**Le chapitre 02**

1. **Les contrôleurs "Controllers"**

Les contrôleurs MVC sont chargés de contrôler le déroulement de l'exécution d'une requête HTTP. Lorsque vous faites une demande à une application MVC, c'est une méthode d'action du contrôleur qui traite la demande et retourne une réponse à cette demande.

Par convention, une classe contrôleur doit :

* Résider dans le dossier "***Controllers"*** *de* niveau racine du projet
* Hériter de la classe "Microsoft.AspNetCore.Mvc.**Controller**"
* Le nom de la classe est suffixé par le mot "**Controller**"

La classe "**Controller**" possède les propriétés et méthodes suivantes :

* **ModelState** : contient l'état de validation du modèle.
* **Request** : représente la requête http ;
* **Response** : représente la réponse http ;
* **TempDate** : collection qui contient les données pour l'action ;
* **Url** : C'est un objet qui contient l'url d'appel de l'action ;
* **ViewData** : collection qui contient les données pour la vue (ViewResult)
* **ViewBag** : C'est un objet dynamique qui contient les données pour la vue.
* **HttpContext** : Contient le contexte de la requête http

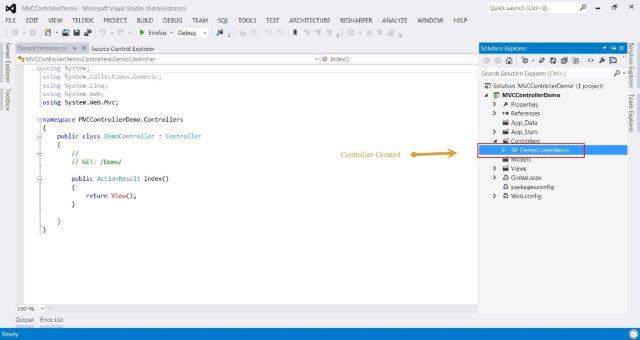
Les méthodes :

* **View**( ) : qui retourne un objet de type **ViewResult**
* **File**( ) : qui retourne un objet de type **VirtualFileResult**
* **Json**( ) : retourne un objet de type **JsonResult**
* **NoContent**( ): retourne un objet de type **NoContentResult**
* **Ok**( ) : retourne un objet de type **OkObjectResult**
* **PartialView**( ) : retourne un objet de type **PartialViewResult**
* **RedirectToAction**( ) : retourne un objet de type **RedirectActionResult**
* **Redirect**( ) : retourne un objet de type RedirectResult

1. **La création d'un contrôleur**

Pour ajouter un contrôleur à votre application, faites un clic droit sur le dossier "**Controllers**" de votre application MVC et sélectionnez l'option de menu Ajouter-> Contrôleur. Après la boîte de dialogue Ajouter Contrôleur est affichée. Donnez un nom à votre contrôleur (le nom est toujours préfixé par le mot "Controller").

Un fichier sera ajouter dans le dossier "**Controllers**" comme ceci:



1. **Les types de retour d'une méthode d'action**

Une méthode d'action du contrôleur peut retourner différents résultats de type "**ActionResult**" à une demande particulière.

Vous pouvez créer des méthodes d'action qui retournent un objet de n'importe quel type, comme une chaîne, un entier ou une valeur booléenne. Ces types de retour sont enveloppés dans un objet de type "**ActionResult**"  avant qu'ils ne soient rendus au flux de réponse.

Le tableau suivant montre les types de résultats des méthodes d'action et les méthodes qui les retournent.

|  |  |  |
| --- | --- | --- |
| **Résultat d'Action** | **La méthode qui le génère** | **Description** |
| [ViewResult](https://msdn.microsoft.com/en-us/library/system.web.mvc.viewresult(v=vs.118).aspx) | [View](https://msdn.microsoft.com/en-us/library/system.web.mvc.controller.view(v=vs.118).aspx#M:System.Web.Mvc.Controller.View) | Rend une vue comme une page Web. |
| [PartialViewResult](https://msdn.microsoft.com/en-us/library/system.web.mvc.partialviewresult(v=vs.118).aspx) | [PartialView](https://msdn.microsoft.com/en-us/library/system.web.mvc.controller.partialview(v=vs.118).aspx#M:System.Web.Mvc.Controller.PartialView) | Rend une vue partielle, qui définit une section d'une vue qui peut être rendu à l'intérieur une autre vue. |
| [RedirectResult](https://msdn.microsoft.com/en-us/library/system.web.mvc.redirectresult(v=vs.118).aspx) | [Redirect](https://msdn.microsoft.com/en-us/library/system.web.mvc.controller.redirect(v=vs.118).aspx#M:System.Web.Mvc.Controller.Redirect(System.String)) | Redirige à une autre méthode d'action en utilisant son URL. |
| [RedirectToRouteResult](https://msdn.microsoft.com/en-us/library/system.web.mvc.redirecttorouteresult(v=vs.118).aspx) | [RedirectToAction](https://msdn.microsoft.com/en-us/library/system.web.mvc.controller.redirecttoaction(v=vs.118).aspx#M:System.Web.Mvc.Controller.RedirectToAction(System.String)) | Redirige à une autre méthode d'action en utilisant son nom. |
| [JsonResult](https://msdn.microsoft.com/en-us/library/system.web.mvc.jsonresult(v=vs.118).aspx) | [Json](https://msdn.microsoft.com/en-us/library/system.web.mvc.controller.json(v=vs.118).aspx#M:System.Web.Mvc.Controller.Json(System.Object)) | Renvoie un objet sérialisé en format JSON. |
| [JavaScriptResult](https://msdn.microsoft.com/en-us/library/system.web.mvc.javascriptresult(v=vs.118).aspx) | [JavaScript](https://msdn.microsoft.com/en-us/library/system.web.mvc.controller.javascript(v=vs.118).aspx#M:System.Web.Mvc.Controller.JavaScript(System.String)) | Renvoie un script qui peut être exécuté sur le client. |
| [FileContentResult](https://msdn.microsoft.com/en-us/library/system.web.mvc.filecontentresult(v=vs.118).aspx) | [File(Byte[], String, String)](https://msdn.microsoft.com/en-us/library/system.web.mvc.controller.file(v=vs.118).aspx#M:System.Web.Mvc.Controller.File(System.Byte[],System.String,System.String)) | Envoie le contenu d'un fichier binaire comme réponse. |
| [FilePathResult](https://msdn.microsoft.com/en-us/library/system.web.mvc.filepathresult(v=vs.118).aspx) | [File(String, String)](https://msdn.microsoft.com/en-us/library/system.web.mvc.controller.file(v=vs.118).aspx#M:System.Web.Mvc.Controller.File(System.String,System.String)) | Envoie le contenu d'un fichier comme réponse. |

**Remarque**:

L'URL - http://localhost/MVCDemo/Home/Index, fait appel à la méthode d'action **Index**() du contrôleur "**HomeController" (voir le fichier Startup.cs qui définit le routage des requêtes http)**.

L'URL suivante n'a pas le paramètre "id". Ceci n'est pas un problème parce que le paramètre id est optionnel.

<http://localhost/MVCDemo/Home/Index>

Si vous voulez recevoir le paramètre "id" dans la méthode d'action "Index", changer la méthode comme suit:

public string Index(string id)  
{  
    return "The value of Id = " + id;  
}

Taper l'URL : http://localhost/MVCDemo/Home/Index/10  
  
Si vous voulez envoyer un deuxième paramètre à la méthode "Index", vous devez le nommer dans l'url.

Exemple: http://localhost/MVCDemo/home/index/10?name=Ahmed  
  
Changer la méthode "Index" de tel sorte qu'elle accepte deux paramètres "id" et "name".

public string Index(string id, string name)  
{  
    return "La valeur du parameter Id = " + id + " et Name = " + name;  
}

On peut aussi accéder à la valeur du paramètre "name" en utilisant la propriété "Query" de l'objet "Request" : this.**Request.Query**

public string Index(string id)  
{  
    return "La valeur du parameter Id = " + id + " et Name = " + this.Request.**Query**["name"];  
}

1. **Les Vues "View"**

Les vues sont des fichiers avec l'extension ".**cshtml**". Elles sont impliquées dans la réalisation de l'interface utilisateur de l'application. Ces vues sont généralement liées aux données du modèle. Pour rendre ces contenus statiques et dynamiques au navigateur, le Framework MVC utilise le moteur de vue "View Engine" qui traduit le code C# en page HTML.

Le Framework MVC est livré avec deux moteurs de vue "**ViewEngine**" intégrés:

1. **Moteur** **Razor:** Razor est une syntaxe de balisage qui permet d'insérer du code C# ou VB dans des pages Web du côté serveur en ajoutant le symbole **"@"** avant la ligne du code. Ce code du côté serveur peut être utilisé pour créer un contenu dynamique lorsque la page Web est en cours de chargement. Razor est un moteur avancé par rapport au moteur ASPX et a été lancé dans les versions ultérieures de MVC.

2. **Moteur** **ASPX:** ASPX ou le moteur Web Forms est le moteur d'affichage par défaut qui est inclus dans le cadre MVC depuis le début. Écrire du code avec ce moteur est très similaire à l'écriture du code dans ASP.NET Web Forms à l'aide des symboles **"<% et %>".**

Exemple de petits extraits de code comparant les deux moteur **Razor** et **ASPX**.

**Le Razor:**

@ Html.ActionLink ("Create New", "useradd")

**ASPX:**

**<%** Html.ActionLink ( "SignUp", "Inscription")**%>**

**Remarque**: Sur ces deux, le moteur **Razor** est plus avancé car il est livré avec une syntaxe compacte. Nous allons utiliser le moteur Razor dans tous nos exemples, car il est le moteur de vues le plus utilisé.

Une vue est de type "**ViewResult**" qui hérite de la classe "**ActionResult**". Elle possède les propriétés et les méthodes suivantes :

Les propriétés

**ContentType** Gets or sets the Content-Type header for the response.

**Context** Contexte de la requete http

**Model** Gets the view data model.

**StatusCode** Gets or sets the HTTP status code.

**TempData** Gets or sets the ITempDataDictionary for this result.

**ViewData** Gets or sets the ViewDataDictionary for this result.

**ViewName** Gets or sets the name or path of the view that is rendered to the response.

Les méthodes

**ExecuteResult**(ActionContext) Executes the result operation of the action method synchronously. This method is called by MVC to process the result of an action method.

**ExecuteResultAsync**(ActionContext)

1. **Les méthodes d'assistance HTML**

Une méthode d'assistance HTML " Helper HTML" est une méthode qui renvoie une chaîne de type "**MvcHtmlString**" ou "**HtmlString**". La chaîne peut représenter tout type de contenu que vous souhaitez. Par exemple, vous pouvez utiliser les méthodes d'assistance HTML pour rendre les balises HTML standards comme les tags HTML <input> et <img>.

Vous pouvez également utiliser les méthodes d'assistance HTML pour rendre le contenu plus complexe, comme une bande de tabulation ou un tableau HTML des données de base de données.

L'infrastructure ASP.NET MVC inclut des méthodes d'assistance qui offrent un moyen simple pour la génération du rendu HTML dans une vue. La liste suivante répertorie certains des programmes d'assistance HTML actuellement disponibles:

**BeginForm** : marque le début d'un formulaire et établit un lien vers la méthode d'action qui restitue le formulaire.

**CheckBox** : restitue une case à cocher.

**DropDownList** : restitue une liste déroulante.

**ListBox** : restitue une zone de liste.

**Password** : restitue une zone de texte pour l'entrée d'un mot de passe.

**RadioButton** : restitue une case d'option.

**TextArea** : restitue un espace de texte (zone de texte multi ligne).

**TextBox** : restitue une zone de texte.

**ActionLink** : établit un lien vers une méthode d'action.

**Hidden** : incorpore les informations du formulaire qui ne sont pas restituées pour que l'utilisateur puisse les consulter.

1. **Utilisation de la méthode d'assistance BeginForm**

La méthode d'assistance "BeginForm" génère la balise html "form" qui marque le début d'un formulaire et établit un lien vers la méthode d'action qui restitue le formulaire.

**@using (Html.BeginForm("action", "controller", la méthode post/get))**

**Utilisation de la méthode d'assistance TextBox**

La méthode d'assistance TextBox restitue une zone de texte portant le nom spécifié. Le balisage suivant correspond à celui de la méthode d'assistance TextBox.

**@Html.Label("Prenom", "Prénom:",new { id = "lblprenom" })**

**@Html.TextBox("txtPrenom","Allo")**

1. **Utilisation de la méthode d'assistance CheckBox**

La méthode d'assistance CheckBox restitue une case à cocher portant le nom que vous spécifiez. Le contrôle restitué retourne une valeur booléenne (true ou false).

**@Html.Label("Arabe", new { id = "lblarabe" })**

**@Html.CheckBox("Arabe",true, new { value="Arabe"})**

1. **Utilisation de la méthode d'assistance DropDownList**

Le programme d'assistance DropDownList restitue une liste déroulante. Dans sa forme la plus simple, DropDownList accepte un paramètre, le nom de la clé ViewData dont la valeur est de type SelectList et qui contient les valeurs d'option de la liste déroulante. L'infrastructure MVC utilise la propriété ModelState de ViewData pour déterminer la valeur sélectionnée. Si la propriété ModelState est vide, l'infrastructure recherche un élément dont la propriété Selected est définie.

**@Html.DropDownList("age",new List<SelectListItem> {**

**new SelectListItem(){Text="1", Value="1"},**

**new SelectListItem(){Text="2", Value="2"},**

**new SelectListItem(){Text="3", Value="2"}**

**}, "Selectionner une valeur")**

1. **Utilisation de la méthode d'assistance RadioButton**

La méthode d'assistance RadioButton restitue une case d'option. Dans sa forme la plus simple, la méthode accepte trois paramètres : le nom du groupe de contrôles, la valeur d'option et une valeur Boolean qui détermine si la case d'option est sélectionnée initialement. Le balisage suivant correspond à celui de la méthode d'assistance RadioButton.

**@Html.Label("Masculin", new { id = "lbmasculin" })**

**@Html.RadioButton("Sexe", "Masculin", true, new {id="rdMasculin" })**

**@Html.Label("Féminin", new { id = "lblfeminin" })**

**@Html.RadioButton("Sexe", "Fémnin", new { id = "rdFeminin" })**

**Exemple** :

<body>

<div>

@using (Html.BeginForm()) {

<table>

<tr>

<td>@Html.Label("txtNumero", "Numéro:")</td>

<td>@Html.TextBox(</td>

</tr>

<tr>

<td>@Html.Label("txtNom", "Nom:")</td>

<td>@Html.TextBox("txtNom")</td>

</tr>

<tr>

<td>@Html.Label("txtPrenom", "Prénom:")</td>

<td>@Html.TextBox("txtPrenom")</td>

</tr>

<tr>

<td>Genre:</td>

<td>@Html.RadioButton("rdGenre","m",true)@Html.Label("Masculin")<br />

@Html.RadioButton("rdGenre", "f", false)@Html.Label("Féminin")</td>

</tr>

<tr>

<td>La Langue:</td>

<td>

@Html.DropDownList("langue",new List<SelectListItem>

{

new SelectListItem(){Text="Arabe", Value="arabe"},

new SelectListItem(){Text="Anglais", Value="anglais"},

new SelectListItem(){Text="Français", Value="francais"},

})

</td>

</tr>

<tr><td colspan="2" align="center">

<input type="reset" value="Reset"/>

<input type="submit" value="Submit" />

</td></tr>

</table>

}

</div>

</body>

1. **La création d'une méthode d'assistance HTML personnalisée**

La meilleure façon de créer une nouvelle méthode d'assistance HTML est de créer une méthode **static** **d'extension** **de la classe HtmlHelper** qui retourne une chaîne de type "**MvcHtmlString**".

**Remarque** :

* La classe qui contient les méthodes d'extension doit être **static**;
* La méthode d'extension doit être **static** et retourne un objet de type **MvcHtmlString**;
* La méthode doit avoir un paramètre de type "**HtmlHelper** " qui représente la classe à étendre et qui est précédé par le mot clé **this**.

Imaginez, par exemple, que vous décidez de créer une nouvelle méthode d'assistance HTML qui rend un bouton "**submit"**. Voici le code de la classe:

1. Utilisation de la classe **MvcHtmlString**

public **static** class SubmitButtonHelper {

public **static** **MvcHtmlString** SubmitButton(**this** **HtmlHelper** helper, string buttonText)

{

string rep= String.Format("<input type=\"submit\" value=\"{0}\" />", buttonText);

return **MvcHtmlString.Create(rep);**

}

}

1. Utilisation de la classe **TagBuilder**

public **static** class ButtonHelper

{

public **static** **MvcHtmlString** Button(**this** **HtmlHelper** helper, string buttonText,ButtonType buttonType, Object htmlAttributes)

{

var submitTag = new TagBuilder("input");

submitTag.**MergeAttribute**("type", buttonType.ToString());

submitTag.**MergeAttribute**("value", buttonText);

submitTag.**MergeAttributes**(HtmlHelper.**ObjectToDictionary**(htmlAttributes));

ou

//submitTag.MergeAttributes(HtmlHelper.**AnonymousObjectToHtmlAttributes**(htmlAttributes));

ou

//submitTag.MergeAttributes(new **RouteValueDictionary**(htmlAttributes));

return MvcHtmlString.Create(submitTag.ToString(TagRenderMode.SelfClosing));

}

}

public enum ButtonType {button, submit, reset}

}

La vue qui utilise le bouton

@using WebApplication2.Models

@{

ViewBag.Title = "Test";

}

<h2>Bienvenue dans mon site Web</h2>

<p>On est le :@DateTime.Now.ToString("dd/MM/yyyy")</p>

@Html.BeginForm()

@Html.Label("nom", "Nom:")

@Html.TextBox("nom")

@Html.Label("prenom", "Prénom:")

@Html.TextBox("prenom")

@Html.Button("Envoyer",ButtonType.submit,new {id="btSubmit" })

@{Html.EndForm();}

1. **La création des tags d'aide personnalisés**

Les balises d'assistance nous permettent d'exécuter le code côté serveur pour générer du code HTML dans une vue.

Une balise d'assistance ce n'est rien d'autre qu'une classe qui implémente l'interface "**ITagHelper"** ou qui dérive de la classe **"TagHelper"**.

Pour créer un tag d'assistance suivez les étapes suivantes :

1. Créer une classe qui hérite de la classe "**TagHelper**". Cette classe a deux méthodes virtuelle pour générer des balises HTML. Une synchrone nommée "**Process**" et l'autre asynchrone nommée "**ProcessAsync**".
2. **public** **virtual** Task **ProcessAsync**(TagHelperContext context, TagHelperOutput output);
3. **public** **virtual** **void** **Process**(TagHelperContext context, TagHelperOutput output);

On implémente l'une de ces deux ou les deux méthodes, en fonction de nos besoins. La méthode Process (ou ProcessAsync) est chargée de générer le code HTML à envoyer au navigateur.

Cette méthode reçoit deux paramètres, un de type "**TagHelperContext**" et le deuxième de type "**TagHelperOutput**".

Exemple : Dans cette exemple on va créer une balise nommée "dropdownlist".

[**HtmlTargetElement**("dropdownlist")]

public class DropDownListTagHelper :**TagHelper**

{

}

Remarque : A l'aide de l'attribut [**HtmlTargetElement( )**], on peut associer un nom à notre nouvelle balise

1. Ajouter à cette classe les propriétés que vous voulez

Dans notre cas, le tag "dropdownlist" a besoin des attributs (name, la liste des valeurs, la liste du texte à afficher et d'autres attributs que l'utilisateur veut ajouter à la balise).

[HtmlTargetElement("dropdownlist")]

public class DropDownListTagHelper :TagHelper

{

public string Name { get; set; }

[HtmlAttributeName("asp-values")]

public List<string> AspValues { get; set; }

[HtmlAttributeName("asp-texte")]

public List<string> AspTexte { get; set; }

[HtmlAttributeName("asp-attributes")]

public object HtmlAttributes { get; set; }

}

Remarque : A l'aide de l'attribut [**HtmlAttributeName( )**], on peut associer un nom à nos propriétés de la balise.

1. Implémentation de la méthode "Process" ou "ProcessAsync"

La classe "**TagHelperOutput**" possède les propriétés suivantes :

* **TagName** : représente le nom de la balise HTML cible
* **TagMode** : représente le type de la balise (avec contenu ou sans contenu)
* **Attributes** : C'est une collection des attributs de la balise cible
* **Content** : représente le contenu de la balise cible

public class DropDownListTagHelper :TagHelper

{

public string Name { get; set; }

[HtmlAttributeName("asp-values")]

public List<string> AspValues { get; set; }

[HtmlAttributeName("asp-texte")]

public List<string> AspTexte { get; set; }

[HtmlAttributeName("asp-attributes")] public object HtmlAttributes { get; set; }

public override void Process(TagHelperContext context, TagHelperOutput output)

{

output.TagName = "select";

output.TagMode = TagMode.StartTagAndEndTag;

output.Attributes.Add("name", this.Name);

foreach(var element in HtmlHelper.ObjectToDictionary(this.HtmlAttributes))

{

output.Attributes.Add(element.Key, element.Value);

}

var sb = new StringBuilder();

for (int index= 0; index<this.AspValues.Count; index++)

{

sb.AppendFormat("<option value=\"{0}\">{1}</option>", this.AspValues[index],this.AspTexte[index]);

}

output.Content.SetHtmlContent(sb.ToString());

}

}

1. L'étape suivante consiste à **enregistrer** la balise d'assistance dans le fichier "\_**ViewImports**.**cshtml**" afin qu'elle soit utilisable dans les vues.

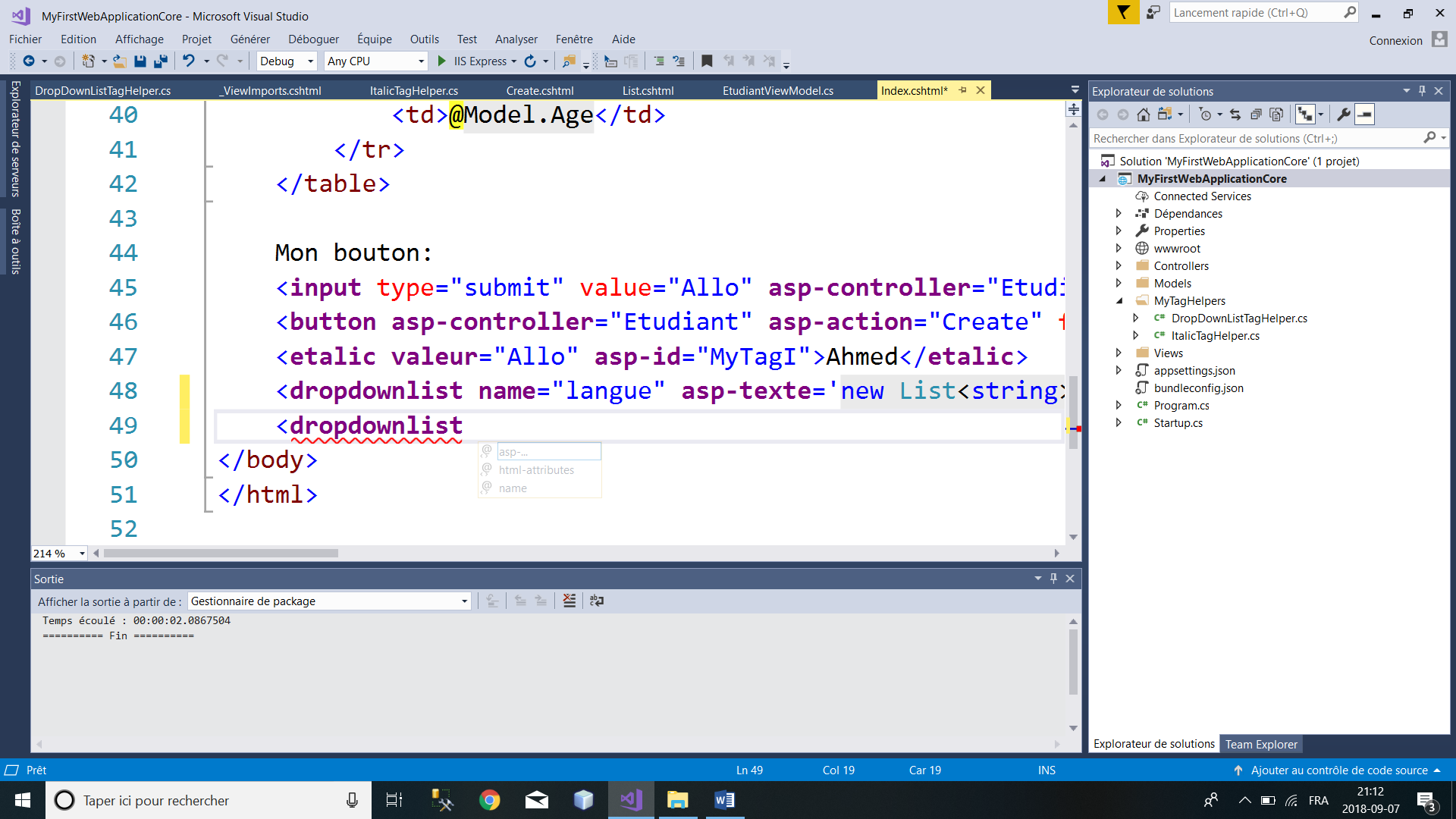
@using MyFirstWebApplicationCore

@using MyFirstWebApplicationCore.Models

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

@addTagHelper \*, **MyFirstWebApplicationCore**

1. Enfin, nous pouvons utiliser notre balise d'aide dans nos vues.



<**dropdownlist** **name**="langue" **asp-texte**='new List<string> {"Arabe", "Francais", "Anglais" }' **asp-values**='new List<string> {"1", "2", "3" }' **html-attributes**='new { id="drop1", color="gris" }'></**dropdownlist**>

La sortie

<select name="langue" id="drop1" color="gris">

<option value="1">Arabe</option>

<option value="2">Francais</option>

<option value="3">Anglais</option>

</select>

**Transmettre un modèle de données à un tag d'aide**

Nous pouvons également transmettre les données du modèle à l'assistant de balise via la liaison de modèle en créant des propriétés de type "**ModelExpression**".

En utilisant la propriété **Model** de cette classe, nous pouvons obtenir la valeur envoyée à la propriété du tag.

Dans l'exemple suivant, j'ai créé un tag d'aide pour afficher les détails de l'employé, tels que le nom et la profession.

namespace CustomTagHelper.TagHelpers

{

using Microsoft.AspNetCore.Mvc.ViewFeatures;

using Microsoft.AspNetCore.Razor.TagHelpers;

using System.Text;

[HtmlTargetElement("employee-details")]

public class EmployeeDetailTagHelper : TagHelper

{

[HtmlAttributeName("for-name")]

public ModelExpression EmployeeName { get; set; }

[HtmlAttributeName("for-profession")]

public ModelExpression Profession { get; set; }

public override void Process(TagHelperContext context, TagHelperOutput output)

{

output.TagName = "EmployeeDetails";

output.TagMode = TagMode.StartTagAndEndTag;

var sb = new StringBuilder();

sb.AppendFormat("<span>Name: {0}</span> <br/>", this.EmployeeName.Model);

sb.AppendFormat("<span>Designation: {0}</span>", this.Designation.Model);

output.PreContent.SetHtmlContent(sb.ToString());

}

}

}

Model

namespace CustomTagHelper.Models

{

using System.ComponentModel.DataAnnotations;

public class EmployeeViewModel

{

public string Name { get; set; }

public string Profession { get; set; }

}

}

Controller

public IActionResult Index1()

{

EmployeeViewModel e = new EmployeeViewModel();

e.Name = "Ahmed Bounouar ";

e.Profession = "Professeur en informatique";

return View(e);

}

View

@model CustomTagHelper.Models.EmployeeViewModel

<div class="row">

<employee-details for-name ="Name" for-profession="Profession"></employee-details>

</div>

1. **Passage des données dans une application ASP.NET MVC**

L'infrastructure ASP.NET MVC fournit des conteneurs qui nous permettent de passer des données entre les contrôleurs et les vues. Cette rubrique explique comment passer des données faiblement et fortement typées dans une application MVC. Elle indique également comment passer des données d'état temporaires entre des méthodes d'action.

1. [**Passage de données entre un contrôleur et une vue**](javascript:void(0))

Pour générer une vue à partir d'un contrôleur, vous devez utiliser la méthode **View**() de la classe Controller.

1. Utilisation de la collection "**ViewData**".

Pour passer des données à la vue à partir d'un contrôleur, vous pouvez utiliser la propriété "[ViewData](https://msdn.microsoft.com/fr-fr/library/dd492506(v=vs.100).aspx)" de la classe "[ControllerBase](https://msdn.microsoft.com/fr-fr/library/system.web.mvc.viewpage(v=vs.100).aspx)". Cette propriété retourne un objet de type "[ViewDataDictionary](https://msdn.microsoft.com/fr-fr/library/system.web.mvc.viewdatadictionary(v=vs.100).aspx)" comportant des clés chaîne de caractères qui ne respectent pas la casse. Pour passer des données à la vue, il suffit d'assigner des valeurs au dictionnaire.

Remarque:

1. ViewData est utilisé pour transmettre des données à partir du contrôleur pour la vue.
2. Elle est de type "ViewDataDictionary"
3. Elle est disponible pour la requête en cours uniquement
4. Vous devez faire un **casting** de type pour lire les données complexes. N'oublier pas de vérifier les valeurs nulles pour éviter les erreurs
5. Si la redirection se produit, sa valeur devient nulle

Exemple 1:

**Le contenu du contrôleur**

public ViewResult GetViewData()

{

Etudiant e = new Etudiant() { Numero = 1, Nom = "Bounouar", PreNom = "Ahmed", Telephone = "819 664-6475", DateNaissance = DateTime.Now };

**ViewData["Etudiant"] = e;**

return View();

}

**Remarque**:

Si vous appelez la méthode "View" sans paramètres (comme indiqué dans l'exemple précédent), la propriété ViewData de l'objet contrôleur est passée à la **vue** qui **porte le même nom que la méthode d'action**.

Dans la page de vue, vous pouvez accéder à la propriété "**ViewData**" de la classe "**ViewPage**" pour obtenir les données passées à la vue. La propriété **ViewData** est un dictionnaire qui prend en charge un indexeur acceptant les clés de dictionnaire.

**Le contenue de la vue**:

@{

ViewBag.Title = "GetViewData";

}

@using CoursMVC.Models

@{

**var etud = (Etudiant)ViewData["Etudiant"];**

}

<h2>GetViewData</h2>

<table>

<tr>

<td>@Html.DisplayName("Numéro")</td><td>@etud.Numero</td>

</tr>

<tr>

<td>@Html.DisplayName("Nom")</td><td>@etud.Nom</td>

</tr>

<tr>

<td>@Html.DisplayName("Date de naissance")</td> <td>@etud.DateNaissance.ToString("dd/MM/yyyy")</td>

</tr>

</table>

Exemple 2:

public ActionResult Index()  
{  
    // Store the list of Countries in ViewData.    
    **ViewData["**Countries"] = new List<string>()  
    {  
         "Algeria",  
         "US",  
         "UK",  
         "Canada"  
    };  
  
    // Finally return a view  
    return View();  
}  
Le contenu de la vue

@{  ViewData["Title"] = "Countries List";}

@{

**var listeCounties = (List<string>)ViewData["Countries"];**

}

<h2>Countries List</h2>  
<ul>  
@foreach (string strCountry in listeCountries)  
{  
    <li>@strCountry</li>  
}  
</ul>

1. Utilisation de la propriété "**ViewBag**"

Pour passer des données à la vue à partir d'un contrôleur, vous pouvez utiliser la propriété "[ViewBag](https://msdn.microsoft.com/fr-fr/library/dd492506(v=vs.100).aspx)" de la classe "[ControllerBase](https://msdn.microsoft.com/fr-fr/library/system.web.mvc.viewpage(v=vs.100).aspx)".

La propriété "[ViewBag](https://msdn.microsoft.com/fr-fr/library/system.web.mvc.controllerbase.viewbag.aspx#P:System.Web.Mvc.ControllerBase.ViewBag)" vous permet de passer dynamiquement les valeurs du contrôleur et la vue. Le "ViewBag" est un objet dynamique qui signifie qu'il n'a pas de propriétés prédéfinies.

Vous pouvez ajouter une propriété à l'objet "[ViewBag](https://msdn.microsoft.com/fr-fr/library/system.web.mvc.controllerbase.viewbag.aspx#P:System.Web.Mvc.ControllerBase.ViewBag)" simplement en l'ajoutant à l'objet.

Pour passer des données à la vue, vous pouvez assigner une valeur à une propriété de l'objet "ViewBag"

Remarque:

1. ViewBag est également utilisé pour transmettre des données à partir du contrôleur à la vue respective
2. ViewBag est une propriété dynamique qui tire parti des nouvelles fonctionnalités dynamiques en C # 4.0
3. Il est également disponible pour la requête en cours uniquement
4. Si la redirection se produit, sa valeur devient nulle
5. Ne nécessite pas de transtypage pour le type de données complexes

Exemple 1:

public ViewResult GetViewBag()

{

Etudiant e = new Etudiant() { Numero = 1, Nom = "Bounouar", PreNom = "Ahmed", Telephone = "819 664-6475", DateNaissance = DateTime.Now };

**ViewBag.Etudiant = e;**

return View();

}

Dans la vue, vous récupérez les valeurs en utilisant le même nom de la propriété de l'objet "**ViewBag**" de la classe "**ViewPage**".

Voir le code suivant:

@{

Layout = null;

}

@using CoursMVC.Models

@{

**var etud = ViewBag.Etudiant;**

}

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>GetViewBag</title>

</head>

<body>

<div>

<h2>GetViewBag</h2>

<table>

<tr>

<td>@Html.DisplayName("Numéro")</td>

<td**>@etud.Numero**</td>

</tr>

<tr>

<td>@Html.DisplayName("Nom")</td>

<td>@etud.Nom</td>

</tr>

<tr>

<td>@Html.DisplayName("Date de naissance")</td>

<td>@etud.DateNaissance.ToString("dd/MM/yyyy")</td>

</tr>

</table>

</div>

</body>

</html>

Exemple 2:

public ActionResult Index()  
{  
    // Store the list of Countries in ViewBag.    
    **ViewBag**.Countries = new List<string>()  
    {  
         "Algeria",  
         "US",  
         "UK",  
         "Canada"  
    };  
  
    // Finally return a view  
    return View();  
}  
Le contenu de la vue

@{  ViewBag.Title = "Countries List";}  
<h2>Countries List</h2>  
<ul>  
@foreach (string strCountry in ViewBag.Countries)  
{  
    <li>@strCountry</li>  
}  
</ul>

1. **Utilisation de l’objet** **Model**

Il suffit d’envoyer un objet comme paramètre à la vue dans la méthode d’action.

public ViewResult GetViewBag()

{

Etudiant etud = new Etudiant() { Numero = 1, Nom = "Bounouar", PreNom = "Ahmed", Telephone = "819 664-6475", DateNaissance = DateTime.Now };

return **View(etud);**

}

Dans la vue, vous récupérez les valeurs en utilisant la propriété "**Model**" de la classe "**ViewPage**". Cette propriété est en lecture seulement.

Voir le code suivant:

@{

Layout = null;

}

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>GetViewBag</title>

</head>

<body>

<div>

<h2>GetViewBag</h2>

<table>

<tr>

<td>@Html.DisplayName("Numéro")</td>

<td**>@Model.Numero**</td>

</tr>

<tr>

<td>@Html.DisplayName("Nom")</td>

<td>@Model.Nom</td>

</tr>

<tr>

<td>@Html.DisplayName("Date de naissance")</td>

<td>@Model.DateNaissance.ToString("dd/MM/yyyy")</td>

</tr>

</table>

</div>

</body>

</html>

1. **Utilisation des vues fortement typées**

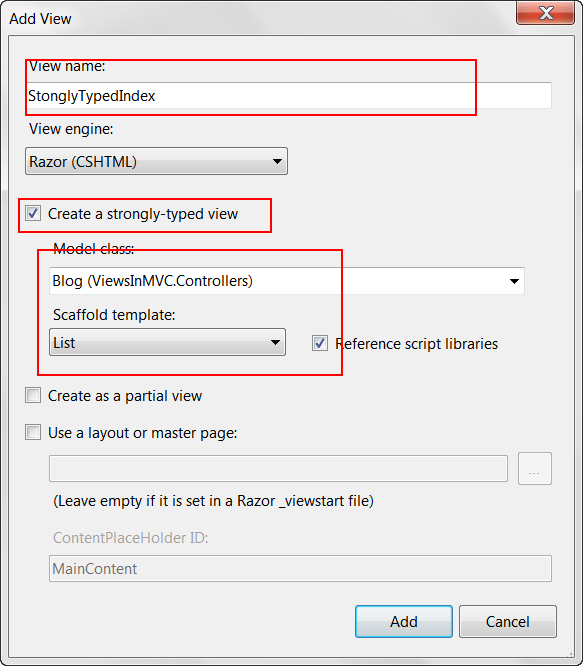
Lorsque vous passez des données entre un contrôleur et une vue à l'aide de la propriété "[ViewData](https://msdn.microsoft.com/fr-fr/library/dd492506(v=vs.100).aspx)" de la classe "ControllerBase", celles-ci ne sont pas fortement typées.

Si vous souhaitez passer des données fortement typées à la vue, il suffit de cocher l'option "Fortement typée" et de sélectionner le type d'objet à transmettre à la vue au moment de la création de la vue. Le système ajoute la ligne suivante dans le code la vue:

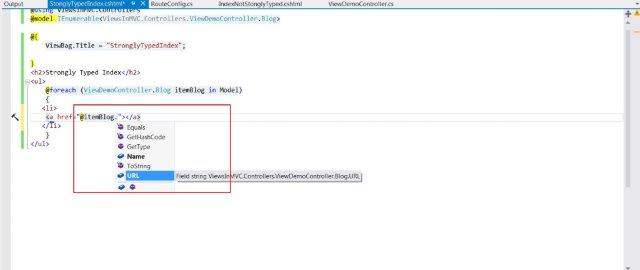
**@model CoursMVC.Models.Etudiant**

**Remarque** : La valeur envoyée par le contrôleur à la vue est accessible par le biais de l’objet « **Model** » au niveau de la vue.

Le modèle est une classe contenant des propriétés pour chaque élément de données que vous souhaitez passer.



Un fichier de vue similaire à la capture d'écran ci-dessous sera créé. Comme vous pouvez le constater, il a inclus la classe Blog de modèle du ViewDemoController au sommet. Vous serez également en mesure d'utiliser IntelliSense dans votre code avec cette approche.



Exemple:

Le modèle:

public class Etudiant

{

[DisplayName("Numéro")]

public int Numero { get; set; }

public string Nom { get; set; }

public string PreNom { get; set; }

[Phone(ErrorMessage="Téléphone non valide")]

[RegularExpression(@"\d{4}", ErrorMessage = "Téléphone non valide")]

public string Telephone { get; set; }

[DisplayName("Date de naissance")]

[Required(ErrorMessage="Le champ date de naissance ne peut etre vide")]

public DateTime DateNaissance { get; set; }

}

La méthode d'action du contrôleur

public ViewResult GetFortementtype()

{

Etudiant e = new Etudiant() { Numero = 56, Nom = "Bounouar", PreNom = "Ahmed", Telephone = "819 664-6475", DateNaissance = DateTime.Now };

return View(e);

}

Le contenu de la vue:

@model CoursMVC.Models.Etudiant

@{

Layout = null;

}

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>GetFortementtype</title>

</head>

<body>

<fieldset>

<legend>Vue fortement typée</legend>

<div class="display-label">

@Html.DisplayNameFor(model => model.Numero)

</div>

<div class="display-field">

@Html.DisplayFor(model => model.Numero)

</div>

<div class="display-label">

@Html.DisplayNameFor(model => model.Nom)

</div>

<div class="display-field">

@Html.DisplayFor(model => model.Nom)

</div>

<div class="display-label">

@Html.DisplayNameFor(model => model.PreNom)

</div>

<div class="display-field">

@Html.DisplayFor(model => model.PreNom)

</div>

<div class="display-label">

@Html.DisplayNameFor(model => model.Telephone)

</div>

<div class="display-field">

@Html.DisplayFor(model => model.Telephone)

</div>

<div class="display-label">

@Html.DisplayNameFor(model => model.DateNaissance)

</div>

<div class="display-field">

@Html.DisplayFor(model => model.DateNaissance)

</div>

</fieldset>

</body>

</html>

Exemple 2:

Le modèle:

public class Blog

{

public string Nom;

public string URL;

}

Le contrôleur :

public class ViewDemoController: Contrôleur

{

public ActionResult StonglyTypedIndex ()

{

List<Blog> topBlogs = new List<Blog>

{

new Blog {Name = "Joe Delage", URL = "http: // cegepoutaouais / joe /"},

new Blog {Name = "Mark Dsouza", URL = "http: // cegepoutaouais / marque"},

new Blog {Name = "Michael Shawn", URL = "http: // cegepoutaouais / michael"}

};

return View(topBlogs);

}

}

}

Le contenu de la vue:

@model CoursMVC.Models.Blog

@ {

ViewBag.Title = "ViewStonglyTyped";

}

<H2> View stongly typed </ h2>

<P>

<Ul>

@foreach (var blog in Model) {

<Li>

<a href="@blog.URL"> @ blog.Name </a>

</ Li>

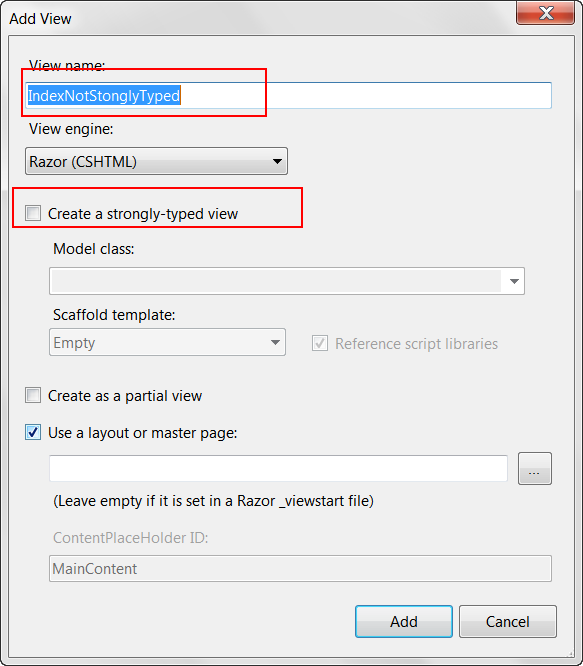
}

</ Ul>

</ P>

1. **Les vues dynamiquement typées**:

Si vous souhaitez passer des données non fortement typées à la vue, il suffit de ne pas cocher l'option "Fortement typée" au moment de la création de la vue.



Ajouter la ligne suivante au code de la vue générée par le système:

@model dynamic

Exemple: en utilisant le même modèle "Etudiant"

@using CoursMVC.Models

@model dynamic

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>GetDynamictype</title>

</head>

<body>

<div>

<h2>Type dynamique (utilisation de la propriété Model)</h2>

<table>

<tr>

<td>@Html.DisplayName("Numéro")</td>

<td>@Model.Numero</td>

</tr>

<tr>

<td>@Html.DisplayName("Nom")</td>

<td>@Model.Nom</td>

</tr>

<tr>

<td>@Html.DisplayName("DateNaissance")</td>

<td>@Model.DateNaissance.ToString("dd/MM/yyyy")</td>

</tr>

</table>

</div>

</body>

</html>

Exemple 2:

Le contenu de la vue:

**@model dynamic**

@ {

ViewBag.Title = "IndexNotStonglyTyped";

}

<H2> Index Non stongly typed </ h2>

<P>

<Ul>

@foreach (var blog in Model) {

<Li>

<a href="@blog.URL"> @ blog.Name </a>

</ Li>

}

</ Ul>

</ P>

**Remarque**:

**Vous ne seriez pas en mesure d'utiliser** **IntelliSense** cette fois parce que cette fois la liaison se fera au moment de l'exécution.

Une vue fortement typée est considérée comme une meilleure approche puisque nous savons déjà ce que les données sont transmis en tant que modèle à la différence de vues dynamiquement typées dans lequel les données se lient à l'exécution et peut conduire à des erreurs d'exécution si quelque chose change dans le modèle lié.

1. [**Passage de données entre les méthodes d'action**](javascript:void(0)) **du contrôleur**
2. **Utilisation de la propriété "TempData"**

Une méthode d'action peut stocker des données dans l'objet "**TempData**" de type  [TempDataDictionary](https://msdn.microsoft.com/fr-fr/library/system.web.mvc.tempdatadictionary(v=vs.100).aspx) du contrôleur avant d'appeler la méthode [RedirectToAction](https://msdn.microsoft.com/fr-fr/library/system.web.mvc.controller.redirecttoaction(v=vs.100).aspx) du contrôleur pour invoquer l'action suivante. La valeur de la propriété TempData est stockée dans l'état de session.

La valeur de "**TempData**" **persiste** jusqu'à sa lecture ou jusqu'à l'expiration de la **session**. La persistance de TempData de cette façon permet des scénarios tels que la redirection, parce que les valeurs de TempData sont disponibles au-delà d'une requête unique.

**Remarque**:

1. TempData est dérivé de la classe TempDataDictionary
2. TempData est utilisé pour transmettre les données de la requête en cours à la prochaine demande
3. Il conserve les informations pour le moment d'une requête HTTP. Cela signifie que d'une page à une autre. Il aide à maintenir les données lorsque l'on passe d'un contrôleur à un autre contrôleur ou d'une action à une autre action
4. Il faut faire un casting de type pour lire des données complexes. En général, il est utilisé pour stocker seulement un message de temps comme les messages d'erreur et les messages de validation
5. **Utilisation de la variable Session**
6. Dans ASP.NET MVC, la propriété **Session** de type "I**Session". Cette objet** est une propriété de la classe "**HttpContext**".
7. Session est également utilisée pour transmettre des données dans l'application ASP.NET MVC et Contrairement à "TempData", elle persiste jusqu'à son temps d'expiration (heure d'expiration de la session par défaut est de 20 minutes, mais elle peut être augmentée).
8. Session est valable pour toutes les demandes, et non pas pour une seule redirection.
9. Il est également nécessaire de faire un casting de type pour lire les données et vérifier les valeurs NULL pour éviter les erreurs.

