Dotnet France
Technologies Sharepoint, SQL Server & .NET

Association Dotnet France

ASP .NET Dynamic Data : premiers pas

Version 1.0

James RAVAILLE
http://blogs.dotnet-france.com/jamesr

Sommaire

1 Présentation du projet à créer

1.1 Présentation générale

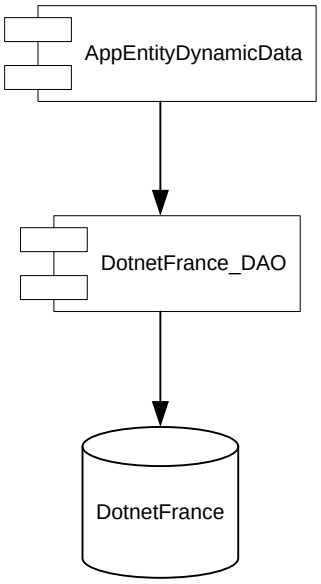
L'application que nous allons développer, permet gérer des stagiaires et les cours auxquels ils sont inscrits. Elle devra permettre de naviguer au travers des données, et de pourvoir gérer toute donnée en mode CRUD (Create, Read, Update et Delete).

1.2 Composants de l'application

Cette application est constituée de deux projets :

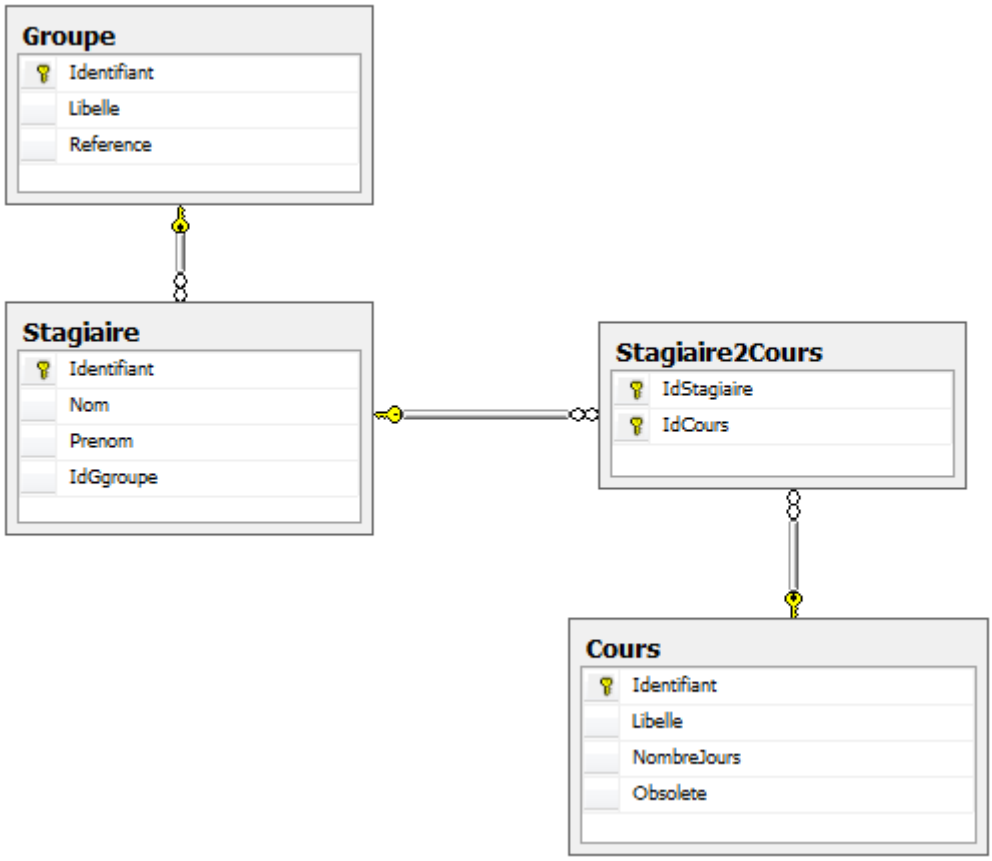
- Un projet contenant le code d'accès aux données, nommé *DotnetFrance_DAO*.
- Un projet permettant d'afficher les données, et recueillir les demandes de l'utilisateur (navigation et gestion de données), nommé *AppEntityDynamicData*.

Voici un schéma décrivant l'architecture de l'application, que nous allons réaliser :



1.3 Création de la base de données

Voici le schéma de la base de données DotnetFrance :



Voici quelques explications sur ce schéma :

- Un groupe est un ensemble de stagiaires.
- Un stagiaire peut être inscrit à plusieurs cours.
- Plusieurs stagiaires peuvent suivre un cours.

Voici un script permettant de créer ce schéma :

```
-- SQL

USE [DotnetFrance]
GO

/** Object:  Table [dbo].[Groupe]    Script Date: 05/19/2009 23:35:49 **/
CREATE TABLE [dbo].[Groupe](
      [Identifiant] [int] IDENTITY(1,1) NOT NULL,
      [Libelle] [varchar](50) NOT NULL,
      [Reference] [varchar](10) NULL,
 CONSTRAINT [PK_Groupe] PRIMARY KEY CLUSTERED
(
      [Identifiant] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

```
-- SQL

/****** Object:  Table [dbo].[Cours]    Script Date: 05/19/2009 23:35:49
******/
CREATE TABLE [dbo].[Cours](
	[Identifiant] [int] IDENTITY(1,1) NOT NULL,
	[Libelle] [varchar](50) NOT NULL,
	[NombreJours] [int] NOT NULL,
	[Obsolete] [bit] NULL,
 CONSTRAINT [PK_Cours] PRIMARY KEY CLUSTERED
(
	[Identifiant] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY],
 CONSTRAINT [UQ__Cours__023D5A04] UNIQUE NONCLUSTERED
(
	[Identifiant] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

/****** Object:  Table [dbo].[Stagiaire]    Script Date: 05/19/2009
23:35:49 ******/
CREATE TABLE [dbo].[Stagiaire](
	[Identifiant] [int] IDENTITY(1,1) NOT NULL,
	[Nom] [varchar](50) NOT NULL,
	[Prenom] [varchar](50) NOT NULL,
	[IdGgroupe] [int] NULL,
 CONSTRAINT [PK_Stagiaire] PRIMARY KEY CLUSTERED
(
	[Identifiant] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY],
 CONSTRAINT [UQ__Stagiaire__7F60ED59] UNIQUE NONCLUSTERED
(
	[Identifiant] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

/****** Object:  Table [dbo].[Stagiaire2Cours]    Script Date: 05/19/2009
23:35:49 ******/
CREATE TABLE [dbo].[Stagiaire2Cours](
	[IdStagiaire] [int] NOT NULL,
	[IdCours] [int] NOT NULL,
 CONSTRAINT [PK_Stagiaire2Cours] PRIMARY KEY CLUSTERED
(
	[IdStagiaire] ASC,
	[IdCours] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

```
-- SQL

/****** Object:  ForeignKey [FK_Stagiaire_Groupe1]    Script Date:
05/19/2009 23:35:49 ******/
ALTER TABLE [dbo].[Stagiaire]  WITH CHECK ADD  CONSTRAINT
[FK_Stagiaire_Groupe1] FOREIGN KEY([IdGgroupe])
REFERENCES [dbo].[Groupe] ([Identifiant])
ON UPDATE CASCADE
ON DELETE CASCADE
GO
ALTER TABLE [dbo].[Stagiaire] CHECK CONSTRAINT [FK_Stagiaire_Groupe1]
GO

/****** Object:  ForeignKey [FK_Stagiaire2Cours_Cours]    Script Date:
05/19/2009 23:35:49 ******/
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] CHECK CONSTRAINT
[FK_Stagiaire2Cours_Cours]
GO

/****** Object:  ForeignKey [FK_Stagiaire2Cours_Stagiaire]    Script
Date: 05/19/2009 23:35:49 ******/
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
ALTER TABLE [dbo].[Stagiaire2Cours] CHECK CONSTRAINT
[FK_Stagiaire2Cours_Stagiaire]
GO
```

Aussi, sur le serveur SQL Server, nous créons un utilisateur nommé *userTest*, ayant les doits permettant d'accéder à toutes les tables de la base de donnes DotnetFrance, afin de pouvoir les consulter, ajouter / modifier et supprimer des données. Le mot de passe de cet utilisateur est *passwd*.

Voici un script permettant d'alimenter cette base de données :

```
-- SQL

SET IDENTITY_INSERT Groupe ON
INSERT Groupe (Identifiant, Libelle, Reference) VALUES (1, N'Groupe n°1',
N'GP010')
INSERT Groupe (Identifiant, Libelle, Reference) VALUES (2, N'Groupe n°2',
N'GP055')
INSERT Groupe (Identifiant, Libelle, Reference) VALUES (3, N'Groupe n°3',
N'GP012')
INSERT Groupe (Identifiant, Libelle, Reference) VALUES (4, N'Groupe n°4',
N'GP34')
SET IDENTITY_INSERT Groupe OFF
```

```
-- SQL

SET IDENTITY_INSERT Cours ON
INSERT Cours (Identifiant, Libelle, NombreJours, Obsolete) VALUES (1,
N'SQL Server - Administration de serveurs', 3, 0)
INSERT Cours (Identifiant, Libelle, NombreJours, Obsolete) VALUES (2,
N'XHTML / CSS', 3, NULL)
INSERT Cours (Identifiant, Libelle, NombreJours, Obsolete) VALUES (3,
N'C#', 5, NULL)
INSERT Cours (Identifiant, Libelle, NombreJours, Obsolete) VALUES (4,
N'ASP .NET 3.5', 5, NULL)
INSERT Cours (Identifiant, Libelle, NombreJours, Obsolete) VALUES (5,
N'ASP .NET AJAX', 3, NULL)
INSERT Cours (Identifiant, Libelle, NombreJours, Obsolete) VALUES (7,
N'SQL Server - Administration de serveurs', 5, NULL)
INSERT Cours (Identifiant, Libelle, NombreJours, Obsolete) VALUES (8,
N'XHTML / CSS', 3, NULL)
SET IDENTITY_INSERT Cours OFF

SET IDENTITY_INSERT Stagiaire ON
INSERT Stagiaire (Identifiant, Nom, Prenom, IdGgroupe) VALUES (1,
N'DEROUX', N'Alain', 1)
INSERT Stagiaire (Identifiant, Nom, Prenom, IdGgroupe) VALUES (2,
N'RAVAILLE', N'James', 1)
INSERT Stagiaire (Identifiant, Nom, Prenom, IdGgroupe) VALUES (3,
N'SIRON', N'Karl', 1)
INSERT Stagiaire (Identifiant, Nom, Prenom, IdGgroupe) VALUES (4,
N'EMATO', N'Julie', 2)
INSERT Stagiaire (Identifiant, Nom, Prenom, IdGgroupe) VALUES (47,
N'VERON', N'Hamed', 2)
INSERT Stagiaire (Identifiant, Nom, Prenom, IdGgroupe) VALUES (48,
N'RIVEAU', N'Alain', 2)
INSERT Stagiaire (Identifiant, Nom, Prenom, IdGgroupe) VALUES (52,
N'DEJOIE', N'Patrick', 3)
SET IDENTITY_INSERT Stagiaire OFF

INSERT Stagiaire2Cours (IdStagiaire, IdCours) VALUES (1, 1)
INSERT Stagiaire2Cours (IdStagiaire, IdCours) VALUES (1, 3)
INSERT Stagiaire2Cours (IdStagiaire, IdCours) VALUES (2, 1)
INSERT Stagiaire2Cours (IdStagiaire, IdCours) VALUES (2, 2)
INSERT Stagiaire2Cours (IdStagiaire, IdCours) VALUES (2, 3)
INSERT Stagiaire2Cours (IdStagiaire, IdCours) VALUES (2, 4)
INSERT Stagiaire2Cours (IdStagiaire, IdCours) VALUES (3, 1)
INSERT Stagiaire2Cours (IdStagiaire, IdCours) VALUES (3, 4)
INSERT Stagiaire2Cours (IdStagiaire, IdCours) VALUES (3, 5)
INSERT Stagiaire2Cours (IdStagiaire, IdCours) VALUES (4, 4)
INSERT Stagiaire2Cours (IdStagiaire, IdCours) VALUES (4, 5)
```

2 Création du composant d'accès aux données

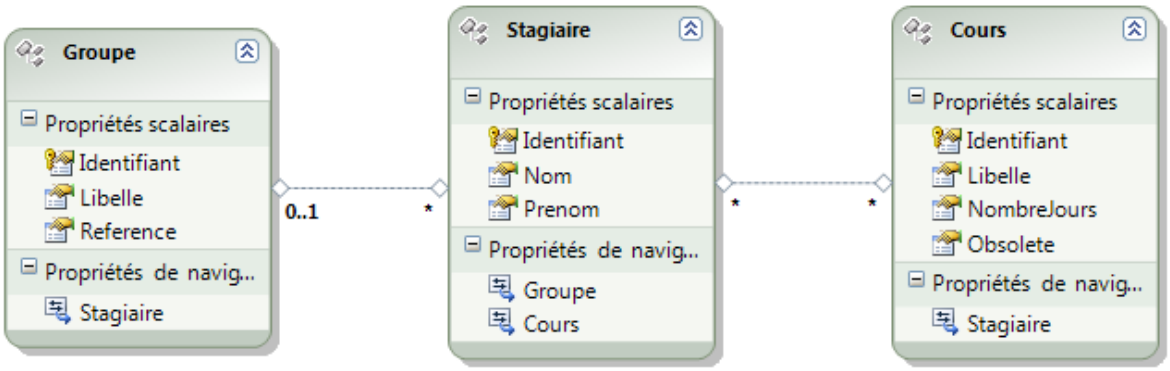
2.1 Création du projet

Commençons par créer un projet de type *bibliothèque de classes*, que nous appellerons *DotnetFrance_DAO*. Ce projet contiendra le modèle d'entités, ainsi que toutes classes permettant d'étendre les classes qu'il contient.

Ce modèle d'entités servira de modèle de base pour notre application Web de gestion de données créée avec ASP .NET Dynamic Data.

2.2 Concernant la création du modèle d'entités

Dans ce chapitre, nous n'allons pas nous étendre sur la création du modèle d'entités, permettant de mettre en œuvre le Framework Entity dans notre application. Cette création est notamment explicitée dans le cours « Framework Entity : premiers pas », publié dans la catégorie de cours ADO .NET sur Dotnet-France, que nous vous invitons à lire. Ainsi, dans ce cours, nous fournirons directement le modèle d'entités. Pour le créer, nous ajoutons et configurons un composant de type *Entity Data Model* nommé DotnetFrance, et nous obtenons le résultat suivant :



Nous n'apportons aucune modification particulière (aucune personnalisation) au modèle d'entités généré par Visual Studio. Si vous souhaitez le personnaliser nous vous invitons à lire le cours « Framework Entity : personnaliser le modèle d'entités », publié dans la catégorie de cours ADO .NET sur Dotnet-France.

3 Création d'un projet ASP .NET Dynamic Data

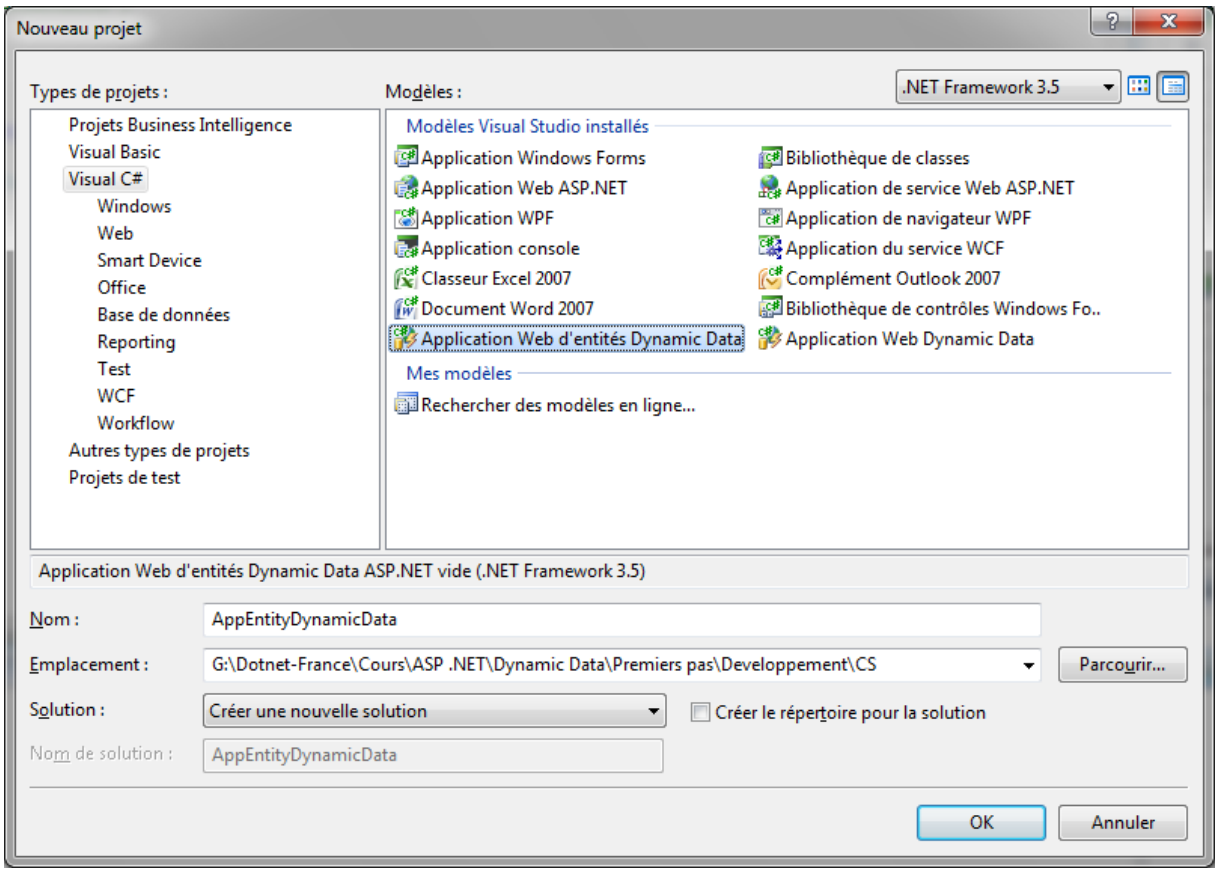
Comme pour tout projet Web, il est possible de créer une application Web ou un site Web de type Dynamic Data (je vous invite à consulter les cours ASP .NET, pour en savoir plus sur les différences entre les applications Web et les sites Web).

Pour rappel, lors de la création d'une application Web ou d'un site Web, il est possible de créer deux types de projets Web Dynamic Data :

- Applications Web Dynamic Data : ce type de projet permet de créer une application ASP .NET Dynamic Data, qui s'appuiera sur le modèle de données de type Linq To SQL (à savoir qu'il nécessite l'utilisation du composant Classes Linq To SQL). Dans les pages ASP .NET, pour accéder aux données, le contrôle d'accès aux données *LinqDataSource* sera utilisé.
- Applications Web d'entités Dynamic Data : ce type de projet permet de créer une application ASP .NET Dynamic Data, qui s'appuiera sur le modèle de données de type Framework Entity (à savoir qu'il nécessite l'utilisation du composant Entity Data Model). Dans les pages ASP .NET, pour accéder aux données, le contrôle d'accès aux données *EntityDataSource* sera utilisé.

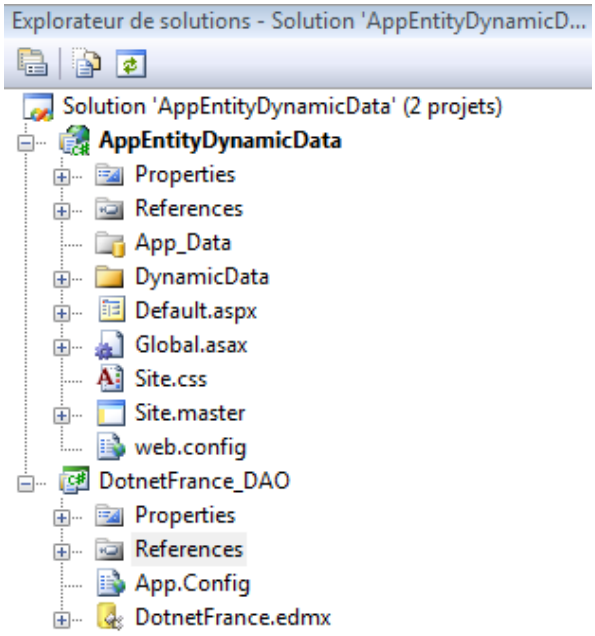
Au delà du contrôle source de données différent entre les deux types de projet Web Dynamic Data, il existe de nettes différences entre une application Dynamic Data s'appuyant sur Linq To SQL et une autre s'appuyant sur le Framework entity, notamment dans la personnalisation de l'application Web générée.

Dans le développement notre application, nous nous appuyons sur un modèle de données de type Entity Framework. Ainsi, nous allons créons une application Web d'entités Dynamic Data :



3.1 Composition de l'application

Une fois le projet créé, nous pouvons observer dans l'explorateur de solutions qu'il contient de nombreux fichiers :



- Un fichier de configuration, qui présente quelques différences par rapport à un fichier de configuration d'une application .NET standard. Nous présenterons ces différences ultérieurement.
- Une feuille de style CSS nommée *Site.css* permettant de mettre en forme et structurer les pages Web dans le navigateur.
- Le fichier *global.asax* permettant de définir lors du démarrage de l'application, le modèle de données (d'entités) à utiliser.
- Une page *Default.aspx*, permettant d'accéder à l'ensemble des entités de notre modèle de données. Il s'agit du point d'entrée de l'application.
- Un répertoire nommé *DynamicData* contenant tous fichiers permettant de parcourir, afficher et modifier les données. Il contient des pages et contrôles utilisateurs.

Le répertoire *DynamicData\PageTemplates* contient les modèles de pages de l'application. Il s'agit de pages ASP .NET utilisées lors de l'exécution de l'application, permettant de naviguer, consulter, ajouter, modifier … les données des entités. Ces modèles de page sont les suivants :

- *List.aspx* : cette page permet d'afficher l'ensemble des données d'une entité du méta-modèle dans une grille de données (contrôle *GridView*). Elle propose des contrôles des contrôles de type *DropDownList* permettant de filtrer les données pour chaque colonne constituant une clé étrangère ou booléenne. Elle propose aussi des possibilités de tri et de pagination. Il s'agit du modèle de page utilisé par défaut.
- *ListDetails.aspx* : cette page propose des fonctionnalités similaires à la page *List.aspx*. En supplément, elle propose et un contrôle de type *DetailsView* pour afficher les données de la ligne sélectionnée, ainsi que pour l'ajout d'un nouvel enregistrement.
- *Details.aspx* : cette page permet d'afficher le détail d'une ligne, via un contrôle de type *DetailsView*, dans le seul but de les consulter.
- *Edit*.aspx : cette page permet d'afficher le détail d'une ligne, via un contrôle de type *DetailsView*, dans le cadre d'une modification.
- *Insert.aspx* : cette page permet de créer une nouvelle instance d'une entité et de la persister dans la base de données.

Ces pages sont des pages de contenues, qui s'exécutent au sein d'une Master Page.

Lors de l'exécution de l'application, nous utiliserons ces pages, pour parcourir et modifier toutes les données du modèle. Dans le chapitre suivant intitulé « ASP .NET Dynamic Data : développement avancé », nous verrons qu'il est possible de personnaliser ces pages.

3.2 Ajout de références au projet

Dans notre projet ASP .NET Dynamic Data, nous ajoutons une référence vers le projet *DotnetFrance_DAO*. L'ajout de cette référence a pour conséquence de devoir ajouter une autre référence, vers le composant *System.Data.Entity.dll* du Framework .NET.

3.3 Définition du modèle de données

Positionnons-nous dans la méthode *RegisterRoutes* du fichier *global.asax*. Elle est appelée lors du démarrage de l'application. Son contenu initial est le suivant :

```
// C#

public class Global : System.Web.HttpApplication
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        MetaModel model = new MetaModel();

        model.RegisterContext(typeof(DotnetFranceEntities), new
ContextConfiguration() { ScaffoldAllTables = true });

        routes.Add(new DynamicDataRoute("{table}/{action}.aspx")
        {
            Constraints = new RouteValueDictionary(new { action =
"List|Details|Edit|Insert" }),
            Model = model
        });
    }

    void Application_Start(object sender, EventArgs e)
    {
        RegisterRoutes(RouteTable.Routes);
    }
}
```

```
' VB

Public Shared Sub RegisterRoutes(ByVal routes As RouteCollection)
    Dim model As New MetaModel()

    model.RegisterContext(GetType(DotnetFranceEntities), New
ContextConfiguration() With {.ScaffoldAllTables = True})

    routes.Add(New DynamicDataRoute("{table}/{action}.aspx") With { _
        .Constraints = New RouteValueDictionary(New With {.Action =
"List|Details|Edit|Insert"}), _
        .Model = model _
    })
End Sub

Private Sub Application_Start(ByVal sender As Object, ByVal e As
EventArgs)
    RegisterRoutes(RouteTable.Routes)
End Sub
```

Cette méthode possède trois instructions, dont l'écriture ne vous est peut être pas familière. Elle utilise les nouveautés des langages C# 3.0 et VB 9.0. Pour en savoir plus sur les nouveautés sur ces langages, nous vous invitons à consulter les cours publiés sur Dotnet-France. Voici une brève description de ces instructions :

- La première instruction permet de créer une instance du méta-modèle de données.
- La seconde instruction permet :
    - De définir le modèle de données du méta-modèle.
    - De créer et configurer le méta modèle. La propriété *ScaffoldAllTables* obtient ou définit une valeur, qui indique si la génération de modèles automatique est activée. Sa valeur par défaut est *False*. Si cette propriété n'est pas valorisée à *True*, et que nous souhaitons consulter la liste des entités du méta-modèle, alors la chaîne une exception de type *InvalidOperationException* est levée, avec le message d'erreur « Il n'y a aucune table accessible. Assurez-vous qu'au moins un modèle de données est inscrit dans le fichier *Global.asax* et que la structure est activée, ou implémentez des pages personnalisées. ».
- La troisième instruction permet de configurer le routage (l'itinéraire) entre les entités du méta-modèle, et de définir les actions CRUD qu'il est possible d'effectuer sur chacune d'entre elles.

3.4 Fichier de configuration

Le fichier de configuration Web.Config permet de configurer l'ensemble des paramètres de l'application. Nous allons commencer par spécifier la chaîne de connexion, qui sera utilisée par notre modèle d'entité, pour gérer les données de la base de données Dotnet-France. Ainsi sous l'élément Configuration, on ajoute le flux XML suivant :

```
// Xml

<connectionStrings>
      <add name="DotnetFranceEntities"
connectionString="metadata=res://*/DotnetFrance.csdl|res://*/DotnetFrance
.ssdl|res://*/DotnetFrance.msl;provider=System.Data.SqlClient;provider
connection string=&quot;Data Source=localhost\sql2008;Initial
Catalog=DotnetFrance;User
ID=userTest;Password=passwd;MultipleActiveResultSets=True&quot;"
providerName="System.Data.EntityClient" />
</connectionStrings>
```

Aussi, par rapport à un fichier de configuration d'une application ASP .NET « standard », il comporte quelques modifications, présentées ci-dessous :

- Dans la section compilation/assemblies, ajout de références vers les assemblies du Framework .NET permettant de mettre en œuvre ASP .NET Dynamic Data :
  - System.Web.Abstractions.dll
  - System.Web.Routing.dll
  - System.Web.DynamicData.dll
  - System.ComponentModel.DataAnnotations.dll
- Dans la section pages/controls, importation des contrôles DynamicData, contenu dans l'espace de noms *System.Web.DynamicData* du composant *System.Web.DynamicData.dll*. Ainsi cet espace de noms n'est pas importer dans les pages de l'application utilisant ces contrôles.
- Définition d'un module HTTP supplémentaire nommé *UrlRoutingModule*. Il permet de faire correspondre une requête HTTP à un itinéraire dans une application ASP.NET, un itinéraire étant constitué d'un ensemble de routes, que nous pouvons définir dans le fichier *global.asax*.

4 Exécution de l'application

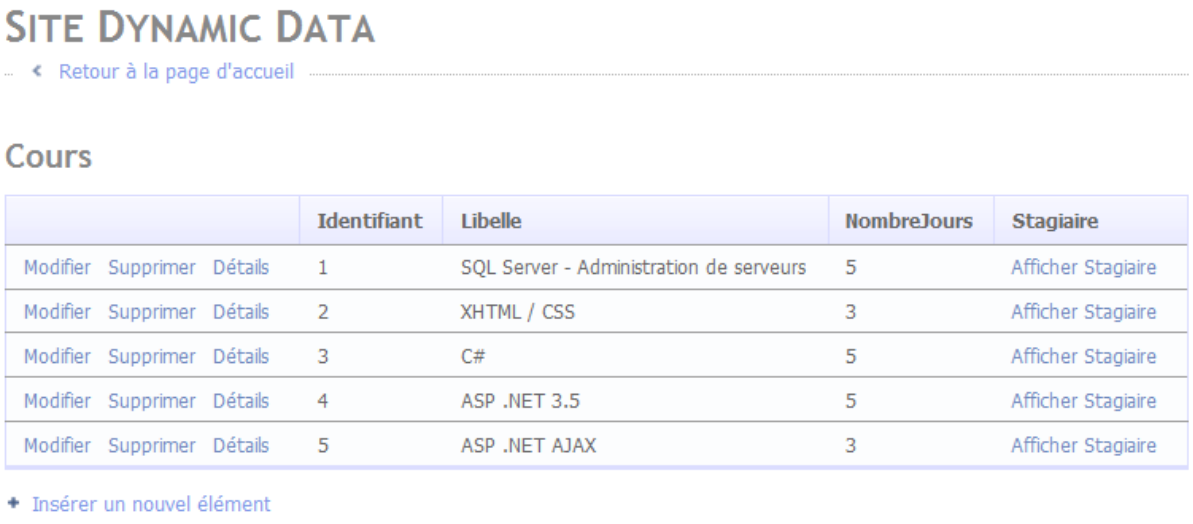
4.1 Accès à la page d'accueil

Pour exécuter l'application, exécutons la page Default.aspx, située à la racine de l'application. La page suivante apparaît :

Elle affiche la liste des entités exposées par le modèle d'entités. Il est possible de choisir la liste des entités à afficher. Nous verrons comment procéder dans le chapitre « ASP .NET Dynamic Data : développement avancé ».

4.2 Consulter les données d'une entité

En cliquant sur le nom d'une table, il est possible de visualiser les données qu'elle contient. Voici les données de la table cours :



Par défaut, à partir de 10 entités, un pied de tableau est affiché. Il permet d'accéder aux différents éléments de manière paginée.

Aussi, il est possible de trier les enregistrements affichés, en cliquant sur les entêtes de colonne de données.

4.3 Ajout d'une entité

Cet écran permet de gérer chacun des cours de notre table. En cliquant sur le lien « insérer un nouvel élément », on obtient la page suivante :

Pour chacun des champs, des contrôles de validation ont été générés. Ces contrôles de validation sont issus des contraintes du modèle de données sur lequel s'appuie notre application.

Dans notre cas, un identifiant est requis, alors que le champ correspondant est défini comme colonne identité dans SQL Server. Dans le cours « développement avancé sur ASP .NET Dynamic Data », nous verrons comment masquer ce champ. Dans le cas actuel, nous pouvons saisir un identifiant, afin que le contrôle de validation ne bloque pas l'ajout du cours dans la base de données.

4.4 Modification d'une entité

La demande de modification d'une entité affiche le résultat suivant :

Toutes les informations concernant le stagiaire sont affichées (on retrouve les propriétés scalaires et de navigation de l'entité Stagiaire). Seul l'identifiant n'est pas modifiable.

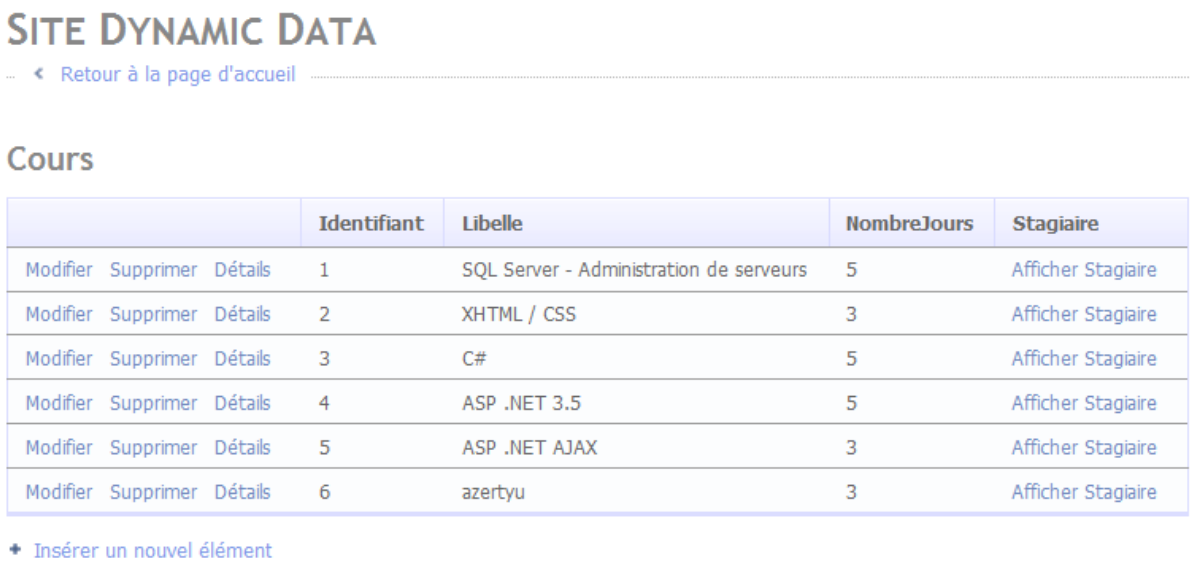
Les propriétés scalaires permettent de modifier les données du stagiaire ; les propriétés Nom et Prenom sont modifiables via un contrôle TextBox.

Les propriétés de navigation permettent d'accéder aux données liées (en l'occurrence dans notre cas, les cours des stagiaires) ; la propriété Groupe permet de modifier le groupe au travers d'une liste de données pré-remplie avec la liste des groupes.

4.5 Consultation des données d'une entité

Dans la modification d'une entité, les propriétés de navigation permettent de naviguer au travers des données liées dans le modèle de données.

Par exemple, si on clique sur le lien « Afficher Cours » de la propriété de navigation Cours, dans le formulaire de modification d'un stagiaire, l'écran suivant apparaît :

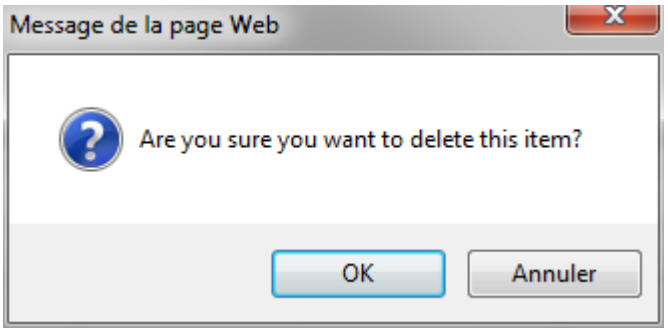


En plus d'afficher la liste des cours auxquels est inscrit un stagiaire, il permet :

- De consulter (via le lien Détails), ajouter, modifier et supprimer un cours.
- De consulter les cours d'un stagiaire.
- D'ajouter un cours.

4.6 Suppression d'une entité

Lors de la suppression d'une entité, une demande de confirmation est effectuée :



Dans le cas d'une confirmation, un ordre de suppression est envoyé à la base de données, via le modèle de données.

5 Conclusion

Ce cours nous a permis de prendre connaissance des différents types de projets de type ASP .NET Dynamic Data. Puis d'en créer un, afin de créer rapidement une application ASP .NET permettant de consulter et modifier les données, contenues dans une base de données SQL Server, au travers d'un modèle d'entités (cf : le Framework Entity).

Le prochain chapitre sur ASP .NET dynamic Data publié sur Dotnet-France, traitera de la personnalisation d'application ASP .NET Dynamic Data. Pour mettre en œuvre les possibilités de personnalisation, nous nous appuierons sur l'application développée dans ce cours.