/bower.html).

Ainsi qu’un fichier de configuration NPM :

*package.json* :

{

"version": "1.0.0",

"name": "ASP.NET",

"devDependencies": {

"gulp": "3.8.11",

"gulp-concat": "2.5.2",

"gulp-cssmin": "0.1.7",

"gulp-uglify": "1.2.0",

"rimraf": "2.2.8"

}

}

Et un scripts *gulpfile.js* pour rapetisser les fichiers :

/// <binding Clean='clean' />

"use strict";

var gulp = require("gulp"),

rimraf = require("rimraf"),

concat = require("gulp-concat"),

cssmin = require("gulp-cssmin"),

uglify = require("gulp-uglify");

var paths = {

webroot: "./wwwroot/"

};

paths.js = paths.webroot + "js/\*\*/\*.js";

paths.minJs = paths.webroot + "js/\*\*/\*.min.js";

paths.css = paths.webroot + "css/\*\*/\*.css";

paths.minCss = paths.webroot + "css/\*\*/\*.min.css";

paths.concatJsDest = paths.webroot + "js/site.min.js";

paths.concatCssDest = paths.webroot + "css/site.min.css";

gulp.task("clean:js", function (cb) {

rimraf(paths.concatJsDest, cb);

});

gulp.task("clean:css", function (cb) {

rimraf(paths.concatCssDest, cb);

});

gulp.task("clean", ["clean:js", "clean:css"]);

gulp.task("min:js", function () {

return gulp.src([paths.js, "!" + paths.minJs], { base: "." })

.pipe(concat(paths.concatJsDest))

.pipe(uglify())

.pipe(gulp.dest("."));

});

gulp.task("min:css", function () {

return gulp.src([paths.css, "!" + paths.minCss])

.pipe(concat(paths.concatCssDest))

.pipe(cssmin())

.pipe(gulp.dest("."));

});

gulp.task("min", ["min:js", "min:css"]);

Note Pour de plus amples informations, vous pouvez consulter [la documentation](https://docs.asp.net/en/latest/client-side/using-gulp.html).

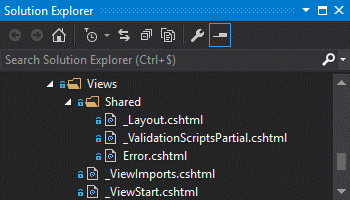
Modifiez également le fichier *project.json* pour la publication de l’API :

"scripts": {

"prepublish": [ "npm install", "bower install", "gulp clean", "gulp min" ]

}

Enfin, ajoutez les vues communes selon cette disposition :



Organisation des vues

*\_ViewStart.cshtml* :

@{

Layout = "\_Layout";

}

*\_ViewImports.cshtml* :

@addTagHelper "\*, Microsoft.AspNetCore.Mvc.TagHelpers"

*\_Layout.cshtml* :

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Fournisseur d'identité ASP.NET Core</title>

<link rel="icon" type="image/x-icon" href="~/favicon.ico" />

<link rel="shortcut icon" type="image/x-icon" href="~/favicon.ico" />

<link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />

<link rel="stylesheet" href="~/css/site.css" />

</head>

<body>

<div class="navbar navbar-inverse navbar-fixed-top">

<div class="container">

<div class="navbar-header">

<span class="navbar-brand">Fournisseur d'identité ASP.NET Core</span>

</div>

</div>

</div>

<div class="container body-content">

@RenderBody()

<hr />

<footer>

<p>&copy; 2016 - Microsoft France</p>

</footer>

</div>

<script src="~/lib/jquery/dist/jquery.js"></script>

<script src="~/lib/bootstrap/dist/js/bootstrap.js"></script>

@RenderSection("scripts", required: false)

</body>

</html>

*\_ValidationScriptsPartial.cshtml* :

<environment names="Development">

<script src="~/lib/jquery-validation/dist/jquery.validate.js"></script>

<script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.js"></script>

</environment>

<environment names="Staging,Production">

<script src="https://ajax.aspnetcdn.com/ajax/jquery.validate/1.14.0/jquery.validate.min.js"

asp-fallback-src="~/lib/jquery-validation/dist/jquery.validate.min.js"

asp-fallback-test="window.jQuery && window.jQuery.validator">

</script>

<script src="https://ajax.aspnetcdn.com/ajax/mvc/5.2.3/jquery.validate.unobtrusive.min.js"

asp-fallback-src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js"

asp-fallback-test="window.jQuery && window.jQuery.validator && window.jQuery.validator.unobtrusive">

</script>

</environment>

*Error.cshtml* :

@{

ViewData["Title"] = "Erreur";

}

<h1 class="text-danger">Erreur.</h1>

<h2 class="text-danger">Une erreur s'est produite lors du traitement de votre demande.</h2>

### Ajouter la vue Login

Tout d’abord, créez une classe **SignInService** dans le dossier **Services** :

public class SignInService

{

private readonly List<InMemoryUser> \_users;

public SignInService(List<InMemoryUser> users)

{

\_users = users;

}

public bool ValidateCredentials(string username, string password)

{

var user = FindByUsername(username);

if (user != null)

{

return user.Password.Equals(password);

}

return false;

}

public InMemoryUser FindByUsername(string username)

{

return \_users.FirstOrDefault(x => x.Username.Equals(username, StringComparison.OrdinalIgnoreCase));

}

}

Ensuite, ajoutez un contrôleur *LoginController* :

public class LoginController : Controller

{

private readonly SignInService \_signInService;

private readonly SignInInteraction \_signInInteraction;

public LoginController(

SignInService signInService,

SignInInteraction signInInteraction)

{

\_signInService = signInService;

\_signInInteraction = signInInteraction;

}

[HttpGet(Constants.RoutePaths.Login, Name = "Login")]

public async Task<IActionResult> Index(string id)

{

var vm = new LoginViewModel();

if (!string.IsNullOrEmpty(id))

{

var request = await \_signInInteraction.GetRequestAsync(id);

if (request != null)

{

vm.Username = request.LoginHint;

vm.SignInId = id;

}

}

return View(vm);

}

[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.SignInId != null)

{

return new SignInResult(model.SignInId);

}

return Redirect("~/");

}

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

}

return View(model);

}

private async Task IssueCookie(InMemoryUser user, string identityProvider, string authenticationType)

{

var name = user.Claims.Where(x => x.Type == JwtClaimTypes.Name).Select(x => x.Value).FirstOrDefault() ?? user.Username;

var claims = new Claim[]

{

new Claim(JwtClaimTypes.Subject, user.Subject),

new Claim(JwtClaimTypes.Name, name),

new Claim(JwtClaimTypes.IdentityProvider, identityProvider),

new Claim(JwtClaimTypes.AuthenticationTime, DateTime.UtcNow.ToEpochTime().ToString()),

};

var claimIdentity = new ClaimsIdentity(claims, authenticationType, JwtClaimTypes.Name, JwtClaimTypes.Role);

var claimPricipal = new ClaimsPrincipal(claimIdentity);

await HttpContext.Authentication.SignInAsync(Constants.PrimaryAuthenticationType, claimPricipal);

}

}

public class SignInResult : IActionResult

{

private readonly string \_requestId;

public SignInResult(string requestId)

{

\_requestId = requestId;

}

public async Task ExecuteResultAsync(ActionContext context)

{

var interaction = context.HttpContext.RequestServices.GetRequiredService<SignInInteraction>();

await interaction.ProcessResponseAsync(\_requestId, new IdentityServer4.Core.Models.SignInResponse());

}

}

Puis créez 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 string ErrorMessage { get; set; }

}

Enfin, créez la vue correspondante :

@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">

<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>

<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>

### Ajouter la vue Consentement

Une fois que l’utilisateur s’est authentifié, il doit consentir à ce que le fournisseur d’identité retourne ses informations à FranceConnect

Dans un premier temps, créez le contrôleur **ConsentController**:

public class ConsentController : Controller

{

private readonly ILogger<ConsentController> \_logger;

private readonly IClientStore \_clientStore;

private readonly ConsentInteraction \_consentInteraction;

private readonly IScopeStore \_scopeStore;

public ConsentController(

ILogger<ConsentController> logger,

ConsentInteraction consentInteraction,

IClientStore clientStore,

IScopeStore scopeStore)

{

\_logger = logger;

\_consentInteraction = consentInteraction;

\_clientStore = clientStore;

\_scopeStore = scopeStore;

}

[HttpGet(Constants.RoutePaths.Consent, Name = "Consent")]

public async Task<IActionResult> Index(string id)

{

if (id != null)

{

var request = await \_consentInteraction.GetRequestAsync(id);

if (request != null)

{

var client = await \_clientStore.FindClientByIdAsync(request.ClientId);

if (client != null)

{

var scopes = await \_scopeStore.FindScopesAsync(request.ScopesRequested);

if (scopes != null && scopes.Any())

{

return View(new ConsentViewModel

{

ClientName = client.ClientName,

ConsentId = id,

Scopes = scopes.Select(s => s.Name)

});

}

else

{

\_logger.LogError("No scopes matching: {0}", request.ScopesRequested.Aggregate((x, y) => x + ", " + y));

}

}

else

{

\_logger.LogError("Invalid client id: {0}", request.ClientId);

}

}

else

{

\_logger.LogError("No consent request matching id: {0}", id);

}

}

else

{

\_logger.LogError("No id passed");

}

return View("Error");

}

[HttpPost(Constants.RoutePaths.Consent)]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Index(string button, string id)

{

if (id != null)

{

var request = await \_consentInteraction.GetRequestAsync(id);

if (request != null)

{

var scopes = await \_scopeStore.FindScopesAsync(request.ScopesRequested);

if (scopes != null && scopes.Any())

{

if (button == "no")

{

return new ConsentResult(id, ConsentResponse.Denied);

}

else if (button == "yes")

{

return new ConsentResult(id, new ConsentResponse

{

ScopesConsented = scopes.Select(s => s.Name)

});

}

else

{

ModelState.AddModelError("", "Invalid Selection");

}

var client = await \_clientStore.FindClientByIdAsync(request.ClientId);

if (client != null)

{

return View(new ConsentViewModel

{

ClientName = client.ClientName,

ConsentId = id,

Scopes = scopes.Select(s => s.Name)

});

}

else

{

\_logger.LogError("Invalid client id: {0}", request.ClientId);

}

}

else

{

\_logger.LogError("No scopes matching: {0}", request.ScopesRequested.Aggregate((x, y) => x + ", " + y));

}

}

else

{

\_logger.LogError("No consent request matching id: {0}", id);

}

}

else

{

\_logger.LogError("No id passed");

}

return View("Error");

}

async Task<IActionResult> BuildConsentResponse(string id, string[] scopesConsented, bool rememberConsent)

{

if (id != null)

{

var request = await \_consentInteraction.GetRequestAsync(id);

}

return View("Error");

}

}

public class ConsentResult : IActionResult

{

private readonly string \_requestId;

private readonly ConsentResponse \_response;

public ConsentResult(string requestId, ConsentResponse response)

{

\_requestId = requestId;

\_response = response;

}

public async Task ExecuteResultAsync(ActionContext context)

{

var interaction = context.HttpContext.RequestServices.GetRequiredService<ConsentInteraction>();

await interaction.ProcessResponseAsync(\_requestId, \_response);

}

}

Puis, ajoutez le modèle de vue *ConsentViewModel* :

public class ConsentViewModel

{

public string ConsentId { get; set; }

public string ClientName { get; set; }

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

}

Enfin, créez la vue correspondante :

@model WebApi\_Identity\_Provider\_DotNet.ViewModels.Consent.ConsentViewModel

<h2>Consentez-vous à donner ces informations à @Model.ClientName</h2>

<div class="row">

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

<section>

<form asp-route="Consent" asp-route-id="@Model.ConsentId" class="form-horizontal">

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

<div class="form-group">

<label class="col-md-3 control-label">Informations</label>

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

<ul class="list-group">

@foreach (var scope in Model.Scopes)

{

<li class="list-group-item">@scope</li>

}

</ul>

</div>

</div>

<div class="form-group">

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

<button name="button" value="yes" class="btn btn-primary" autofocus>Accepter</button>

<button name="button" value="no" class="btn">Refuser</button>

</div>

</div>

</form>

</section>

</div>

</div>

# 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)[[9]](#footnote-9).

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/)[[10]](#footnote-10), [GitHub](https://github.com/FranceConnectSamples)[[11]](#footnote-11) ou [Visual Studio](https://msdn.microsoft.com/vstudio/aa718325.aspx)[[12]](#footnote-12). 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/)[[13]](#footnote-13) 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.

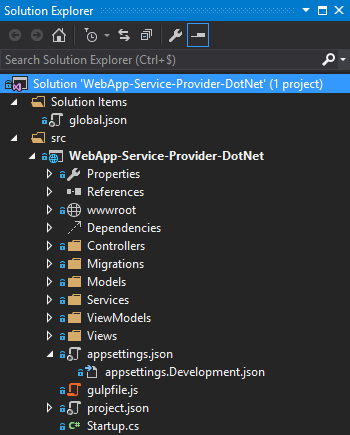
Remarque importante Nous faisons référence au dossier où se situe le code source du canevas de fournisseur d’identité (FI) comme étant le dossier Kit-Demarrage dans la suite de ce document.

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

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.

{

"Data": {

"DefaultConnection": {

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

}

},

"Logging": {

"IncludeScopes": false,

"LogLevel": {

"Default": "Verbose",

"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>",

"DataProviders": [

{

"Name": "<nom du provider>",

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

"Endpoint": "<endpoint de la ressource>"

}

]

}

}

|  |  |
| --- | --- |
| **Section de configuration** | **Description** |
| Data | 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 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

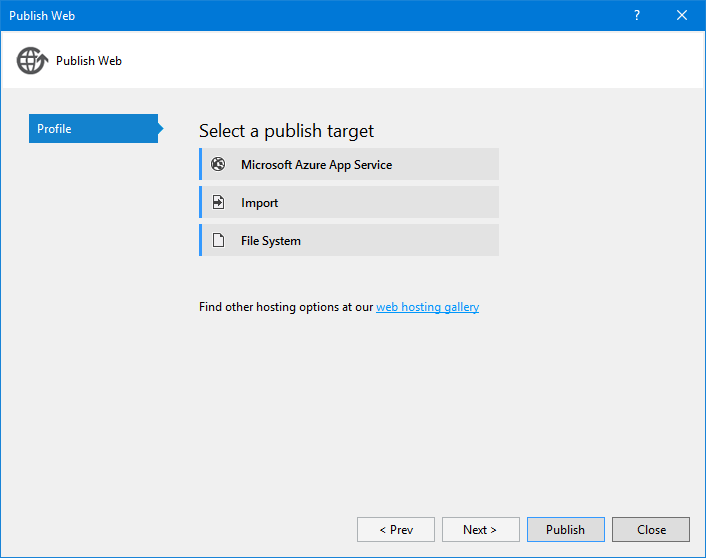
Pour créer l’application web relative à votre fournisseur d’identité dans Microsoft Azure, procédez comme suit :

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 endpoints :
   1. Authorization endpoint : <*FI URL*>/connect/authorize
   2. Token endpoint : <FI URL>/consent/token
   3. Userinfo endpoin : <FI URL>/consent/userinfo

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

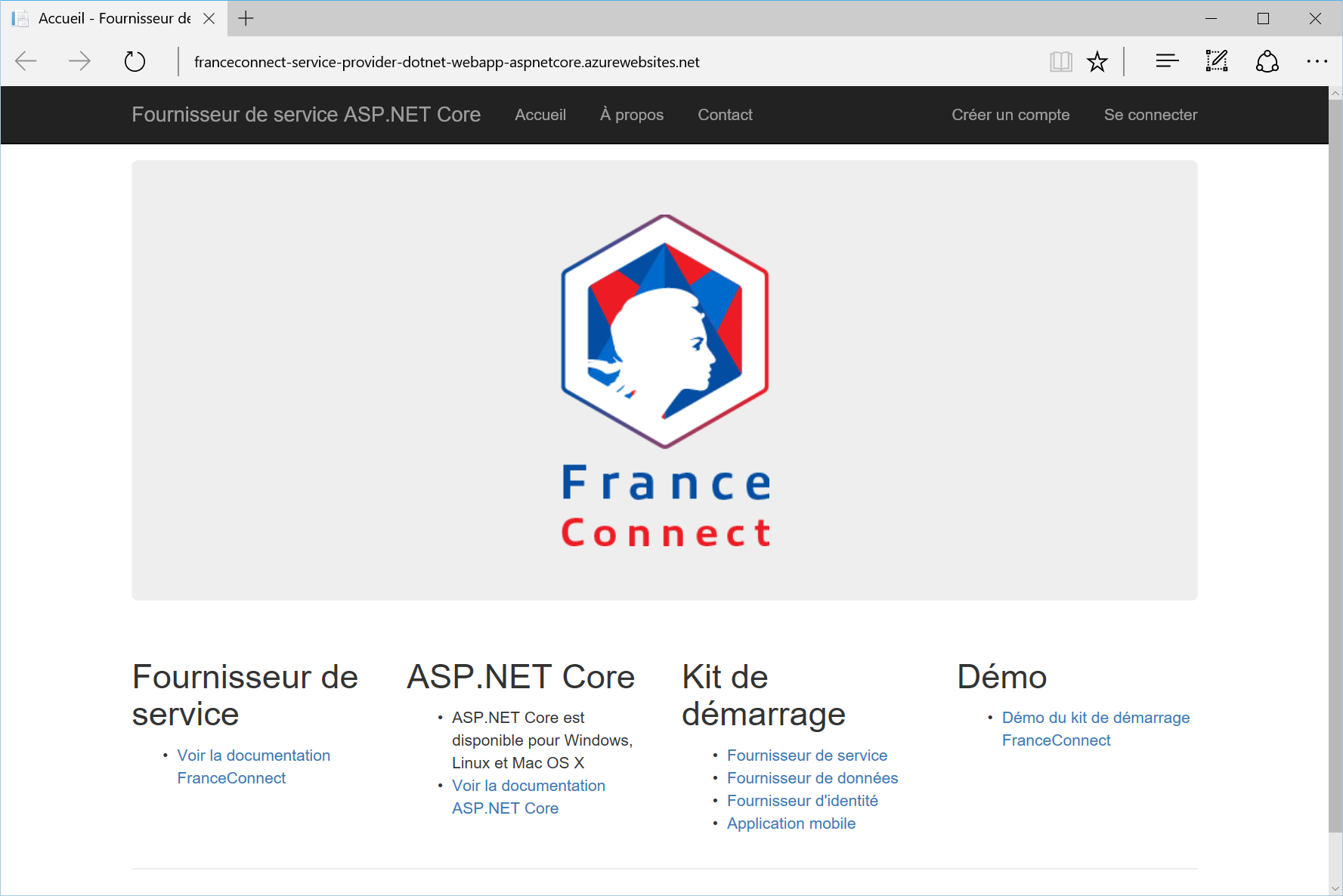
1. Générer un client ID et un client secret pour FranceConnect

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

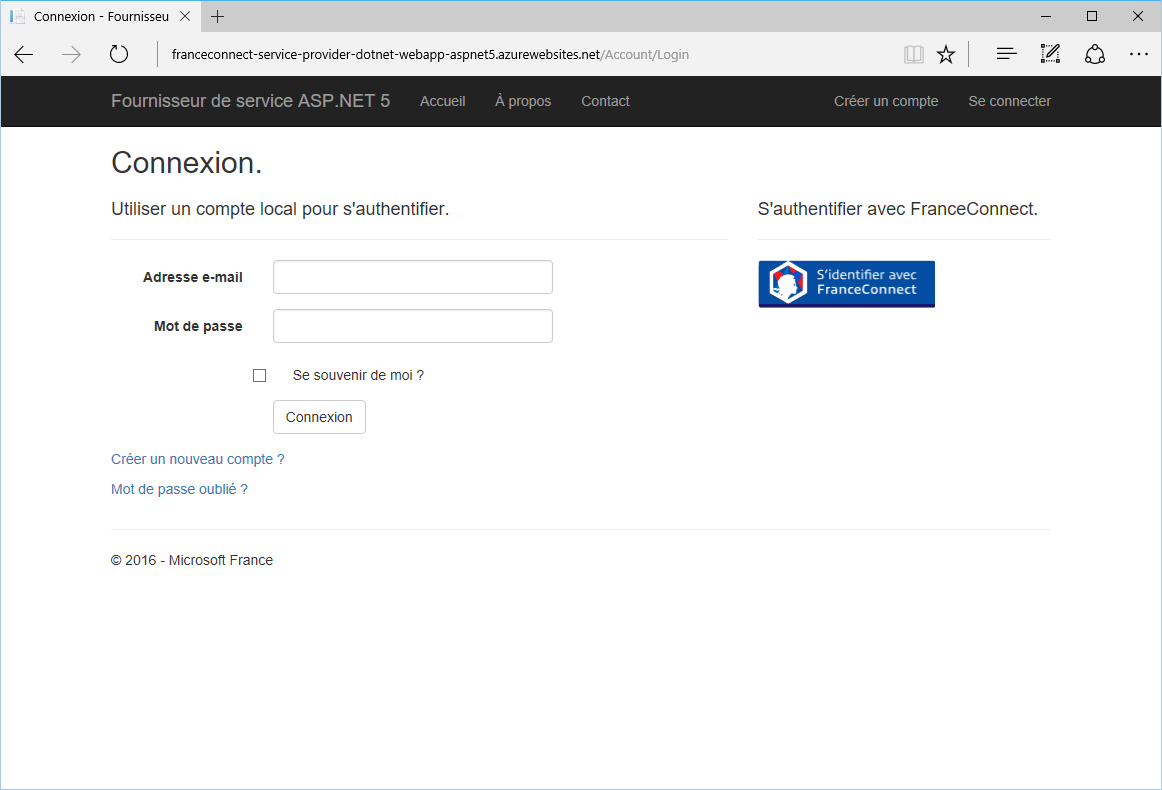
## Se connecter avec le fournisseur d’identité

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

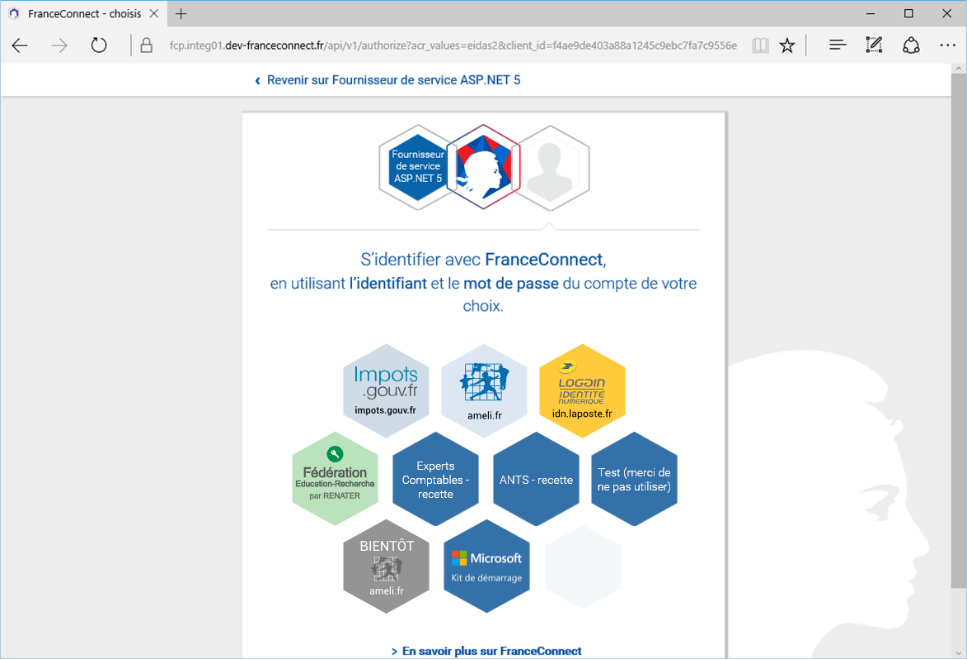
1. Ouvrez une session browser et naviguez vers l’instance de démonstration du canevas de fournisseur de service à l’adresse <http://aka.ms/FranceConnect-FS>.



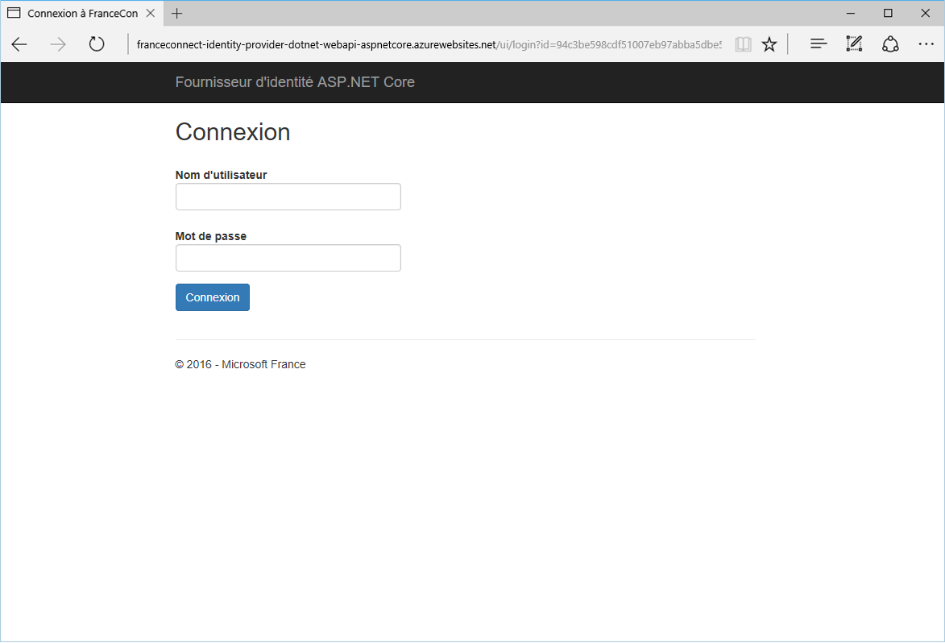
1. Cliquez sur **Se connecter** dans le coin supérieur droit de la fenêtre du browser.



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



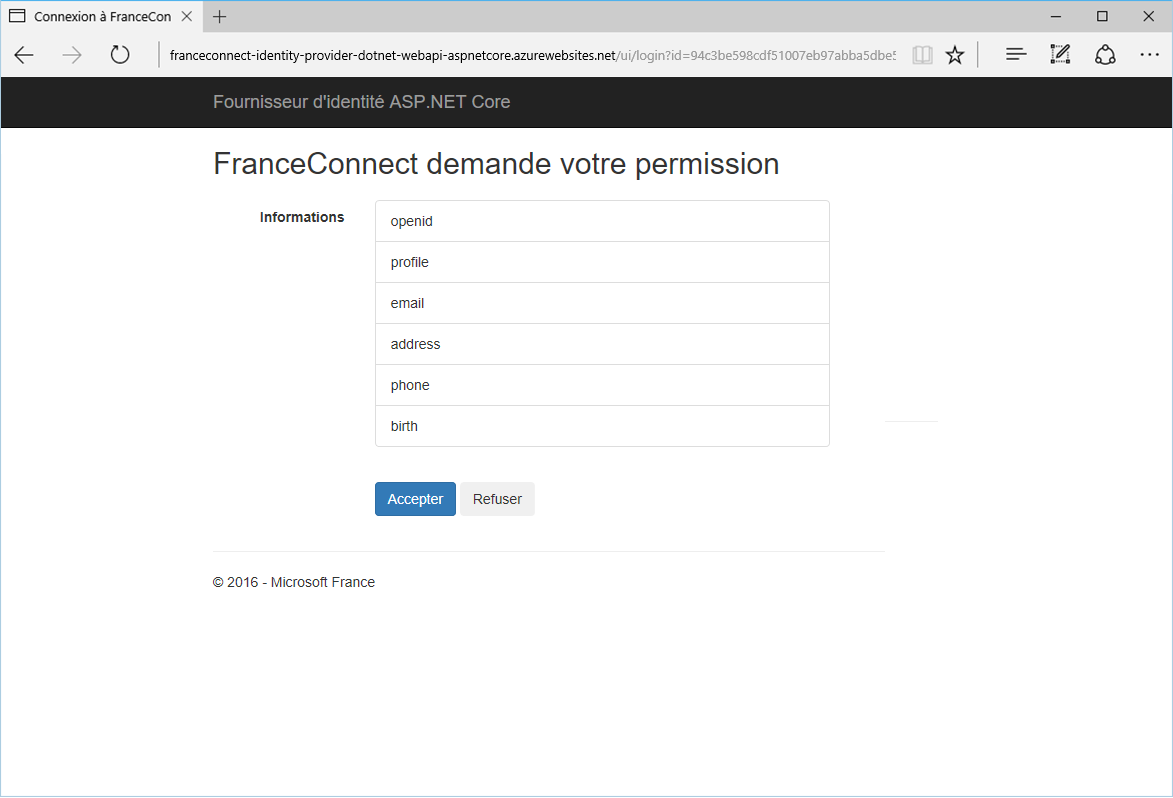
1. Sélectionnez votre canevas de fournisseur d’identité ou le fournisseur d’identité factice **Microsoft Kit de démarrage** pour l’instance de démonstration du canevas de fournisseur d’identité. Vous êtes alors redirigé(e)s vers ce fournisseur d’identité.



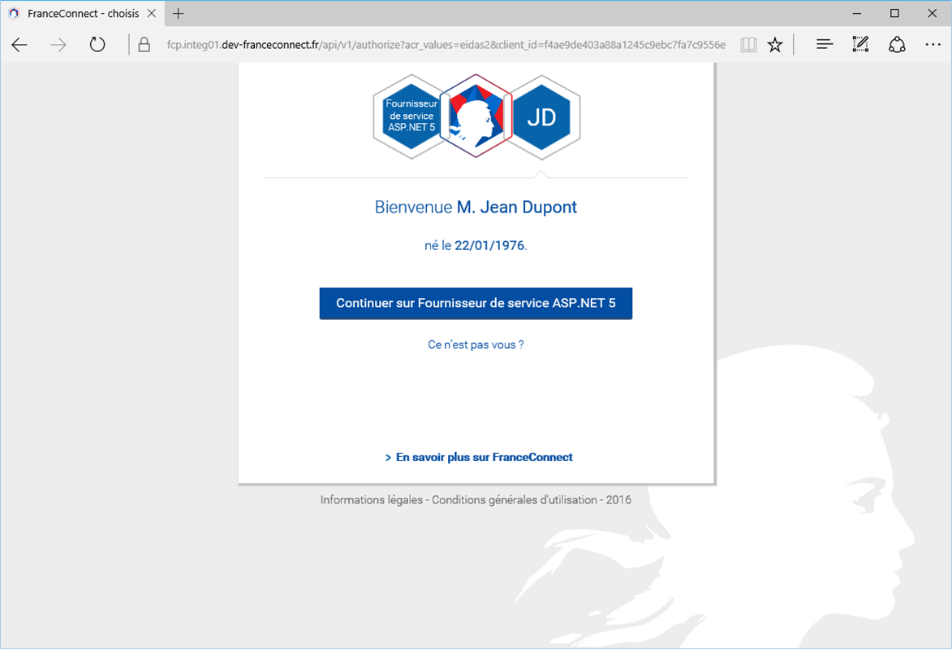
1. Au niveau du fournisseur d’identité, identifiez-vous. Si vous avez sélectionnez l’instance de démonstration du canevas de fournisseur d’identité, identifiez-vous en tant que l’utilisateur Jean Dupont. Précisez pour cela les informations suivantes dans le formulaire qui vous est proposé :

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

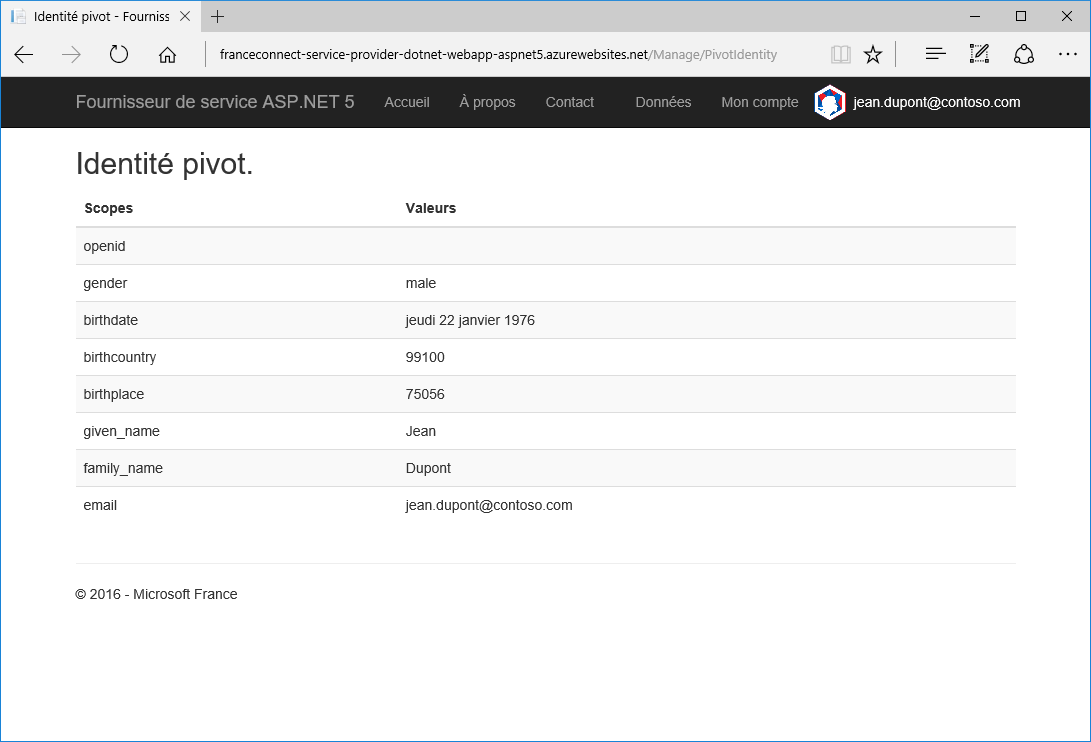
1. Cliquez sur **Connexion**. Une page de consentement s’affiche au niveau du fournisseur d’identité.



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.



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)[[14]](#footnote-14).

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)[[15]](#footnote-15).

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. franceconnect-identity-provider-dotnet-webapi-aspnetcore : https://github.com/FranceConnectSamples/franceconnect-identity-provider-dotnet-webapi-aspnetcore [↑](#footnote-ref-8)
9. franceconnect-identity-provider-dotnet-webapi-aspnetcore : https://github.com/FranceConnectSamples/franceconnect-identity-provider-dotnet-webapi-aspnetcore [↑](#footnote-ref-9)
10. Git : http://git-scm.com/ [↑](#footnote-ref-10)
11. GitHub : https://github.com/FranceConnectSamples [↑](#footnote-ref-11)
12. Visual Studio : https://msdn.microsoft.com/vstudio/aa718325.aspx [↑](#footnote-ref-12)
13. Set up Git : https://help.github.com/articles/set-up-git/ [↑](#footnote-ref-13)
14. FranceConnect : https://doc.integ01.dev-franceconnect.fr/ [↑](#footnote-ref-14)
15. Documentation – Fournisseur d’identité : https://doc.integ01.dev-franceconnect.fr/fournisseur-identite [↑](#footnote-ref-15)