

# Développement Web avec le Framework .NET v.2.0

février 06

## Résumé

Avoir un framework et un outil de développement, c'est bien, savoir s'en servir, c'est mieux. Nous vous proposons, via une démonstration, d'appliquer les bons principes d'architecture et de conception dans la réalisation d'une petite application Web. Rien de révolutionnaire, mais une utilisation optimisée des outils standards.

Le cas de démonstration :

- Une application permettant de visualiser des cours des formateurs, et leurs certifications.

La réalisation :

- Partant des données sous forme XML ou dans des Bases, on en déduit le DAL (Data Access Layer). On ne codera rien manuellement.
- On montrera comment un ajout de spécification impacte l'application et comment enrichir le DAL
- Après les données, les traitements (très simples dans notre cas de démonstration), dans le BLL (Business Logic Layer) chargée d'exposer les données métier (générées automatiquement) à la couche suivante.
- La présentation utilisera les nouveaux composants et les techniques de liaison de données.

Les finitions :

- Comment rendre l'application agréable du point de vue de l'utilisateur ...

# Table des matières

<b>1. PRÉSENTATION DU BESOIN</b>	<b>4</b>
<b>2. ARCHITECTURE GÉNÉRALE</b>	<b>5</b>
2.1 Vue de l'architecture cible .....	5
2.2 Stratégie de développement .....	5
2.3 Les données.....	5
2.4 Le Data Access Layer (DAL).....	5
2.5 Le Business Logic Layer (BLL) .....	5
2.6 La Présentation .....	6
2.7 Un Service ?.....	6
<b>3. LA MISE EN OEUVRE</b>	<b>6</b>
3.1 Réalisation de l'accès aux données.....	6
3.2 Un test anticipé .....	7
3.3 Réalisation de la partie Métier.....	8
3.4 Réalisation de la partie présentation.....	9
<b>4. LES FINITIONS : PENSER À L'UTILISATEUR</b>	<b>10</b>
4.1 Faciliter la navigation .....	10
4.2 Améliorer l'aspect graphique.....	11
4.2.1 Une première mise en facteur : une structure de page commune .....	11
4.2.2 Une deuxième mise en facteur : des attributs de présentation communs.....	12
<b>5. CONCLUSION</b>	<b>13</b>

## Liste des Figures

<i>Figure 1 : les informations de départ .....</i>	<i>4</i>
<i>Figure 2 : le Diagramme d'Architecture.....</i>	<i>5</i>
<i>Figure 3: le schéma XSD des données de la base.....</i>	<i>6</i>
<i>Figure 4 : Affichage des Cours dans la GridView.....</i>	<i>7</i>
<i>Figure 5 Méthode de mapping Métier/Données .....</i>	<i>8</i>
<i>Figure 7 : le lien entre le composant GridView et notre BLL.....</i>	<i>9</i>
<i>Figure 8 : ici un TreeView connecté au SiteMap.....</i>	<i>10</i>
<i>Figure 9 : Une page utilisant une MasterPage.....</i>	<i>11</i>
<i>Figure 10 : La pollution d'une GridView par les attributs de présentation .....</i>	<i>12</i>
<i>Figure 11 : un exemple de fichier SKIN .....</i>	<i>12</i>
<i>Figure 12 : Une GridView débarrassée des attributs de formattage.....</i>	<i>13</i>

## 1. Présentation du besoin

On se propose dans cette démonstration de réaliser une application Web permettant de visualiser les cours et les formateurs, ainsi que d'attribuer une certification à ces derniers.

Pour cela, nous disposons des documents suivants :

- de fichiers XML donnant la liste des cours et leur description
- d'une petite base de données contenant les Cours, les Formateurs et la Certification

Nous allons construire une Application Web sur une architecture trois niveaux (3 tiers).

Après consultation des documents de départ, nous nous attaquerons successivement aux différentes couches en partant des données, en passant par le niveau métier pour finir par le niveau présentation.

Les formateurs : identifiant, nom, prenom, localisation (base)

Les Cours : identifiant, titre, description, ...

Les Certifications : idFormateur, idCours, Date



Figure 1 : les informations de départ

## 2. Architecture générale

### 2.1 Vue de l'architecture cible

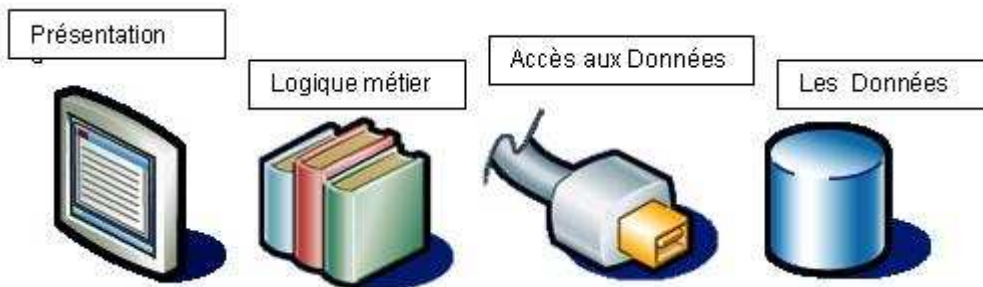


Figure 2 : le Diagramme d'Architecture

### 2.2 Stratégie de développement

Pour chaque niveau, nous préconisons un projet séparé. Ainsi, les interfaces entre les différents niveaux seront bien spécifiées.

### 2.3 Les données

Nous ne développerons pas de code dans ce niveau.

Des procédures stockées peuvent être réalisées ici pour, entre autres raisons, les performances ou la sécurité.

### 2.4 Le Data Access Layer (DAL)

Les données sont structurées. Que ce soient dans des fichiers XML (ou sous d'autres formats), ou encore dans des bases de données, les structures sont connues et nous pouvons nous appuyer dessus. Les outils aussi... et nous allons en profiter pour faire générer le DAL complet à Visual Studio. Nous attendons de sa part un ensemble de classes techniques, reproduction fidèle dans le monde C# des tables décrites dans la base de données.

De plus, nous allons confier les fonctionnalités de CRUD (Create/Retrieve/Update/Delete) à ces classes sous la forme de méthodes d'accès aux données.

Conclusion : pas de code créé manuellement non plus dans ce niveau

### 2.5 Le Business Logic Layer (BLL)

Plutôt que de négocier avec les objets techniques du DAL, nous allons créer des objets métier dans ce niveau. Ils seront, chargés de traiter les actions métier provenant de la couche de présentation en s'appuyant sur le DAL. Dans notre exemple très simple, un seul objet sera nécessaire pour gérer l'association entre les Formateurs et les Cours que l'on a nommé Certification.

Remarquons que seul ce niveau nous amène à développer... mais c'est du code « métier ».

## 2.6 La Présentation

La dernière partie est la partie visible de l'application : la présentation. Nous allons connecter des objets graphiques du framework à nos objets métier.

La bonne surprise est la facilité et la souplesse d'emploi de ces objets.

On citera par exemple le GridView ou encore le DataList...

Peu de code dans cette partie. Il suffit d'intercepter les actions utilisateur et de les rediriger vers le niveau BLL.

## 2.7 Un Service ?

Et si d'autres applications avaient besoin de récupérer des informations manipulées par notre niveau métier ?

Un Web Service peut être développé pour exposer de facilement les données voulues.

# 3. La mise en oeuvre

## 3.1 Réalisation de l'accès aux données

Nous avons l'intention de matérialiser ce niveau par un composant réutilisable, un projet de type librairie fait l'affaire.

La Base de données doit être intégrée dans le projet, nous choisissons de l'inclure par drag&drop dans le projet.

L'assistant nous permet de choisir les objets à inclure : ici les trois tables « Formateur » « Cours » et « Certification »

On remarquera la création d'un fichier XSD, représentant la structure des tables. On pourra renommer ce fichier « Formation.XSD »

Un double click pour activer sa vue graphique et nous explorons la structure importée.

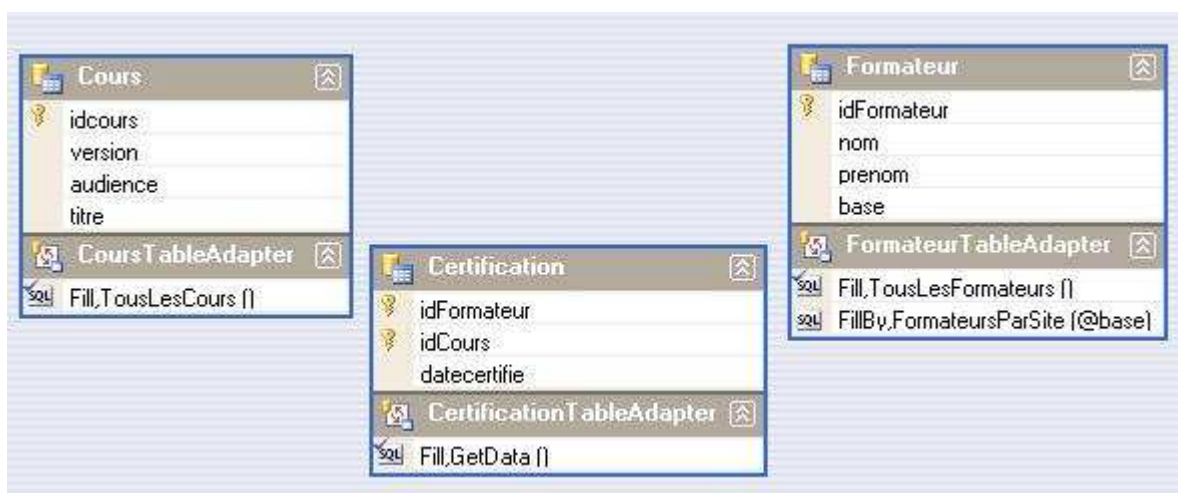


Figure 3: le schéma XSD des données de la base

Le but du DAL (et son nom l'indique) est d'encapsuler les accès aux données.

Dans la vue des classes, on remarque que les objets sont préparés pour des requêtes par défaut



On devra munir ces classes des actions spécifiques désirées :  
Ici on se contentera de renommer les méthodes

Afin de préparer le DAL pour négocier avec les niveaux suivant, on fera générer le fichier Formation.cs à l'outil XSD.EXE. Et on l'inclura dans le projet.

Ensuite : compilons le projet pour obtenir notre assembly : DAL.DLL dans le répertoire bin/debug

### 3.2 Un test anticipé

Il ne fait pas de mal de vérifier à chaque étape que tout va pour le mieux.

Créons donc un petit test sous la forme d'un nouveau site web nommé Test muni d'une page web test.aspx.

Remarque : pas besoin de IIS pour cela, Visual Studio est livré avec un moteur web local.

Pour pouvoir utiliser le DAL, on ajoutera une référence de type « projet » au DAL, ainsi qu'une clause using

```
using DAL.FormatTableAdapters;
```

Remarquez les classes du namespace **DAL.FormatTableAdapters**, elles vont nous permettre de récupérer la liste des Formateurs et effectuer une restitution sur la page Web.

Placer un contrôle , par exemple un GridView, et le lier par le code suivant au chargement de la page :

```
gridView.DataSource= new CoursTableAdapter().TousLesCours();  
gridView.DataBind();
```

Voici le résultat :

#### Liste des Cours

idcours	version	audience	titre
ACOD			C++ Avancé
ADPC			Design Patterns avec C++
ADPJ			Design Patterns avec Java
AJOD			Java Avancé
AMP			Apache MySql Php
ASP.NET			Concevoir et Développer une application Web

Figure 4 : Affichage des Cours dans la GridView

Pour pouvoir afficher seulement les formateurs de Toulouse ou de Paris, il suffit de créer une méthode FormateursParSite(site)

Modifions le DAL en conséquence et testons grâce à une combobox (autopostback) de sélection du site ('T' ou 'P'). La DataList sera alors connectée à la base de données via l'objet FormateurTableAdapter du DAL.

Attention à bien retirer le code du chargement de page avant de tester.

### 3.3 Réalisation de la partie Métier

On crée un nouveau projet de type librairie.

Ici, nous voulons manipuler les objets Cours, Formateur et Certification renvoyés par le DAL.  
On importera donc le fichier XSD représentant leurs structures. (formation.xsd)

Une seule classe nommée FormationService va nous suffire.

Son rôle est la fourniture des objets métier aux couches suivantes (présentation).

En particulier, la liste des Cours et des Formateurs.

On utilisera plutôt les nouvelles collections typées du Framework 2.0 : les génériques.

FormationService comportera la méthode suivante permettant de renvoyer des objets métier plutôt que des structures de base de données.

la clause using correspondant au namespace du DAL sera ajoutée :

```
public List<FormationFormateur> GetTousLesFormateurs ()
{
    List<FormationFormateur> liste = new List<FormationFormateur> ();
    foreach (Formation.FormateurRow f in new
        FormateurTableAdapter().TousLesFormateurs())
    {
        FormationFormateur ff = new FormationFormateur ();
        ff.@base = f._base;
        ff.nom = f.nom;
        ff.prenom = f.prenom;
        ff.idFormateur = f.idFormateur;
        liste.Add(ff);
    }
    return liste;
}
```

Figure 5 Méthode de mapping Métier/Données

C'est maintenant sur cet objet et sur ces méthodes que nous pourrons pratiquer la liaison de données des contrôles.

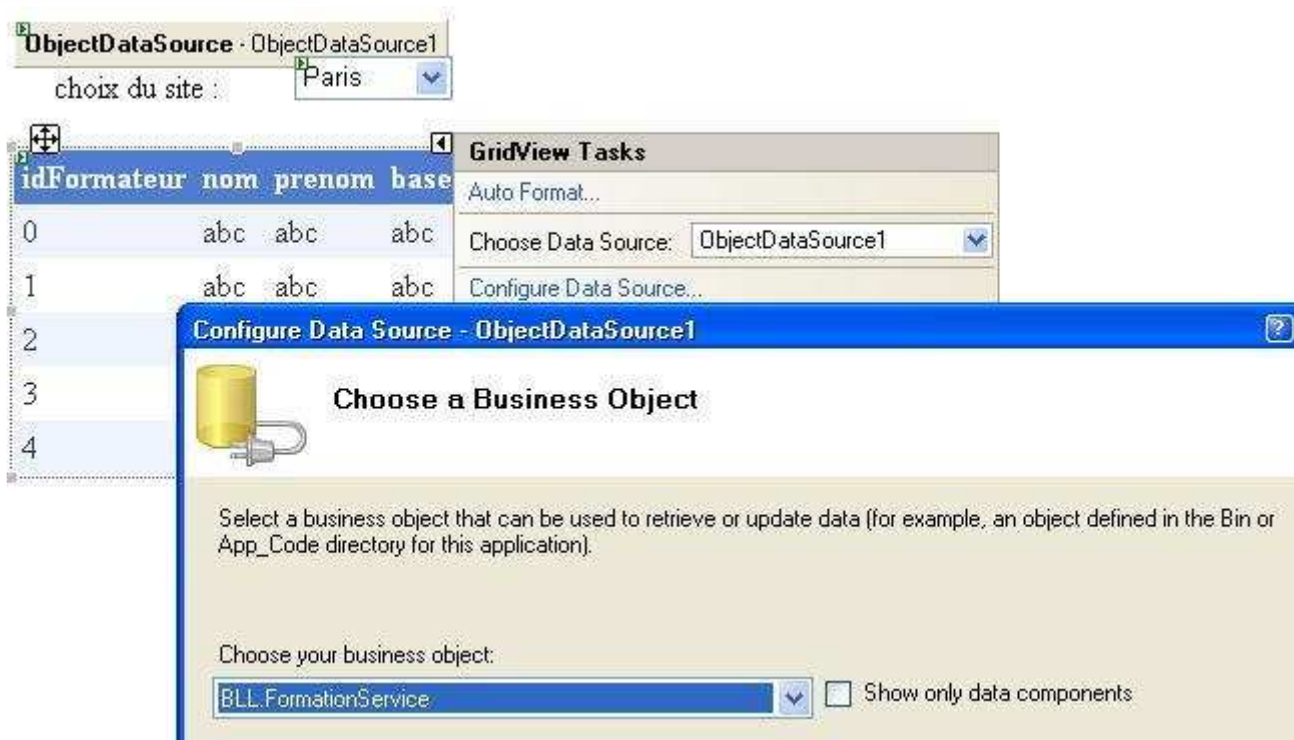


Figure 6 : le lien entre le composant GridView et notre BLL

On développera aussi une action de mise à jour (qui devra persister dans la base de donnée), l'ajout d'une certification à un formateur.

Pour cela, on vérifiera que le DAL comprend bien cette action, et on l'encapsulera dans le niveau BLL.

### 3.4 Réalisation de la partie présentation

Pour gagner du temps, nous allons réutiliser notre site de Test pour la partie présentation

Il s'agit de développer une page pour visualiser les formateurs, les cours et d'associer les deux pour la certification.

On montrera comment les composants graphiques permettent « sans aucun code » de supporter les notions de sélection, de mise à jour, de tri, de filtre ainsi que de pagination.

Ajouter une référence sur le projet BLL.

Ainsi, nos GridView pourront être connectés à la classe FormationService et profiter de toute ses méthodes.

## 4. Les finitions : penser à l'utilisateur

### 4.1 Faciliter la navigation

L'utilisateur aime bien savoir où il est situé par rapport à un site web, il en est de même par rapport à une application web.

Cette tâche est relativement répétitive et sujette à erreurs.

L'approche proposée est une approche centralisée : un seul endroit contiendra les informations d'architecture logique de notre application : nous avons choisi le plus simple, un fichier XML décrivant la vue logique.

Remarquons que d'autres possibilités (base SQL par exemple) sont possibles.

Les pages apparaissent hiérarchisées à travers le SiteMap, mais peuvent tout à fait être placées physiquement selon une organisation différente et résider par exemple dans le même répertoire.

En fait, il suffit d'ajouter à notre projet un composant SiteMap du contenu suivant :

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
  <siteMapNode url="Home.aspx" title="Demarrage" description="" >
    <siteMapNode url="Default.aspx" title="Defaut" description="" />
    <siteMapNode url="TestDAL.aspx" title="Test D A L" description="" />
    <siteMapNode url="TestBLL.aspx" title="Test B L L" description="" />
    <siteMapNode url="Certifier.aspx" title=" Certifier" description="" />
  </siteMapNode>
</siteMap>
```

Figure 1 : La structure de l'application

Ceci n'est qu'un préalable. Pour réellement informer l'utilisateur sur sa position dans l'application, nous allons utiliser un contrôle **SiteMapPath**

La bonne nouvelle est qu'il suffit de paramétrer ce contrôle en lui indiquant le sitemap (ou tout autre fournisseur des données) et c'est opérationnel. Nous disposons d'un marqueur de localisation nous donnant le chainage de la structure du site vers la page racine.

Ce contrôle ne permet toutefois pas une vue globale.

Pour cela, ajoutons un contrôle treeview ou un contrôle menu... connecté au SiteMap

On peut alors tester la page et effectuer la navigation aisément.

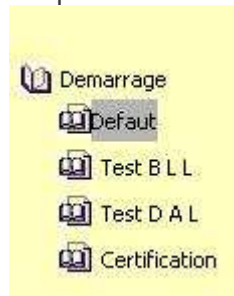


Figure 7 : ici un TreeView connecté au SiteMap

Remarque : aucune ligne de code n'a été tapée pour obtenir ces fonctionnalités.

## 4.2 Améliorer l'aspect graphique

### 4.2.1 Une première mise en facteur : une structure de page commune

Allons nous effectuer tout le travail précédent pour chacune des pages ?  
 N'y a-t-il pas moyen de mettre en facteur commun ces fonctionnalités ?  
 C'est le but des **MasterPages**

La notion de MasterPage a été créée de façon à homogénéiser le cadre de page à tout un site. Nous pouvons organiser nos plans de pages comme nous l'entendons, mais restons classiques : plaçons une zone horizontale en haut de page pour recevoir un bandeau d'en-tête, ajoutons une zone verticale prête à recevoir les menus de navigation, ceux qui ont été créés précédemment directement sur la page.

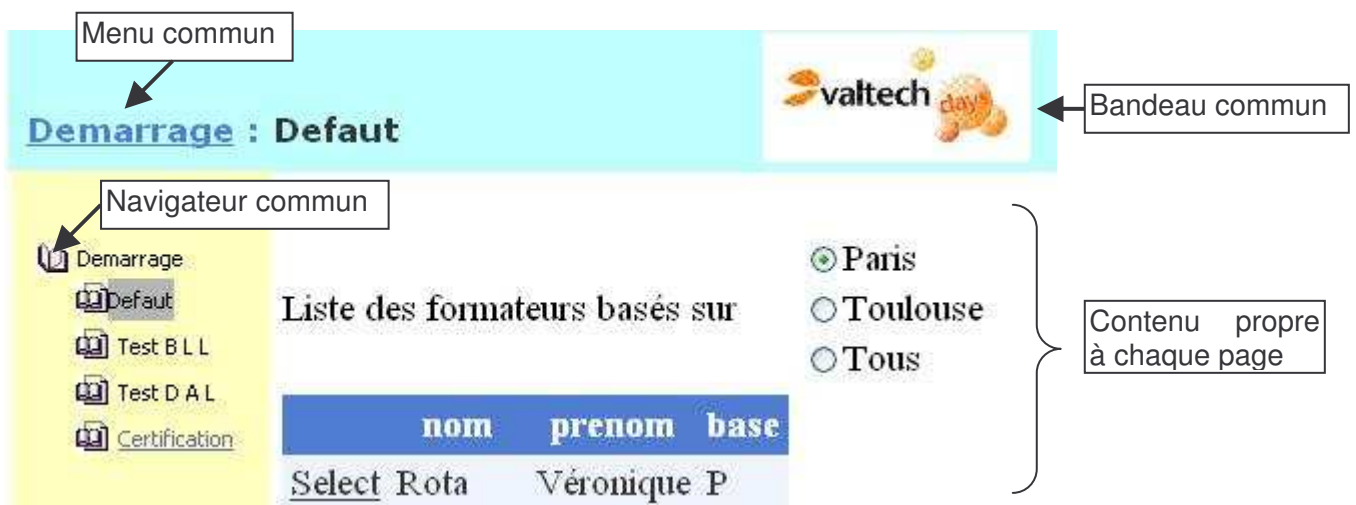


Figure 8 : Une page utilisant une MasterPage

Remarquons qu'il est possible d'utiliser une ou plusieurs masterpages. Il est aussi faisable de les appliquer pour différents browsers (IE / Firefox / ...) et ceci de façon uniquement déclarative. Une bonne utilisation consiste à proposer un rendu différent selon les types de navigateurs, en particulier ceux à affichage réduit.

## 4.2.2 Une deuxième mise en facteur : des attributs de présentation communs

Jetons un coup d'œil à une GridView formatée : regardons sa partie HTML  
On notera beaucoup de lignes de formatage qui polluent les données elle mêmes

```
<asp:GridView ID="gvFormateur" runat="server" AutoGenerateColumns="False" CellPadding="4"
DataKeyNames="idFormateur" ForeColor="#333333" GridLines="None">
  <FooterStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
  <Columns>
    <asp:CommandField ShowSelectButton="True" />
    <asp:BoundField DataField="nom" HeaderText="nom" SortExpression="nom" />
    <asp:BoundField DataField="prenom" HeaderText="prenom" SortExpression="prenom" />
    <asp:BoundField DataField="base" HeaderText="base" SortExpression="base" />
  </Columns>
  <RowStyle BackColor="#EFF3FB" />
  <EditRowStyle BackColor="#2461BF" />
  <SelectedRowStyle BackColor="#D1DDF1" Font-Bold="True" ForeColor="#333333" />
  <PagerStyle BackColor="#2461BF" ForeColor="White" HorizontalAlign="Center" />
  <HeaderStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
  <AlternatingRowStyle BackColor="White" />
</asp:GridView>
```

Figure 9 : La pollution d'une GridView par les attributs de présentation

Y a-t-il moyen de regrouper tout cela en un seul endroit ?

On utilisera des Thèmes (skins et css)

La création de skin est facilitée par l'environnement.

Ajouter un ressource de type skin, nommé «monSkin » dans un dossier spécifique App\_Themes Et l'éditer. En général, on définit un aspect par défaut et un ou plusieurs dotés d'un skinID :

```
<asp:GridView runat="server" CellPadding="4" ForeColor="#333333" GridLines="None">
  <FooterStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
  <RowStyle BackColor="#EFF3FB" />
  <EditRowStyle BackColor="#2461BF" />
  <SelectedRowStyle BackColor="#D1DDF1" Font-Bold="True" ForeColor="#333333" />
  <PagerStyle BackColor="#2461BF" ForeColor="White" HorizontalAlign="Center" />
  <HeaderStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
  <AlternatingRowStyle BackColor="White" />
</asp:GridView>
<asp:GridView skinID="gvSkin" runat="server" CellPadding="4" BackColor="White"
BorderColor="#CC9966" BorderStyle="None" BorderWidth="1px">
  <FooterStyle BackColor="#FFFFCC" ForeColor="#330099" />
  <RowStyle BackColor="White" ForeColor="#330099" />
  <SelectedRowStyle BackColor="#FFCC66" Font-Bold="True" ForeColor="#663399" />
  <PagerStyle BackColor="#FFFFCC" ForeColor="#330099" HorizontalAlign="Center" />
  <HeaderStyle BackColor="#990000" Font-Bold="True" ForeColor="#FFFFCC" />
</asp:GridView>
```

Figure 10 : un exemple de fichier SKIN

Il faut ensuite le déclarer au projet dans le fichier de configuration : Web.config

```
<system.web>
  <pages theme="monSkin" masterPageFile="~/MasterPage.master"></pages>
  ...
```



Voici le résultat après retrait des attributs au niveau de chaque GridView

```
<asp:GridView ID="gvFormateur" runat="server" AutoGenerateColumns="False"
DataKeyNames="idFormateur">
  <Columns>
    <asp:CommandField ShowSelectButton="True" />
    <asp:BoundField DataField="nom" HeaderText="nom" SortExpression="nom" />
    <asp:BoundField DataField="prenom" HeaderText="prenom" SortExpression="prenom" />
    <asp:BoundField DataField="base" HeaderText="base" SortExpression="base" />
  </Columns>
</asp:GridView>
```

Figure 11 : Une GridView débarrassée des attributs de formattage

Pour un rendu graphique identique .

Il est aussi conseillé d'utiliser des fichiers CSS

On peut avoir plusieurs themes et les appliquer à loisir.

## 5. Conclusion

En conclusion, le framework dotnet couplé à un outil comme Visual Studio permet du véritable développement RAD.

Les nombreux assistants mettent à notre disposition des fonctionnalités sans codage.

Ils sont assez matures pour ne pas se contenter de générer du « one shot » mais sont suffisamment souples pour être réutilisés en modification de projet et pour se plier à des besoins d'architecture.