Dotnet France
Technologies Sharepoint, SQL Server & .NET

Association Dotnet France

ASP .NET MVC : premiers pas

Version 1.0

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

Sommaire

1 Introduction

1.1 Présentation

Dans ce cours, nous allons réaliser un projet, relativement simple, permettant de mettre en pratique les notions abordées dans le cours de présentation d'ASP .NET MVC. Nous aurons l'occasion de définir un modèle, puis les contrôleurs et vues de l'application. Le modèle créé ne sera pas interfacé avec une base de données, au travers des composants d'accès aux données tels que Linq to SQL ou Framework Entity. Pour acquérir le contenu de ce cours, vous n'êtes pas obligés de connaître le fonctionnement et la mise en œuvre de ces composants.

1.2 Pré-requis

Avant de lire ce cours, nous vous conseillons :

- De maîtriser la conception et le développement d'applications Web avec la technologie ASP .NET.
- D'avoir lu le cours de présentation d'ASP .NET MVC publié sur Dotnet-France.

Ce cours a été écris avec Visual Studio 2008 et ASP .NET MVC 2 Preview.

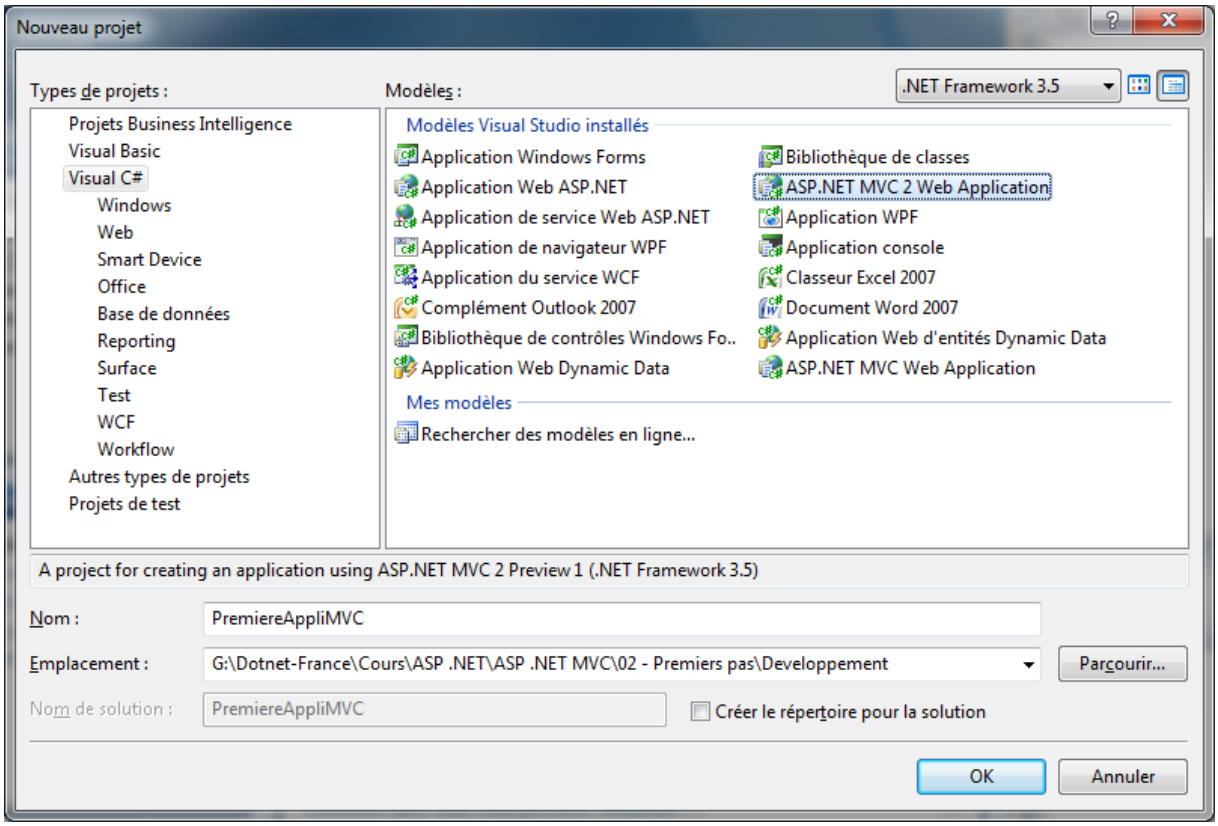
1.3 Présentation du projet

Dans ce cours, nous allons réaliser un projet, permettant d'afficher des informations sur des stagiaires et des cours qu'ils suivent dans un centre de formation. L'application ASP .NET MVC que nous allons créer permet doit permettre :

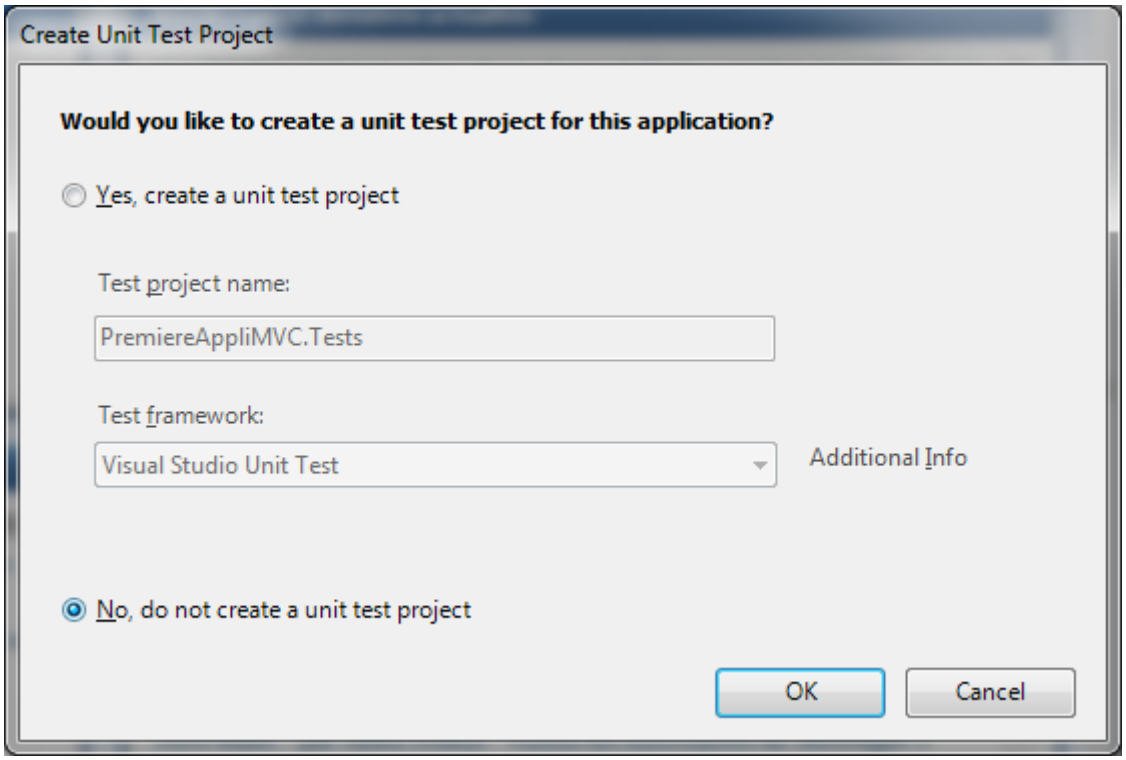
- D'afficher la liste de tous les stagiaires du centre de formation. Pour chacun d'entre eux, elle devra permettre d'afficher la liste des cours qui lui sont dispensés.
- D'afficher la liste de tous les cours dispensés, tous stagiaires confondus. Elle devra permettre pour chacun d'entre eux, de fournir la liste des stagiaires qui le suivront.

2 Création du projet ASP .NET MVC

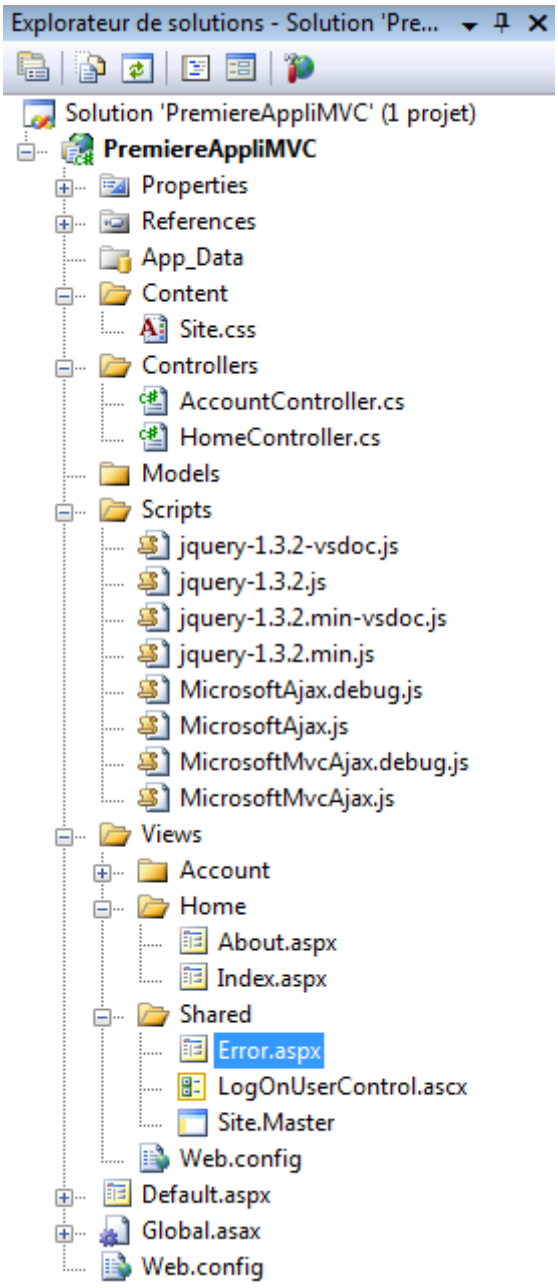
Après avoir lancé Visual Studio 2008, nous allons créer un projet Web de type ASP .NET MVC 2. Pour ce faire, dans la liste des modèles de projet, nous sélectionnons l'élément « ASP .NET MVC 2 Web Application » :



Après avoir cliqué sur le bouton OK, la fenêtre suivante apparaît :



Cette fenêtre nous propose de créer un projet de test pour l'application ASP .NET MVC que nous créons. Nous allons refuser cette proposition, car les tests d'applications MVC feront l'objet d'un autre cours, spécialisé sur ce sujet. Nous cliquons sur le bouton OK et notre projet est créé. Dans l'explorateur de solution, nous pouvons observer la structure suivante :



Dans ce projet, nous pouvons observer que chaque partie du modèle MVC correspond à un dossier spécifique automatiquement créé :

- *Models* : permet de définir des classes constituant un modèle propre à l'application.
- *Controllers* : permet de définir les contrôleurs de l'application.
- *Views* : permet de définir les vues (pages ASPX et contrôles utilisateurs Web).

Nous pouvons aussi nous rendre compte que chaque vue du modèle est une page ASPX différente des pages ASPX que nous étions habitués à créer :

- Elles ne contiennent pas de classe d'arrière plan (*classe code-behind*). En effet, ces pages sont directement liées à un contrôleur approprié qui se chargera d'exécuter les actions demandées par l'utilisateur.

- Elles dérivent non pas de classe *System.Web.UI.Page*, mais de la classe *System.Web.Mvc.ViewPage*.

Nous pouvons aussi observer que deux autres dossiers sont créés :

- Le dossier *Content* qui contient les fichiers *css* de l'application.
- Le dossier *Scripts* qui contient des librairies *Jquery*, qui permettront de mettre en œuvre *Jquery* dans notre application. Nous ne traiterons pas de ces aspects dans ce cours, car celui-ci fer l'objet d'un autre cours, spécialisé sur ce sujet.

La page *Default.aspx* est le point d'entrée de l'application.

On remarque aussi la présence du fichier « global.asax », dont le code est le suivant :

```
// C#

public class MvcApplication : System.Web.HttpApplication
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

        routes.MapRoute(
            "Default",                                              //
Route name
            "{controller}/{action}/{id}",                           //
URL with parameters
            new { controller = "Home", action = "Index", id = "" }  //
Parameter defaults
        );
    }

    protected void Application_Start()
    {
        RegisterRoutes(RouteTable.Routes);
    }
}
```

Lors du démarrage de l'application, deux routes sont définis :

- La première permet de signifier au process ASP .NET MVC de ne pas prendre en compte les requêtes HTTP visant les handlers HTTP.
- La seconde permet de définir le modèle principal des Url, permettant de demander l'exécuter d'une action d'un contrôleur, avec optionnellement la présence d'un paramètre. Le contrôleur par défaut est *Home* (*HomeControler*). La méthode (action) exécutée par défaut au sein des contrôleurs est *Index*. La valeur du paramètre de cette méthode s'il est présent est la valeur « chaîne vide ».

3 Création du modèle

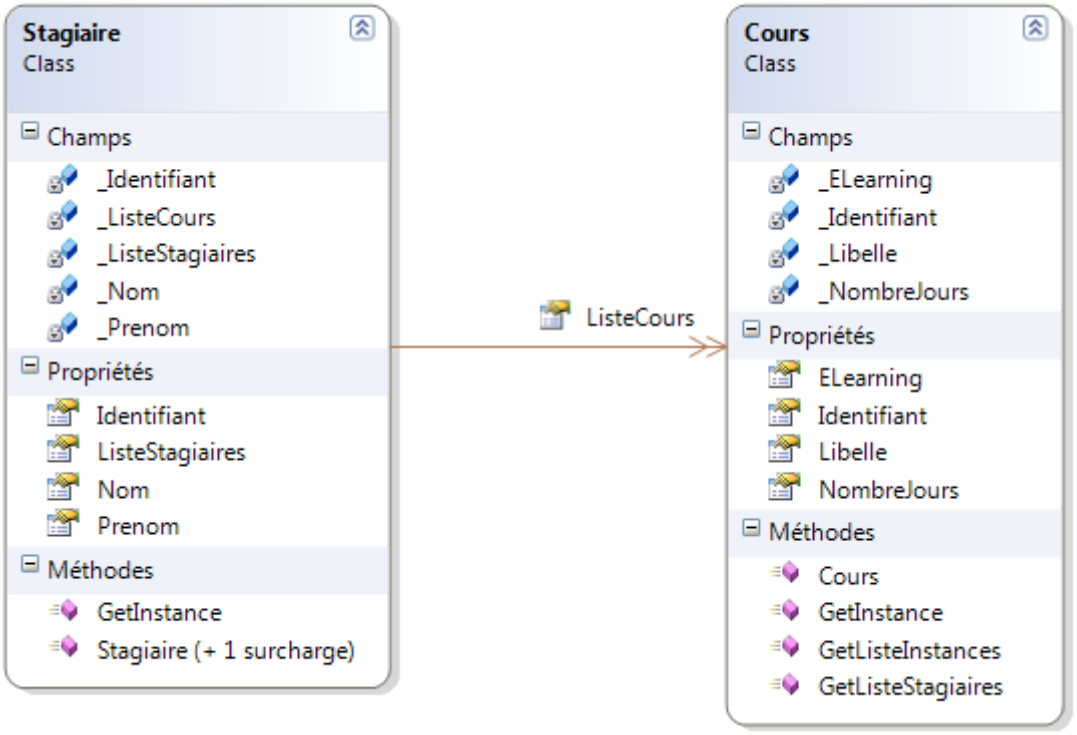
Dans ce chapitre, nous allons créer le modèle de notre application MVC.

3.1 Modélisation

Dans notre application, le modèle sera constitué de deux classes :

- La classe *Stagiaire* : représente un stagiaire d'un centre de formation.
- La classe *Cours* : représente les cours suivis par les stagiaires dans le centre de formation.

Voici le diagramme de classe de notre modèle :



3.2 Implémentation

Voici l'implémentation de la classe *Cours* :

```
// C#

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using PremiereAppliMVC.Models.Personne;

namespace PremiereAppliMVC.Models.Formations
{
    public class Cours
    {
        #region Attributs et Accesseurs
        private int _Identifiant;
        public int Identifiant
        {
            get { return _Identifiant; }
            set { _Identifiant = value; }
        }

        private string _Libelle;
        public string Libelle
        {
            get { return _Libelle; }
            set { _Libelle = value; }
        }

        private int _NombreJours;
        public int NombreJours
        {
            get { return _NombreJours; }
            set { _NombreJours = value; }
        }

        private bool _ELearning;
        public bool ELearning
        {
            get { return _ELearning; }
            set { _ELearning = value; }
        }
        #endregion

        #region Constructeurs
        public Cours(int aIdentifiant, string aLibelle, int aNombreJours,
bool aEstELearning)
        {
            // Initialisation des attributs.
            this.Identifiant = aIdentifiant;
            this.Libelle = aLibelle;
            this.NombreJours = aNombreJours;
            this.ELearning = aEstELearning;
        }
        #endregion
```

```
// C#


        #region Méthodes
            public static List<Cours> GetListeInstances()
            {
                return (from oStagiaire in Stagiaire.ListeStagiaires
                        from oCours in oStagiaire.ListeCours
                        select oCours)
                    .Distinct(new CoursComparer())
                    .OrderBy(oCours => oCours.Identifiant)
                    .ToList();
            }

            public static Cours GetInstance(int aIdCours)
            {
                return (from cours in Cours.GetListeInstances()
                        where cours.Identifiant == aIdCours
                        select cours).FirstOrDefault();
            }

            public List<Stagiaire> GetListeStagiaires()
            {
                return (from oStagiaire in Stagiaire.ListeStagiaires
                        from oCours in oStagiaire.ListeCours
                        where oCours.Identifiant == this.Identifiant
                        select oStagiaire).ToList();
            }
        #endregion
    }

    public class CoursComparer : IEqualityComparer<Cours>
    {
        #region IEqualityComparer<Cours> Membres

            public bool Equals(Cours x, Cours y)
            {
                return x.Identifiant == y.Identifiant;
            }

            public int GetHashCode(Cours obj)
            {
                return obj.Identifiant;
            }

        #endregion
    }
}
```

La classe *CoursComparer* a été créée afin de pouvoir comparer deux cours. Elle est utilisée pour enlever les doublons d'une liste de cours. Dans le code ci-dessus, on remarque dans la méthode *GetListeInstances*, l'utilisation de la méthode *Distinct* pour supprimer les doublons de la liste des cours fournis par une requête LINQ. Pour que cette méthode s'exécute de manière attendue, nous utilisons une instance de la classe *CoursComparer*.

Voici l'implémentation de la classe *Stagiaire* :

```
// C#

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

using PremiereAppliMVC.Models.Formations;


namespace PremiereAppliMVC.Models.Personne
{
    public class Stagiaire
    {
        #region Attributs et Accesseurs
        private int _Identifiant;
        public int Identifiant
        {
            get { return _Identifiant; }
            set { _Identifiant = value; }
        }

        private string _Nom;
        public string Nom
        {
            get { return _Nom; }
            set { _Nom = value; }
        }

        private string _Prenom;
        public string Prenom
        {
            get { return _Prenom; }
            set { _Prenom = value; }
        }

        private List<Cours> _ListeCours;
        public List<Cours> ListeCours
        {
            get { return _ListeCours; }
            set { _ListeCours = value; }
        }
```

```
// C#

        private static List<Stagiaire> _ListeStagiaires;
        public static List<Stagiaire> ListeStagiaires
        {
            get
            {
                if (_ListeStagiaires == null)
                {
                    _ListeStagiaires = new List<Stagiaire>()
                    {
                        new Stagiaire(1, "RAVAILLE", "James", new
List<Cours>()
                            {
                                new Cours(1, "Formation C# 3.0", 5,
true),
                                new Cours(2, "ASP .NET 3.5", 4, false),
                                new Cours(3, "Architecture
d'applications", 2, false),
                                new Cours(4, "Conception de bases de
données SQL Server", 3, true)
                            }),
                        new Stagiaire(2, "PAGES", "Anthony", new
List<Cours>()
                            {
                                new Cours(5, "Formation VB 9.0", 5,
true),
                                new Cours(2, "ASP .NET 3.5", 4, false),
                                new Cours(6, "Administration de bases de
données SQL Server", 5, false)
                            }),
                        new Stagiaire(3, "DOLLON", "Julien", new
List<Cours>()
                            {
                                new Cours(7, "Windows Presentation
Foundation", 5, true),
                                new Cours(8, "Silverlight", 2, true),
                                new Cours(9, ".NET RIA Services", 3,
false),
                                new Cours(10, "MOOS 2007", 5, false)
                            }),
                        new Stagiaire(4, "VERGNAULT", "Bertrand",new
List<Cours>()
                            {
                                new Cours(7, "Windows Presentation
Foundation", 5, true),
                                new Cours(1, "Formation C# 3.0", 5,
false),
                                new Cours(6, "Administration de bases de
données SQL Server", 5, false)
                            })
                    };
                }
                return _ListeStagiaires;
            }
        }
        #endregion
```

```
// C#

        #region Constructeurs
            public Stagiaire(int aIdentifiant, string aNom, string
aPrenom)
            {
                // Initialisation des attributs.
                this.Identifiant = aIdentifiant;
                this.Nom = aNom;
                this.Prenom = aPrenom;
                this.ListeCours = new List<Cours>();
            }

            public Stagiaire(int aIdentifiant, string aNom, string
aPrenom, List<Cours> aListeFormations)
                : this(aIdentifiant, aNom, aPrenom)
            {
                // Initialisation des attributs.
                this.ListeCours = aListeFormations;
            }
        #endregion

        #region Méthodes
            public static Stagiaire GetInstance(int aIdentifiant)
            {
                return (from oStagiaire in ListeStagiaires
                        where oStagiaire.Identifiant == aIdentifiant
                            select oStagiaire).FirstOrDefault();
            }
        #endregion
    }
}
```

Le projet ASP .NET MVC que nous créons, ne va pas gérer les données contenues dans une base de données. Le modèle va fournir les données sous forme d'un « bouchon de données ». Pour créer ce bouchon, nous avons créé dans la classe *Stagiaire* un accesseur en lecture seule nommé *ListeStagiaires* de type *List<Stagiaire>*, avec son attribut. Cet accesseur construit la liste des stagiaires lorsqu'il est appelé la première fois. Lors des appels suivants, cette même liste est renvoyée.

4 Création du contrôleur

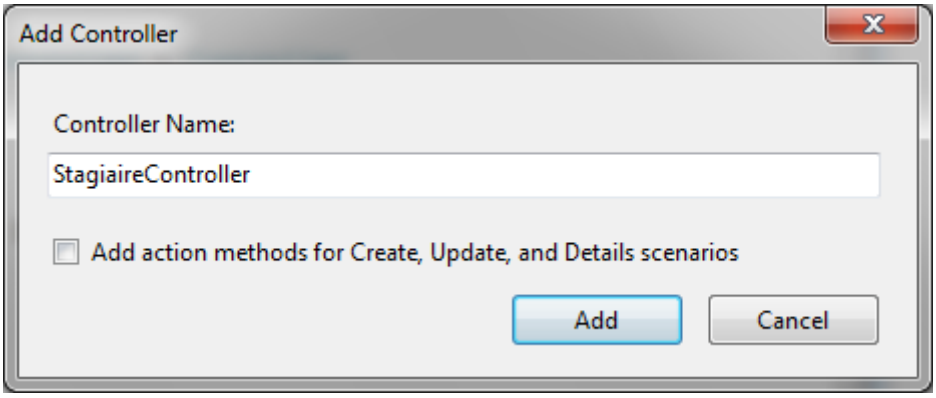
Dans ce chapitre, nous allons créer les contrôleurs de notre application MVC. Nous allons créer deux contrôleurs :

- Un contrôleur nommé *Stagiaire*, implémenté dans la classe *StagiaireController*.
- Un contrôleur nommé *Cours*, implémenté dans la classe *CoursController*.

Tous les contrôleurs de notre application ASP .NET MVC sont contenus dans le répertoire *Controllers* situé à la racine de l'application.

4.1 Création du contrôleur *Stagiaire*

Pour créer le contrôleur *Stagiaire*, nous allons afficher le menu contextuel sur le répertoire *Controllers* situé à la racine de notre projet, cliquer sur l'item *Ajouter* puis *Controller*. La fenêtre suivante apparaît :



Nous saisissons le nom de notre contrôleur, par défaut suffixé par *Controller*, et nous ne cochons pas la case à cocher « Ajouter des méthodes d'action pour la création, mise à jour, détail ». Puis nous validons et nous obtenons une classe avec une méthode dont l'implémentation est la suivante :

```
// C#

public ActionResult Index()
{
    return View();
}
```

Cette méthode retourne simplement la vue à laquelle cette méthode (action) est associée. Nous allons la modifier, afin qu'elle réponde aux besoins suivants :

- Demander au modèle d'obtenir la liste de tous les stagiaires.
- D'obtenir la liste des stagiaires d'un cours particulier.

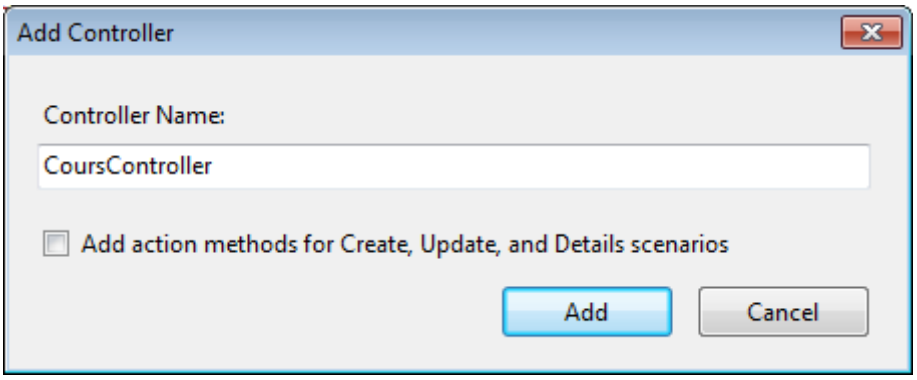
Pour ce faire, nous allons ajouter un paramètre de type *int* (type de l'attribut *Identifiant* de la classe *Stagiaire*), qui sera *nullable*. De cette manière, si lors de l'exécution de cette méthode une valeur est fournie pour ce paramètre, alors la vue attachée à cette méthode affichera la liste des

stagiaires du cours identifié par cette valeur. Le cas échéant, la liste de tous les stagiaires sera affichée. Nous obtenons donc le code suivant :

```
// C#

public ActionResult Index(int? aIdentifiantCours)
{
    List<Stagiaire> oListeStagiaires = null;
    if (aIdentifiantCours.HasValue)
    {
        Cours oCours = Cours.GetInstance(aIdentifiantCours.Value);
        oListeStagiaires = oCours.GetListeStagiaires();
    }
    else
    {
        oListeStagiaires = Stagiaire.ListeStagiaires;
    }
    return View(oListeStagiaires);
}
```

4.2 Création du contrôleur *Cours*

De la même manière que pour le contrôleur *Stagiaire*, nous créons le contrôleur *Cours* :



Nous validons et nous obtenons une classe avec une méthode dont l'implémentation est la suivante :

```
// C#

public ActionResult Index()
{
    return View();
}
```

De la même manière que pour le contrôleur Stagiaire, la méthode (action) Index doit être capable de retourner la liste de tous les cours, ainsi que la liste des cours d'un stagiaire. Pour ce faire, nous modifions l'implémentation de la méthode *Index* :

```
// C#

public ActionResult Index(int? aIdentifiantStagiaire)
{
    List<Cours> oListeCours = null;
    if (aIdentifiantStagiaire.HasValue)
    {
        oListeCours =
Stagiaire.GetInstance(aIdentifiantStagiaire.Value).ListeCours;
    }
    else
    {
        oListeCours =  Cours.GetListeInstances();
    }
    return View(oListeCours);
}
```

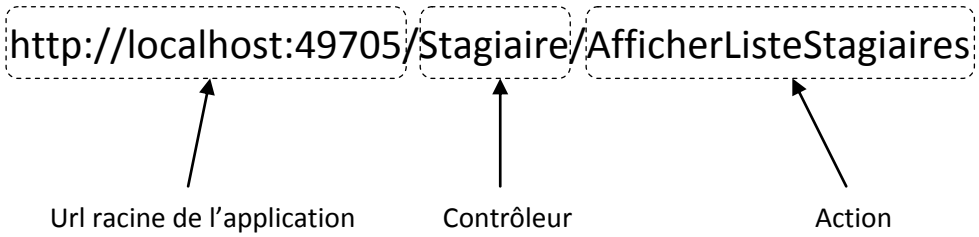
4.3 Test des contrôleurs

Pour le moment, nous n'avons pas créé les vues correspondant aux méthodes (actions) de nos deux contrôleurs. Nous allons donc créer une méthode (action) spécifique, dans le sens où elle n'est pas rattachée à la vue, puis l'exécuter.

Dans le contrôleur *Stagiaire*, nous allons ajouter une méthode nommée *AfficherListeStagiaires*, dont l'implémentation est la suivante :

```
// C#

public void AfficherListeStagiaires()
{
    Response.Write("Vous avez exécutez l'action Tester du contrôleur
StagiaireController.");
    Response.Write("<br /><br />");
    Response.Write("Voici la liste des stagiaires : ");
    Response.Write("<ul>");
    foreach (Stagiaire oStagiaire in Stagiaire.GetListeInstances())
    {
        Response.Write("<li>" + oStagiaire.Nom + " " + oStagiaire.Prenom
+ "</li>");
    }
    Response.Write("</ul>");
}
```

Pour exécuter ce contrôleur, nous allons ouvrir un navigateur Web et exécuter l'URL suivante :



Et nous obtenons le résultat suivant :

Vous avez exécutez l'action AfficherListeStagiaires du contrôleur StagiaireController.

Voici la liste des stagiaires :

- RAVAILLE James
- PAGES Anthony
- DOLLON Julien
- VERGNAULT Bertrand

5 Création des vues

5.1 Création de la vue des stagiaires

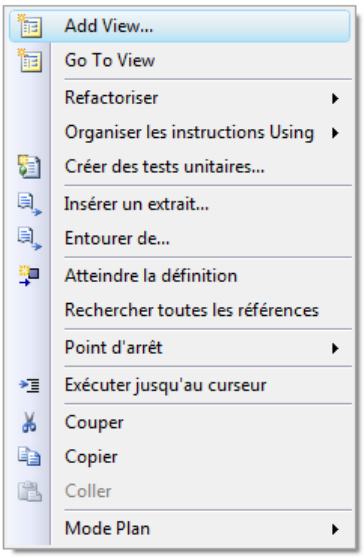
La vue des stagiaires doit permettre d'afficher la liste de tous les stagiaires du centre de formation. Pour chacun d'entre eux, elle devra permettre d'afficher la liste des cours qui lui sont dispensés.

5.1.1 Création de la vue

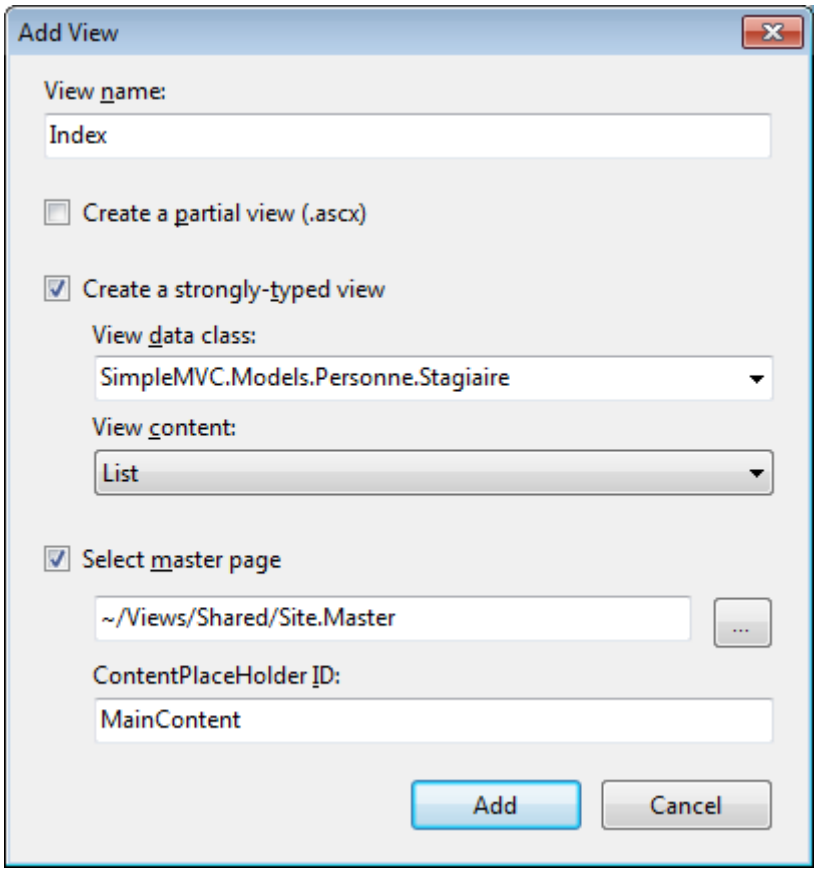
Visual Studio propose de manière de créer une vue :

- Soit via le menu contextuel du répertoire *Views* dans l'explorateur de solution.
- Soit directement à partir d'une méthode (action) du contrôleur. Dans notre projet, nous utiliserons cette méthode pour créer notre vue de stagiaires :

```
public class StagiaireController : Controller
{
    //
    // GET: /Stagiaire/

    public ActionResult Index()
    {
        return View();
    }

}
```



La fenêtre suivante apparaît alors :



Cette fenêtre demande les informations suivantes :

- Nom de la vue : nom de la page ASP .NET MVC. Nom donnons le même nom que l'action.
- Créer une vue partielle : si elle est cochée, la vue créer sera un contrôle utilisateur Web. Sinon une page Web le cas échéant. Nous créerons une page.
- Créer une vue fortement typée : nous créons une vue pour gérer les stagiaires de notre modèle. Alors nous sélectionnons la classe Stagiaire, afin de signifier que la vue gère des stagiaire.
- Comme contenu de la vue, nous choisissons l'élément *List*, afin de signifier que nous voulons lister les stagiaires.
- Nous créons une page de contenu. Elle s'exécutera au sein du contrôle ContentPlaceHolder identifié par *MainContent* de la page de l'application Maître nommée *Site.Master*.

Lorsque nous validons, un répertoire Stagiaire est créé dans le répertoire *Views*. Ce répertoire contient la vue *GestionStagiaires.aspx*. Cette page sera alors exécutée et fournie à l'utilisateur, à chaque fois que l'action *Index* sera invoquée sur le contrôleur *CoursController*. Cette vue est une page ASP .NET MVC contenant le code suivant :

```
// XHTML

<%@ Page Title="" Language="C#"
MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage<IEnumerable<SimpleMVC.Models.Personne.S
tagiaire>>" %>

<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent"
runat="server">
      GestionStagiaires
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent"
runat="server">

    <h2>GestionStagiaires</h2>

    <table>
        <tr>
            <th></th>
            <th>
                Identifiant
            </th>
            <th>
                Nom
            </th>
            <th>
                Prenom
            </th>
        </tr>

    <% foreach (var item in Model) { %>

        <tr>
            <td>
                <%= Html.ActionLink("Edit", "Edit", new { /*
id=item.PrimaryKey */ }) %> |
                <%= Html.ActionLink("Details", "Details", new { /*
id=item.PrimaryKey */ })%>
            </td>
            <td>
                <%= Html.Encode(item.Identifiant) %>
            </td>
            <td>
                <%= Html.Encode(item.Nom) %>
            </td>
            <td>
                <%= Html.Encode(item.Prenom) %>
            </td>
        </tr>

    <% } %>

    </table>

    <p>
        <%= Html.ActionLink("Create New", "Create") %>
    </p>

</asp:Content>
```

Cette vue l'ensemble des informations sur les stagiaires : identifiant, nom, prénom, au travers de l'objet Model, qui est alimenté par le contrôleur. Etant donné que nous ne n'allons pas modifier les stagiaires, mettons en commentaire :

- Le code correspondant à la première colonne, permettant de modifier et consulter le détail d'un stagiaire :
    - L'entête de la colonne :

```
// XHTML

<th></th>
```

    - Le contenu de la colonne :

```
// XHTML

<td>
 <%= Html.ActionLink("Edit", "Edit", new { /* id=item.PrimaryKey */ }) %>
|
 <%= Html.ActionLink("Details", "Details", new { /* id=item.PrimaryKey */
})%>
</td>
```

- Le code permettant de créer un stagiaire :

```
// XHTML

<p>
      <%= Html.ActionLink("Create New", "Create") %>
</p>
```

Nous reviendrons sur ces instructions en détail dans le chapitre suivant. Ouvrons un navigateur et exécutons l'URL permettant d'exécuter cette vue :

Index

Identifiant	Nom	Prenom
1	RAVAILLE	James
2	PAGES	Anthony
3	DOLLON	Julien
4	VERGNAULT	Bertrand

Pour pouvoir accéder à la liste des cours de chacun des stagiaires, nous allons ajouter une nouvelle colonne où chaque item permettra d'accéder à la liste des cours de chaque stagiaire. Dans cette vue, nous ajoutons le code marqué en gras ci-dessous :

```
// XHTML

        <tr>
            <%--<td>
                <%= Html.ActionLink("Edit", "Edit", new { /*
id=item.PrimaryKey */ }) %> |
                <%= Html.ActionLink("Details", "Details", new { /*
id=item.PrimaryKey */ })%>
            </td>--%>
            <td>
                <%= Html.Encode(item.Identifiant) %>
            </td>
            <td>
                <%= Html.Encode(item.Nom) %>
            </td>
            <td>
             <%= Html.Encode(item.Prenom) %>
            </td>
            <td>
                 <%= Html.ActionLink("Cours", "Index", "Cours", new
{ aIdentifiantStagiaire = item.Identifiant }, null)%>
             </td>
        </tr>
```

Le résultat est le suivant :

Index

Identifiant	Nom	Prenom	
1	RAVAILLE	James	Cours
2	PAGES	Anthony	Cours
3	DOLLON	Julien	Cours
4	VERGNAULT	Bertrand	Cours

Le bloc de code ajouté permet d'ajouter un lien hypertext, dont le libellé sera « Cours ». Lorsque l'utilisateur cliquera sur ce lien, alors une requête HTTP sera envoyée au serveur Web, dont l'URL est la suivante :

http://localhost:49705/Cours?aIdentifiantStagiaire=1

Cette URL permet d'exécuter la méthode (action) Index (méthode par défaut) en passant la valeur 1 en paramètre.

Si nous souhaitons consulter la liste des cours de « James RAVAILLE », nous cliquons sur le lien « Cours » correspondant, et la page suivante apparaît :

Index

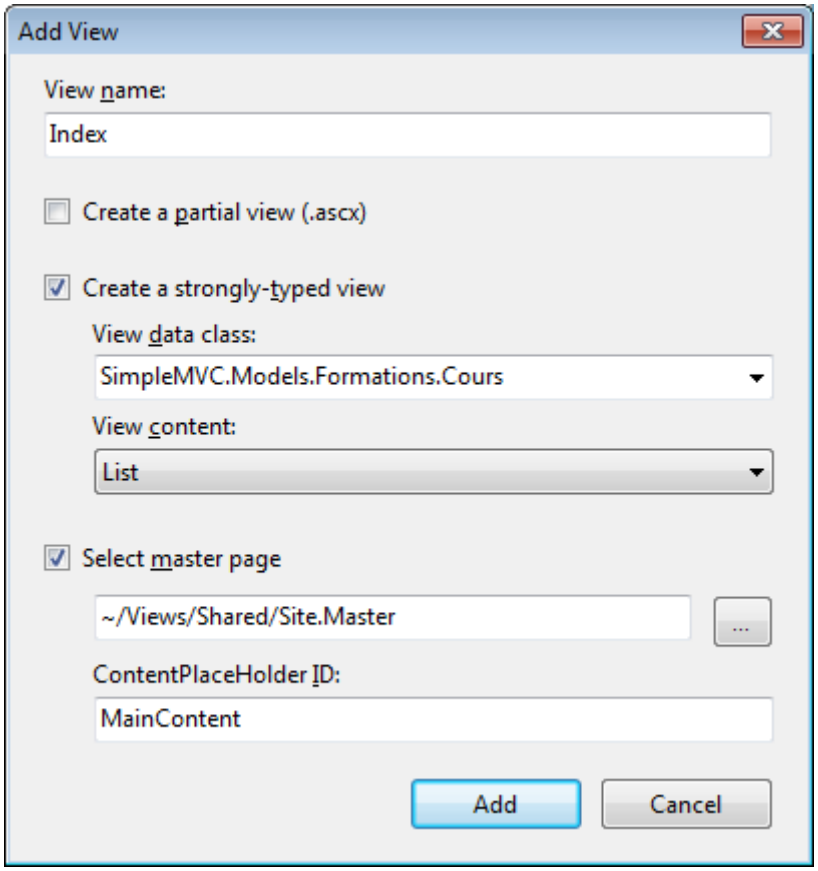
Identifiant	Libelle	NombreJours	ELearning
1	Formation C# 3.0	5	True
2	ASP .NET 3.5	4	False
3	Architecture d'applications	2	False
4	Conception de bases de données SQL Server	3	True

5.2 Création de la vue des cours

La vue des stagiaires doit permettre d'afficher la liste de tous les cours dispensés, tous stagiaires confondus. Elle devra permettre pour chacun d'entre de fournir la liste des stagiaires qui le suivront.

5.2.1 Création de la vue

De manière analogue à la création de la vue des stagiaires, nous créons la vue des cours :



Nous apportons les mêmes modifications que pour la création de la vue des stagiaires :

- Mise en commentaire de la colonne permettant de modifier le contenu d'un cours.
- Mise en commentaire du code permettant de créer un cours.

Index

Identifiant	Libelle	NombreJours	ELearning
1	Formation C# 3.0	5	True
2	ASP .NET 3.5	4	False
3	Architecture d'applications	2	False
4	Conception de bases de données SQL Server	3	True
5	Formation VB 9.0	5	True
6	Administration de bases de données SQL Server	5	False
7	Windows Presentation Foundation	5	True
8	Silverlight	2	True
9	.NET RIA Services	3	False
10	MOOS 2007	5	False

6 Cadre de l'application

6.1 Modification de la Master Page

6.1.1 Présentation

Dans l'exemple de projet que fourni Microsoft, la Master Page définit la structure générale de l'application. Elle est contenue dans le répertoire */Views/Shared* de l'application. Elle contient un contrôle de type *ContentPlaceHolder*, dans lequel s'exécuteront les vues que nous avons créées.

6.1.2 Modification du titre de l'application

Pour modifier le titre de l'application, il suffit de se positionner dans la division identifiée par *title*, contenant le titre symbolisé par l'élément XHTML *H1*. Nous le modifions de la manière suivante :

```
<div id="title">
    <h1>Gestion du centre de formation</h1>
</div>
```

6.1.3 Modification des éléments du menu

Les éléments du menu vont nous permettre d'accéder aux différentes pages de l'application. Nous allons ajouter deux éléments de menu :

- Un permettant d'accéder à la vue des stagiaires, afin d'afficher la liste de tous les stagiaires du centre de formation. Nous devons alors faire appel à l'action « Index » du contrôleur *Stagiaire*.
- Un permettant d'accéder à la vue des cours, afin d'afficher la liste de tous les cours proposés par le centre de formation. Nous devons alors faire appel à l'action « Index » du contrôleur *Cours*.

```
// XHTML

<ul id="menu">
    <%--<li><%= Html.ActionLink("Home", "Index", "Home")%></li>
    <li><%= Html.ActionLink("About", "About", "Home")%></li>--%>

    <li><%= Html.ActionLink("Stagiaire", "Index", "Stagiaire")%></li>
    <li><%= Html.ActionLink("Cours", "Index", "Cours")%></li>
</ul>
```

`Html.ActionLink`

Le menu est mis en forme au travers de la feuille de style *CSS/Content/Site.css*.

6.2 Modification de la page de démarrage

La page de démarrage de l'application correspond à la vue Index.aspx contenue dans le répertoire */Views/Home* de l'application. Nous modifions son contenu afin d'obtenir le bloc de code suivant :

```
// XHTML

<%@ Page Language="C#" MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage" %>

<asp:Content ID="indexTitle" ContentPlaceHolderID="TitleContent"
runat="server">
    Page de démarrage
</asp:Content>

<asp:Content ID="indexContent" ContentPlaceHolderID="MainContent"
runat="server">
    <h2><%= Html.Encode(ViewData["Message"]) %></h2>
    <p>
        Simple démonstration de la création d'une application ASP .NET
MVC.
    </p>
</asp:Content>
```

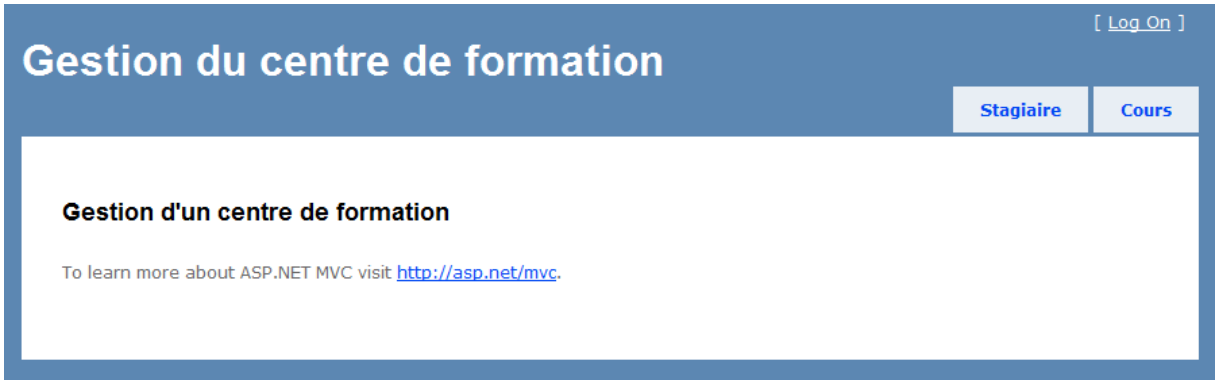
L'instruction *Html.Encode(ViewData["Message"])* récupère le message défini dans la méthode (action) *Index* du contrôleur *Home*, dont le bloc de code est le suivant :

```
// XHTML

public ActionResult Index()
{
    ViewData["Message"] = "Gestion d'un centre de formation";

    return View();
}
```

7 Exécution de l'application

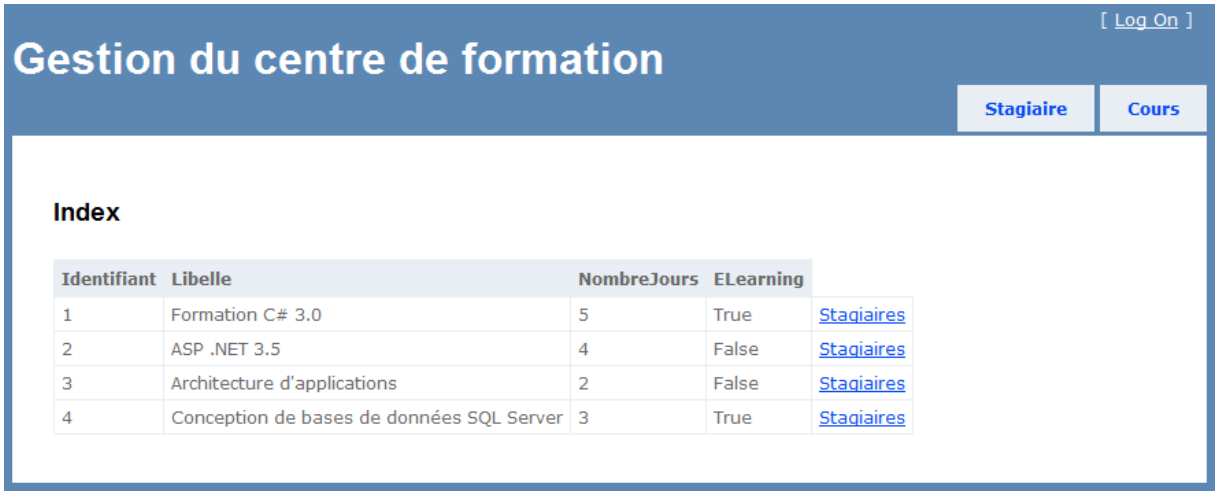
Dans notre application, la page *Default.aspx* est la seule page que nous pouvons exécuter directement, sans avoir à passer par un contrôleur. Les vues ne peuvent être exécutées directement.

Pour exécuter la page *Default.aspx*, dans la fenêtre Explorateur de solutions de Visual Studio, affichons le menu contextuel sur ce fichier et cliquons sur l'item « Afficher dans le navigateur ». La page suivante apparaît :



Lorsqu'on clique sur l'élément du menu *Stagiaire*, la page suivante apparaît :

Puis, si on clique un lien hypertext Cours du stagiaire « James RAVAILLE » , la page suivante apparaît :



Enfin, il est possible de consulter la liste des stagiaires d'une formation (sous-ensemble de stagiaires du centre de formation), en cliquant sur le lien *Stagiaires* du cours « Formation C# 3.0 » :

8 Conclusion

Le but de ce cours est de vous montrer en quoi consiste un simple modèle, de simples vues et contrôleurs d'une application ASP .NET MVC, de manière à mettre en pratique les concepts vus dans le chapitre de présentation du modèle ASP .NET MVC.

Ces premiers pas dans ASP .NET MVC ne sont pas suffisants pour en explorer tous les concepts, et être autonome. Les chapitres suivants publiés sur Dotnet-France concernant ASP .NET MVC permettront de détailler chacun de ces concepts.