Mettre en œuvre un fournisseur de données 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](#_Toc455082658)

[A propos du kit de démarrage FranceConnect 4](#_Toc455082659)

[Licence 4](#_Toc455082660)

[Commentaires 4](#_Toc455082661)

[Prérequis 5](#_Toc455082662)

[Connaissances et attendu 5](#_Toc455082663)

[Créer un fournisseur de données FranceConnect avec ASP.NET Core 6](#_Toc455082664)

[Créer un fournisseur de données 6](#_Toc455082665)

[Créer un middleware 6](#_Toc455082666)

[Créer un filtre 10](#_Toc455082667)

[Utiliser le package NuGet 11](#_Toc455082668)

[Configurer l'API 11](#_Toc455082669)

[Modifier le fichier de configuration 11](#_Toc455082670)

[Utiliser le middleware *FranceConnect.DataProvider* 12](#_Toc455082671)

[Définir le modèle de donnée 12](#_Toc455082672)

[Appliquer la migration 13](#_Toc455082673)

[Exposer des données 13](#_Toc455082674)

[Créer un formulaire de création de compte 14](#_Toc455082675)

[Configurer le projet pour les vues 14](#_Toc455082676)

[Ajouter le formulaire 18](#_Toc455082677)

[Déployer le fournisseur de données préconfiguré du kit de démarrage 20](#_Toc455082678)

[Obtenir les sources du fournisseur de données 20](#_Toc455082679)

[Télécharger les sources du fournisseur de données 20](#_Toc455082680)

[Cloner les sources du fournisseur de données 21](#_Toc455082681)

[Organisation du code source du canevas de fournisseur de données 22](#_Toc455082682)

[Comprendre l’organisation du projet 22](#_Toc455082683)

[Appréhender les éléments de configuration 22](#_Toc455082684)

[Déployer le projet dans Microsoft Azure 23](#_Toc455082685)

[Créer l’API web dans Microsoft Azure 23](#_Toc455082686)

[Publier le fournisseur de données dans Microsoft Azure 23](#_Toc455082687)

[Tester le fournisseur de donnée 25](#_Toc455082688)

[Créer un compte sur le fournisseur de données 25](#_Toc455082689)

[Obtenir des données 25](#_Toc455082690)

[Annexe A. Références techniques 26](#_Toc455082691)

# A propos du système FranceConnect

La plateforme FranceConnect est un nouveau système d’identification à l’initiative de la Direction interministérielle du numérique et du système d’information et de communication de l’État (DINSIC) visant à faciliter l’accès des usagers aux services numériques de l’administration en ligne en évitant à tout un chacun de devoir créer un nouveau compte lors de l’accès à un nouveau service et donc à se remémorer différents mots de passe pour l’ensemble des services accédés.

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

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

Le système d’identification FranceConnect couvre les rôles suivants :

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

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

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

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

# A propos du kit de démarrage FranceConnect

Le kit de démarrage FranceConnect sous licence libre est une initiative menée par les équipes Microsoft en charge du [Secteur Public](http://www.microsoft.com/france/entreprises/secteur-public/solutions-pour-secteur-public.aspx)[[1]](#footnote-1). Il propose un ensemble d’exemples/accélérateurs facilitant la mise en œuvre des différents rôles prévus par le système d’identification FranceConnect avec les services, produits et technologies Microsoft.



Plus précisément, ce document détaille la mise en œuvre d’un canevas de fournisseur de données (FD) compatible FranceConnect. Il s’appuie pour cela sur la technologie [ASP.NET Core](http://www.dotnetfoundation.org/aspnet-core)[[2]](#footnote-2) mise à disposition sous licence libre par la [fondation .NET](http://www.dotnetfoundation.org/)[[3]](#footnote-3).

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

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

Un accompagnement pas-à-pas vous est proposé pour toutes les étapes du processus de mise en œuvre d’un canevas de fournisseur de données (FD) en partant si vous le souhaitez d’un projet générique ASP.NET Core 1.0. Une solution complète prête à être configurée et déployée vous est également proposée.

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

## Licence

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

## Commentaires

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

## Prérequis

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

## Connaissances et attendu

La mise en œuvre d’un fournisseur de données (FD) avec le kit de démarrage FranceConnect requiert une expérience préalable :

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

Note L’article [Introduction to ASP.NET Core](https://docs.asp.net/en/latest/conceptual-overview/aspnet.html)[[5]](#footnote-5) propose comme son nom l’indique une première introduction à ASP.NET Core (anciennement ASP.NET 5). La page [ASP.NET Core Documentation](https://docs.asp.net/en/latest/)[[6]](#footnote-6) donne par ailleurs accès à l’ensemble de la documentation disponible à date.

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

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

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

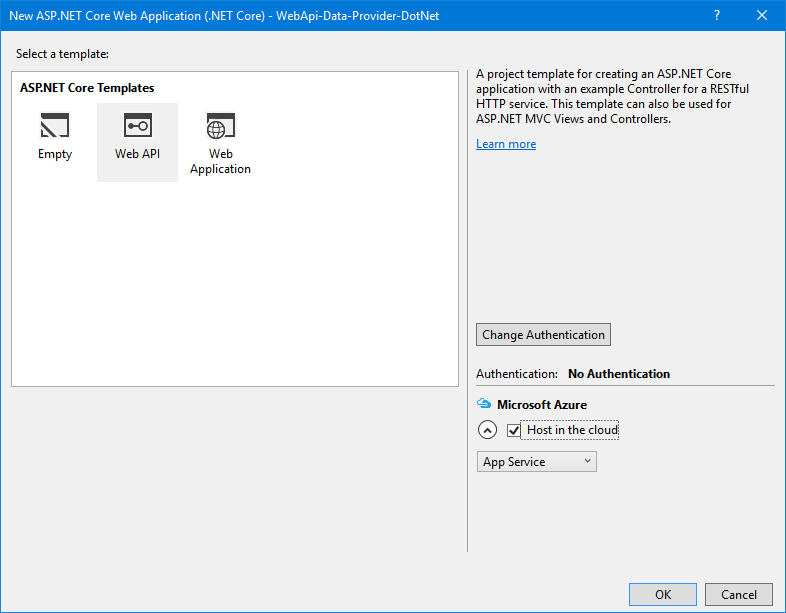
Cette section décrit comment créer un canevas de fournisseur de données (FD) compatible FranceConnect depuis Visual Studio 2015 en partant de la création d’un projet générique.

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

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

## Créer un fournisseur de données

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



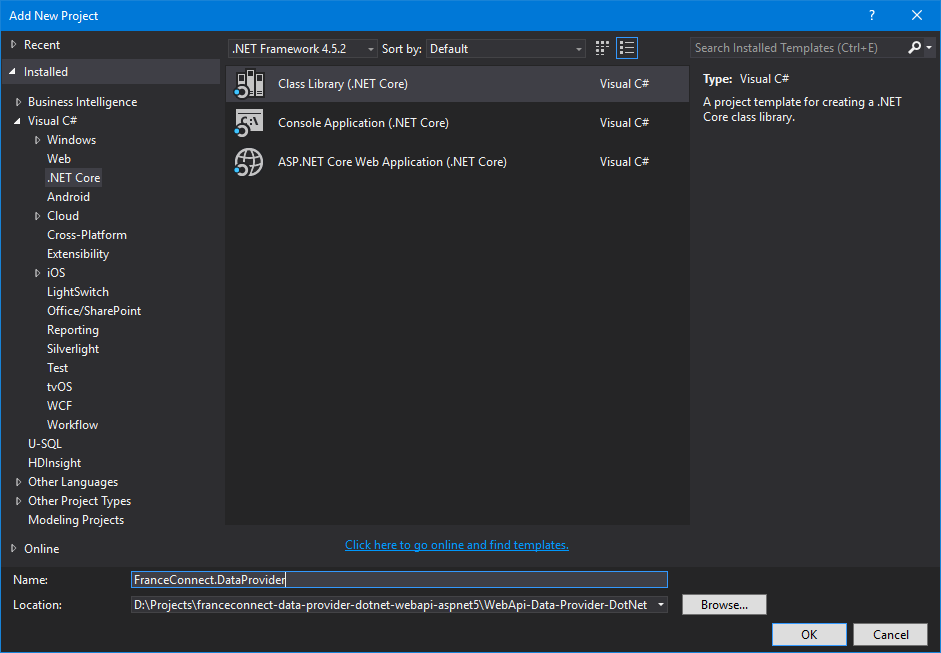
Assistant de création de projet

## Créer un middleware

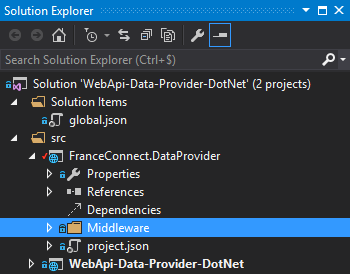
Le fournisseur de données (FD) doit vérifier le jeton d’accès (access token) que le fournisseur de service (FS) lui envoi à chaque requête. Pour simplifier ce traitement, vous allez créer un middleware.

Note Pour de plus amples informations, vous pouvez consulter la documentation [ici](https://docs.asp.net/en/latest/fundamentals/middleware.html)[[8]](#footnote-8).

Ajoutez un nouveau projet à la solution :



Ajoutez un nouveau dossier pour le middleware :



Créez une classe *DataProviderMiddleware* :

public class DataProviderMiddleware  
{  
 private readonly RequestDelegate \_next;

public DataProviderMiddleware(RequestDelegate next, DataProviderOptions options)  
 {  
 if (next == null)  
 {  
 throw new ArgumentNullException(nameof(next));  
 }

if (options == null)  
 {  
 throw new ArgumentNullException(nameof(options));  
 }

if (options.ChecktokenEndpoint == null)  
 {  
 throw new ArgumentNullException(nameof(options.ChecktokenEndpoint));  
 }

\_next = next;  
 Options = options;  
 }

public DataProviderOptions Options { get; set; }

public async Task Invoke(HttpContext context)  
 {  
 if (!context.Request.Path.Value.StartsWith("/api"))  
 {  
 await \_next(context);  
 return;  
 }

string authorization = context.Request.Headers["Authorization"];  
 string token = string.Empty;  
   
 if (string.IsNullOrEmpty(authorization))  
 {  
 await InvalidAuthorizationHeader(context);  
 return;  
 }

if (authorization.StartsWith("Bearer ", StringComparison.OrdinalIgnoreCase))  
 {  
 token = authorization.Substring("Bearer ".Length).Trim();  
 }  
   
 if (string.IsNullOrEmpty(token))  
 {  
 await AccessTokenNotFound(context);  
 return;  
 }

var client = new HttpClient();  
 var httpContent = new  
 {  
 token = token  
 };  
 var response = await client.PostAsync(Options.ChecktokenEndpoint, new StringContent(JsonConvert.SerializeObject(httpContent), Encoding.UTF8, "application/json"));  
 if (response.IsSuccessStatusCode)  
 {  
 var json = response.Content.ReadAsStringAsync().Result;  
 var checktokenResponse = JsonConvert.DeserializeObject<ChecktokenResponse>(json);  
 context.Items["scope"] = checktokenResponse.Scope;  
 context.Items["email"] = checktokenResponse.Identity.Email;  
 }  
 else  
 {  
 await SendFranceConnectError(context, response.StatusCode, response.Content.ReadAsStringAsync().Result);  
 }

await \_next(context);  
 }

#region Helpers

private static async Task InvalidAuthorizationHeader(HttpContext context)  
 {  
 await SendError(context, HttpStatusCode.BadRequest, new Error  
 {  
 error = "invalid\_request",  
 message = "The request does not contain authorization header"  
 });  
 }

private static async Task AccessTokenNotFound(HttpContext context)  
 {  
 await SendError(context, HttpStatusCode.Unauthorized, new Error  
 {  
 error = "invalid\_token",  
 message = "Access token not found"  
 });  
 }

private static async Task SendError(HttpContext context, HttpStatusCode statusCode, Error error)  
 {  
 context.Response.StatusCode = (int)statusCode;  
 context.Response.ContentType = "application/json";  
 await context.Response.WriteAsync(JsonConvert.SerializeObject(error));  
 }

private static async Task SendFranceConnectError(HttpContext context, HttpStatusCode statusCode, string error)  
 {  
 context.Response.StatusCode = (int)statusCode;  
 context.Response.ContentType = "application/json";  
 await context.Response.WriteAsync(error);  
 }

#endregion  
}

Le middleware va récupérer le jeton envoyé par le fournisseur de données (FS), puis le transférer à FranceConnect pour le vérifier et obtenir l’identité pivot et les scopes. Ces deux éléments seront ensuite sauvegardés dans le contexte de l’application pour être utilisé par les contrôleurs.

Ajoutez une classe *Error* pour retourner des erreurs au format JSON :

public class Error

{

public string error { get; set; }

public string message { get; set; }

}

Ajoutez une classe *DataProviderOptions* pour configurer le middleware :

public class DataProviderOptions

{

public string ChecktokenEndpoint { get; set; }

}

Ajoutez une classe statique *DataProviderAppBuilderExtensions* afin de pouvoir inscrire le middleware dans le pipeline :

public static class DataProviderAppBuilderExtensions

{

public static IApplicationBuilder UseDataProvider(this IApplicationBuilder app, DataProviderOptions options)

{

if (app == null)

{

throw new ArgumentNullException(nameof(app));

}

if (options == null)

{

throw new ArgumentNullException(nameof(options));

}

return app.UseMiddleware<DataProviderMiddleware>(options);

}

public static IApplicationBuilder UseDataProvider(this IApplicationBuilder app, Action<DataProviderOptions> configureOptions)

{

if (app == null)

{

throw new ArgumentNullException(nameof(app));

}

var options = new DataProviderOptions();

configureOptions?.Invoke(options);

return app.UseDataProvider(options);

}

}

Enfin, ajoutez une classe *ChecktokenResponse* pour parser la réponse de FranceConnect :

public class ChecktokenResponse

{

public string[] Scope { get; set; }

public Identity Identity { get; set; }

public Client Client { get; set; }

public string Identity\_provider\_id { get; set; }

public string Identity\_provider\_host { get; set; }

public string acr { get; set; }

}

public class Identity

{

public string Given\_name { get; set; }

public string Family\_name { get; set; }

public DateTimeOffset Birthdate { get; set; }

public string Gender { get; set; }

public int Birthplace { get; set; }

public int Birthcountry { get; set; }

public string Email { get; set; }

}

public class Client

{

public string Client\_id { get; set; }

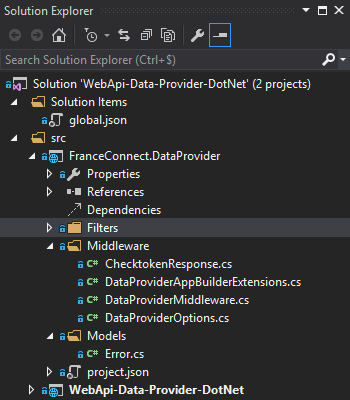
public string Client\_name { get; set; }

}

## Créer un filtre

C’est au fournisseur de données de vérifier que les scopes obtenus sont suffisant pour obtenir les ressources. Pour simplifier cette démarche, vous allez créer un filtre.

Ajoutez un nouveau dossier *Filters* dans le projet *FranceConnect.DataProvider* :



Ajoutez une classe *ConsentFilter* dans ce nouveau dossier :

public class ConsentFilter : ActionFilterAttribute

{

public string[] Scope { get; set; }

public override void OnActionExecuting(ActionExecutingContext context)

{

var recivedScope = context.HttpContext.Items["scope"] as string[];

if (recivedScope == null || !Scope.All(scope => recivedScope.Contains(scope)))

{

context.HttpContext.Response.StatusCode = (int)HttpStatusCode.Unauthorized;

context.HttpContext.Response.ContentType = "application/json";

context.Result = new JsonResult(new Error

{

error = "invalid\_scope",

message = "The request does not contain required scopes"

});

}

}

}

Ce filtre va vérifier que les scopes requis sont présents dans le jeton d’accès (access token) pour accéder à la ressource demandée.

Pour utiliser ce filtre, il suffit d’ajouter l’attribut suivant sur une action de contrôleur :

[ConsentFilter(Scope = new string[] { "scope1", "scope2" })]

## Utiliser le package NuGet

Si vous le souhaitez, vous pouvez créer un package NuGet du middleware et du filtre décrit ci-dessus. Voici la démarche à suivre :

1. Téléchargez les sources du projet depuis GitHub (Cf. section § *Obtenir les sources du fournisseur de données*)
2. Sélectionnez la configuration **Release**.
3. Compilez le projet *FranceConnect.DataProvider*
4. Récupérez le package dans le dossier *artifacts\bin\FranceConnect.DataProvider\Release*.

## Configurer l'API

### Modifier le fichier de configuration

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

Note Pour de plus amples informations, vous pouvez consulter la documentation [ici](https://docs.asp.net/en/latest/fundamentals/configuration.html).

Ajoutez les lignes suivantes pour configurer le middleware *FranceConnect.DataProvider* :

"FranceConnect": {  
 "ChecktokenEndpoint": "https://fcp.integ01.dev-franceconnect.fr/api/v1/checktoken"  
}

Ajoutez un second fichier de configuration nommé *appsettings.Development.json* :

{

"ConnectionStrings": {

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

}

}

Modifiez la classe Startup afin d’utiliser ce nouveau fichier de configuration :

var builder = new ConfigurationBuilder()

.SetBasePath(env.ContentRootPath)

.AddJsonFile("appsettings.json")

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

.AddEnvironmentVariables();

De la même manière, modifiez la méthode ConfigureServices de la classe Startup pour connecter l’application à la base de données :

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<ApplicationDbContext>(options =>

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

// Add framework services.

services.AddMvc();

}

## Utiliser le middleware *FranceConnect.DataProvider*

Dans la méthode *Configure* de la classe *Startup*, ajouter les lignes suivantes avant la ligne *app.UseMvc();* :

app.UseDataProvider(options =>

{

options.ChecktokenEndpoint = Configuration["FranceConnect:ChecktokenEndpoint"];

});

## Définir le modèle de donnée

Dans le dossier *Models*, ajoutez une classe *ApplicationUser* :

public class ApplicationUser

{

public ApplicationUser()

{

Id = Guid.NewGuid().ToString();

}

public ApplicationUser(string email) : this()

{

Email = email;

}

public string Id { get; set; }

public string Email { get; set; }

public string ValueOne { get; set; }

public string ValueTwo { get; set; }

}

Vous pouvez ajouter ou modifier des propriétés selon vos besoins.

Créez également une classe *ApplicationDbContext* :

public class ApplicationDbContext : DbContext

{

public DbSet<ApplicationUser> Users { get; set; }

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<ApplicationUser>(b =>

{

b.HasKey(u => u.Id);

b.HasIndex(u => u.Email).IsUnique(true);

b.Property(u => u.Id).HasMaxLength(256);

b.Property(u => u.Email).HasMaxLength(256);

b.Property(u => u.Email).IsRequired();

});

}

}

La propriété *Email* doit être unique et non nulle car c’est elle qui va permettre de réconcilier l’identité pivot retourné par FranceConnect avec votre base d’utilisateurs.

### Appliquer la migration

Lancez cette commande depuis le répertoire du projet :

> dotnet ef migrations add AddUserProperties

Note Pour de plus amples informations, vous pouvez consulter la [documentation Entity Framework Core](http://docs.efproject.net/en/latest/platforms/aspnetcore/new-db.html)[[9]](#footnote-9).

## Exposer des données

Dans un premier temps, créer un nouveau contrôleur *BaseController* :

public class BaseController : Controller

{

protected ApplicationDbContext \_context;

public BaseController(ApplicationDbContext context)

{

\_context = context;

}

protected override void Dispose(bool disposing)

{

base.Dispose(disposing);

\_context.Dispose();

}

}

Ce contrôleur de base va permettre d’accéder au contexte de la base de données.

Dans un second temps, créez un autre contrôleur *BaseApiController* :

public class BaseApiController : BaseController

{

public BaseApiController(ApplicationDbContext context) : base(context) { }

protected ApplicationUser FindUser()

{

var email = HttpContext.Items["email"] as string;

return \_context.Users.SingleOrDefault(u => u.Email == email);

}

protected ObjectResult ReturnUserNotFound()

{

HttpContext.Response.StatusCode = (int)HttpStatusCode.NotFound;

return new ObjectResult(new Error

{

error = "user\_not\_found",

message = "User not found"

});

}

}

Cet autre contrôleur de base va permettre de réconcilier l’identité pivot avec la base d’utilisateurs.

Enfin, modifiez le contrôleur *ValuesControlleur* :

[Route("api/[controller]")]

public class ValuesController : BaseApiController

{

public ValuesController(ApplicationDbContext context) : base(context) { }

// GET: api/values

[HttpGet]

[ConsentFilter(Scope = new string[] { "value1", "value2" })]

public IActionResult Get()

{

var user = FindUser();

if (user == null)

{

return ReturnUserNotFound();

}

return new ObjectResult(new

{

ValueOne = user.ValueOne,

ValueTwo = user.ValueTwo

});

}

}

L’action *Get* est filtrée par le *ConsentFilter*. De ce fait, si les scopes envoyés ne sont pas suffisant, le client n’aura pas accès à la ressource.

Il est possible de personnaliser les libellés de ces scopes sur la page de consentement de FranceConnect en en faisant la demande par mail à [support@dev-franceconnect.fr](mailto:support@dev-franceconnect.fr).

## Créer un formulaire de création de compte

Pour permettre à vos utilisateurs de créer un compte sur votre API, vous pouvez proposer un formulaire de création de compte.

### Configurer le projet 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.Diagnostics.EntityFrameworkCore* | 1.0.0 |
| *Microsoft.AspNetCore.Razor.Tools* | 1.0.0-preview2-final |
| *Microsoft.VisualStudio.Web.BrowserLink.Loader* | 14.0.0 |

Puis modifiez la méthode *Configure* de la classe *Startup* de la façon suivante :

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

{

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

loggerFactory.AddDebug();

if (env.IsDevelopment())

{

app.UseBrowserLink();

app.UseDeveloperExceptionPage();

app.UseDatabaseErrorPage();

}

else

{

app.UseExceptionHandler("/Account/Error");

}

app.UseStaticFiles();

app.UseDataProvider(options =>

{

options.ChecktokenEndpoint = Configuration["FranceConnect:ChecktokenEndpoint"];

});

app.UseMvc(routes =>

{

routes.MapRoute(

name: "default",

template: "{controller=Account}/{action=Register}/{id?}");

});

}

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).

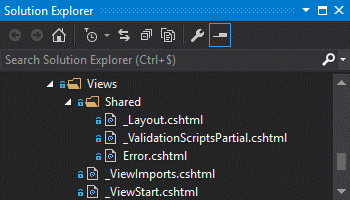
Modifiez également le fichier *project.json* pour la publication de l’API :

"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* :

@using WebApi\_Data\_Provider\_DotNet

@using WebApi\_Data\_Provider\_DotNet.ViewModels

@using WebApi\_Data\_Provider\_DotNet.ViewModels.Account

@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>@ViewData["Title"] - Fournisseur de données ASP.NET Core</title>

<environment names="Development">

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

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

</environment>

<environment names="Staging,Production">

<link rel="stylesheet" href="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.5/css/bootstrap.min.css"

asp-fallback-href="~/lib/bootstrap/dist/css/bootstrap.min.css"

asp-fallback-test-class="sr-only" asp-fallback-test-property="position" asp-fallback-test-value="absolute" />

<link rel="stylesheet" href="~/css/site.min.css" asp-append-version="true" />

</environment>

</head>

<body>

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

<div class="container">

<div class="navbar-header">

<a asp-controller="Home" asp-action="Index" class="navbar-brand">Fournisseur de données ASP.NET Core</a>

</div>

</div>

</div>

<div class="container body-content">

@RenderBody()

<hr />

<footer>

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

</footer>

</div>

<environment names="Development">

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

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

<script src="~/js/site.js" asp-append-version="true"></script>

</environment>

<environment names="Staging,Production">

<script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-2.1.4.min.js"

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

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

</script>

<script src="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.5/bootstrap.min.js"

asp-fallback-src="~/lib/bootstrap/dist/js/bootstrap.min.js"

asp-fallback-test="window.jQuery && window.jQuery.fn && window.jQuery.fn.modal">

</script>

<script src="~/js/site.min.js" asp-append-version="true"></script>

</environment>

@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/jquery.validation.unobtrusive/3.2.6/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 le formulaire

Ajoutez un contrôleur AccountController :

public class AccountController : BaseController

{

public AccountController(ApplicationDbContext context) : base(context) { }

// GET: /Account/Register

[HttpGet]

public IActionResult Register()

{

return View();

}

// POST: /Account/Register

[HttpPost]

public IActionResult Register(RegisterViewModel model)

{

if (!ModelState.IsValid)

{

ModelState.AddModelError("", "Tentative de création de compte non valide.");

return View(model);

}

if (\_context.Users.Any(user => user.Email == model.Email))

{

ModelState.AddModelError("", "Ce compte existe déjà.");

return View(model);

}

try

{

var user = new ApplicationUser

{

Email = model.Email,

ValueOne = model.ValueOne,

ValueTwo = model.ValueTwo

};

\_context.Users.Add(user);

\_context.SaveChanges();

ViewData["Message"] = "Le compte a été créé.";

return View();

}

catch (Exception)

{

ModelState.AddModelError("", "Une erreur est survenue lors de la création du compte. Veuillez réessayer.");

return View(model);

}

}

public IActionResult Error()

{

return View();

}

}

Créez également le modèle de vue *RegisterViewModel* :

namespace WebApi\_Data\_Provider\_DotNet.ViewModels.Account

{

public class RegisterViewModel

{

[Required]

[EmailAddress(ErrorMessage = "Adresse email non valide")]

[Display(Name = "Adresse email")]

public string Email { get; set; }

[Display(Name = "Valeur 1")]

public string ValueOne { get; set; }

[Display(Name = "Valeur 2")]

public string ValueTwo { get; set; }

}

}

Enfin, ajoutez la vue *Register* :

@model RegisterViewModel

@{

ViewData["Title"] = "Créer un compte";

}

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

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

<form asp-controller="Account" asp-action="Register" method="post" class="form-horizontal" role="form">

<h4>Créer un nouveau compte.</h4>

<hr />

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

<div class="form-group">

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

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

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

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

</div>

</div>

<div class="form-group">

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

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

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

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

</div>

</div>

<div class="form-group">

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

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

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

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

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<button type="submit" class="btn btn-default">Créer un compte</button>

</div>

</div>

</form>

@section Scripts {

@{ await Html.RenderPartialAsync("\_ValidationScriptsPartial"); }

}

# Déployer le fournisseur de données préconfiguré du kit de démarrage

## imageObtenir les sources du fournisseur de données

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

Vous trouverez à cette adresse notamment le projet relatif à l’exemple/accélérateur de fournisseur de données (FD) : [franceconnect-data-provider-dotnet-webapi-aspnetcore](https://github.com/FranceConnectSamples/franceconnect-data-provider-dotnet-webapi-aspnetcore)[[10]](#footnote-10).

Les solutions communautaires évoluant avec l’investissement de la communauté, il est souhaitable de vous assurer que vous disposez de la dernière version, ne serait-ce que pour bénéficier, le cas échéant, de l’ensemble des correctifs.

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

Les deux prochaines sections explorent les deux options possibles.

### Télécharger les sources du fournisseur de données

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 browser et naviguer vers le projet** [franceconnect-data-provider-dotnet-webapi-aspnetcore](https://github.com/FranceConnectSamples/franceconnect-data-provider-dotnet-webapi-aspnetcore)**.**
2. **Cliquez sur Clone or Download.**



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

**Le code source du canevas de fournisseur de données FranceConnect (FS) est alors situé sous le dossier *C:\Code\franceconnect-data-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 de données

 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/)[[11]](#footnote-11), [GitHub](https://github.com/beejones/B2CDemoTools)[[12]](#footnote-12) ou [Visual Studio](https://msdn.microsoft.com/vstudio/aa718325.aspx)[[13]](#footnote-13). 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/)[[14]](#footnote-14) sur le site de GitHub.

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

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

1. **Ouvrez une session browser et naviguer vers le projet** [franceconnect-data-provider-dotnet-webapi-aspnetcore](https://github.com/FranceConnectSamples/franceconnect-data-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-data-provider-dotnet-webapi-aspnetcore

Le clonage démarre.

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

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

Le package du canevas de fournisseur de données FranceConnect (FD) est disponible sous le dossier *franceconnect-data-provider-dotnet-webapi-aspnetcore* situé sous *%UserProfile%\Documents\GitHub*.

**Le code source associé est alors situé sous le dossier** *%UserProfile%\Documents\GitHub****\franceconnect-data-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.

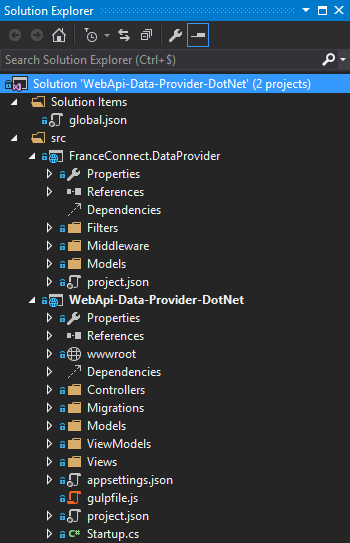
Remarque importante Nous faisons référence au dossier où se situe le code source du canevas de fournisseur de données (FD) comme étant le dossier Kit-Demarrage dans la suite de ce document.

## Organisation du code source du canevas de fournisseur de données

A l’instar de ce qui est décrit dans la section § [*Créer un fournisseur de données FranceConnect avec ASP.NET Core*](https://microsoft-my.sharepoint.com/personal/philber_microsoft_com/Documents/Drafts/Kit%20de%20démarrage%20FranceConnect/Versions/2016-06/franceconnect-data-provider-dotnet-webapi-aspnetcore/Documentation/Mettre%20en%20oeuvre%20un%20fournisseur%20de%20données%20FranceConnect%20avec%20ASP.NET%20Core-PHILBER002.docx#_Créer_un_fournisseur), la solution préconfigurée prête à l’emploi est constituée d’un unique projet ASP.NET Core MVC 6 de type **Web Application**. Ouvrez le fichier solution Visual Studio .sln depuis le dossier **Kit-Demarrage***.*

### Comprendre l’organisation du projet

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



Architecture du projet

### Appréhender les éléments de configuration

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

{

"ConnectionStrings": {

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

},

"Logging": {

"IncludeScopes": false,

"LogLevel": {

"Default": "Debug",

"System": "Information",

"Microsoft": "Information"

}

},

"FranceConnect": {

"ChecktokenEndpoint": "https://fcp.integ01.dev-franceconnect.fr/api/v1/checktoken"

}

}

|  |  |
| --- | --- |
| **Section de configuration** | **Description** |
| ConnectionStrings | Contient la chaine de connexion à la base de données |
| Logging | Contient les paramètres du loggeur |
| FranceConnect | Contient l’URL de validation du jeton auprès de FranceConnect |

## Déployer le projet dans Microsoft Azure

Pour déployer le fournisseur de données dans Microsoft Azure, vous allez devoir créer une nouvelle application web.

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

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

### Créer l’API web dans Microsoft Azure

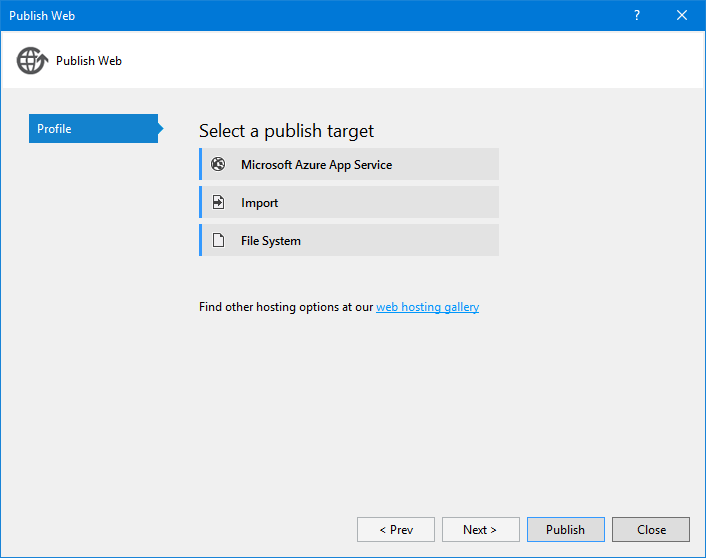
Pour créer l’API web relative à votre fournisseur de données dans Microsoft Azure, procédez comme suit :

1. **Ouvrez une session browser et** rendez-vous sur le portail Azure à l’adresse https://portal.azure.com/
2. Cliquez sur **Nouveau**.
3. Dans la catégorie **Web + mobile**, choisissez **Application web + SQL**.
4. Renseignez les champs demandés.

### Publier le fournisseur de données dans Microsoft Azure

Pour publier votre fournisseur de données 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**.



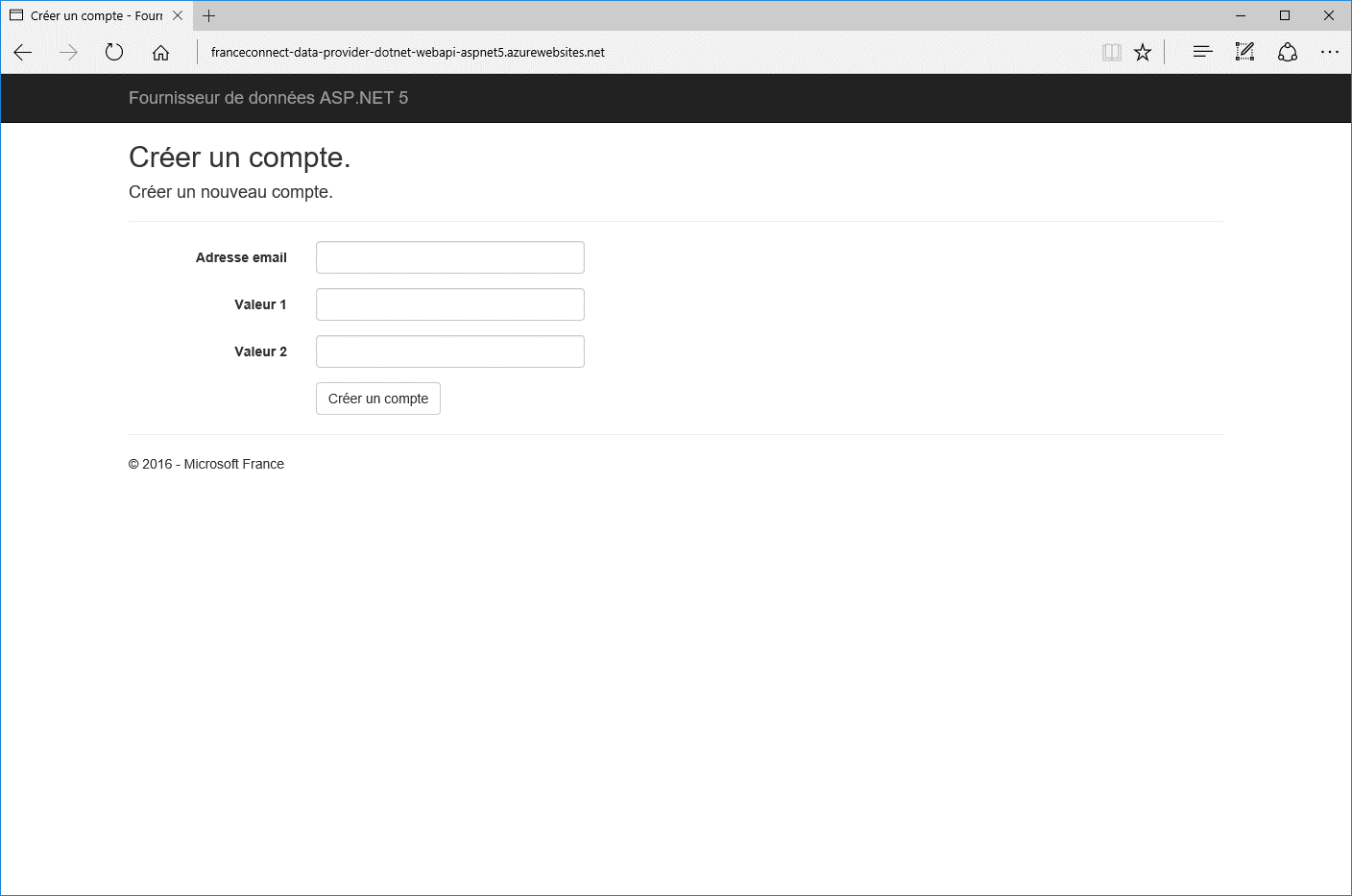
1. Sélectionnez l’API web que vous venez de créer.
2. Cliquez sur **Publish**.

# Tester le fournisseur de donnée

## Créer un compte sur le fournisseur de données

Pour créer un compte au niveau du canevas de fournisseur de données, procédez comme suite :

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



1. Renseignez une adresse mèl d’un compte enregistré auprès d’un fournisseur d’identité.

## Obtenir des données

Invoquez le fournisseur de données depuis un fournisseur de service en précisant l’URL de votre ressource et les scopes nécessaires.

Ceci conclut ce guide.

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

Il repose plus particulièrement pour la définition des éléments de mise en œuvre du fournisseur de données FranceConnect ainsi proposé sur la [documentation](https://doc.integ01.dev-franceconnect.fr/fournisseur-donnees)[[16]](#footnote-16).

1. Site portail Web Microsoft Secteur Public : http://www.microsoft.com/france/entreprises/secteur-public/solutions-pour-secteur-public.aspx [↑](#footnote-ref-1)
2. ASP.NET Core : http://www.dotnetfoundation.org/aspnet-core [↑](#footnote-ref-2)
3. Fondation .NET : http://www.dotnetfoundation.org/ [↑](#footnote-ref-3)
4. Visual Studio Community 2015 : https://go.microsoft.com/fwlink/?LinkId=691978&clcid=0x40c [↑](#footnote-ref-4)
5. Introduction to ASP.NET Core : https://docs.asp.net/en/latest/conceptual-overview/aspnet.html [↑](#footnote-ref-5)
6. ASP.NET Core Documentation : https://docs.asp.net/en/latest/ [↑](#footnote-ref-6)
7. franceconnect-data-provider-dotnet-webapi-aspnetcore : https://github.com/FranceConnectSamples/franceconnect-data-provider-dotnet-webapi-aspnetcore [↑](#footnote-ref-7)
8. https://docs.asp.net/en/latest/fundamentals/middleware.html [↑](#footnote-ref-8)
9. ASP.NET Core Application to New Database : http://docs.efproject.net/en/latest/platforms/aspnetcore/new-db.html [↑](#footnote-ref-9)
10. franceconnect-data-provider-dotnet-webapi-aspnetcore : https://github.com/FranceConnectSamples/franceconnect-data-provider-dotnet-webapi-aspnetcore [↑](#footnote-ref-10)
11. Git : http://git-scm.com/ [↑](#footnote-ref-11)
12. GitHub : https://github.com/beejones/B2CDemoTools [↑](#footnote-ref-12)
13. Visual Studio : https://msdn.microsoft.com/vstudio/aa718325.aspx [↑](#footnote-ref-13)
14. Set up Git : https://help.github.com/articles/set-up-git/ [↑](#footnote-ref-14)
15. FranceConnect : https://doc.integ01.dev-franceconnect.fr/ [↑](#footnote-ref-15)
16. Documentation – Fournisseur de données : https://doc.integ01.dev-franceconnect.fr/fournisseur-donnees [↑](#footnote-ref-16)