

PAGES WEB ASP.NET

Syntaxe Razor

- Le symbole `@` différencie le code C# du code HTML dans une vue.
- `@@`: affiche le symbole `@`
- `@`: définit une ligne de texte dans le code C#, pour plusieurs lignes il faut utiliser `<text> </text>`
- `@*` commentaire razor `*@`
- Par défaut Razor encode les chaînes de caractères avant de les envoyer au navigateur, par exemple si `@Model.Categorie` contient "`<p>Salade</p>`", alors il remplace les caractères `<` et `>` par `'<'` et `'>'`.
- Pour générer les chaînes sans encodage HTML, il faut utiliser `@Html.Raw(Model.Categorie)`

Exemple 1

```
<!DOCTYPE html>
<html>
  <head>
    <title>Exemple 1</title>
  </head>
<body>
  <h2>Exemple de page razor</h2>
  <!--Un commentaire HTML-->
  @* Un commentaire RAZOR*@
  @{ //Un commentaire C#
    /*
    un commentaire C#
    */
    string message = "Une première page ASP.NET ";

  }

  <p>
    Page:@message
  </p>

</body>
</html>
```

Exemple 2

```
<!DOCTYPE html>
<html>
  <head>
    <title>Exemple 2</title>
  </head>
  <body>
    <ul>
      @{
        for (int i = 1; i <= 20; i++)
        {
          <li> Elément @i </li>
        }
        //le code suivant sera affiché dans la page et non pas
        exécuté.
        @:for (int i = 1; i <= 20; i++)
        <br />

        //Les lignes de code suivantes seront affichées et non pas
        exécutées
        <text>
          if (a > 37)
          {

          }
        </text>
      }
    </ul>

  </body>
</html>
```

Les formulaires: Requêtes HTTP

- Selon la méthode d'envoi du formulaire, il existe deux collections pour accéder aux valeurs des champs:
 - La collection `Request.Form` est utilisée dans le cas d'un envoi par la méthode `Post`, alors que la collection `Request.QueryString` permet l'accès aux champs dans le cas d'un envoi par `Get`.
- La collection `Request` contient les paramètres envoyés par `Post` et par `Get`.
- La propriété booléenne `IsPost` retourne `true`, si la page est affichée après un envoi par `Post`.

```
if (IsPost)
{

}
```

- Méthodes associées aux valeurs des paramètres
 - `IsEmpty()`: retourne `true` si la valeur du champ est vide ou nulle.
 - Méthodes pour détecter le type de données d'un paramètre: `IsInt()`, `IsFloat()`, `IsBoolean()`, `IsDateTime()`...
 - Exemple: `if (Request["nom_champ"].IsDateTime()) { ... }`

Exemple 3: Envoi d'un formulaire par GET

```
<!DOCTYPE html>
<html>
  <head>
    <title>Exemple 3</title>
  </head>
  <body>
    @{
      if (!Request.QueryString["id"].IsEmpty())
      {
        string id = Request.QueryString["id"];
        string nom = Request["nom"];
        <div>
          ID:@id<br />
          Nom:@nom<br />
        </div>
      }
      else
      {
        <form method="get">
          ID:<input type="text" name="id"/><br />
          Nom:<input type="text" name="nom"/><br />
          <input type="submit" value="Envoyer"/>
        </form>
      }
    }</body></html>
```

Exemple 4: Envoi d'un formulaire par POST

```
<!DOCTYPE html>
<html>
  <head>
    <title>Exemple 4</title>
  </head>
  <body>
    @{
      //if (!Request.Form["id"].IsEmpty()) ou bien
      if (IsPost)
      {
        string id = Request.Form["id"];
        string nom = Request["nom"];
        <div>
          ID:@id<br />
          Nom:@nom<br />
        </div>
      }
      else
      {
        <form method="post">
          ID:<input type="text" name="id"/><br />
          Nom:<input type="text" name="nom"/><br />
          <input type="submit" value="Envoyer"/>
        </form>
      }
    }</body></html>
```

Exemple 4: Afficher les éléments d'une collection dans une liste.

Classes dans le dossier app_code

```
// Description résumée de Intervenant
public class Intervenant
{
    public int Id { get; set; }
    public string Nom { get; set; }
}

//La classe ListeIntervenants
public class ListeIntervenants
{
    public List<Intervenant> getIntervenants()
    {
        List<Intervenant> liste = new List<Intervenant>()
        {new Intervenant {Id=1,Nom="Intervenant 1" },
        new Intervenant {Id=2,Nom="Intervenant 2" },
        new Intervenant {Id=3,Nom="Intervenant 3" },
        new Intervenant {Id=4,Nom="Intervenant 4" }

        };
        return liste;
    }
}
```

Page intervenants.aspx

```
<!DOCTYPE html>
<html>
    <head>
        <title>Exemple 4 liste des intervenants</title>
    </head>
    <body>
        <h2>Intervenants</h2>
        <table border="1">
            <tr>
                <th>Id</th>
                <th>Nom</th>
            </tr>
            @{
                List<Intervenant> liste = (new ListeIntervenants()).getIntervenants();
                foreach (Intervenant intervenant in liste)
                {
                    <tr>
                        <td>@intervenant.Id</td>
                        <td>@intervenant.Nom</td>
                    </tr>
                }
            }
        </table>

    </body>
</html>
```

Accès aux données

Chaîne de connexion

Les informations de connexion à la base de données doivent être définies dans le fichier web.config à l'aide d'une chaîne de connexion nommée, exemple:

```
<configuration>

  <connectionStrings>
    ...
    <add name="csFilms" connectionString="data source=.;initial
catalog=films;integrated security=true"
providerName="System.Data.SqlClient" />

  </connectionStrings>
  ...
</configuration>
```

Accès aux données

Connexion à la base de données

- La classe Database contient les méthodes de connexion à la base de données:
 - Open(string nom): accepte le nom d'une chaîne de connexion définie dans le fichier web.config
 - OpenConnectionString(string chaine): accepte en paramètre une chaîne de connexion.
 - Les deux méthodes Open et OpenConnectionString retourne une instance de type Database.

Méthodes d'exécution d'une requête

- Une instade type Database possède 4 méthodes pour exécuter une requête :
 - Query(string requête): exécute une requête SQL qui retourne un jeu d'enregsitrements
 - QuerySingle(string requête): exécute une requête qui retourne une seule ligne.
 - QueryValue(string requête) : exécute une requête qui retourne une seule valeur (exemple select count(*)).
 - Execute(string requête): exécute une requête qui ne retourne pas de données (de type Insert, Update ou Delete ou bien une requête de définition de données comme create table).

Exemple 5 : Liste des catégories dans une table

```
@{
```

```
    Database db = Database.Open("csFilms");  
    var sql = "select * from categories";
```

```
    var data = db.Query("select * from films");
```

```
}
```

```
<!DOCTYPE html>
```

```
<html>
```

```
    <head>
```

```
        <title></title>
```

```
    </head>
```

```
    <body>
```

```
        <h2>Liste des catégories </h2>
```

```
        <table>
```

```
            <tr>
```

```
                <th>Id</th>
```

```
                <th>Nom</th>:
```

```
            </tr>
```

```
            @foreach (var ligne in db.Query(sql))
```

```
            {
```

```
                <tr>
```

```
                    <td>@ligne.Id</td>
```

```
                    <td>@ligne.Nom</td>
```

```
                </tr>
```

```
            }
```

```
        </table>
```

```
    </body>
```

```
</html>
```

Le helper WebGrid

- Le helper WebGrid permet d'afficher les données d'une collection dans une table HTML, les données sont passées au helper à l'aide du constructeur WeGrid(data) ou bien WebGrid(source:data).
- Le helper supporte des propriétés pour définir la structure et les styles de la table HTML.
- Exemple d'utilisation:

```
@{
```

```
Database db = Database.Open("csFilms");  
var data = db.Query("select * from films");  
var w = new WebGrid(source: data);
```

```
}
```

```
<!DOCTYPE html>
```

```
<html>
```

```
    <head>
```

```
    </head>
```

```
<body>
```

```
@w.GetHtml()
```

```
</body>
```

```
</html>
```

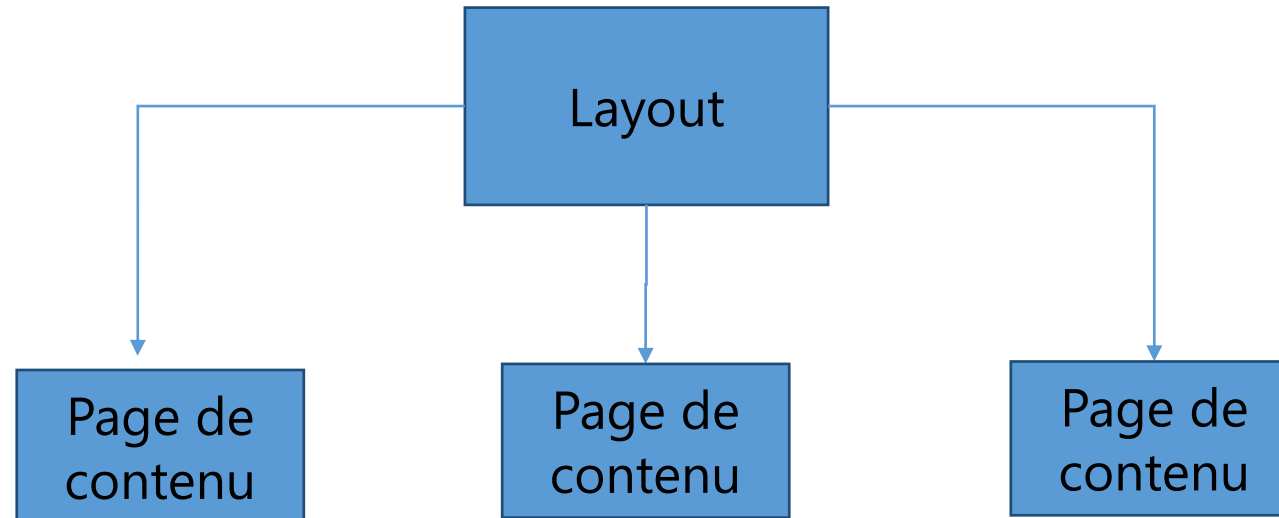
	<u>Id</u>	<u>catid</u>	<u>titre</u>	<u>Réalisateur</u>	<u>date</u>	<u>enSalle</u>	<u>BoxOffice</u>	<u>Description</u>
1	3		Titanic	James Cameron	21/06/1997 00:00:00	False	600000000,0000	
2	1		Star Wars	George Lucas	01/06/1977 00:00:00	False	500000000,0000	
3	1		Jurassic Park	Steven Spielberg	17/06/1993 00:00:00	False	400000000,0000	
4	4		Jaws	Steven Spielberg	30/05/1975 00:00:00	False	300000000,0000	
5	3		Ghost	Jerry Zucker	14/06/1990 00:00:00	False	200000000,0000	
7	3		Forrest Gump	Robert Zemeckis	18/06/1994 00:00:00	True	300000000,0000	
8	2		Ice Age	Chris Wedge	26/06/2002 00:00:00	False	200000000,0000	
9	2		Shrek	Andrew Adamson	25/06/2001 00:00:00	False	400000000,0000	
10	1		Independence Day	Roland Emmerich	20/06/1996 00:00:00	False	300000000,0000	
11	4		The Ring	Gore Verbinski	26/06/2002 00:00:00	False	100000000,0000	

Validation des champs de formulaire

- La validation des champs de formulaire se passe en trois étapes:
 1. Enregistrement des champs à valider au début de la page avec le message d'erreur à afficher éventuellement si le champ est vide:
`Validation.RequireField(nom_duchamp, message);`
 2. Après envoi du formulaire, la méthode `Validation.IsValid()` retourne `true` si tous les champs enregistrés sont valides.
 3. Dans le code HTML, le helper `@Html.ValidationMessage(nom_champ, message)` permet d'afficher « message » dans le cas où `nom_champ` est vide ou null.
- Le helper `@Html.ValidationSummary` permet d'afficher dans une liste `<ul>`, les messages enregistrés par la méthode `Validation.RequiredField` pour tous les champs non valides.

Les pages de disposition (Layout)

- Un layout (ou page de disposition) définit un template pour les pages asp.net
- La page de disposition peut contenir par exemple l'entête, la barre de navigation , le pied de page et une zone modifiable (placeholder) qui doit accueillir le contenu des pages asp.net
- Le contenu des vues est inséré dans le layout à l'aide de la méthode `@RenderBody()`.



Exemple

la méthode `@RenderSection` permet d'insérer le contenu d'une section définie dans la page de contenu à l'aide de l'instruction:

```
@section nom_section {  
    ...  
}
```

Si l'attribut `required` est égal à `true` alors une exception sera déclenchée dans le cas où la section n'est pas définie dans la page de contenu.

```
<!DOCTYPE html>  
<html>  
<head>  
    <meta name="viewport" content="width=device-width" />  
    <title>@Page.Title</title>  
</head>  
<body>  
    <div id="section1">@RenderSection("section1",required:false)</div>  
    <div>  
        @RenderBody()  
    </div>  
</body>  
</html>
```

Lier une vue à un layout

- La directive Layout permet d'associer un layout à une vue.
- Le code de la page de contenu s'exécute avant le code du layout

```
@{  
    Page.Title = "Titre de la page";  
    Layout = "~/Views/Shared/_Layout1.cshtml";  
}
```

- La directive @section définit une section dans la vue.

```
@section section1{  
    <p> C'est une section</p>  
}
```

Exemples du cours:categories.cshtml

```
• @{
•     var db = Database.Open("csFilms");
•     var sql = "select * from categories";
•     var categories = db.Query(sql);
• }
• <!DOCTYPE html>
• <html>
•     <head>
•         <title>Liste des catégories</title>
•     </head>
•     <body>
•         <h2>Liste des films</h2>
•         <a href="ajouterCategorie.cshtml">ajouter une catégorie</a>
•         <table>
•             <tr>
•                 <th>Id</th>
•                 <th>Nom</th>
•                 <th></th>
•                 <th></th>
•             </tr>
•             @foreach(var cat in categories) {
•                 <tr>
•                     <td>@cat.id</td>
•                     <td>@cat.nom</td>
•                     <td><a href="modifierCategorie.cshtml?id=@cat.id">Modifier</a></td>
•                     <td><a href="supprimerCategorie.cshtml?id=@cat.id">Supprimer</a></td>
•                 </tr>
•             }
•         </table>
•     </body>
• </html>
```

ajoutCategorie.cshtml

```
@{
    Validation.RequireField("txtNom", "Le champ catégorie est obligatoire");
    if (IsPost && Validation.IsValid())
    {
        var nom = Request["txtNom"];
        var db = Database.Open("csFilms");
        var sql = "insert into categories values(@0)";
        db.Execute(sql, nom);

        //Redirection
        Response.Redirect("categories.cshtml");
    }
}

<!DOCTYPE html>
<html>
<head>
    <title>Ajouter Une catégorie</title>
    <style type="text/css">
        .field-validation-error {
            font-weight: bold;
            color: red;
        }

        .validation-summary-errors {
```

```
            border: 2px solid #000000;
            color: red;
            font-weight: bold;
            margin: 12px;
        }
    </style>
</head>
<body>
    <h2>Ajouter une catégorie</h2>
    <form method="post">
        <fieldset>
            <legend>Informations de la catégorie</legend>
            <div>
                <label for="txtNom">Nom:</label>
                <input type="text" name="txtNom" />
                @Html.ValidationMessage("txtNom", "")
            </div>
            <input type="submit" name="ajouter" />
        </fieldset>
    </form>
    @Html.ValidationSummary()
</body>
</html>
```

modifierCategorie.cshtml

```
@{
    var db = Database.Open("csFilms");
    var id = "";
    var nom = "";
    if (IsPost)
    {
        //Enregistrer les modifications
        id = Request["txtId"];
        nom = Request["txtNom"];
        var sql = "update categories set nom=@1 where id=@0";
        db.Execute(sql, id, nom);
        Response.Redirect("categories.cshtml");
    }
    else {
        //Afficher le formulaire

        if (!Request["id"].IsEmpty())
        {
            id = Request["id"];
            var sql = "select * from categories where id=@0";
            var cat = db.Query(sql, id).First();
            nom = cat.nom;
        }
    }
}
```

```
<!DOCTYPE html>
<html>
    <head>
        <title>Ajout d'une catégorie</title>
    </head>
    <body>
        <h2>Ajout d'une catégorie</h2>
        <form method="post">
            <fieldset>
                <legend>Informations de la catégorie</legend>
                <div>
                    <label for="txtId">Nom:</label>
                    <input type="text" readonly name="txtId" value="@id"/>
                </div>
                <div>
                    <label for="txtNom">Nom:</label>
                    <input type="text" name="txtNom" value="@nom" />
                </div>
                <input type="submit" name="cmdAjouter" value="Ajouter" />
            </fieldset>
        </form>
    </body>
</html>
```

graph.cshtml

```
@{
    var db = Database.Open("csFilms");
    var data = db.Query("SELECT titre, BoxOffice FROM films");
    var graph = new Chart(width: 600, height: 400, theme: ChartTheme.Vanilla3D)
        .AddTitle("CA")
        .AddSeries("nomSerie", chartType: "Pie",
xValue: data, xField: "titre",
yValues: data, yFields: "BoxOffice")
        .Write();
}
```

films.cshtml

```
@{
    Page.Title= "Films";
    Layout = "~/_siteLayout.cshtml";
    var db = Database.Open("csFilms");
    var films = db.Query("select * from films");
    var grille = new WebGrid(source:films,rowsPerPage:5);
}
```

```
<h2> Liste des films </h2>
```

```
@grille.GetHtml(
```

```
    @*Application des styles*@
```

```
    tableStyle: "sTable",
```

```
    headerStyle: "sHeader",
```

```
    alternatingRowStyle: "sAlter",
```

```
    @*Choix des colonnes*@
```

```
    columns: grille.Columns(grille.Column("titre"),
```

```
        grille.Column("réalisateur"),
```

```
        grille.Column("BoxOffice"),
```

```
    grille.Column(format:@<a href="modifier.cshtml?id=@item.ID">Modifier</a>))
```

```
)
```

```

```

```
@section styles{
```

```
<style type="text/css">
```

```
    .sTable {
```

```
        margin: 4px;
```

```
        border-collapse: collapse;
```

```
        width: 600px;
```

```
    }
```

```
    .sTable th, .grid td {
```

```
        border: 1px solid #C0C0C0;
```

```
        padding: 5px;
```

```
    }
```

```
    .sHeader {
```

```
        background-color: #2ad5b6;
```

```
        font-weight: bold;
```

```
        color: #FFF;
```

```
    }
```

```
    .sAlter {
```

```
        background-color: #E8E8E8;
```

```
        color: #000;
```

```
    }
```

```
</style> }
```