(Leçon 07)
Les vues dans les applications ASP.NET

LOTFI DERBALI, PH.D
DERBALI.IT@GMAIL.COM
HTTPS://GITHUB.COM/DERBALI-IT

Contenu

- Rendu des pages Web
- Ajout d'un contenu dynamique
- Syntaxe Razor
- Les vues fortement typées

Rendu des pages Web

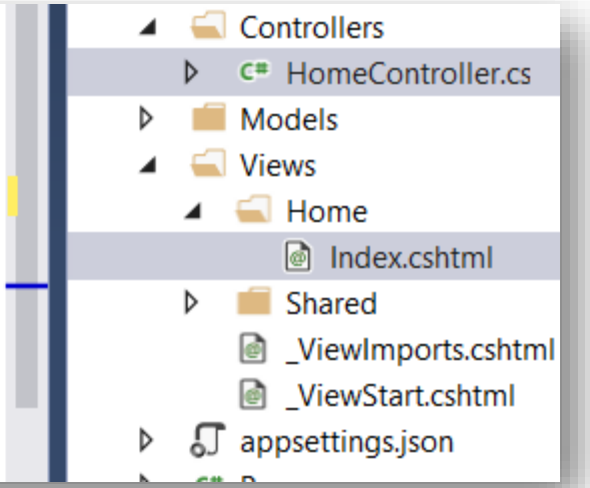
- Pour produire une réponse HTML à une requête HTTP, il y a utilisation d'une **vue**

- Une **vue**
  - Fait l'interface avec l'utilisateur
  - Reçoit tous les évènements d'un utilisateur (clic sur un bouton, sélection dans une liste, …)
  - Associe chaque évènement à une action dans le contrôleur
  - Représente les données encapsulées via un modèle
  - …

Dans une application ASP.NET MVC

- Razor est le moteur des vues
  - Extension du fichier (.cshtml)
- Les vues se placent dans le répertoire /Views
  - Plus précisément dans le sous répertoire correspondant au nom du contrôleur lié à la vue
  - Le nom de la vue est généralement lié au nom de l'action du contrôleur
  - Par exemple: la vue retournée par l'action Index du contrôleur Home se placera dans le répertoire /Views/Home et s'appellera Index.cshtml

```
public class HomeController : Controller
{
    public IActionResult Index()
    {
        return View();
    }
}
```



Comprendre les chemins d'accès (Routes)

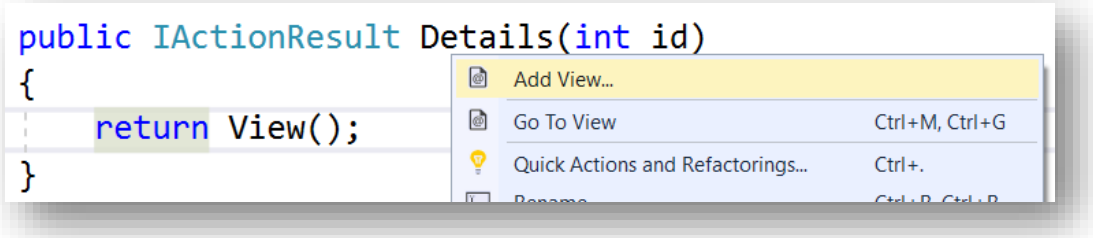
- Les applications MVC utilisent le système de routage ASP.NET, qui décide comment les URL correspondent aux contrôleurs et aux actions.
  - Dans le fichier **Startup.cs**, il y a le système de routage ASP.NET par défaut:

```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");
});
```

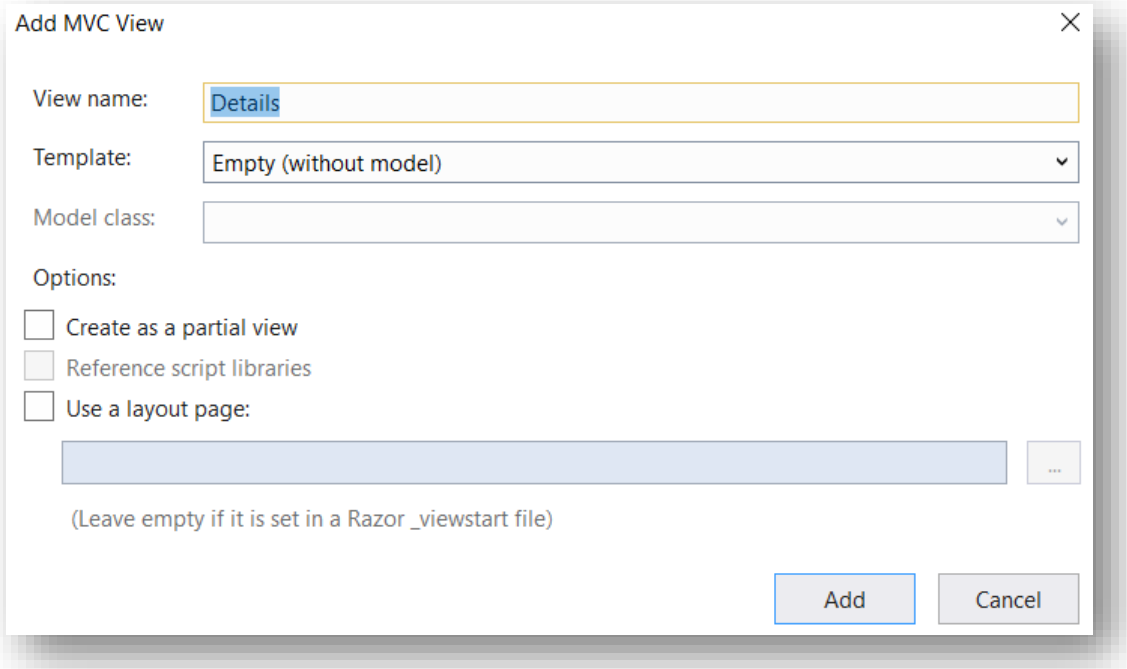
- Si vous ne fournissez pas un contrôleur, alors la valeur par défaut sera prise à savoir Home. Pour l'action il s'agira d'Index et pour le paramètre Id, il s'agira d'une chaine vide
- Imaginez que vous entrez l'URL suivant dans la barre d'adresse de votre navigateur : /Student/Update/3
  - Le système de routage ASP.NET lie cette URL aux paramètres suivants :
    - Controller = Student
    - Action = Update
    - Id = 3

Ajout d'une vue vide

- Clic droit à l'intérieur de l'action en question
- Choix de l'option « Ajouter une vue »

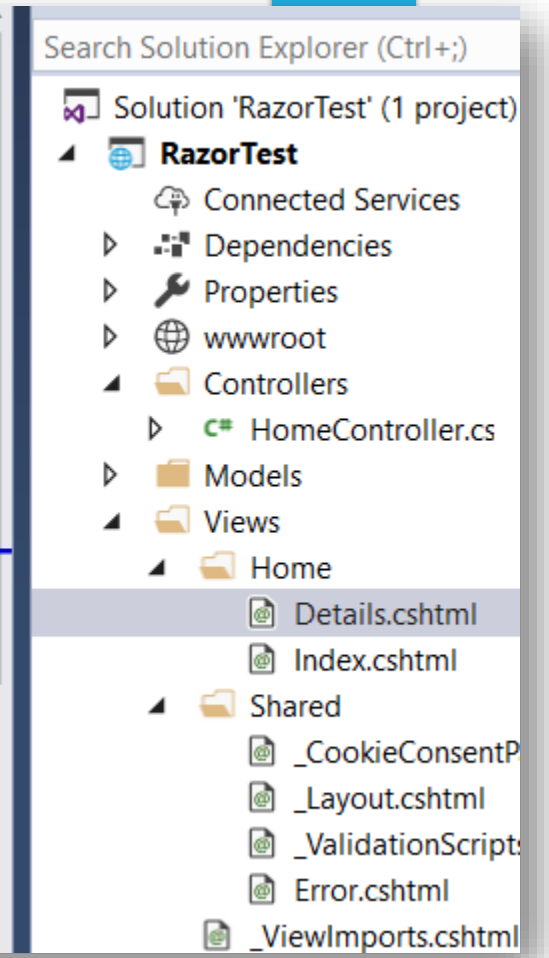


- Décocher « Utiliser une page de disposition »
- Clic sur « Ajouter »



- L'opération pécédente génère la vue suivante:

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Details</title>
</head>
<body>
</body>
</html>
```



Ajout d'un contenu dynamique

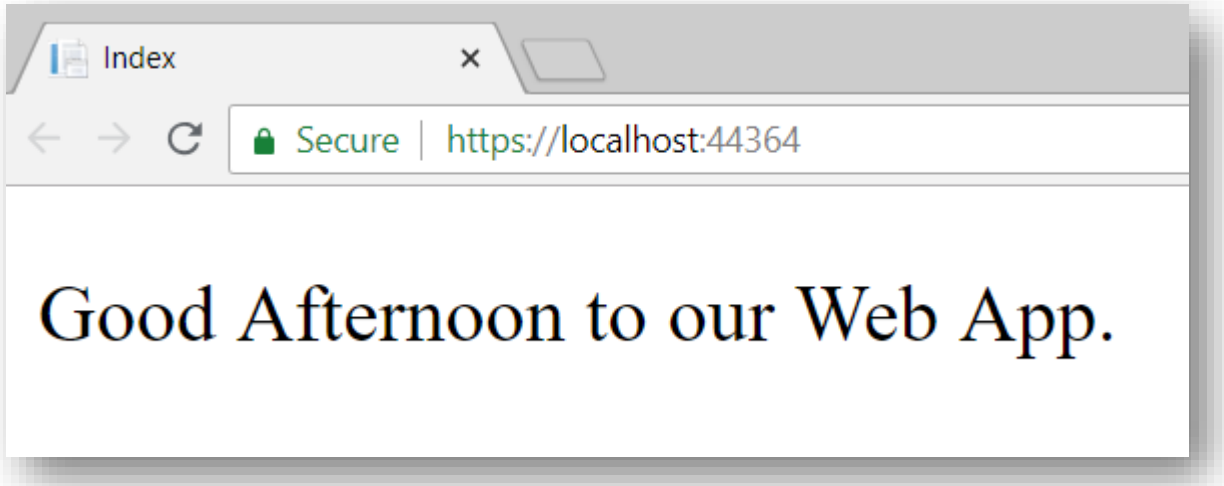
- Dans MVC, le contrôleur a pour tâche de construire des données et de les transmettre à la vue, qui est chargée de les rendre au format HTML.
- En ASP.NET, une façon de transmettre des données du contrôleur à la vue consiste à utiliser l'objet **ViewBag**, qui est membre de la classe de base Controller.
  - ViewBag est un objet dynamique auquel vous pouvez assigner des propriétés arbitraires, rendant ces valeurs disponibles dans n'importe quelle vue rendue par la suite

Exemple: Afficher un message dynamique de bienvenue


```
public class HomeController : Controller
    {
        public IActionResult Index()
        {
            int hour = DateTime.Now.Hour;
            ViewBag.Greeting = hour < 12 ? "Good Morning" : "Good Afternoon";
            return View();
        }
```

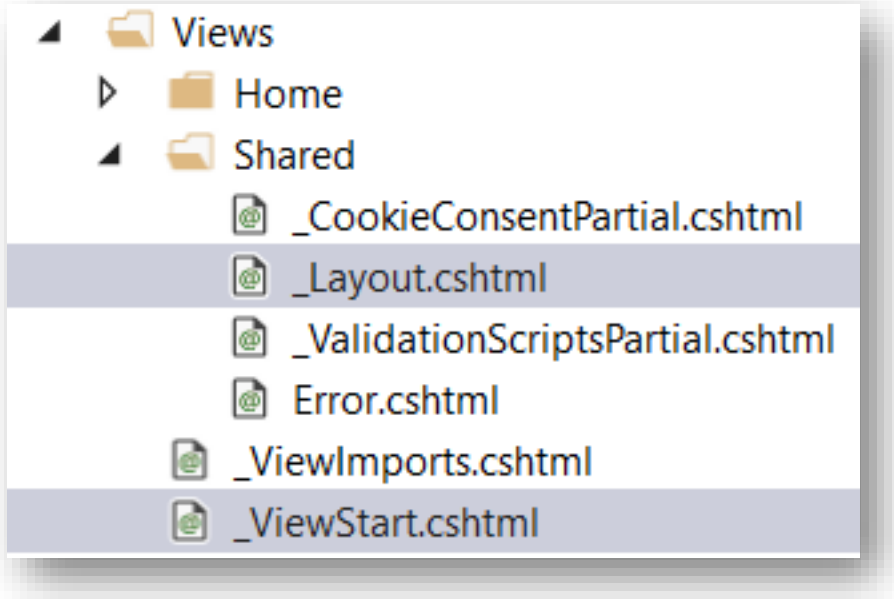
Index.cshtml

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <p> @ViewBag.Greeting to our Web App.</p>
</body>
</html>
```



Layout et ViewStart

- Un **Layout** définit une page « modèle ». Toutes les autres pages pourront hériter de cette page modèle.
  - Principe DRY (Don't Repeat Yourself)
  - Lorsque vous créez un projet ASP.NET MVC, il y a déjà un layout par défaut qui se nomme _Layout.cshtml (situé dans **Views/Shared**).
  - Le fichier _ViewStart,cshtml définit le layout par défaut
    - Il est possible de placer un fichier _ViewStart.cshtml dans un sous répertoire



Layout et ViewStart

- Extrait du code dans le fichier _Layout.cshtml

```
<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
    @RenderSection("styles", required: false)
</head>
<body>
    <div id="main">
        @RenderBody()
    </div>

    @RenderSection("scripts", required: false)
</body>
</html>
```

- Le **@RenderBody()** permet de déterminer l'endroit où le code de la page utilisant le Layout va être rendu.

- Le **@RenderSection()** avec la valeur required permet d'indiquer si la vue utilisant le Layout doit comporter ou non une section script.

Syntaxe Razor

- Razor évalue les expressions C# et les affiche en format HTML.
  - Utilisation de @
    - Combinaison du code C# avec HTML
    - Extension du fichier (.cshtml)
  - Pour définir un bloc de code

```
@{
    ....
}
```

  - Commentaires en Razor: @* .... *@

Blocs, conditions et boucles


```
@{
    /* Bloc de code sur plusieurs lignes */
    string uneVariable = "valeur initiale";
    uneVariable += "un autre morceau";
}

<p>Une variable vaut : @uneVariable.</p>
```

```
@if (Model.IsEnabled) {
    <p>Le produit est activé !</p>
}
else {
    <p>Le produit n'est pas actif.</p>
}
```

Toutes les instructions en C# seront utilisées avec le symbole @.

```
<ul>
@foreach (var p in Model.Products) {
    <li>@p.Name</li>
}
</ul>
```

HTML Helpers

- Ce sont des méthodes qui génèrent du code HTML dans une vue (MVC)
- Exemple:
  - En HTML

```
<input type="text" id="nom" name="nom" />
```

  - Avec TextBox HTML Helper (plusieurs contructeurs)
    - Version 1

```
@Html.TextBox("nom")
```

    - Version 2 : avec une valeur

```
@Html.TextBox("nom", "Felix")
```

    - Version 3 : avec une valeur et quelques propriétéss

```
@Html.TextBox("nom", "Felix", new { style = "background-color:orange; color:white", title="Entrer votre nom"})
```

HTML Helpers (autres méthodes)

- Pour générer une étiquette

```
@Html.Label("nom","Votre Nom")
```

- Pour générer un Textbox pour un mot de passe

```
@Html.Password("passwd")
```

- Pour générer TextBox à lignes multiples

```
@Html.TextArea("comments","",5,20,null)
```

- Pour générer un TextBox caché

```
@Html.Hidden("id")
```

Formulaire

Html.BeginForm

```
@using (Html.BeginForm("Index", "Home"))
{
    @Html.Label("estMajeur", "Cochez la case si vous êtes majeur")
    @Html.CheckBox("estMajeur", true)
    <input type="submit" value="Envoyer" />
}
```

Autres méthodes

HtmlHelper	Strogly Typed HtmlHelpers	Html Control
Html.ActionLink		It generates Anchor link
Html.Label	Html.LabelFor	It generates Label
Html.TextBox	Html.TextBoxFor	It generates Textbox
Html.TextArea	Html.TextAreaFor	It generates TextArea
Html.Display	Html.DisplayFor	It generates Html text
Html.RadioButton	Html.RadioButtonFor	It generates Radio button
Html.ListBox	Html.ListBoxFor	It generates multi-select list box
Html.DropDownList	Html.DropDownListFor	It generates Dropdown, combobox
Html.Hidden	Html.HiddenFor	It generates Hidden field
Password	Html.PasswordFor	It generates Password textbox
Html.CheckBox	Html.CheckBoxFor	It generates Checkbox
Html.Editor	Html.EditorFor	It generates Html controls based on data type of specified model property e.g. textbox for string property, numeric field for int, double or other numeric type.

Les vues fortement typées

- L'utilisation de ViewBag pour passer de données entre le contrôleur et la vue n'est vraiment pas pratique
- Il existe en effet une surcharge de la méthode View() qui permet de passer le modèle directement à la vue.
  - Par exemple

```
public IActionResult Details()
        {
            GuestResponse g = new GuestResponse()
            {
                Name = "Felix", Phone = "514-887-9556",Email = "felix@gmail.com"
            };
            return View(g);
        }
```

Utilisation du modèle dans la vue


```
@model RazorTest.Models.GuestResponse

@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Details</title>
</head>
<body>
    <h4>Guest Response</h4>
    <hr />
    <ul>
        <li>@Model.Name</li>
        <li>@Model.Email</li>
        <li>@Model.Phone</li>
    </ul>
</body>
</html>
```

Les HTML Helpers fortement typés

- Ils fonctionnent grâce au pouvoir des expressions lambda

```
@model RazorTest.Models.GuestResponse

@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width"
/>
    <title>Details</title>
</head>
<body>
    <h4>Guest Response</h4>
    <ul>
        <li>@Html.DisplayFor(model => model.Name)</li>
        <li>@Html.DisplayFor(model => model.Email)</li>
        <li>@Html.DisplayFor(model => model.Phone)</li>
    </ul>
</body>
</html>
```