

Programmation avancée en c#.NET

Ing.Meryem OUARRACHI

Plan du module

Programmation WEB

- ☐ Généralités outils Web
- ☐ **ASP MVC**
- ☐ ASP.Net core
- ☐ Angular

Génie logiciel en .Net

BI en Self Service

Programmation distribuée avancée

- ☐ Web API
- ☐ GraphQL

CHAPITRE 2:

ASP MVC

Plan du chapitre

1. Structure du projet MVC
2. Gestion d'Etat
3. Mise en forme
4. Entity Framework
5. Razor
6. Les helpers HTML
7. Les validateurs
8. Exemple de l'utilisation de quelques composants
9. Ajax
10. Internationalisation en ASP MVC
11. Sécurité du projet ASP MVC

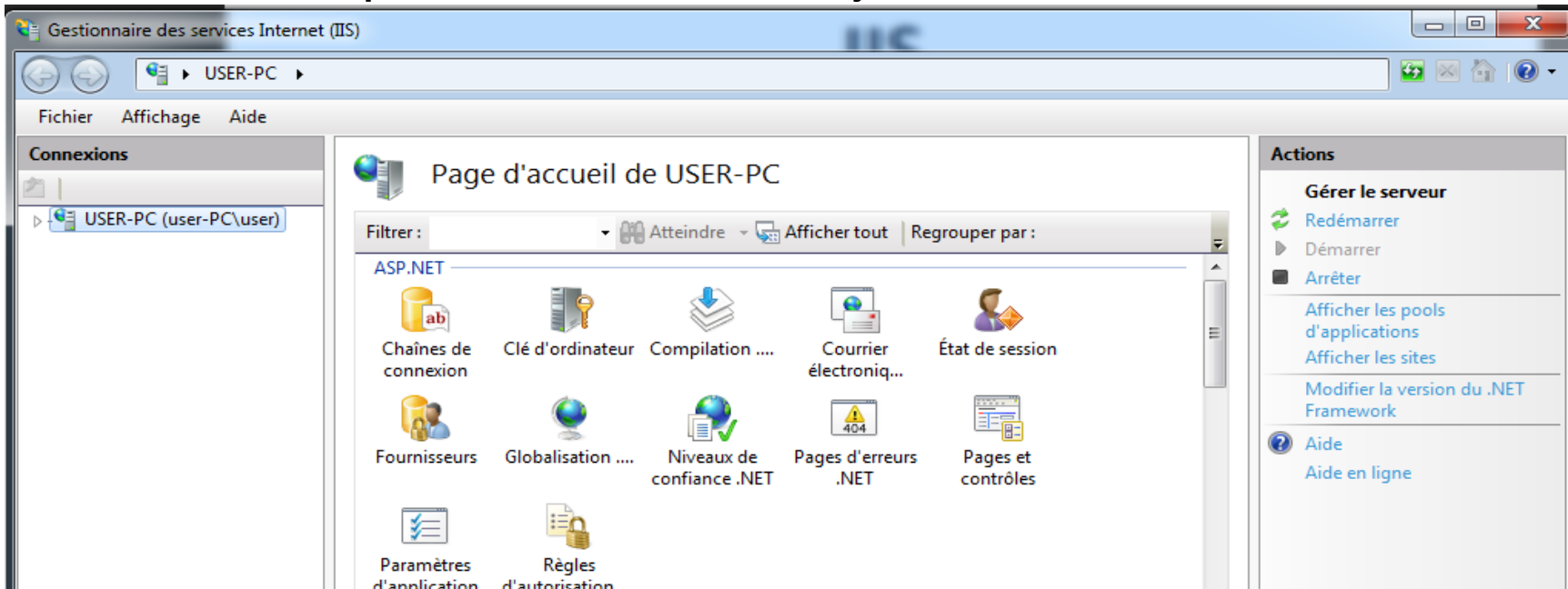
Spécificités de ASP

Objectif

- .Net possède un ensemble de fonctionnalités dédiées à la création et à la gestion de sites Web.
- ASP.NET permet de créer des sites Web dynamiques.
- IIS est le serveur web pour les projets web en .Net

IIS

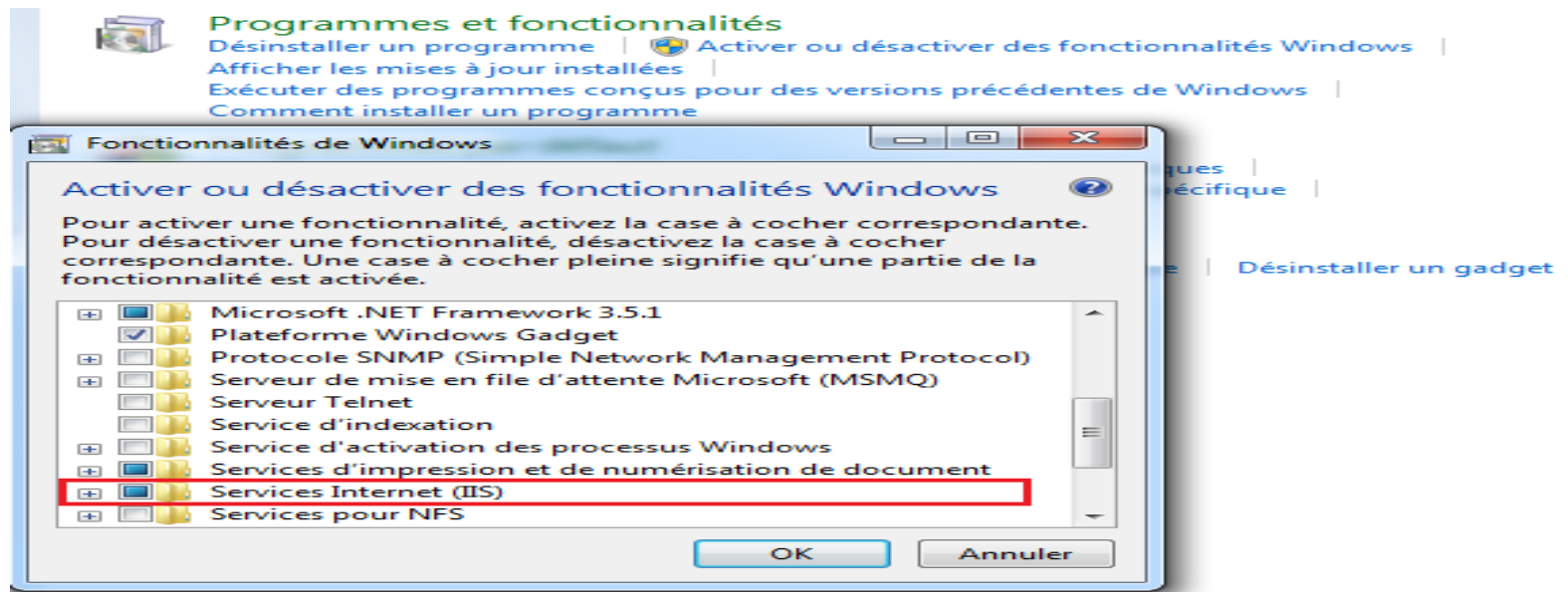
- S'installe avec le système d'exploitation windows
- Pour ouvrir IIS → taper **inetmgr** sur la barre de recherche mais assurer que le service est déjà activé



IIS

-Activation de IIS

Panneau de configuration → ajouter supprimer des programmes → Activer ou désactiver des fonctionnalités windows



Privilèges d'ASP

- Plus sécurisée :Selon le *cabinet de sécurité WhiteHat Security*, la sécurité est légèrement meilleure avec *ASP .NET* qu'avec *JSP*, à cause du fait qu'il y a une meilleure orientation de la sécurité pour les développeurs. Mais, les chiffres sont très proches l'un de l'autre, la densité de vulnérabilité est de 27,2 pour le .NET et de 30,0 pour le Java.
- Solution utilisée par des nombreux gouvernements et institutions financières.
- Exemple des sites en ASP:Massar,Stackoverflow

Modèles de programmation ASP.Net

ASP.NET prend en charge un certain nombre de modèles de programmation pour la création d'applications Web

❑ **ASP WEB Forms:** Framework pour la construction de pages modulaires à partir de composants, avec des événements d'interface utilisateur en cours de traitement côté serveur.

❑ **ASP MVC:** permet de créer des pages Web à l'aide du modèle de conception Modèle – Vue – Contrôleur

❑ **ASP .Net Core:** framework open source multiplateforme et hautes performances pour la création d'applications web modernes.

Structure du projet MVC

ASP Web forms vs ASP MVC

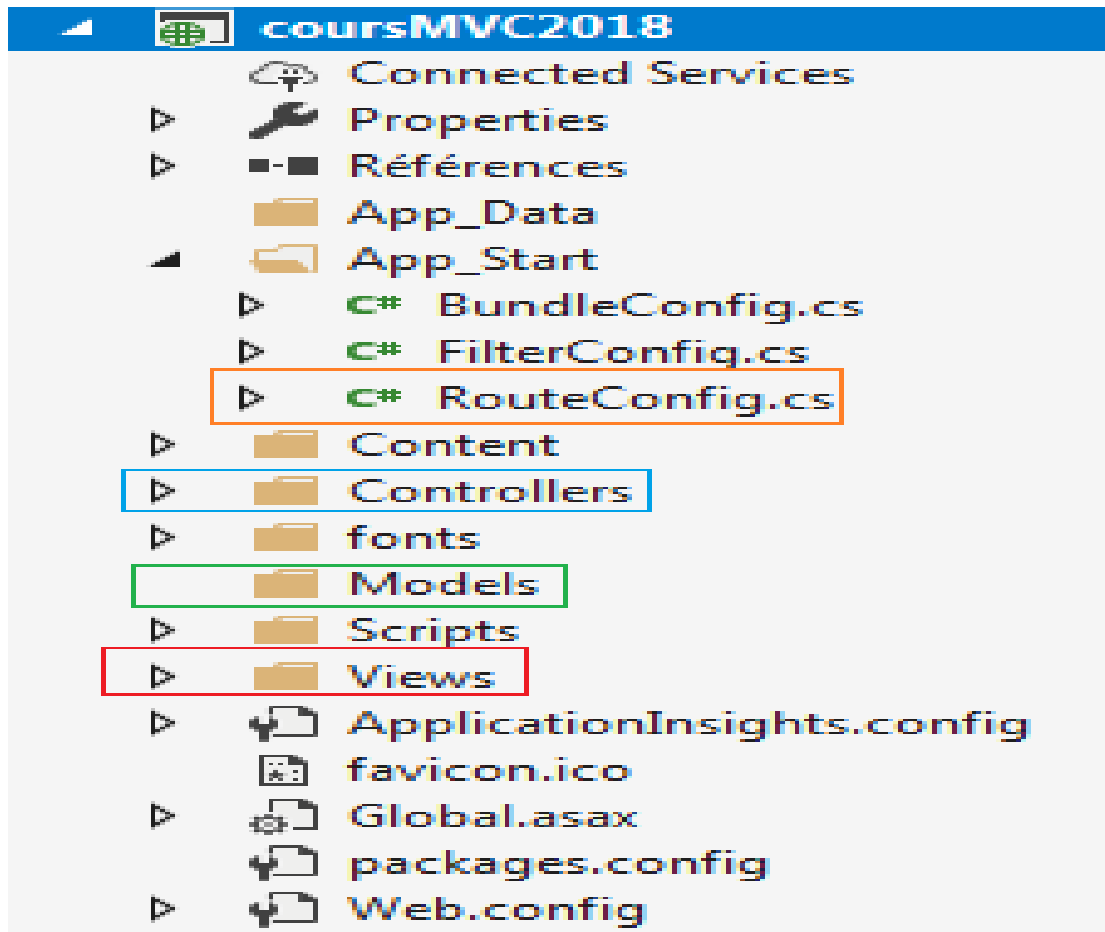
- ASP MVC plus léger car il ne stocke pas les données des variables dans les ViewState.
- L'entrée en ASP Web forms est la vue en ASPMVC est le controller → facilité d'automatisation des tests unitaires puisque on peut tester juste les fonctions de projet.
- La création de l'interface graphique en ASP Webforms se base sur glisser/déposer en ASPMVC il faut avoir des connaissances en html et css.

Le patron de conception MVC

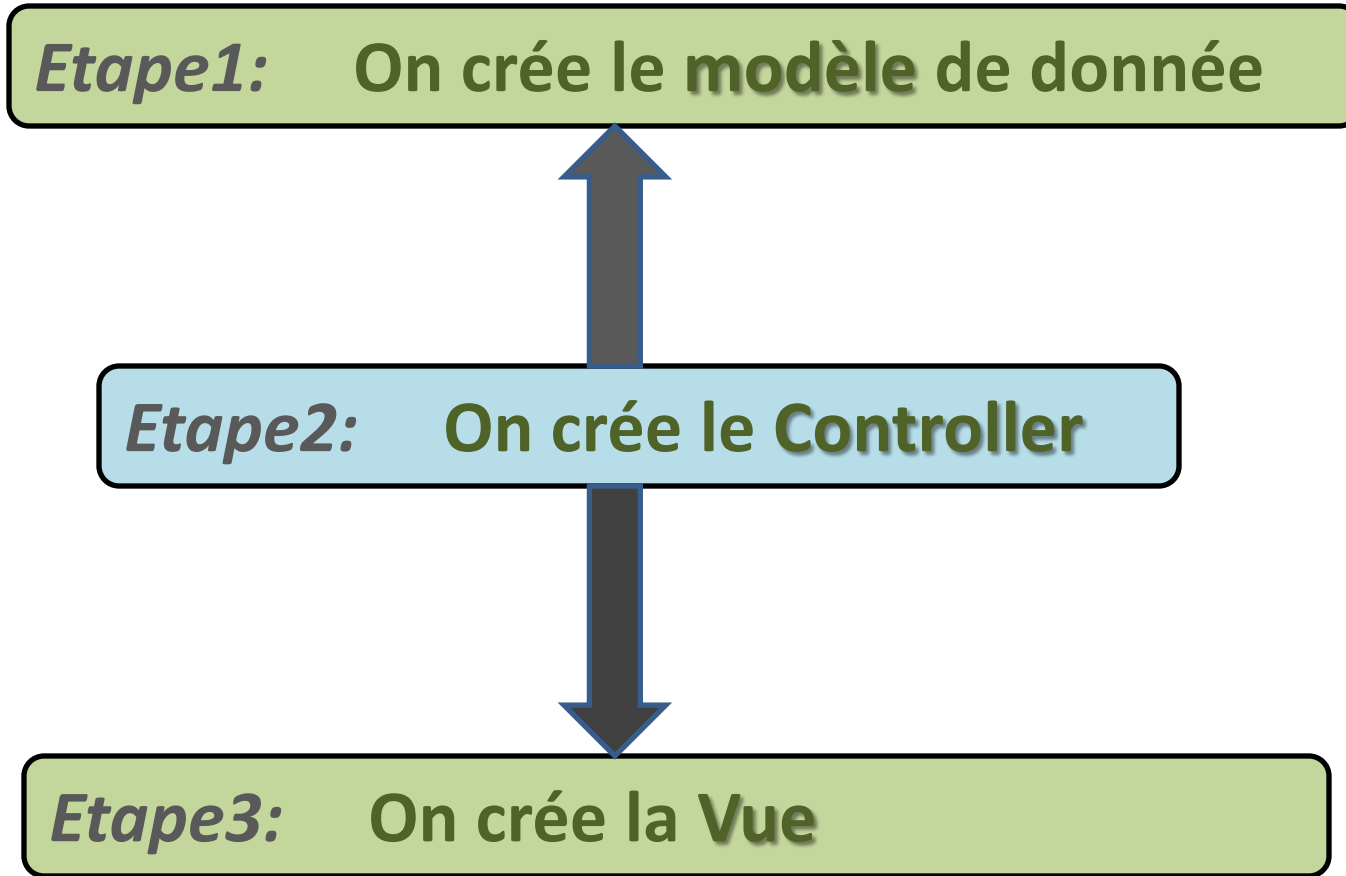
- Le MVC est un pattern architectural qui sépare les données (le modèle), l'interface homme-machine (la vue) et la logique de contrôle (le contrôleur).
- Ce modèle de conception impose donc une séparation en 3 couches :
 - Le modèle** : Il représente les données de l'application. Il définit aussi l'interaction avec la base de données et le traitement de ces données.
 - La vue** : Elle représente l'interface utilisateur, ce avec quoi il interagit. Elle n'effectue aucun traitement, elle se contente simplement d'afficher les données que lui fournit le modèle.
 - Le contrôleur** : Il gère l'interface entre le modèle et le client. Il va interpréter la requête de ce dernier pour lui envoyer la vue correspondante. Il effectue la synchronisation entre le modèle et les vues.

Architecture d'une application ASP MVC

New project → Application web ASP.Net → Choisir MVC



Etape de création des pages ASP MVC



Modèle

-C'est une classe qui devra se créer dans le dossier Model, porte les données de projets

```
namespace Mvccours2017.Models
{
    public class Etudiant
    {
        public int idEt;
        public string nom;
        public string prenom;
        public int age;
    }
}
```


Controller

- Il représente le point de démarrage de projet ASP MVC
- Il a pour rôle d'interpréter les requêtes de client pour lui envoyer les réponses(vues).
- C'est une classe qui va contenir un ensemble des actions.
- Création du controller: click droit sur le dossier controller→Ajouter un controller →il existe plusieurs modèles de contrôleur. Pour l'instant, nous choisirons le contrôleur MVC - Vide.

Controller

-La classe Controller fournit des méthodes qui répondent aux requêtes HTTP envoyées à un site Web ASP.NET MVC.

```
public class EtudiantController : Controller
{
    //
    // GET: /Etudiant/

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

Controller

- Dans une classe de contrôleur, chaque méthode publique représente une action.
- ActionResult signifie que la méthode va réaliser l'action correspondant à son nom.
- Dans cet exemple l'application donnera l'URL suivante <http://localhost/Etudiant/Index>.
- L'exécution de cet exemple va déclencher une erreur: car on a pas encore créer la vue.

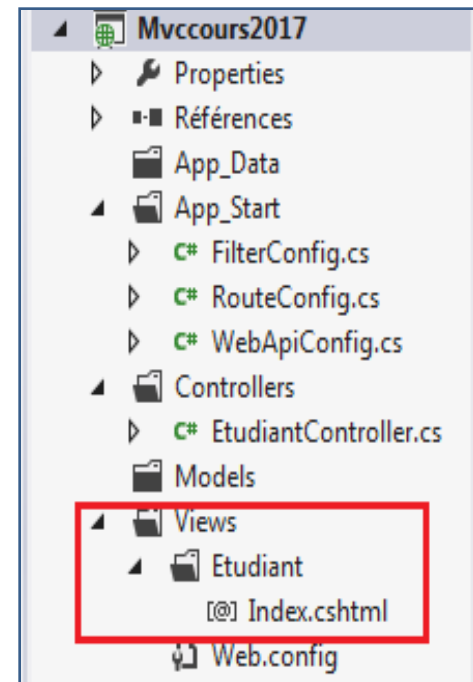
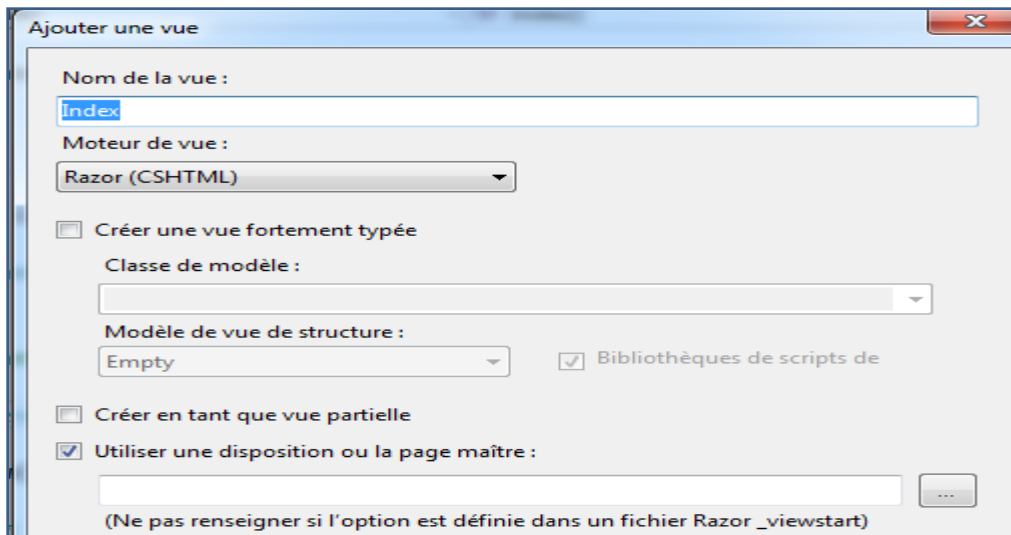
View

- `return View()` dans le controller: amène le programme à chercher dans le dossier View, un sous-répertoire portant le même nom que le controller dans le quel on trouve la vue (a le même nom que l'action)
- Càd dans le dossier View on doit avoir un sous-répertoire Etudiant portant une page Index (nom de l'action exécuté).
- A savoir que par défaut, ASP.NET MVC va chercher la vue dans le sous-répertoire du même nom que le contrôleur, mais que si elle n'existe pas, il ira voir ensuite dans le sous-répertoire Shared.

View

-Création de vue:

On clique au dessus de l'action du controller qui va être responsable de l'affichage de vue et on choisit ajouter une vue



View

-Si on veut appeler une vue qui porte un nom différent que l'action on met `return View("Index2")` mais toujours la recherche est dans le même sous répertoire.

```
public ActionResult Index()  
{  
    return View("Index2");  
}
```

Gestion d'Etat

Synchronisation entre le modèle et la vue

- **Envoi des données via le controller:**

-Pour envoyer des données du modèle vers la vue,il faut les passer dans les paramètres de View.

```
public ActionResult Index2()
{
    Etudiant e = new Etudiant();
    e.idEt = 1;
    e.prenom = "sara";
    e.nom = "hafidi";
    e.age = 20;
    return View(e);
}
```


Synchronisation entre le modèle et la vue

- **Récupération des données dans la vue:**

- Chaque vue a une propriété *Model* qui sauvegarde l'objet envoyé par le controller

@Model.nom

Synchronisation entre le modèle et la vue

- **Récupération des données dans la vue**

- Si on veut afficher une liste des valeur,on doit passer par l'instruction foreach

- Toujours lors d'une instruction foreach dans la vue,il faut ajouter

@model IEnumerable<MvcTP2.Models.etudiant>

Au lieu de @model MvcTp2.Models.etudiant

ViewData

- **Passer des valeurs du contrôleur vers la vue:**

- On utilise le dictionnaire de clé/valeur ViewData

Dans le controller: `ViewData["Nom"] = "yyy";`

- **Récupération de valeur:**

- Dans la vue `@ViewData["Nom"]`

ViewBag

- Autre façon pour passer les données de controller vers la vue est la propriété ViewBag
- Création:** `ViewBag.Nom="tttt"`
- Récupération:** `@ViewBag.Nom;`
- On peut passer des données Via le ViewData et les récupérer via ViewBag et vice versa
- Le ViewBag n'accepte pas des espaces.

RedirectToAction

- Si le controller ne veut pas afficher une vue mais il veut se rediriger vers une autre action:

RedirectToAction("Index3");

```
public ActionResult Index3()
{
    ViewData["nom"] = "xxxx";
    return View();
}

public ActionResult Index4()
{
    int valeur = 10;
    return RedirectToAction("Index3");
}
```

- <http://localhost:4979/Etudiant/Index3> → affiche xxxx

Redirect(Url.Action(...))

- Si le controller veut se rediriger vers une action qui se trouve dans un autre controller:

```
Redirect(Url.Action("Index", "accueil"));
```

Envoi des données via le View

-Supposons que l'utilisateur cette fois qui veut envoyer des donnée dans un lien

<http://localhost:4979/Etudiant/Index5?x=eee&y=fff>

Récupération: utiliser ces données comme des paramètres des actions.

```
public ActionResult Index5(string x ,string y)
{
    ViewData["nom"] = x;
    ViewData["prenom"] = y;
    return View();
}
```

Cookies

-Création:

```
HttpCookie cookie = new HttpCookie("Cookie");  
cookie.Value = "xxx" ;  
this.ControllerContext.HttpContext.Response.Cookies.  
Add(cookie);
```

-Récupération:

```
String s =  
this.ControllerContext.HttpContext.Request.Cookies["Cookie"].  
Value;
```


Cookies

-Vérification

```
this.ControllerContext.HttpContext.Request.Cookies.AllKeys.Contains("Cookie")==true
```

-Suppression:

```
HttpCookie cookie =  
this.ControllerContext.HttpContext.Request.Cookies["Cookie"];  
cookie.Expires = DateTime.Now.AddDays(-1);  
this.ControllerContext.HttpContext.Response.Cookies.  
Add(cookie);
```

Session

-Création:

```
Session["Cookie"] = "xxx";
```

-Récupération:

```
String s = Session["Cookie"];
```

Vérification

```
Session["Cookie"] = null
```

Supprimer:

```
Session["Cookie"]=null;
```

RouteConfig

-C'est un fichier dans le répertoire App_Start,il correspond à l'emplacement où on définit la première page qui va être exécuter dans le site

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

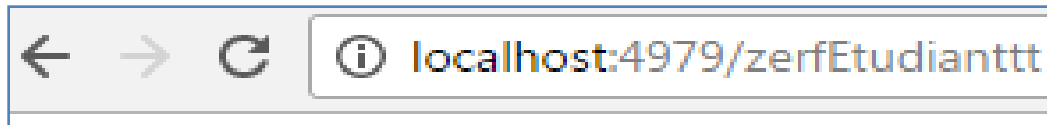
    routes.MapRoute(
        name: "Default",
        url: "{controller}/{action}/{id}",
        defaults: new { controller = "Etudiant", action = "Index1", id = UrlParameter.Optional }
    );
}
```

RouteConfig

-Des fois on aime cacher le langage utilisé pour développer nos sites

```
routes.MapRoute(  
    name: "Default",  
    url: "zerf{controller}tt/{action}/{id}",  
    defaults: new { controller = "Etudiant",  
                    action = "Index2",  
                    id = UrlParameter.Optional }  
);
```

-Pour appeler



Mise en forme

Page Layout

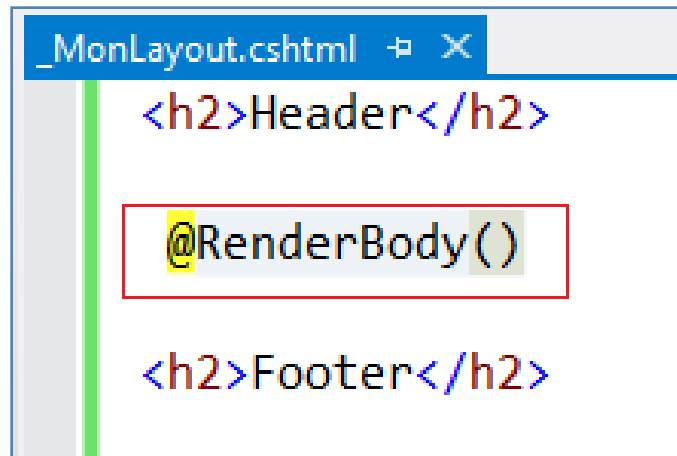
- C'est l'équivalent de Master page en ASP WebForms
- Voir _Layout.chnl

```
        <ul id="menu">
            <li>@Html.ActionLink("Accueil", "Index", "Home")</li>
            <li>@Html.ActionLink("À propos de", "About", "Home")</li>
            <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
        </ul>
    </nav>
</div>
</div>
</header>
<div id="body">
    @RenderSection("featured", required: false)
    <section class="content-wrapper main-content clear-fix">
        @RenderBody()
    </section>
</div>
<footer>
    <div class="content-wrapper">
```

- RenderBody(): permet d'affecter le contenu de page qui exploite layout

Page Layout

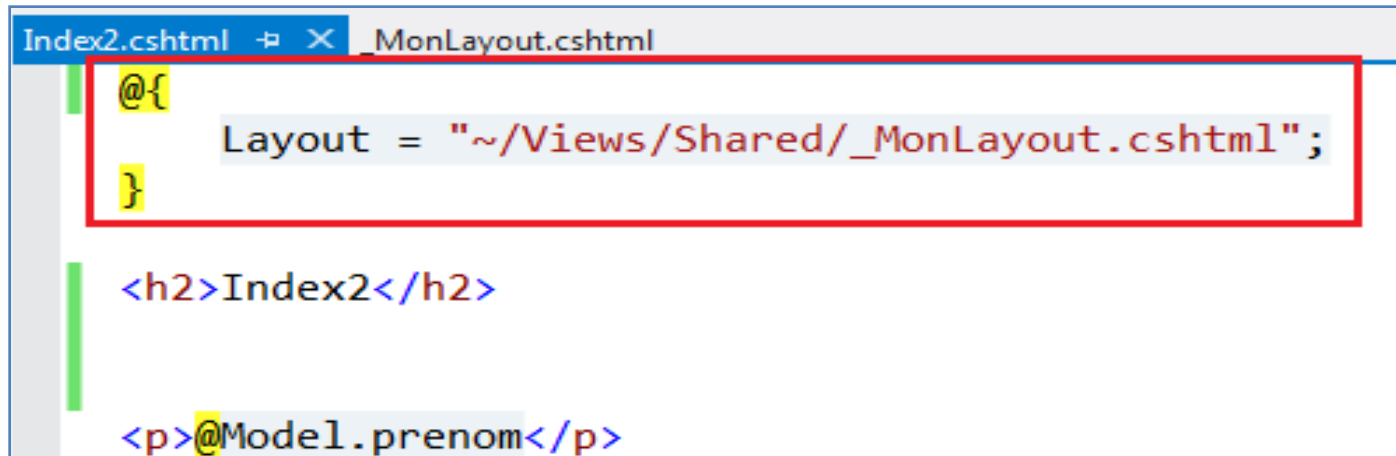
- Chaque vue a une variable nommée Layout qui détermine la Template à appliquer sur une page.
- Cette propriété est affecté dans la première ligne de la vue
- Création de Layout:** on le crée dans le répertoire Shared de projet→ajouter une vue



```
_MonLayout.cshtml  [icon] [X]  
  
<h2>Header</h2>  
  
@RenderBody()  
  
<h2>Footer</h2>
```

Page Layout

-Appel de Layout: on le crée dans le répertoire Shared de projet→ajouter une vue



```
Index2.cshtml  X  _MonLayout.cshtml
@{
    Layout = "~/Views/Shared/_MonLayout.cshtml";
}

<h2>Index2</h2>

<p>@Model.prenom</p>
```

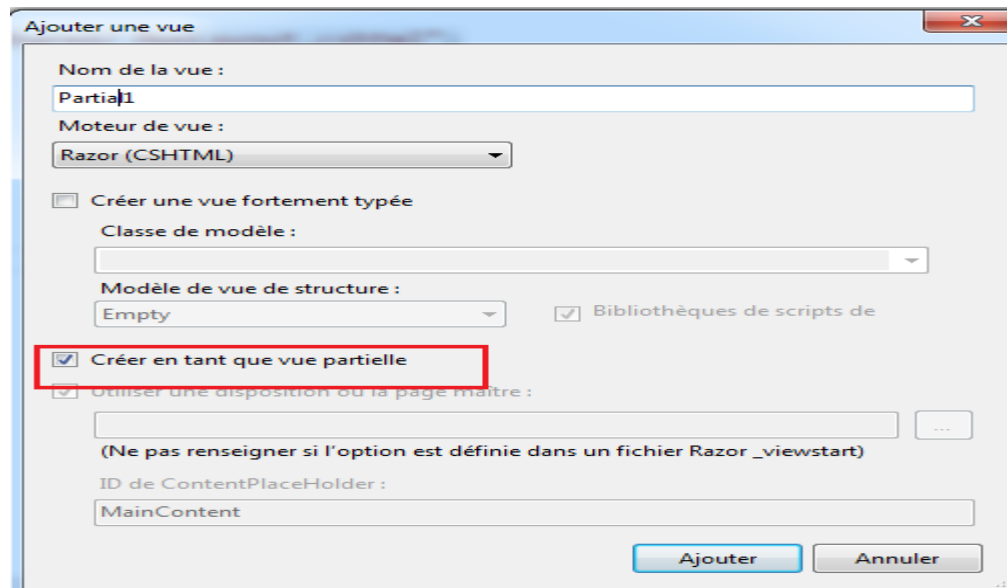
Remarque: par recommandation les noms des pages partagés doivent commencer par « _ »

Page Layout

Remarque:- Si on n'a pas une ligne au début d'une vue qui appelle le layout à utiliser, le programme cherche le layout défini dans la page `_ViewStart.chnml` du projet.

Partial View

- C'est l'équivalent de User contrôle en ASP WebForms.
- Il s'agit d'une partie de view qu'on peut l'intégrer dans d'autres view.
- **Création:** ajouter une nouvelle vue



Ajouter une vue

Nom de la vue :
Partia1

Moteur de vue :
Razor (CSHTML)

☐ Créer une vue fortement typée
Classe de modèle :
Modèle de vue de structure :
Empty

☒ **Créer en tant que vue partielle**

☒ Utiliser une disposition ou la page maître :
(Ne pas renseigner si l'option est définie dans un fichier Razor _viewstart)

ID de ContentPlaceHolder :
MainContent

Ajouter Annuler

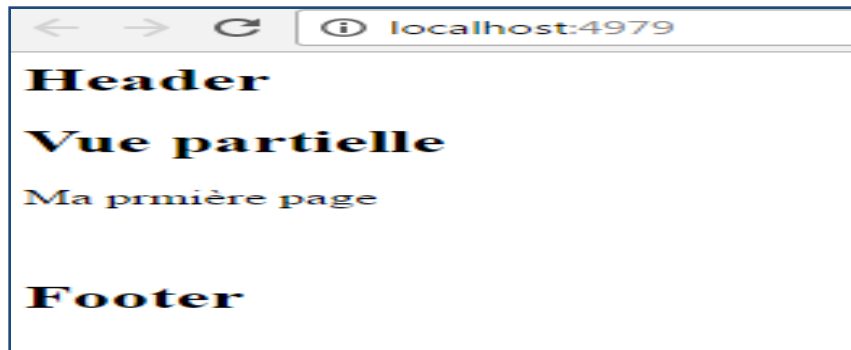
Partial View

- Appel :

-Intégrer cette ligne dans l'emplacement où on veut appeler ce partial view

```
<div>  
    @Html.Partial("Partial1")  
</div>
```

- Exécution:



Section

❑ **Problématique:** Supposons qu'on a un ensemble des pages qui utilisent le même Layout page mais on veut laisser une partie dans Layout se change selon la vue à afficher

-Autrement dit rendre une section dans layout modifiable selon la vue à utiliser .

❑ **Solution:** Utiliser la notion de section

Section

- **Création:** On définit la section dans Layout page via l'écriture suivante càd dans cet endroit la section va être appeler

```
@RenderSection("Section1", required: false)
```

- **Appel:** Ensuite on définit le contenu de section dans la vue qui utilise le précédent layout page

```
@section Section1{  
    <p>C'est la section de la vue1</p>  
}
```

Appeler un CSS

-Dans la page chtml on met

```
@Styles.Render("~/Content/css")  
@Scripts.Render("~/bundles/modernizr")
```

-On met la liaison avec les fichiers de script dans le fichier App_Start /Bundle.config

```
bundles.Add(new StyleBundle("~/Content/css").Include("~/Content/site.css"));
```

```
bundles.Add(new ScriptBundle("~/bundles/jquery").Include(  
    "~/Scripts/jquery-{version}.js"));  
  
bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(  
    "~/Scripts/modernizr-*"));
```