



Dotnet France
Technologies Sharepoint, SQL Server & .NET

Association Dotnet France

Premiers pas avec ADO .NET Data Services

Version 1.0



James RAVAILLE

<http://blogs.dotnet-france.com/jamesr>

Sommaire

1	Introduction.....	3
2	Réalisation d'une application	4
2.1	Présentation	4
2.1.1	Création de la base de données	4
2.1.2	Alimentation de la base de données	6
2.1.3	Architecture de l'application	6
2.1.4	Création des projets	7
3	Création du service d'accès et de gestion des données.....	8
3.1	Création du composant Entity Data Model.....	8
3.2	Création du service d'exposition des données.....	14
3.3	Définition des règles d'accès sur les entités.....	16
3.4	Définir des opérations de service.....	17
4	Affichage et gestion des données	19
4.1	Création d'une référence de service	19
4.2	Création d'une classe de contexte pour les formulaires.....	21
4.3	Création du formulaire <i>FrmDetailStagiaire</i>	22
4.3.1	Design du formulaire	22
4.3.2	Code-behind du formulaire	23
4.4	Création du formulaire <i>FrmGestionListeStagiaires</i>	25
4.4.1	Design du formulaire	25
4.4.2	Code-behind du formulaire	27
5	Exécution de l'application	35
5.1	Exécution des fonctionnalités de l'application.....	35
5.1.1	Ajout d'un stagiaire	35
5.1.2	Modification d'un stagiaire	36
5.1.3	Suppression d'un stagiaire	37
5.2	Observation des traces HTTP	37
6	Conclusion	38

1 Introduction

Ce cours permet de vous montrer, comment mettre en œuvre de manière pratique ADO .NET Data Services, afin de vous permettre de comprendre ses principes fondamentaux. De manière volontaire, nous ne rentrerons pas dans les détails. En revanche, les chapitres suivants publiés sur Dotnet-France, détailleront chacune des parties de ce document.

Ce cours se décompose de la manière suivante :

- Présentation de l'application de gestion, créée tout au long de ce cours.
- Création du service de gestion de données ADO .NET. Ce service de données exposera les données obtenues au travers d'un modèle de données, créé avec le Framework Entity. Pour une bonne compréhension de ce cours, nous vous recommandons de consulter les cours publiés sur Dotnet-France sur ce sujet.
- Création d'une application Windows Forms, qui affiche et gère les données, au travers du service de données ADO .NET.

2 Réalisation d'une application

2.1 Présentation

L'application que nous allons développer tout au long de ce cours, aura pour but de consulter et de gérer une liste de stagiaires, et des cours auxquels ils sont inscrits. Elle sera constituée de deux formulaires :

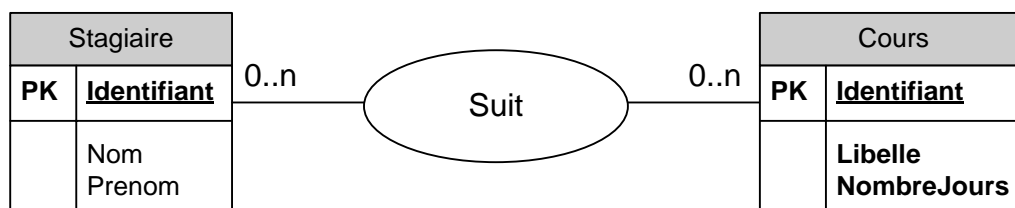
- Un affichant la liste des stagiaires.
- Un affichant les informations sur un stagiaire.

2.1.1 Création de la base de données

2.1.1.1 Le modèle conceptuel de données

Le modèle conceptuel des données (aussi appelé MCD) permet de représenter de façon formelle, sous forme d'un schéma, les données qui seront utilisées par une application. Ce schéma est composé d'entités, reliées entre elles par des relations d'association.

Voici le modèle conceptuel de données de notre application :



Ce modèle décrit les entités Stagiaire et Cours. Il met aussi en évidence :

- Qu'un stagiaire peut suivre un ou plusieurs cours.
- Qu'un cours peut être suivi par aucun, un ou plusieurs stagiaires.

2.1.1.2 Le modèle logique de données

Nous allons maintenant créer le modèle logique de données. La relation n-aire bilatérale entre les tables Stagiaire et Cours entraîne la création d'une table supplémentaire, que nous appelons Stagiaire2Cours. Cette table agrège les champs constituant la clé primaire des tables Cours et Stagiaire. Le modèle logique de données obtenu est donc le suivant :



Pour créer ce schéma de base de données, lancer SQL Server Management Studio (interface d'administration de SQL Server). Créez une base de données nommée DotnetFrance, puis dans une fenêtre de requête, exécutez le jeu d'instructions Transact-SQL suivant :



```
-- SQL

USE DotnetFrance

CREATE TABLE [dbo].[Stagiaire] (
    [Identifiant] [int] IDENTITY(1,1) NOT NULL,
    [Nom] [varchar](50) NOT NULL,
    [Prenom] [varchar](50) NOT NULL,
    CONSTRAINT [PK_Stagiaire] PRIMARY KEY CLUSTERED
(
    [Identifiant] ASC
) WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY],
    CONSTRAINT [UQ__Stagiaire__7F60ED59] UNIQUE NONCLUSTERED
(
    [Identifiant] ASC
) WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[Cours] (
    [Identifiant] [int] IDENTITY(1,1) NOT NULL,
    [Libelle] [varchar](50) NOT NULL,
    [NombreJours] [int] NOT NULL,
    CONSTRAINT [PK_Cours] PRIMARY KEY CLUSTERED
(
    [Identifiant] ASC
) WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY],
    CONSTRAINT [UQ__Cours__023D5A04] UNIQUE NONCLUSTERED
(
    [Identifiant] ASC
) WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[Stagiaire2Cours] (
    [IdStagiaire] [int] NOT NULL,
    [IdCours] [int] NOT NULL,
    CONSTRAINT [PK_Stagiaire2Cours] PRIMARY KEY CLUSTERED
(
    [IdStagiaire] ASC,
    [IdCours] ASC
) WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Stagiaire2Cours] WITH CHECK ADD CONSTRAINT
[FK_Stagiaire2Cours_Cours] FOREIGN KEY([IdCours])
REFERENCES [dbo].[Cours] ([Identifiant])
ON UPDATE CASCADE
ON DELETE CASCADE
GO

ALTER TABLE [dbo].[Stagiaire2Cours] WITH CHECK ADD CONSTRAINT
[FK_Stagiaire2Cours_Stagiaire] FOREIGN KEY([IdStagiaire])
REFERENCES [dbo].[Stagiaire] ([Identifiant])
ON UPDATE CASCADE
ON DELETE CASCADE
GO
```

Voici une brève description des tables :

- **Stagiaire** : contient toutes les informations relatives aux stagiaires.
- **Cours** : contient toutes les informations relatives aux cours. Les stagiaires doivent suivre des cours afin de pouvoir travailler sur un projet d'un client.
- **Stagiaire2Cours** : permet de lier les stagiaires à un cours. Un stagiaire peut assister à plusieurs cours, et un cours peut concerner plusieurs stagiaires.

2.1.2 Alimentation de la base de données

Voici un jeu d'instructions Transact-SQL, permettant d'alimenter les tables en données :

```
-- SQL

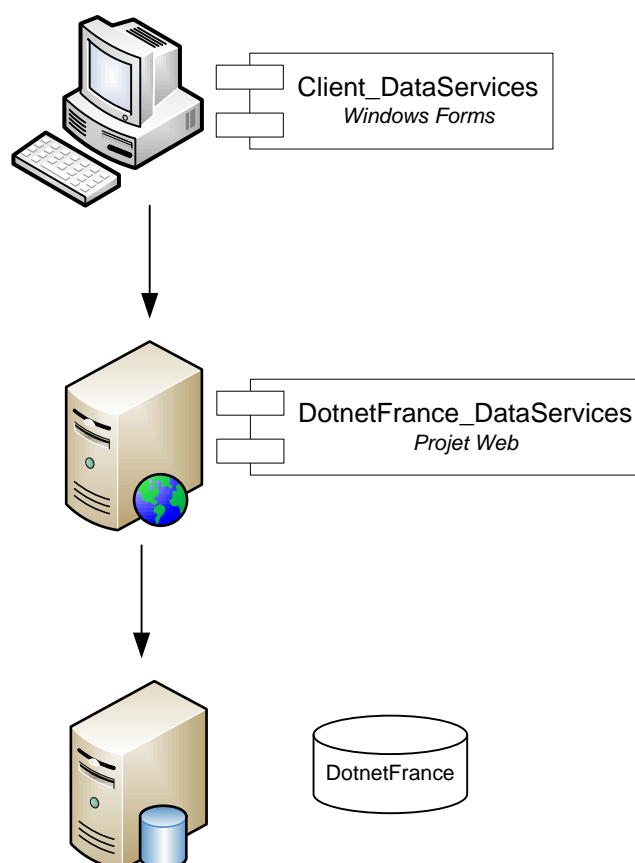
-- Alimentation de la table des stagiaires.
INSERT INTO Stagiaire(Nom, Prenom) VALUES ('DEROUX', 'Alain')
INSERT INTO Stagiaire(Nom, Prenom) VALUES ('RAVAILLE', 'James')
INSERT INTO Stagiaire(Nom, Prenom) VALUES ('SIRON', 'Karl')
INSERT INTO Stagiaire(Nom, Prenom) VALUES ('EMATO', 'Julie')

-- Alimentation de la table des cours.
INSERT INTO Cours (Libelle, NombreJours) VALUES ('SQL Server -
Administration de serveurs', 5)
INSERT INTO Cours (Libelle, NombreJours) VALUES ('XHTML / CSS', 3)
INSERT INTO Cours (Libelle, NombreJours) VALUES ('C#', 5)
INSERT INTO Cours (Libelle, NombreJours) VALUES ('ASP .NET 3.5', 5)
INSERT INTO Cours (Libelle, NombreJours) VALUES ('ASP .NET AJAX', 3)

-- Affectation des cours aux stagiaires.
INSERT INTO Stagiaire2Cours (IdStagiaire, IdCours) VALUES (1, 1)
INSERT INTO Stagiaire2Cours (IdStagiaire, IdCours) VALUES (1, 3)
INSERT INTO Stagiaire2Cours (IdStagiaire, IdCours) VALUES (2, 2)
INSERT INTO Stagiaire2Cours (IdStagiaire, IdCours) VALUES (2, 1)
INSERT INTO Stagiaire2Cours (IdStagiaire, IdCours) VALUES (2, 3)
INSERT INTO Stagiaire2Cours (IdStagiaire, IdCours) VALUES (2, 4)
INSERT INTO Stagiaire2Cours (IdStagiaire, IdCours) VALUES (3, 1)
INSERT INTO Stagiaire2Cours (IdStagiaire, IdCours) VALUES (3, 4)
INSERT INTO Stagiaire2Cours (IdStagiaire, IdCours) VALUES (3, 5)
INSERT INTO Stagiaire2Cours (IdStagiaire, IdCours) VALUES (4, 4)
INSERT INTO Stagiaire2Cours (IdStagiaire, IdCours) VALUES (4, 5)
```

2.1.3 Architecture de l'application

Voici l'architecture de l'application :



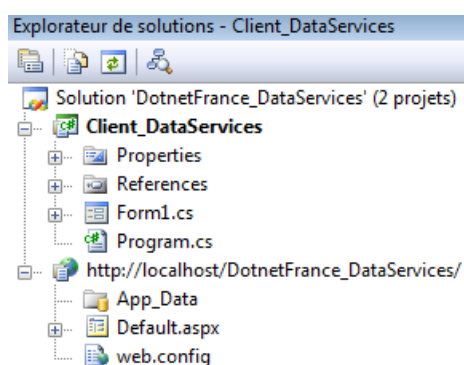
L'application est composée de deux projets :

- *Client_DataServices* : projet permettant d'afficher les données de la base de données *DotnetFrance*, et d'interagir avec les utilisateurs pour gérer ces données.
- *DotnetFrance_DataServices* : projet Web, contenant le service de données ADO .NET, permettant de consulter et de gérer les données dans la base de données *DotnetFrance*.

2.1.4 Création des projets

Dans Visual Studio 2008, créons deux projets :

- Un projet de type Windows Forms, nommé *Client_DataServices*. Définir ce projet comme projet de démarrage.
- Un projet de type Site Web, nommé *DotnetFrance_DataServices*.



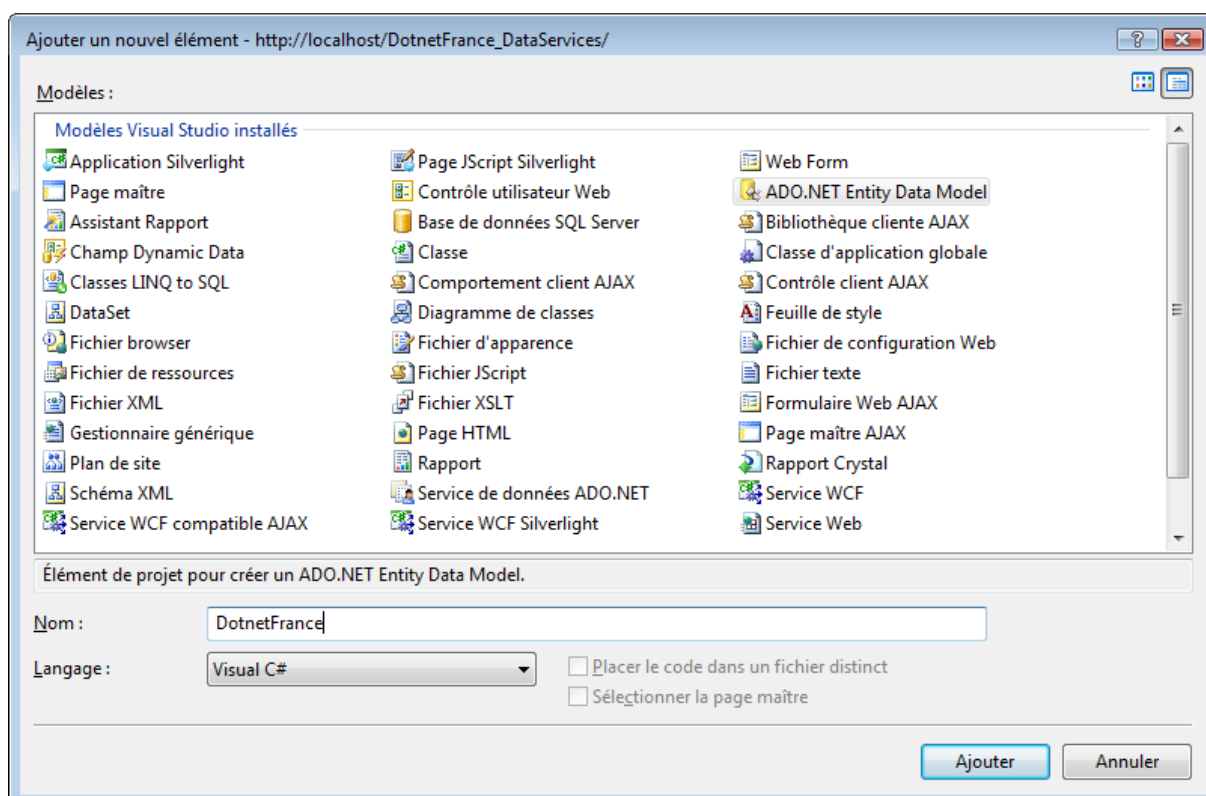
3 Création du service d'accès et de gestion des données

Dans ce chapitre, nous allons créer le service d'accès aux données avec ADO .NET Data Services. Les étapes de création de ce service sont les suivantes :

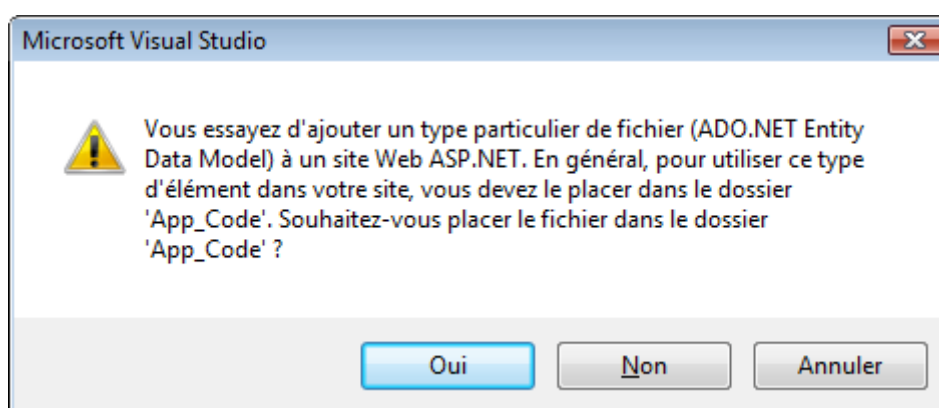
- Création d'un modèle d'entités.
- Création du service d'exposition des données.

3.1 Création du composant Entity Data Model

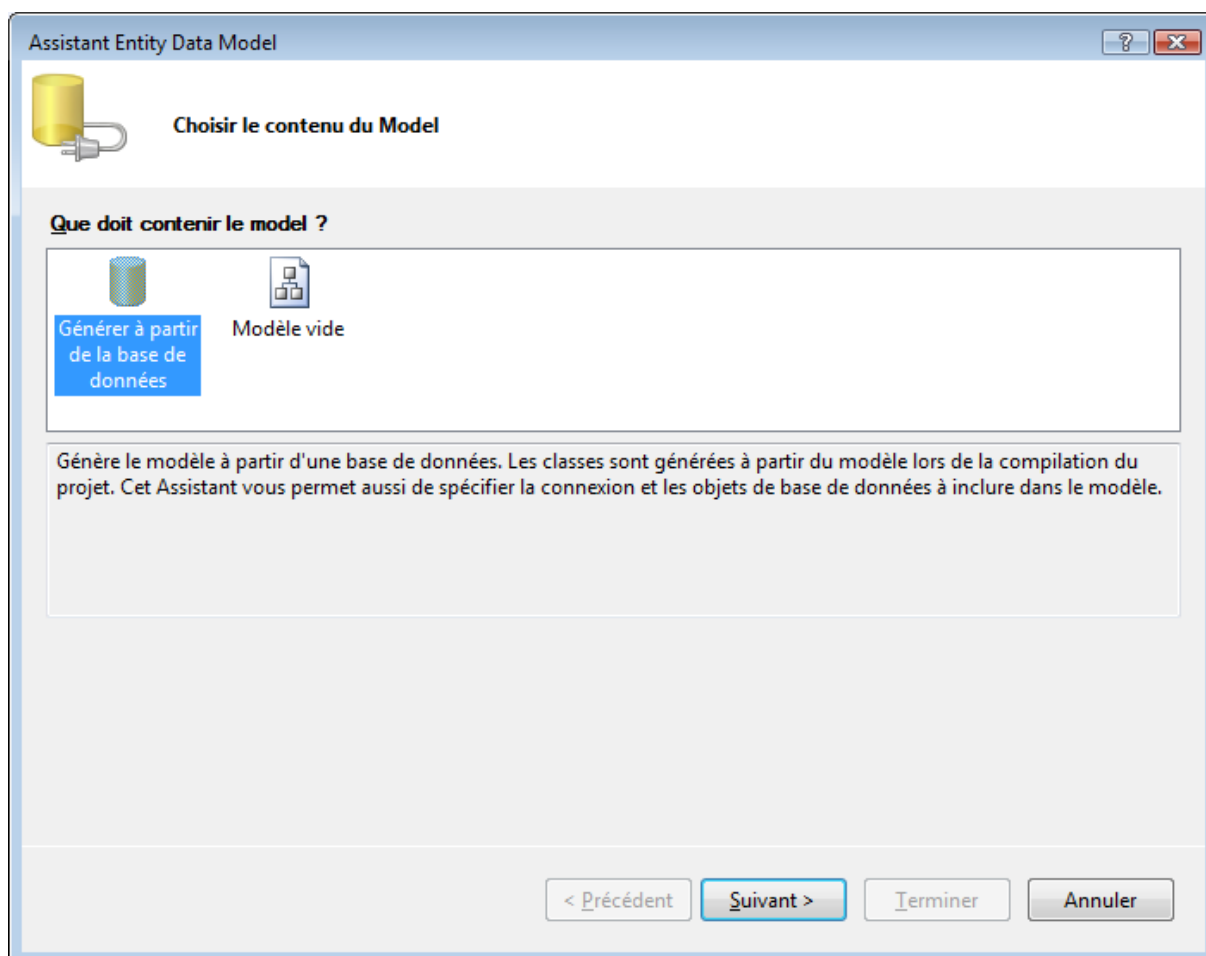
Dans le projet *DotnetFrance_DataServices*, commençons par ajouter un composant de type *Entity Data Model*, nommé *DotnetFrance*. Ce composant constituera notre modèle d'entités, qui sera utilisé ultérieurement par notre service de données ADO .NET :



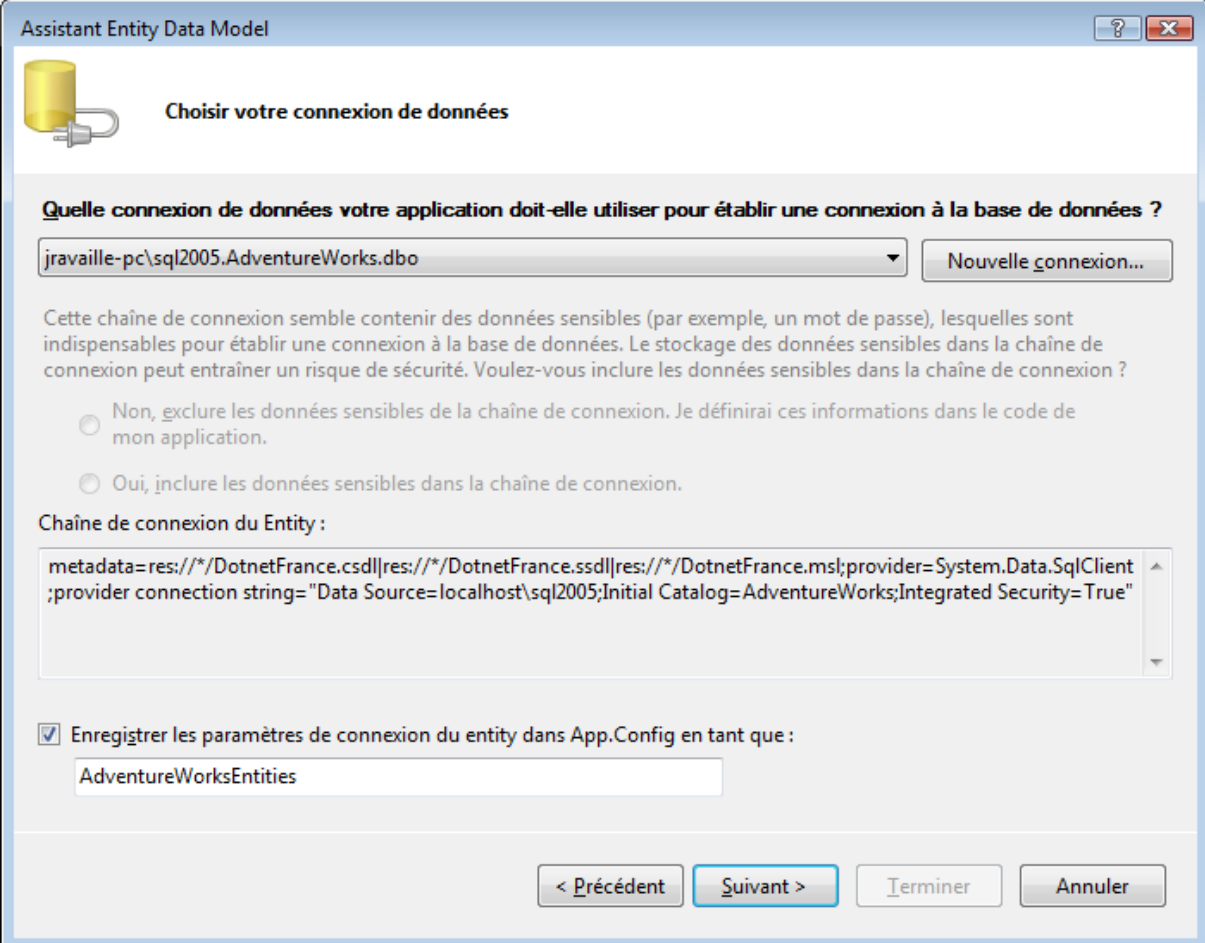
Après avoir validé, la fenêtre suivante apparaît :




Cliquons sur le bouton « oui », pour simplifier le développement et l'utilisation ultérieure de ce service. La fenêtre suivante apparaît alors :



Nous allons créer notre composant d'accès et gestion de données à partir de notre base de données DotnetFrance. Nous sélectionnons alors l'item « Générer à partir de la base de données », et cliquons sur le bouton « Suivant ». La fenêtre suivante apparaît :



Assistant Entity Data Model

 Choisir votre connexion de données

Quelle connexion de données votre application doit-elle utiliser pour établir une connexion à la base de données ?

Nouvelle connexion...

Cette chaîne de connexion semble contenir des données sensibles (par exemple, un mot de passe), lesquelles sont indispensables pour établir une connexion à la base de données. Le stockage des données sensibles dans la chaîne de connexion peut entraîner un risque de sécurité. Voulez-vous inclure les données sensibles dans la chaîne de connexion ?

☐ Non, exclure les données sensibles de la chaîne de connexion. Je définirai ces informations dans le code de mon application.

☐ Oui, inclure les données sensibles dans la chaîne de connexion.

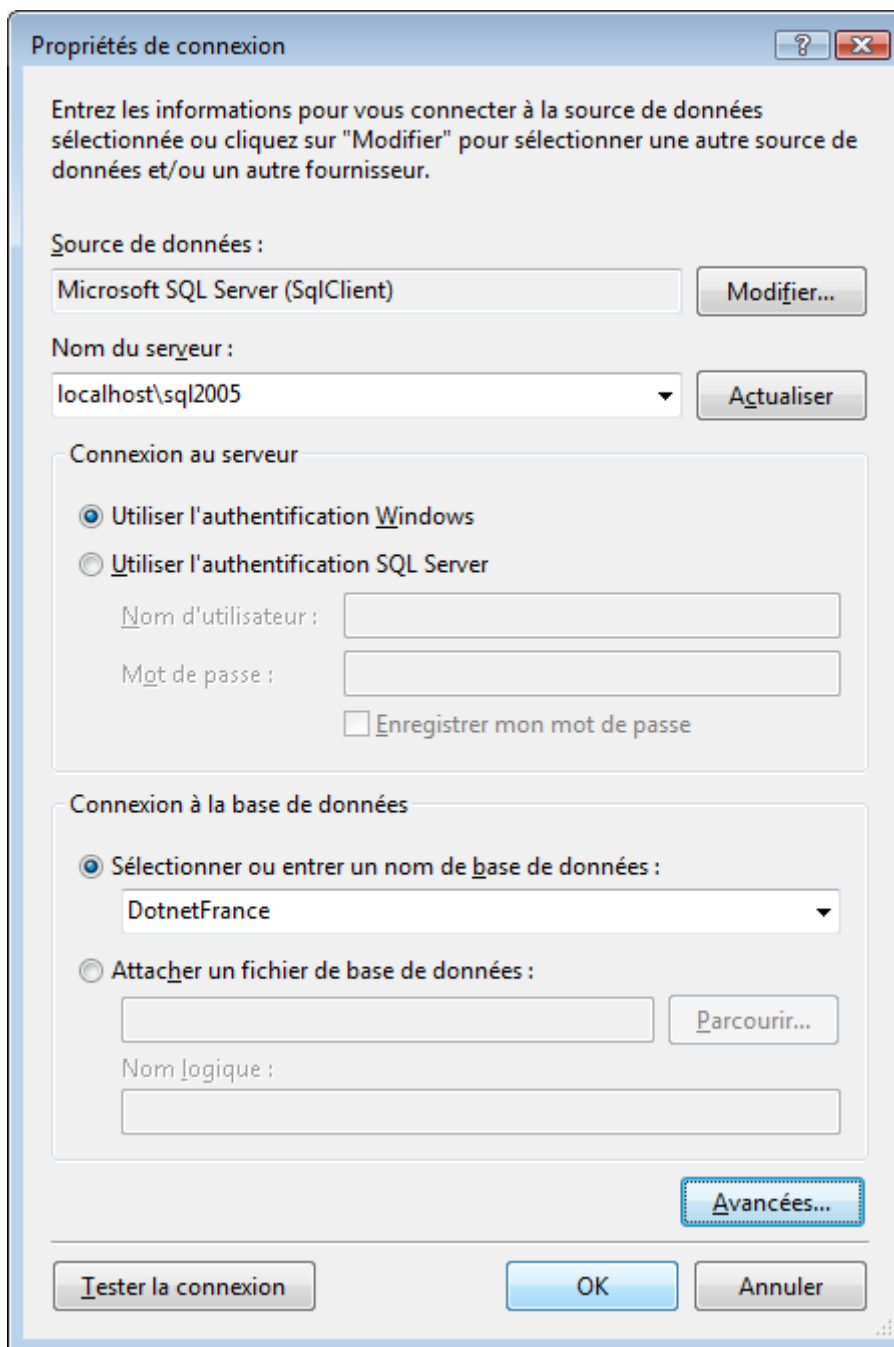
Chaîne de connexion du Entity :

`metadata=res://*/DotnetFrance.csdl|res://*/DotnetFrance.ssdl|res://*/DotnetFrance.msl;provider=System.Data.SqlClient;provider connection string="Data Source=localhost\sql2005;Initial Catalog=AdventureWorks;Integrated Security=True"`

☒ Enregistrer les paramètres de connexion du entity dans App.Config en tant que :

< Précédent Suivant > Terminer Annuler

Dans cette fenêtre, choisir une connexion pointant vers notre base de données, ou alors créer une nouvelle connexion en cliquant sur le bouton « Nouvelle connexion... ». Dans le second cas, la fenêtre suivante apparaîtra :



Propriétés de connexion

Entrez les informations pour vous connecter à la source de données sélectionnée ou cliquez sur "Modifier" pour sélectionner une autre source de données et/ou un autre fournisseur.

Source de données :

Microsoft SQL Server (SqlClient) Modifier...

Nom du serveur :

localhost\\sql2005 Actualiser

Connexion au serveur

☒ Utiliser l'authentification Windows

☐ Utiliser l'authentification SQL Server

Nom d'utilisateur :

Mot de passe :

☐ Enregistrer mon mot de passe

Connexion à la base de données

☒ Sélectionner ou entrer un nom de base de données :

DotnetFrance

☐ Attacher un fichier de base de données :

Parcourir...


Nom logique :

Avancées...

Tester la connexion OK Annuler

Dans cette fenêtre, saisir le nom de votre instance SQL Server, sur laquelle votre base de données est hébergée. Puis choisissez votre mode d'authentification, ainsi que la base de données. Cliquer sur le bouton « Tester la connexion », afin de vérifier si toutes ces informations sont correctes. Cela est nécessaire, car elles seront utilisées pour créer la chaîne de connexion. Une fois cette fenêtre validée, nous revenons à la fenêtre suivante :

Assistant Entity Data Model

 Choisir votre connexion de données

Quelle connexion de données votre application doit-elle utiliser pour établir une connexion à la base de données ?

jrvaille-pc\sql2005.DotnetFrance.dbo Nouvelle connexion...

Cette chaîne de connexion semble contenir des données sensibles (par exemple, un mot de passe), lesquelles sont indispensables pour établir une connexion à la base de données. Le stockage des données sensibles dans la chaîne de connexion peut entraîner un risque de sécurité. Voulez-vous inclure les données sensibles dans la chaîne de connexion ?

☐ Non, exclure les données sensibles de la chaîne de connexion. Je définirai ces informations dans le code de mon application.

☐ Oui, inclure les données sensibles dans la chaîne de connexion.

Chaîne de connexion du Entity :

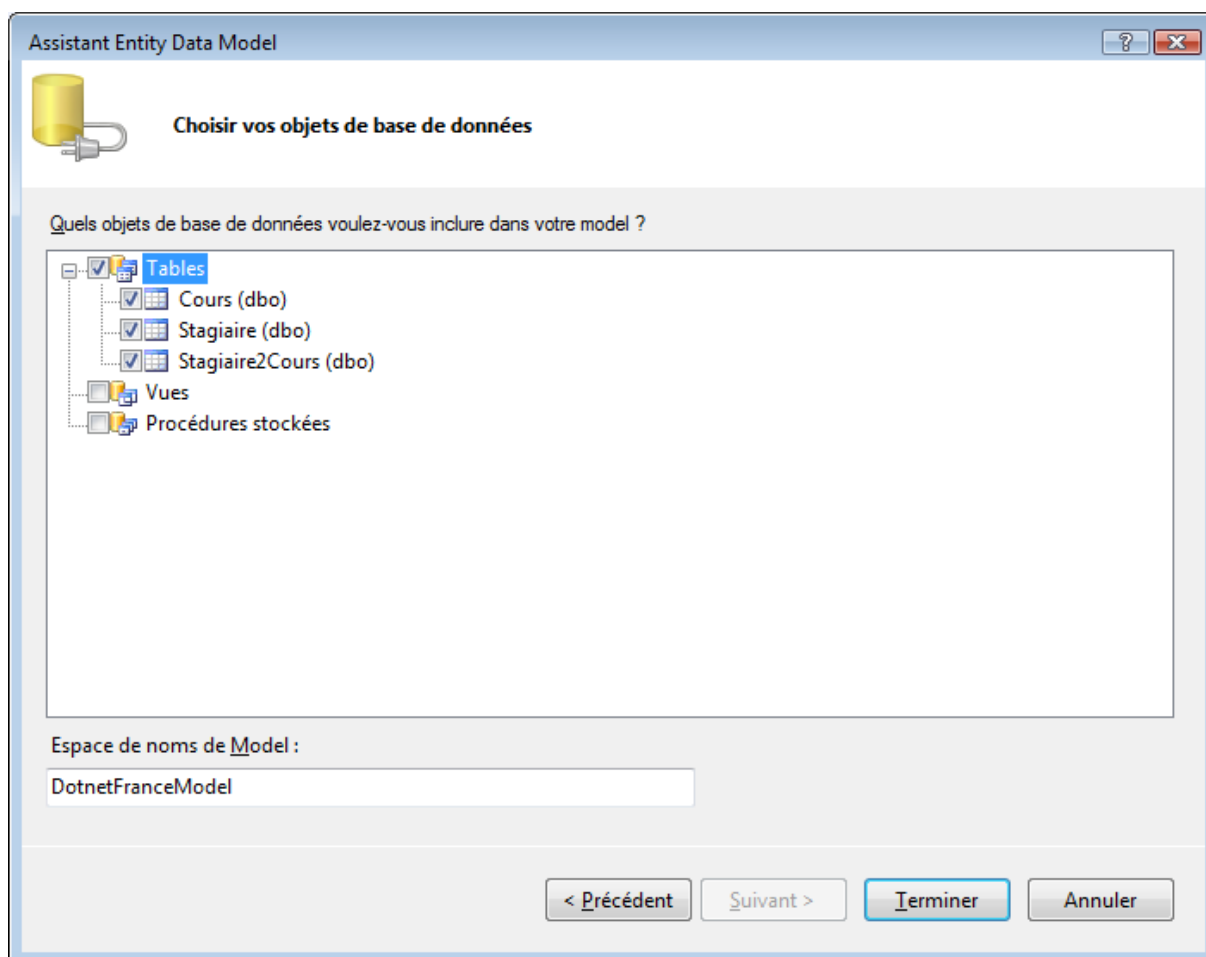
```
metadata=res://*/DotnetFranceEntities.csdl|res://*/DotnetFranceEntities.ssdl|res://*/DotnetFranceEntities.msl;provider=System.Data.SqlClient;provider connection string="Data Source=localhost\sql2005;Initial Catalog=DotnetFrance;Integrated Security=True"
```

☒ Enregistrer les paramètres de connexion du entity dans App.Config en tant que :

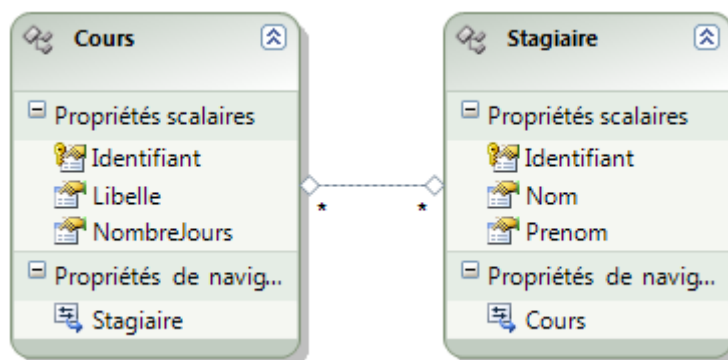
DotnetFranceEntities

< Précédent Suivant > Terminer Annuler

Cliquer alors sur le bouton « Suivant ». La fenêtre suivante apparaîtra :



Choisissons les tables, vues et procédures stockées, que nous allons utiliser dans notre modèle d'entités. Dans notre exemple, nous allons gérer les données contenues dans les tables *Cours*, *Stagiaires* et *Stagiaire2Cours*, que nous sélectionnons. Cliquons ensuite sur le bouton « Terminer ». Le model suivant apparaît :



Vous pouvez observer que la « table de liaison » *Stagiaire2Cours*, présente dans le modèle logique de données de la base de données, n'est pas présente. En effet, le Framework Entity se base sur une vision conceptuelle de la base de données.

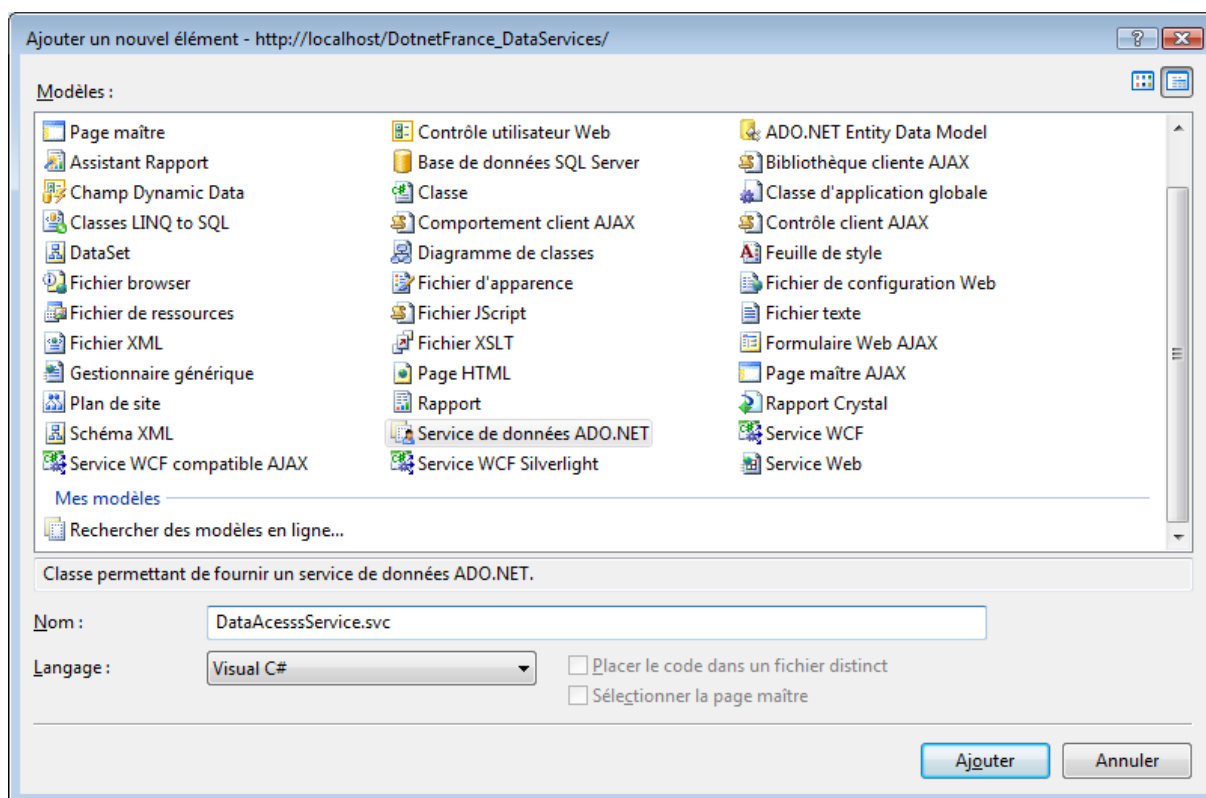
L'implémentation de ces classes est contenue dans le fichier *DotnetFrance.Designer.cs* ou *DotnetFranceEntities.vb*. En ouvrant ce fichier, on peut remarquer :

- Que toutes les classes d'entité dérivent de la classe *System.Data.Objects.DataClasses.EntityObject*.
- Qu'une classe supplémentaire est générée. Elle porte le même nom que le composant *Entity Data Model* précédemment créé, suffixé par « Entities ». Cette classe joue un rôle essentiel dans l'accès et la gestion des entités, car elle permet :
 - o De définir un ensemble de membres communs.
 - o De charger les entités dans un contexte de données.
 - o D'effectuer un suivi des modifications effectuées sur les entités.
 - o ...

Ces informations sont importantes, car elles devront être utilisées ultérieurement, lors de la création de l'application Windows Forms, consommant notre service de données.

3.2 Création du service d'exposition des données

A la racine du projet *DotnetFrance_DataServices*, ajoutons un composant de type *Service de données ADO .NET*, nommé *DataAccessService.svc* :



La création de ce composant provoque la création d'un service WCF (Windows Communication Foundation), composé :

- Du point d'entrée du service, représenté par le fichier *DataAccessService.svc*.
- Du fichier *DataAccessService.cs*, contenu dans le répertoire *App_Code*, contenant la classe d'arrière plan de notre service, dont le code est le suivant (par défaut) :

```
// C#

using System;
using System.Data.Services;
using System.Collections.Generic;
using System.Linq;
using System.ServiceModel.Web;

public class DataAccessService : DataService< /* TODO : placez ici le nom
de votre classe source de données */ >
{
    // Cette méthode n'est appelée qu'une seule fois pour initialiser les
    stratégies au niveau des services.
    public static void InitializeService(IDataServiceConfiguration
config)
    {
        // TODO : définissez des règles pour indiquer les jeux d'entités
        et opérations de service visibles, pouvant être mis à jour, etc.
        // Exemples :
        // config.SetEntitySetAccessRule("MyEntityset",
EntitySetRights.AllRead);
        // config.SetServiceOperationAccessRule("MyServiceOperation",
ServiceOperationRights.All);
    }
}
```

```
' VB .NET

Imports System.Data.Services
Imports System.Linq
Imports System.ServiceModel.Web

Public Class DataAccessService
    ' TODO : remplacez [[class name]] par le nom de votre classe de
    données
    Inherits DataService(Of [[class name]])

    ' Cette méthode n'est appelée qu'une seule fois pour initialiser les
    stratégies au niveau des services.
    Public Shared Sub InitializeService(ByVal config As
IDataServiceConfiguration)
        ' TODO : définissez des règles pour indiquer les jeux d'entités
        et opérations de service visibles, pouvant être mis à jour, etc.
        ' Exemples :
        ' config.SetEntitySetAccessRule("MyEntityset",
EntitySetRights.AllRead)
        ' config.SetServiceOperationAccessRule("MyServiceOperation",
ServiceOperationRights.All)
    End Sub
```

La classe de notre composant d'accès aux données ADO .NET étend la classe générique *System.Data.Services.DataService<T>*. Elle sera le point d'entrée principal pour notre service de données ADO .NET. Le type précisé en paramètre correspond à la classe permettant d'accéder et gérer les données. Dans notre cas, il s'agit de la classe de gestion des entités : *DotnetFranceEntities*. La définition de la classe obtenue est alors la suivante :

```
// C#

using DotnetFranceModel;

public class DataAccessService : DataService<DotnetFranceEntities>
{
}

```

```
' VB .NET

Imports DotnetFranceModel

Public Class DataAccessService
    Inherits DataService(Of DotnetFranceEntities)

End Class

```

La classe *DataAccessService* contient une méthode statique nommée *InitializeService*, et acceptant en paramètre un objet de type *IDataServiceConfiguration*. Ce paramètre permet de définir les autorisations sur les entités et les verbes HTTP autorisés.

3.3 Définition des règles d'accès sur les entités

Notre modèle d'entités permet d'accéder et gérer deux types d'entités : des stagiaires et des cours. Nous allons définir les règles suivantes :

- Autoriser la lecture / ajout / modification / suppression des stagiaires.
- Autoriser uniquement la lecture des cours, auxquels les stagiaires sont inscrits.

Dans la méthode *InitializeService*, nous ajoutons les instructions suivantes :

```
// C#

public static void InitializeService(IDataServiceConfiguration config)
{
    config.SetEntitySetAccessRule("Stagiaire", EntitySetRights.All);
    config.SetEntitySetAccessRule("Cours", EntitySetRights.AllRead);
}

```

```
' VB .NET

Public Shared Sub InitializeService(ByVal config As
IDataServiceConfiguration)
    config.SetEntitySetAccessRule("Stagiaire", EntitySetRights.All)
    config.SetEntitySetAccessRule("Cours", EntitySetRights.AllRead)
End Sub

```


3.4 Définir des opérations de service

Dans la classe de votre service de données ADO .NET, vous pouvez ajouter une opération de service, en ajoutant une méthode, qui doit respecter les caractéristiques suivantes :

- Elle doit être publique.
- Pour les développeurs C#, elle ne doit posséder que des paramètres d'entrée (paramètres de sortie interdits).
- Il peut s'agir d'une procédure. Si elle est une fonction, le type de retour doit être les interfaces génériques *IEnumerable<T>* ou *IQueryable<T>*, où T ou une classe primitive telle qu'un entier ou une chaîne de caractères.
- Elle doit être définie avec l'un des attributs de méthode suivants :
 - o *WebGet* : permet d'appeler la méthode à l'aide d'une requête HTTP de type GET.
 - o *WebInvoke* : permet d'appeler la méthode à l'aide d'une requête de type HTTP de type POST, PUT, MERGE, DELETE.

Voici un exemple : soit une méthode, permettant d'obtenir le nombre de stagiaires étant inscrit à au moins un cours :

```
// C#

public class DataAccessService : DataService<DotnetFranceEntities>
{
    // Cette méthode n'est appelée qu'une seule fois pour initialiser les
    // stratégies au niveau des services.
    public static void InitializeService(IDataServiceConfiguration
    config)
    {
        config.SetEntitySetAccessRule("Stagiaire", EntitySetRights.All);
        config.SetEntitySetAccessRule("Cours", EntitySetRights.AllRead);

        config.SetServiceOperationAccessRule("GetNombreStagiairesInscrits",
        ServiceOperationRights.All);
    }

    [WebGet]
    public int GetNombreStagiairesInscrits()
    {
        return this.CurrentDataSource.Stagiaire
            .Count(oStagiaire => oStagiaire.Cours.Count > 0);
    }
}
```



```
' VB .NET

Public Class DataAcessssService
    Inherits DataService(Of DotnetFranceEntities)

    ' Cette méthode n'est appelée qu'une seule fois pour initialiser les
    stratégies au niveau des services.
    Public Shared Sub InitializeService(ByVal config As
IDataServiceConfiguration)
        config.SetEntitySetAccessRule("Stagiaire", EntitySetRights.All)
        config.SetEntitySetAccessRule("Cours", EntitySetRights.AllRead)

config.SetServiceOperationAccessRule("GetNombreStagiairesInscrits",
ServiceOperationRights.All)
    End Sub

    <WebGet()> _
    Public Function GetNombreStagiairesInscrits() As Integer
        Return Me.CurrentDataSource.Stagiaire.Count(Function(oStagiaire)
oStagiaire.Cours.Count > 0)
    End Function
End Class
```

4 Affichage et gestion des données

Dans le projet *Client_DataServices*, nous allons créer des formulaires d'affichage et de gestion des données contenues dans la base de données *DotnetFrance*, au travers de notre service de données ADO .NET.

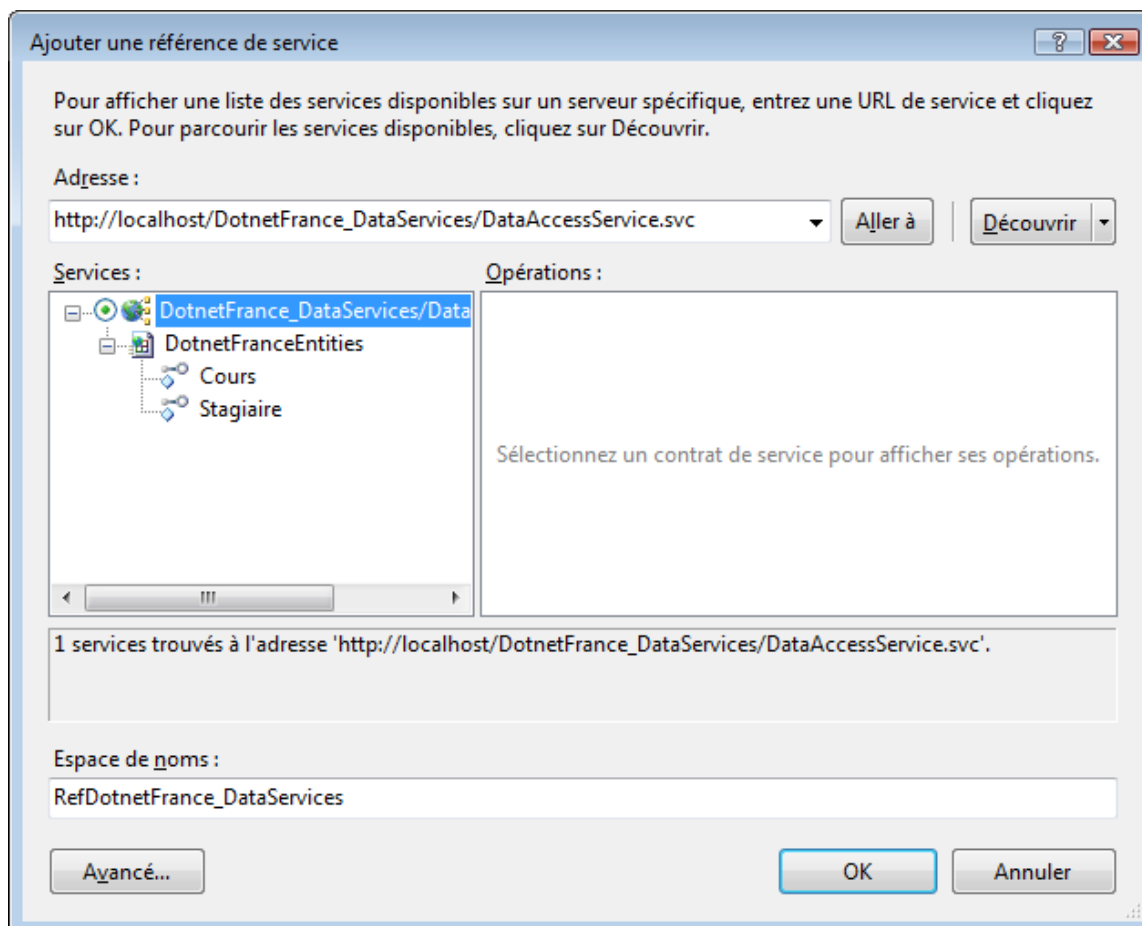
Le projet *Client_DataServices* est composé de deux formulaires :

- Un formulaire nommé *FrmGestionListeStagiaires*. Il permet de consulter et gérer la liste des stagiaires.
- Un formulaire nommé *FrmDetailStagiaire*, qui permet de consulter / modifier les informations relatives à un stagiaire. Ce formulaire sera aussi utilisé pour ajouter un stagiaire.

Afin de pouvoir utiliser les classes d'entité dans le projet *DotnetFrance_IHM*, il est nécessaire d'ajouter une référence vers l'assembly *System.Data.Entity.dll* du Framework .NET.

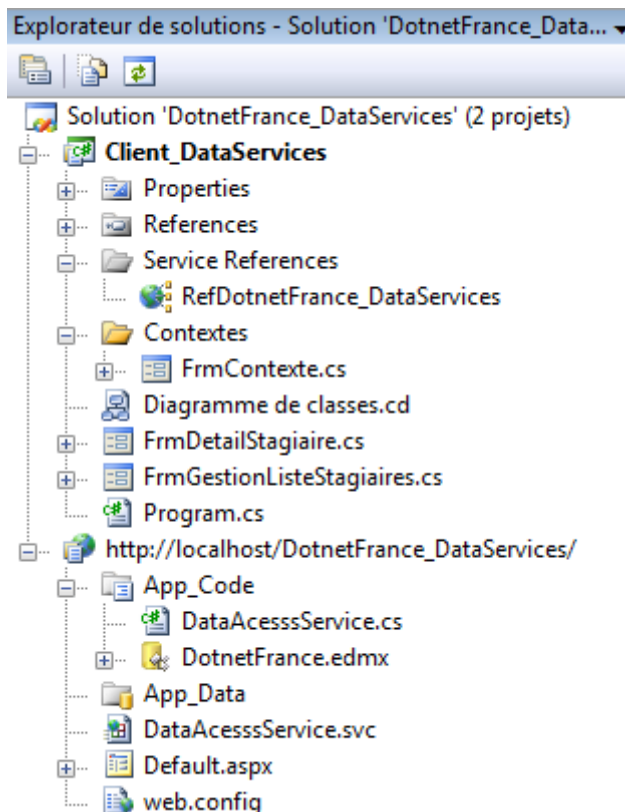
4.1 Création d'une référence de service

Pour accéder et modifier les données, l'application *Client_DataServices* devra consommer le service *DataAccessService* contenu dans l'application *DotnetFrance_DataServices*. Pour ce faire, il est nécessaire d'ajouter dans le projet *Client_DataServices* une référence de service vers ce même service. Dans l'explorateur de solution, affichons alors le menu contextuel du répertoire Service References, et cliquons sur l'item « Ajouter une référence de service ... ». La fenêtre suivante apparaît :



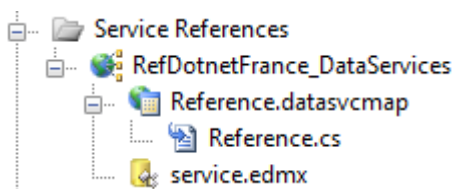
Dans cette fenêtre, recherchez ou spécifiez l'url du service de données à références. Puis, spécifier le nom de la référence de service : *RefDotnetFrance_DataServices*. Puis cliquez sur le bouton OK, pour valider la création de cette référence de service. Les modifications suivantes sont alors apportées au projet :

- Dans l'explorateur de solution :



- Ajout automatique des références suivantes vers des composants du Framework .NET :
 - o System.Data.Services.Client.dll
 - o System.Runtime.Serialization.dll

La référence de service créée ci-dessus crée différents fichiers dans l'application (affichez tous les fichiers dans l'explorateur de solutions pour l'observer :



Le fichier Reference.cs/Reference.vb contient trois classes proxy, contenues dans l'espace de noms *Client_DataServices.RefDotnetFrance_DataServices* : *DotnetFranceEntities*, *Stagiaire* et *Cours*. Ces classes exposent les classes d'entités exposées par le service de données ADO .NET. Dans les formulaires, nous manipulerons des objets créés à partir de ces classes, pour afficher et gérer les

données. Et à travers ces objets, nous consommerons le service de données ADO .NET, que nous venons de créer.

4.2 Création d'une classe de contexte pour les formulaires

Différents formulaires de notre application, vont partager des données et des fonctionnalités communes. Dans notre cas, nous allons partager une instance du contexte de données, que nous utiliserons pour lire et modifier les données.

Pour faciliter ce partage, nous allons créer une classe de contexte que nous appellerons *FrmContexte*. Cette classe dérive de la classe *System.Windows.Forms.Form* du Framework .NET, et sera la classe de base des deux autres formulaires de notre application. On obtient alors le diagramme de classes suivant :

...

A la racine du projet *Client_DataServices*, nous ajoutons un répertoire nommé Contexte, dans lequel nous ajoutons un formulaire Windows nommé *FrmContexte*. Chaque instance de l'application qui s'exécute, doit posséder une seule instance du contexte de données, cette instance devant être unique pour tous les formulaires de l'application. Ainsi, dans le formulaire *FrmContexte*, on ajoute le singleton suivant :

```
' VB .NET

Private Shared _ContexteDonnees As DotnetFranceEntities
Protected Shared Property ContexteDonnees() As DotnetFranceEntities
    Get
        If (FrmContexte._ContexteDonnees Is Nothing) Then
            FrmContexte._ContexteDonnees = New DotnetFranceEntities(New
Uri(ConfigurationManager.AppSettings("UrlServiceDonnees")))
        End If

        Return FrmContexte._ContexteDonnees
    End Get
    Private Set(ByVal value As DotnetFranceEntities)

    End Set
End Property
```

Le contexte de données de l'application est créé une seule fois, lors du premier accès. Vous remarquez qu'il est instancié avec une chaîne de caractère, correspondant à l'URL de notre service de données. Cette URL est définie dans le fichier de configuration de l'application. Ainsi, vous devez :

- Ajouter un fichier de configuration, et y définir le contenu suivant :

```
' Fichier de configuration

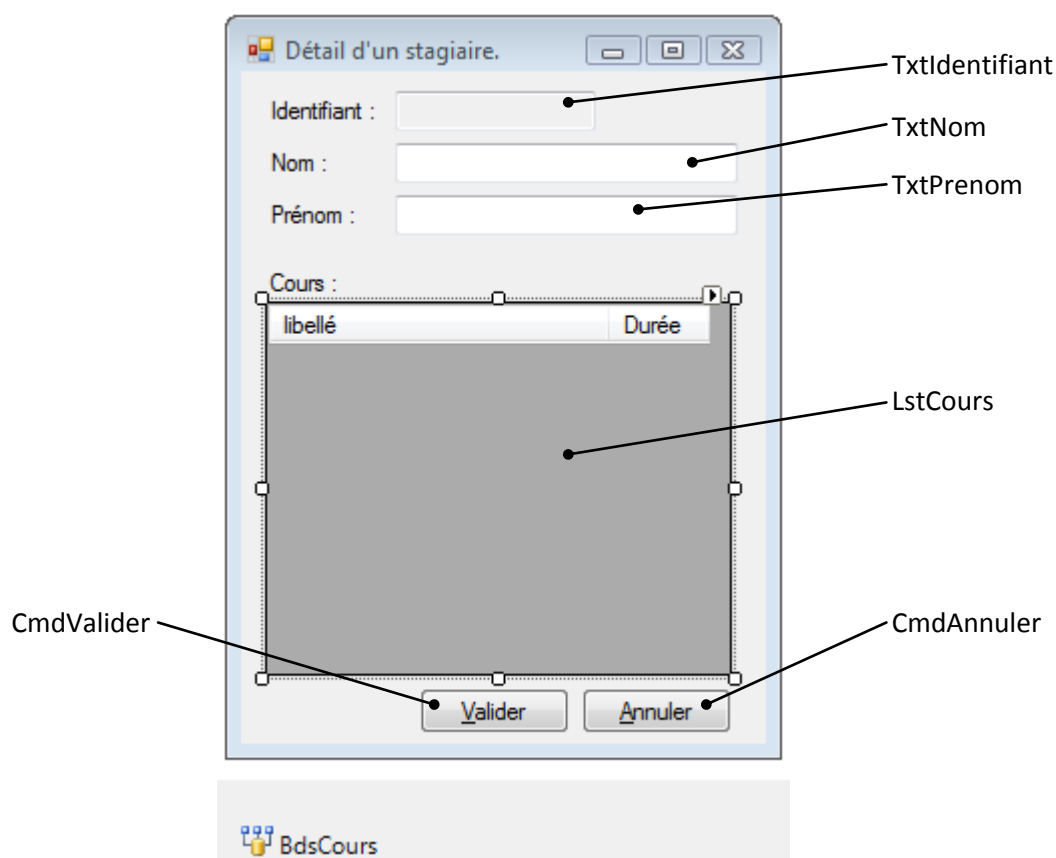
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="UrlServiceDonnees"
value="http://localhost/DotnetFrance_DataServices/DataAccessService.svc"/>
  </appSettings>
</configuration>
```

- Ajouter une référence vers le composant *System.Configuration.dll* du Framework .NET.
- Importer l'espace de nom *System.Configuration*, afin de pouvoir utiliser la classe *ConfigurationManager*.

4.3 Création du formulaire *FrmDetailStagiaire*

4.3.1 Design du formulaire

Voici une présentation de ce formulaire :



Voici quelques propriétés des contrôles de ce formulaire :

Contrôles	Propriétés	Valeurs
TxtIdentifiant		
	ReadOnly	True
LstCours		
	AllowUserToAddRows	False
	AllowUserToDeleteRows	False
	Columns	Ajout de deux colonnes : <ul style="list-style-type: none"> - Libelle : liée à la propriété <i>Libelle</i> - Durée : liée à la propriété <i>NombreJours</i>
	DataSource	BdsCours
	MultiSelect	False
	ReadOnly	True
	RowHeadersVisible	false
	SelectionMode	FullRowSelect

4.3.2 Code-behind du formulaire

Ce formulaire va « gérer » les informations concernant les stagiaires. Nous allons donc :

- Créer un attribut de type *Stagiaire*, avec son accesseur.
- Initialiser cet attribut dans le constructeur.
- Implémenter l'évènement *Load*, de manière à utiliser le databinding pour :

- Afficher l'identifiant. Dans le cas d'un ajout d'un stagiaire, les caractères « -- » seront affichés. En effet, la colonne *Identifiant* de la table *Stagiaire* étant une colonne identité, l'identifiant du stagiaire sera déterminé par SQL Server lors de l'ajout du stagiaire.
- Afficher le nom, prénom, et la liste des cours auxquels le stagiaire est inscrit.
- Implémenter l'évènement *Click* sur le bouton *CmdValider*, afin d'enregistrer que l'utilisateur confirme l'ajout / modification du stagiaire.
- Implémenter l'évènement *Click* sur le bouton *CmdAnnuler*, afin d'enregistrer que l'utilisateur annule l'ajout / modification du stagiaire.

Voici le code behind de ce formulaire :

```
// C#

public partial class FrmDetailStagiaire : FrmContexte
{
    private Stagiaire _oStagiaire;
    public Stagiaire oStagiaire
    {
        get { return _oStagiaire; }
        set { _oStagiaire = value; }
    }

    public FrmDetailStagiaire(Stagiaire aStagiaire)
    {
        InitializeComponent();

        // Initialisation des attributs.
        this.oStagiaire = aStagiaire;
    }

    private void FrmDetailStagiaire_Load(object sender, EventArgs e)
    {
        if (oStagiaire.Identifiant == 0)
        {
            TxtIdentifiant.Text = "--";
        }
        else
        {
            TxtIdentifiant.DataBindings.Add("Text", this.oStagiaire,
            "Identifiant");
        }

        TxtNom.DataBindings.Add("Text", this.oStagiaire, "Nom");
        TxtPrenom.DataBindings.Add("Text", this.oStagiaire, "Prenom");

        BdsCours.DataSource = this.oStagiaire.Cours;
    }

    private void CmdValider_Click(object sender, EventArgs e)
    {
        this.DialogResult = DialogResult.OK;
    }

    private void CmdAnnuler_Click(object sender, EventArgs e)
    {
        this.DialogResult = DialogResult.Cancel;
    }
}
```




```
' VB .NET

Public Class FrmDetailStagiaire

    Private _oStagiaire As Stagiaire
    Private Property oStagiaire() As Stagiaire
        Get
            Return Me._oStagiaire
        End Get
        Set(ByVal value As Stagiaire)
            Me._oStagiaire = value
        End Set
    End Property

    Public Sub New(ByVal aStagiaire As Stagiaire)

        ' Cet appel est requis par le Concepteur Windows Form.
        InitializeComponent()

        ' Ajoutez une initialisation quelconque après l'appel
        InitializeComponent().
        Me.oStagiaire = aStagiaire
    End Sub

    Private Sub FrmDetailStagiaire_Load(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles MyBase.Load
        If (oStagiaire.Identifiant = 0) Then
            TxtIdentifiant.Text = "--"
        Else
            TxtIdentifiant.DataBindings.Add("Text", Me.oStagiaire,
            "Identifiant")
        End If

        TxtNom.DataBindings.Add("Text", Me.oStagiaire, "Nom")
        TxtPrenom.DataBindings.Add("Text", Me.oStagiaire, "Prenom")

        BdsCours.DataSource = Me.oStagiaire.Cours
    End Sub

    Private Sub CmdValider_Click(ByVal sender As System.Object, ByVal e
    As System.EventArgs) Handles CmdValider.Click
        Me.DialogResult = DialogResult.OK
    End Sub

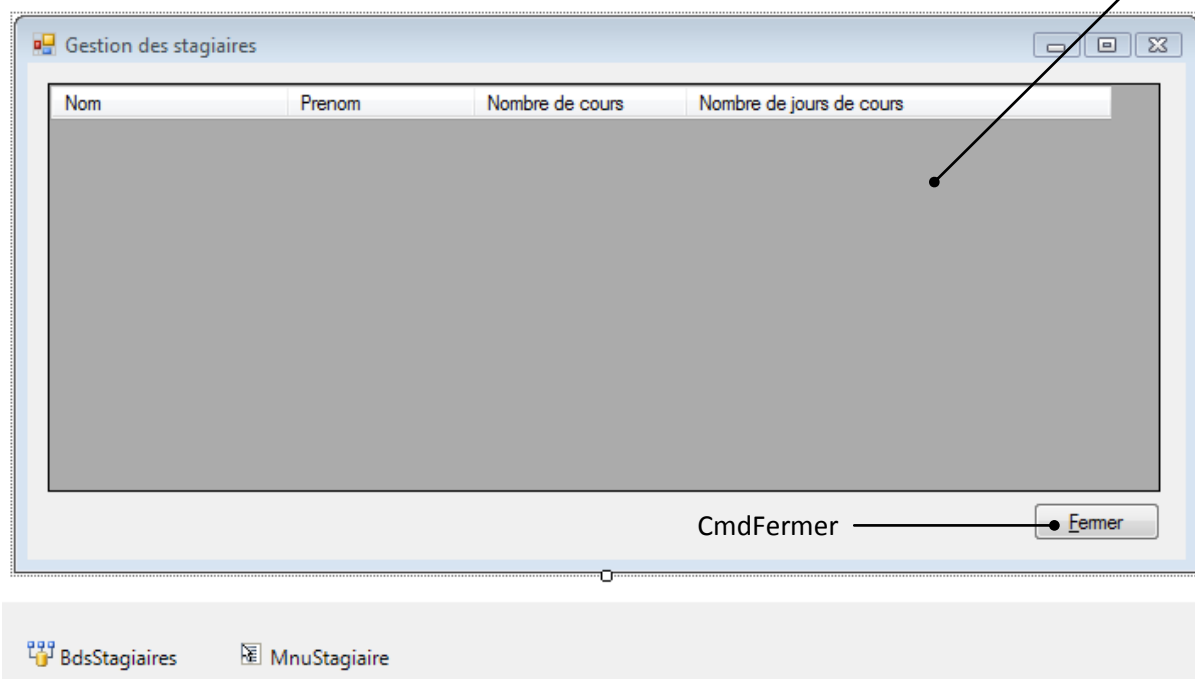
    Private Sub CmdAnnuler_Click(ByVal sender As System.Object, ByVal e
    As System.EventArgs) Handles CmdAnnuler.Click
        Me.DialogResult = DialogResult.Cancel
    End Sub

End Class
```

4.4 Création du formulaire *FrmGestionListeStagiaires*

4.4.1 Design du formulaire

Voici une présentation de ce formulaire :



Voici quelques propriétés des contrôles de ce formulaire :

Contrôles	Propriétés	Valeurs
LstStagiaires		
	AllowUserToAddRows	False
	AllowUserToDeleteRows	False
	Columns	Ajout de quatre colonnes : <ul style="list-style-type: none"> - Nom : liée à la propriété <i>Nom</i> - Prénom : liée à la propriété <i>Prenom</i> - Nombre de cours : <i>NombreCours</i> - Nombre de jours de cours : <i>NombreJoursDeCours</i>
	DataSource	BdsStagiaires
	MultiSelect	False
	ReadOnly	True
	RowHeadersVisible	False
	SelectionMode	FullRowSelect

Le contrôle de type *BindingSource* nommé *BdsStagiaire*, permet de simplifier la mise en œuvre du databinding dans notre formulaire. Nous l'utiliserons pour :

- Afficher la liste des stagiaires et des informations connexes.
- Ajouter / modifier / supprimer un stagiaire dans/de la liste.
- Obtenir à tout moment un objet métier (entité, objet fortement typé) correspondant au stagiaire sélectionné dans la liste.

Le contrôle de type *ContextMenuStrip* nommé *MnuStagiaire*, permet d'afficher un menu contextuel sur un stagiaire sélectionné dans la liste. Les items de ce menu permettent d'ajouter, modifier et supprimer un stagiaire.

4.4.2 Code-behind du formulaire

4.4.2.1 Pour obtenir l'entité du stagiaire courante

Nous allons créer un accesseur en lecture seule, permettant d'obtenir l'entité du stagiaire sélectionné dans la liste :

```
// C#  
  
private Stagiaire StagiaireCourant  
{  
    get  
    {  
        return (Stagiaire)BdsStagiaires.Current;  
    }  
}
```

```
' VB .NET  
  
Private ReadOnly Property StagiaireCourant() As Stagiaire  
    Get  
        Return CType(BdsStagiaires.Current, Stagiaire)  
    End Get  
End Property
```

4.4.2.2 Lors du chargement du formulaire

Lors du chargement du formulaire, obtenir et afficher la liste des stagiaires. Pour ce faire, nous faisons appel à la méthode statique *GetListeInstances* de la classe *Stagiaire* de notre modèle d'entités :

```
// C#  
  
private void FrmListeStagiaires_Load(object sender, EventArgs e)  
{  
    try  
    {  
        // Affichage de la liste des stagiaires.  
        BdsStagiaires.DataSource = ContexteDonnees.Stagiaire.ToList();  
    }  
    catch (Exception aEx)  
    {  
        MessageBox.Show(aEx.Message);  
    }  
}
```

```
' VB .NET

Private Sub FrmGestionListeStagiaires_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
    Try
        ' Affichage de la liste des stagiaires.
        BdsStagiaires.DataSource = ContexteDonnees.Stagiaire.ToList()
    Catch aEx As Exception
        MessageBox.Show(aEx.Message)
    End Try
End Sub
```

4.4.2.3 Pour fermer le formulaire

Voici l'implémentation de l'évènement Click sur le bouton intitulé Fermer :

```
// C#

private void CmdFermer_Click(object sender, EventArgs e)
{
    // Fermeture du formulaire.
    this.Close();
}
```

```
' VB .NET

Private Sub CmdFermer_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles CmdFermer.Click
    ' Fermeture du formulaire.
    Me.Close()
End Sub
```

4.4.2.4 Gestion de l'affichage du menu

Le menu contextuel sur un stagiaire doit être affiché, uniquement un stagiaire est sélectionné dans la liste. Voici l'implémentation de l'évènement *Opening* sur le menu *MnuStagiaire* :

```
// C#

private void MnuStagiaire_Opening(object sender, CancelEventArgs e)
{
    // Si aucun stagiaire n'est sélectionné, alors on n'affiche pas le
    menu contextuel.
    e.Cancel = this.StagiaireCourant == null;
}
```

```
' VB .NET

Private Sub MnuStagiaire_Opening(ByVal sender As System.Object, ByVal e
As System.ComponentModel.CancelEventArgs) Handles MnuStagiaire.Opening
    ' Si aucun stagiaire n'est sélectionné, alors on n'affiche pas le
    menu contextuel.
    e.Cancel = Me.StagiaireCourant Is Nothing
End Sub
```

4.4.2.5 Ajouter un stagiaire

Pour ajouter un stagiaire, nous créons une instance de classe *Stagiaire* de notre modèle d'entités. Puis, nous gérons les données contenues dans cet objet, via une instance du formulaire *FrmDetailStagiaire*. Après la fermeture cette fenêtre, si l'utilisateur a cliqué sur le bouton *Valider*, alors le stagiaire est ajouté dans la table *Stagiaire* du contexte de données, puis enregistré dans la base de données.

```
// C#

private void ajouterToolStripMenuItem_Click(object sender, EventArgs e)
{
    // Variables locales.
    Stagiaire oStagiaire;
    FrmDetailStagiaire oForm;
    DialogResult oResult;

    try
    {
        // Initialisation.
        oStagiaire = new Stagiaire();

        // Création du formulaire de détail sur un stagiaire.
        oForm = new FrmDetailStagiaire(oStagiaire);

        // Affichage.
        oResult = oForm.ShowDialog();

        if (oResult == DialogResult.OK)
        {
            // Ajout du stagiaire en base de données.
            ContexteDonnees.AddToStagiaire(oStagiaire);
            ContexteDonnees.SaveChanges();

            // Affichage du stagiaire dans la grille.
            BdsStagiaires.Add(oStagiaire);
        }
    }
    catch (Exception aEx)
    {
        MessageBox.Show(aEx.Message);
    }
}
```

```
' VB .NET

Private Sub ajouterToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles ajouterToolStripMenuItem.Click
    ' Variables locales.
    Dim oStagiaire As Stagiaire
    Dim oForm As FrmDetailStagiaire
    Dim oResult As DialogResult

    Try
        ' Initialisation.
        oStagiaire = New Stagiaire()

        ' Création du formulaire de détail sur un stagiaire.
        oForm = New FrmDetailStagiaire(oStagiaire)

        ' Affichage.
        oResult = oForm.ShowDialog()

        If (oResult = DialogResult.OK) Then
            ' Ajout du stagiaire en base de données.
            ContexteDonnees.AddToStagiaire(oStagiaire)
            ContexteDonnees.SaveChanges()

            ' Affichage du stagiaire dans la grille.
            BdsStagiaires.Add(oStagiaire)
        End If
    Catch aEx As Exception
        MessageBox.Show(aEx.Message)
    End Try
End Sub
```

4.4.2.6 Modifier un stagiaire

Pour modifier un stagiaire, nous récupérons une instance de classe *Stagiaire* via notre accesseur *StagiaireCourant*. Puis, nous gérons les données contenues dans cet objet, via une instance du formulaire *FrmDetailStagiaire*. Après la fermeture cette fenêtre, si l'utilisateur a cliqué sur le bouton *Valider*, alors les modifications effectuées sur le stagiaire sont enregistrées dans le contexte de données, puis persistées dans la base de données. Le cas échéant, ces modifications effectuées sur les propriétés du stagiaire sont annulées.

```
// C#

private void modifierToolStripMenuItem_Click(object sender, EventArgs e)
{
    // Variables locales.
    Stagiaire oStagiaire;
    FrmDetailStagiaire oForm;
    DialogResult oResult;

    try
    {
        // Initialisation.
        oStagiaire = this.StagiaireCourant;

        if (oStagiaire != null)
        {
            // Création du formulaire de détail sur un stagiaire.
            oForm = new FrmDetailStagiaire(oStagiaire);

            // Affichage.
            oResult = oForm.ShowDialog();

            if (oResult == DialogResult.OK)
            {
                // Enregistrement des modifications dans la base de
données.

                ContexteDonnees.UpdateObject(oStagiaire);
                ContexteDonnees.SaveChanges();
            }
            else
            {
                // Annulation des modifications.
                ContexteDonnees.LoadProperty(oStagiaire, "Nom");
                ContexteDonnees.LoadProperty(oStagiaire, "Prenom");
                BdsStagiaires.ResetCurrentItem();
            }
        }
    }
    catch (Exception aEx)
    {
        MessageBox.Show(aEx.Message);
    }
}
```

```
' VB .NET

Private Sub modifierToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
modifierToolStripMenuItem.Click
    ' Variables locales.
    Dim oStagiaire As Stagiaire
    Dim oForm As FrmDetailStagiaire
    Dim oResult As DialogResult

    Try
        ' Initialisation.
        oStagiaire = Me.StagiaireCourant

        ' Création du formulaire de détail sur un stagiaire.
        oForm = New FrmDetailStagiaire(oStagiaire)

        ' Affichage.
        oResult = oForm.ShowDialog()

        If (oResult = DialogResult.OK) Then
            ' Enregistrement des modifications dans la base de données.
            ContexteDonnees.UpdateObject(oStagiaire)
            ContexteDonnees.SaveChanges()
        Else
            ' Annulation des modifications.
            ContexteDonnees.LoadProperty(oStagiaire, "Nom");
            ContexteDonnees.LoadProperty(oStagiaire, "Prenom");
            BdsStagiaires.ResetCurrentItem();
        End If
    Catch aEx As Exception
        MessageBox.Show(aEx.Message)
    End Try
End Sub
```

4.4.2.7 Supprimer un stagiaire

Pour supprimer un stagiaire, nous récupérons une instance de classe *Stagiaire* via notre accesseur *StagiaireCourant*. Une demande de confirmation est effectuée. Si l'utilisateur valide son choix de suppression, alors le stagiaire est supprimé de la base de données.

Notez que dans la base de données, toutes les relations d'association entres les tables de notre modèle acceptent la suppression en cascade. De ce fait, nous ne supprimons pas les inscriptions des stagiaires aux cours, avant de supprimer un stagiaire.



```
// C#

private void supprimerToolStripMenuItem_Click(object sender, EventArgs e)
{
    // Variables locales.
    Stagiaire oStagiaire;
    DialogResult oResult;

    try
    {
        // Initialisation.
        oStagiaire = this.StagiaireCourant;

        if (oStagiaire != null)
        {
            // Demande de confirmation.
            oResult = MessageBox.Show("Etes-vous sûrs de vouloir
supprimer ce stagiaire ?", "Demande de confirmation",
MessageBoxButtons.YesNo, MessageBoxIcon.Question);

            if (oResult == DialogResult.Yes)
            {
                // Suppression du stagiaire dans la base de données.
                oStagiaire.Supprimer();

                // Suppression du stagiaire dans la grille.
                BdsStagiaires.RemoveCurrent();
            }
        }
    }
    catch (Exception aEx)
    {
        MessageBox.Show(aEx.Message);
    }
}
```



```
' VB .NET

Private Sub supprimerToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
supprimerToolStripMenuItem.Click
    ' Variables locales.
    Dim oStagiaire As Stagiaire
    Dim oResult As DialogResult

    Try
        ' Initialisation.
        oStagiaire = Me.StagiaireCourant

        If (oStagiaire IsNot Nothing) Then
            ' Demande de confirmation.
            oResult = MessageBox.Show("Etes-vous sûrs de vouloir
supprimer ce stagiaire ?", "Demande de confirmation",
MessageBoxButtons.YesNo, MessageBoxIcon.Question)

            If (oResult = DialogResult.Yes) Then
                ' Suppression du stagiaire dans la base de données.
                ContexteDonnees.DeleteObject(oStagiaire)
                ContexteDonnees.SaveChanges()

                ' Suppression du stagiaire dans la grille.
                BdsStagiaires.RemoveCurrent()
            End If
        End If
    Catch aEx As Exception
        MessageBox.Show(aEx.Message)
    End Try
End Sub
```

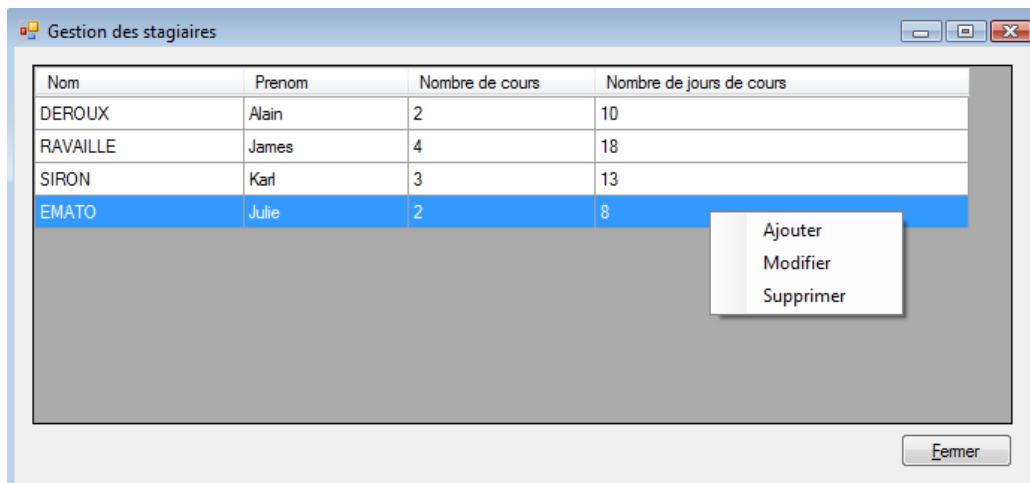
5 Exécution de l'application

5.1 Exécution des fonctionnalités de l'application

Avant d'effectuer les manipulations présentées ci-dessous, nous activons les traces HTTP dans l'application *DotnetFrance_DataServices*. Dans le fichier de configuration Web de cette application Web, dans l'élément *system.web* :

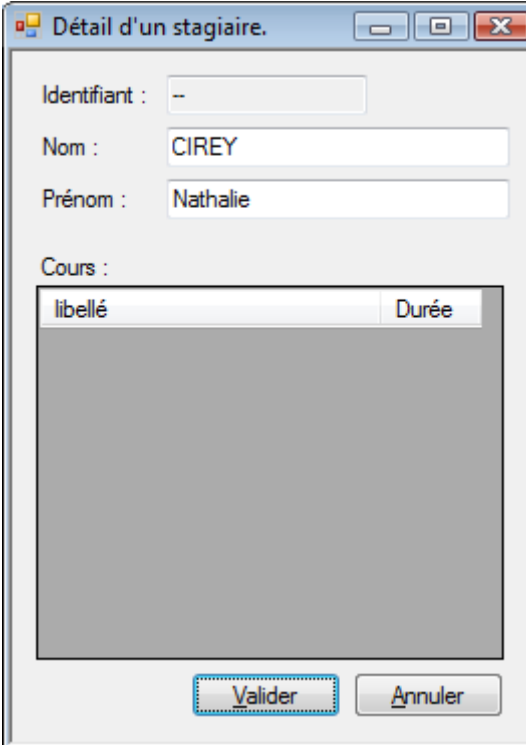
```
' Fichier de configuration  
<trace enabled="true" />
```

Lors de l'exécution de l'application, la fenêtre de gestion des stagiaires apparaît :



5.1.1 Ajout d'un stagiaire

En cliquant sur l'item Ajouter, la fenêtre suivante apparaît. Nous spécifions un nom et un prénom, puis nous validons :



Détail d'un stagiaire.

Identifiant : --

Nom : CIREY

Prénom : Nathalie

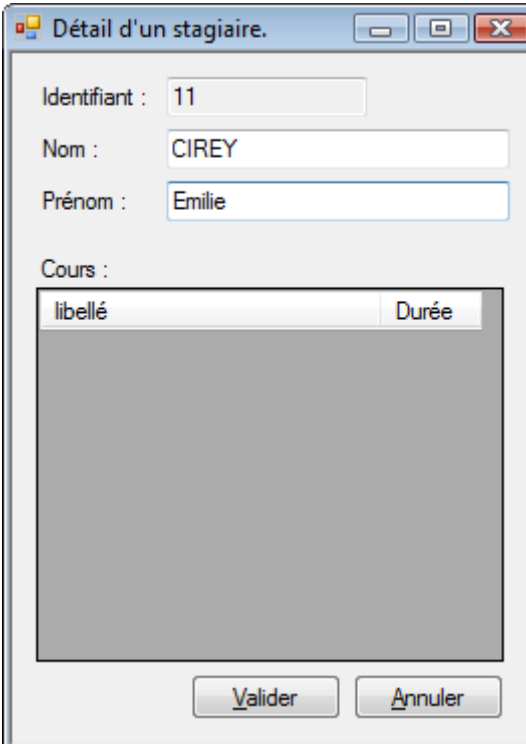
Cours :

libellé	Durée
---------	-------

Valider Annuler

5.1.2 Modification d'un stagiaire

Puis en cliquant sur l'item Modifier du menu contextuel sur le stagiaire que nous venons de créer, nous pouvons modifier son prénom :



Détail d'un stagiaire.

Identifiant : 11

Nom : CIREY

Prénom : Emilie

Cours :

libellé	Durée
---------	-------

Valider Annuler

5.1.3 Suppression d'un stagiaire

Puis en cliquant sur l'item Supprimer du menu contextuel sur le stagiaire que nous venons de créer, nous le supprimons.

5.2 Observation des traces HTTP

Après ces manipulations, dans un navigateur Web, nous pouvons visualiser les traces HTTP enregistrées via l'URL suivante : http://localhost/DotnetFrance_DataServices/Trace.axd. On obtient le résultat suivant :

Trace de l'application DotnetFrance_DataServices

[[effacer la trace actuelle](#)]

Répertoire physique : F:\Dotnet-France\Refonte des cours\ADO .NET et LINQ\ADO.NET Data Services\Premiers pas avec ADO .NET Data Services\Developpement\CS\DotnetFrance_DataServices\

Demandes à cette application					Restant : 6
N°	Heure de la demande	Fichier	Code d'état	verbe	
1	01/03/2009 23:42:55	/DataAccessService.svc/Stagiaire	200	GET	Afficher les détails
2	01/03/2009 23:43:06	/DataAccessService.svc/Stagiaire	201	POST	Afficher les détails
3	01/03/2009 23:43:12	/DataAccessService.svc/Stagiaire (12)	204	MERGE	Afficher les détails
4	01/03/2009 23:43:15	/DataAccessService.svc/Stagiaire (12)	204	DELETE	Afficher les détails

On peut observer que la première requête HTTP envoyée, permet d'accéder à la liste des stagiaires, contenue dans la base de données.

La seconde requête permet d'ajouter un stagiaire dans la base de données.

La troisième permet de persister les modifications effectuées sur un stagiaire, dont l'identifiant est 12.

La dernière permet de supprimer un stagiaire, dont l'identifiant est 12.

6 Conclusion

Tout au long de ce cours, nous avons vu comment :

- Créer un modèle de données avec Entity Data Model (EDM).
- Exposer des données au travers de service ADO .NET.

Ce cours constitue une entrée en matière sur ADO .NET Data Services. Les prochains cours publiés sur Dotnet-France détailleront différents points simplement présentés dans ce support (sécurité, requêtes d'accès aux données définition des droits sur les données, ...).