

# Vues partielles, cookies et redirections

Cours #11



- ◆ Vues partielles  
- ◆ Cookies (Variables de session) 
- ◆ Redirections 
- ◆ Coup de pouce

## ◆ Vue partielles : C'est quoi ? 🤔

- Type de vue **Razor** (.cshtml) qui retourne des éléments HTML dans une autre vue.
- Un peu comme une vue « ordinaire » est insérée dans un **Layout** !

### Vue « ordinaire »

```
@using Ex_Cours11.Models;
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@model List<Dragon>

@{
    Layout = "_Layout";
}

<h2>Vues partielles</h2>
<div class="row m-2"><div class="col"><h3>Tag Helpers</h3></div></div>
<div class="row m-2">
    <div class="col-3 p-2">
        <partial name="_DragonPartial" for="@Model[0]" />
    </div>
</div>
<div class="row m-2">
    <div class="col-3 p-2">
        <partial name="_DragonPartial.cshtml" for="@Model[1]" />
    </div>
</div>
```

### Vue partielle

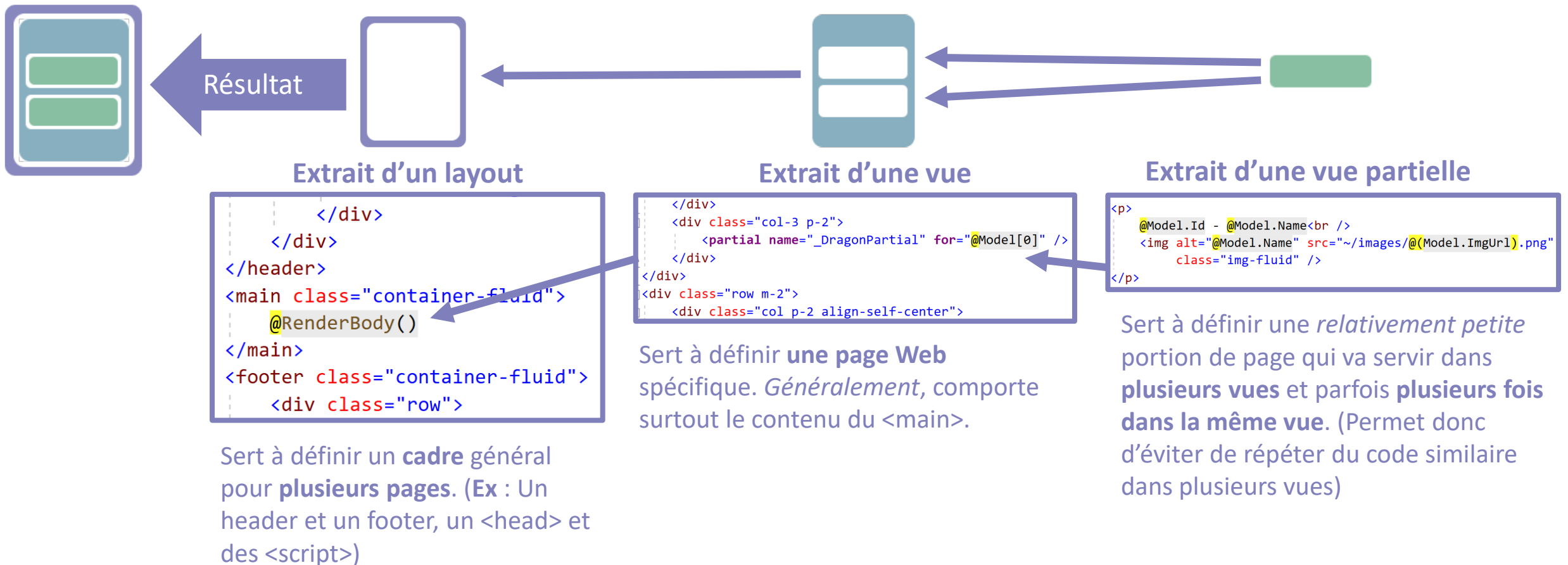
```
@using Ex_Cours11.Models;
@model Dragon

<p>
    @Model.Id - @Model.Name<br />
    
</p>
```

Cette vue **partielle** est insérée à deux endroits dans la vue « ordinaire ».

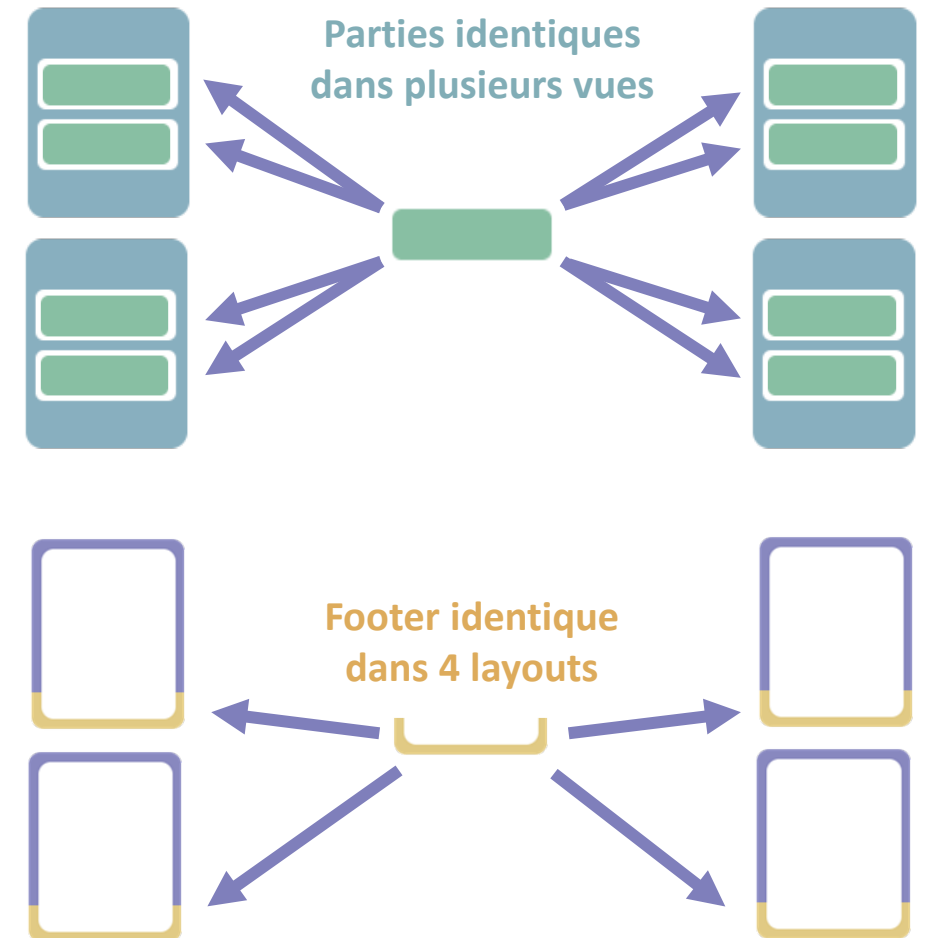
## ◆ Layout, vue et vue partielle

- À ne pas confondre



## ◆ Vues partielles

- À utiliser de la *bonne manière* !
- L'usage de **vues partielles** doit permettre d'éviter la **duplication de code**. Elles sont censées servir dans **plusieurs vues**.
- Les **vues partielles** ne devraient pas servir de **layout**. Par contre, si plusieurs **layouts** nécessitent des portions de code similaires, les **vues partielles** peuvent être utilisées dans ceux-ci. (Ex : Le **footer** est toujours le même dans 4 pages de layout)



## ◆ Vues partielles

- Par exemple, ici, les images et les noms des dragons apparaissent plusieurs fois dans la **vue**.
- Si ce genre d'affichage répétitif ne peut pas être réalisé avec une **boucle** ou s'il est présent dans **plusieurs vues**, c'est l'occasion de créer une **vue partielle**.

```
<partial name="~/ Views / Dragon /  
_DragonPartial.cshtml" for="@Model[2]" />
```

3 - Elizabeth



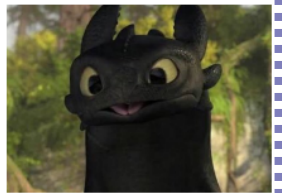
```
<partial name=" / Views / Dragon /  
_DragonPartial.cshtml" for="@Model[3]" />
```

4 - Spyro



```
<partial name=".. / Dragon /  
_DragonPartial.cshtml" model='new Dragon(5,  
"Toothless", "toothless")' />
```

5 - Toothless



Extrait d'une vue ↑

Vue partielle →

```
<div>  
    @Model.Id - @Model.Name<br />  
      
</div>
```

## ◆ Vues partielles

- Comment les utiliser

- 1 Créer la **vue partielle**. Il s'agit simplement d'intégrer les **éléments HTML** désirés et s'il le faut, préciser des **@directive**. Remarquez que, comme une vue ordinaire, une vue partielle peut recevoir un **@model** en paramètre !

### Exemple de vue partielle

```
_DragonPartial.cshtml  _Layout.cshtml  List.cshtml
@using Ex_Cours11.Models;
@model Dragon

<p>
    @Model.Id - @Model.Name<br />
    
</p>
```

## ◆ Vues partielles

- Comment les utiliser

2

Ensuite, il faudra utiliser un **tag helper\*** ou un **Html Helper** pour décider où cette **vue partielle** sera *insérée* dans la vue. Le **tag helper** / **Html Helper** permet aussi de passer un **@model** à la vue partielle, s'il le faut.

### Tag Helper

```
<div class="col-3 p-2">
  <partial name="_DragonPartial" for="@Model[0]" />
</div>
```

### Html Helper

```
<div class="col-3 p-2">
  @await Html.PartialAsync("_DragonPartial", Model[5])
</div>
```

\* Si vous utilisez un **tag helper**, n'oubliez pas la directive « **@addTagHelper \***, **Microsoft.AspNetCore.Mvc.TagHelpers** »



## ◆ Vues partielles

### ○ Comment les utiliser

CSHTML	Description
<code>&lt;partial name="_DragonPartial" for="@Model" /&gt;</code>	La <b>vue</b> (son nom et / ou chemin) est spécifiée avec l'attribut <b>name</b> et le <b>modèle</b> passé est spécifié avec l'attribut <b>for</b> .
<code>@await Html.PartialAsync("_DragonPartial", Model)</code>	Le premier paramètre est la <b>vue partielle</b> (son nom et / ou son chemin) et le deuxième est le <b>modèle</b> passé.
<code>@{ await Html.RenderPartialAsync("_DragonPartial", Model);}</code>	Envoie le résultat directement dans le flux HTTP plutôt que de retourner du contenu. Doit être appelé dans un bloc Razor pour cette raison. Plus performant dans <i>certaines situations</i> .

Ça fait 3 manières d'utiliser les **vues partielles**... laquelle utiliser ? Il est préférable d'utiliser le **Tag Helper**. (Première solution) Il gère d'éventuelles difficultés à notre place grâce à sa syntaxe simple.

**await** : Étant donné que les vues partielles sont générées de manière « *asynchrone* », (La vue continue de se générer simultanément pendant que la vue partielle est récupérée) ce mot-clé est nécessaire.

## ◆ Vues partielles

- **Chemin** vers la vue partielle... et le layout
- Que se passe-t-il lorsqu'on précise un fichier sans chemin ?

```
└─ Views
  └─ Dragon
    ├── _DragonPartial.cshtml
    ├── Index.cshtml
    ├── List.cshtml
    └─ shared
      └─ _Layout.cshtml
```

Nom du fichier de **Layout** sans préciser qu'il est dans **Shared**...

```
@{
    Layout = "_Layout";
}
```

Nom de la **vue partielle** (Qui est dans le même dossier que la vue)

```
<div class="col-3 p-2">
  <partial name="_DragonPartial" for="@Model[0]" />
</div>
```

- Dans ces situations... le fichier est cherché à ces endroits, dans cet ordre :
  1. Recherché dans le **même dossier que la vue** l'appelle.
  2. Recherché dans le **dossier principal du contrôleur de la vue**. (Parfois identique à #1)
  3. Recherché dans le dossier « **Shared** ».

## ◆ Vues partielles : ViewData

- Les données du **ViewData** qui existent dans une **vue** sont transmises aux **vues partielles** qui y sont référencées...
- Mais toute modification faite au **ViewData** dans une **vue partielle** ... n'est pas effective dans les **vues** qui la référencient !

```
@{  
    ViewData["x"] = 1;  
}  
@ViewData["x"] 1 Imprime « 1 »  
<div class="col-3 p-2">  
    <partial name="_DragonPartial.cshtml" for="@Model[1]" />  
</div>  
@ViewData["x"] 3 Imprime « 1 »
```

```
<div>  
    @Model.Id - @Model.Name<br />  
      
    @{  
        ViewData["x"] = 2;  
    }  
    @ViewData["x"] 2 Imprime « 2 »  
</div>
```

- L'exemple ci-dessus illustre cette dynamique. On a changé la valeur de **x** dans la **vue partielle**... mais cela n'a pas d'impact sur la **vue** !

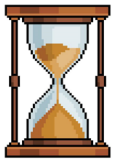


## ◆ Cookies / Variables de session : C'est quoi ? 🤪 🍪

- Ce sont des données spécifiques à un utilisateur qui sont stockées pour la durée de sa « **session** ».

- **Exemples** : Une préférence pour une langue d'affichage, des articles dans un panier virtuel, des informations d'authentification de l'utilisateur, etc.

- À ne pas confondre avec les « **Persistent Cookies** »



- Les variables de **session** durent seulement pour la « **session** » (**ex** : sont supprimés après 20 minutes d'inactivité sur l'application Web) alors que les **Persistent Cookies** sont conservés *durablement*.

- Les variables de session sont spécifiques au **navigateur**.
- Il faut éviter de stocker des **données sensibles** dans les variables de session.



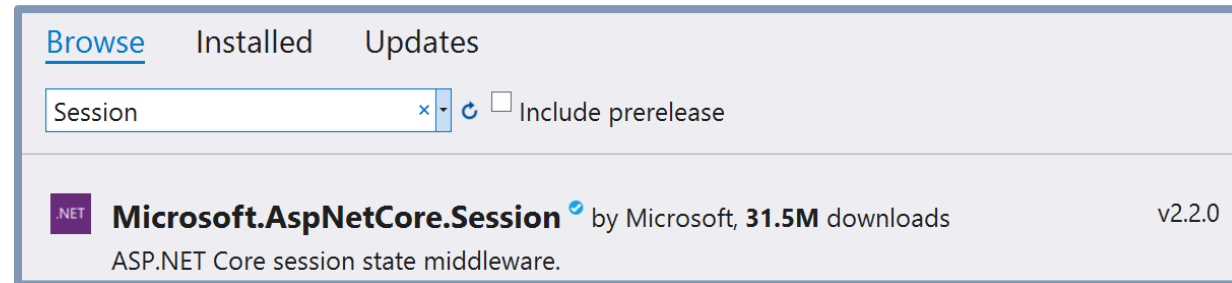
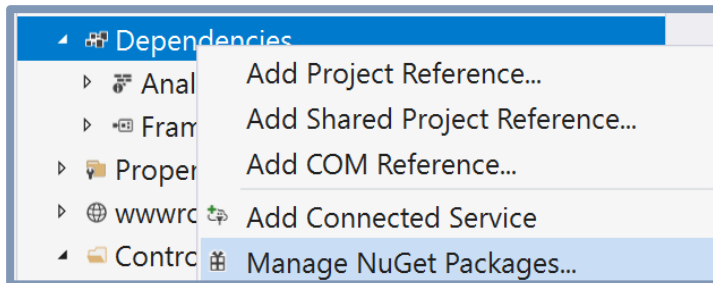
## ◆ Variables de session

- Comment les utiliser ?
- Il y a quelques prérequis ...



1

Installer le package NuGet « **Microsoft.AspNetCore.Session** »



## ◆ Variables de session

### ○ Comment les utiliser ?

2 Ajouter les instructions suivantes dans la classe **Program.cs**

```
builder.Services.AddDistributedMemoryCache(); // Permet l'utilisation de cookies
builder.Services.AddSession(option => { option.IdleTimeout = TimeSpan.FromMinutes(20); });

var app = builder.Build();
```

- Il y a deux « services » à ajouter dans la méthode « **ConfigureServices** »
- Ce nombre permet de définir la durée (en minutes) d'**inactivité** de l'utilisateur suite à laquelle les données des **variables de session** sont supprimées.

La directive `using System;` sera nécessaire pour utiliser « **TimeSpan** »

```
using System;
```

```
app.UseSession();
app.UseRouting();
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller}/{action}/{id?}",
        defaults: new { controller = "Dragon", action = "Index" });
});
```

Il faut également ajouter cette instruction dans la méthode « **Configure** »

```
using Microsoft.AspNetCore.Http;
```

Cette **directive** sera nécessaire pour utiliser **HttpContext**

## ◆ Variables de session

- Comment les utiliser ?

3 Nous sommes maintenant prêts à stocker des données dans les **variables de session** grâce à **HttpContext.Session**

- Les méthodes suivantes permettent de stocker des entiers et des chaînes de caractères sous formes de variables de session :

- `HttpContext.Session.SetString("Key", valeur);`
- `HttpContext.Session.SetInt32("Key", valeur);`

```
HttpContext.Session.SetString("Name", "Antoine");  
HttpContext.Session.SetInt32("Number", 5);
```

- Les méthodes suivantes permettent de récupérer des entiers et des chaînes de caractères stockées comme variables de session :

- `HttpContext.Session.GetString("Key")`
- `HttpContext.Session.GetInt32("Key")`

```
string name = HttpContext.Session.GetString("Name");  
int? number = HttpContext.Session.GetInt32("Number");
```

## ◆ Variables de session

- Exemple simple
- Dans la vue **Index**, on affiche « Bonjour x » si jamais **ViewData["name"]** contient une valeur... et sinon, on demande d'entrer un nom !

```
@if (ViewData["name"] != null)
{
    <p>Bonjour @ViewData["name"] !</p>
}
else
{
    <form method="POST" action="Dragon/Index">
        <label for="Name">Quel est ton nom ? </label>
        <input type="text" id="Name" name="Name" />&nbsp;&nbsp; 
        <button type="submit">Envoyer</button>
    </form>
}
```





## ◆ Variables de session

- Exemple simple (Suite)

- L'action **Index** reçoit la requête de type **POST** avec le nom spécifié, elle stocke cette valeur comme variable de session avec la clé "**Name**", puis elle passe cette valeur à la **vue** à l'aide de **ViewData**.

Quel est ton nom ?

```
[HttpPost]
0 references
public IActionResult Index(String Name)
{
    HttpContext.Session.SetString("Name", Name);
    ViewData["name"] = HttpContext.Session.GetString("Name");
    return View();
}
```

Stockage de la valeur en  
variable de session

Récupération de la  
valeur pour la glisser  
dans **ViewData**

## ◆ Variables de session

- Exemple simple (Suite)
- Cette fois-ci, **ViewData["name"]** contient forcément une valeur... alors on affiche « Bonjour x ».

```
@if (ViewData["name"] != null)
{
    <p>Bonjour @ViewData["name"] !</p>
}
else
{
    <form method="POST" action="Dragon/Index">
        <label for="Name">Quel est ton nom ? </label>
        <input type="text" id="Name" name="Name" />&nbsp;&nbsp; 
        <button type="submit">Envoyer</button>
    </form>
}
```



## ◆ Variables de session

- Exemple simple (Suite)
- D'ailleurs, l'action **Index** qui reçoit les requêtes de type **GET** envoie également cette valeur à la vue avec **ViewData**.
- Ainsi, pour le reste de la durée de la **session** de l'utilisateur, ce sera toujours « Bonjour x » qui sera affiché.

```
public IActionResult Index()
{
    String name = HttpContext.Session.GetString("Name");
    if(name != null)
        ViewData["name"] = name;
    return View();
}

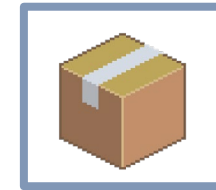
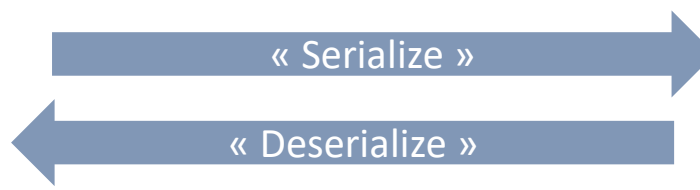
[HttpPost]
0 references
public IActionResult Index(String Name)
{
    HttpContext.Session.SetString("Name", Name);
    ViewData["name"] = HttpContext.Session.GetString("Name");
    return View();
}
```

## ◆ Stocker d'autres types d'objets

- Pour stocker des **objets complexes** (List<>, classes du modèle, etc.), nous devons procéder d'une autre manière.
- Il faudra se munir de deux méthodes :
  - Une méthode qui transforme nos **objets complexes** en **string** ... pour les ranger dans les variables de session avec la méthode **SetString("Key", valeur);**
  - Une méthode qui re-transforme ces **strings** en **objet complexe** ... pour les extraire depuis leur variable de session avec la méthode **GetString("Key");**



**Objet complexe** (Ex : Un « **Dragon** » avec des propriétés « Id », « Nom » et « Couleur »)



**String** (La complexité de notre objet a été encapsulée !)  
On peut le stocker dans une **variable de session**.



## ◆ Stocker d'autres types d'objets

- La [documentation Microsoft](#) pour les variables de session propose deux méthodes encapsulées dans une classe pour réaliser cela !

```
public static class SessionExtensions
{
    public static void Set<T>(this ISession session, string key, T value)
    {
        session.SetString(key, JsonSerializer.Serialize(value));
    }

    public static T Get<T>(this ISession session, string key)
    {
        var value = session.GetString(key);
        return value == null ? default : JsonSerializer.Deserialize<T>(value);
    }
}
```

```
using Microsoft.AspNetCore.Http;
using System.Text.Json;
```



## ◆ Stocker d'autres types d'objets

- La [documentation Microsoft](#) pour les variables de session propose deux méthodes encapsulées dans une classe pour réaliser cela !
- Les fonctions **Set<T>** et **Get<T>** profitent du format **JSON** (Javascript **O**bject **N**otation) pour nous permettent d'encapsuler nos objets complexes dans des **strings**.
- Il n'est pas nécessaire de comprendre leur fonctionnement en détail.
- Les prochaines diapositives illustrent un exemple simple qui utilise ces méthodes.



## ◆ Méthodes Set<T> et Get<T>

- Importer les méthodes dans son projet

Copiez-collez la classe **SessionExtensions** (Elle est quelques diapositives avant...) avec ses deux méthodes dans votre projet dans un dossier / namespace « **Extensions** ».

Désormais, vous pourrez utiliser les méthodes **Set<T>** et **Get<T>** depuis n'importe quelle autre classe de cette manière :

- `HttpContext.Session.Set<T>(key, value)`

**Type** (Ex : `List<Dragon>`)

- `HttpContext.Session.Get<T>(key)`



```
▶ wwwroot
▶ Controllers
▶ Extensions
  ▶ SessionExtensions.cs
▶ Models
▶ Views
▶ appsettings.json
▶ Program.cs
▶ Startup.cs
```



## ◆ Méthodes Set<T> et Get<T>

- Exemple d'utilisation de ces méthodes

- Je souhaite « récupérer » un objet de type « **Dragon** » rangé à la **clé** "mon\_dragon" :

- **Dragon** d = HttpContext.Session.**Get**<**Dragon**>("mon\_dragon");

- Je souhaite « stocker » un objet de type **List<Dragon>** à la **clé** "mes\_dragons" :

- HttpContext.Session.**Set**<**List<Dragon>**>("mes\_dragons", liste)



Ceci est une variable qui contient la List<Dragon> que je souhaite stocker.





## ◆ Méthodes Set<T> et Get<T>

- « Ajouter un article au panier »
- Disons que j'ai une **vue** avec une liste d'objets que je peux ajouter à mon panier :



Chaque **bouton** « Submit » est encapsulé dans un **formulaire** et accompagné d'un champ caché (hidden) avec l'**Id** du Dragon associé.

```
<form method="POST" asp-controller="Dragon" asp-action="Cart">  
  <input type="hidden" name="Id" value="@Model.Id" />  
  <button type="submit">Ajouter au panier</button>  
</form>
```

## ◆ Méthodes Set<T> et Get<T>

### ○ « Ajouter un article au panier » (Suite)

- Cette action est appelée par le formulaire quand on ajoute un Dragon au panier.

```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public IActionResult Cart(int Id)
{
    List<Dragon> dragons = HttpContext.Session.Get<List<Dragon>>("dragons");

    if (dragons == null)
    {
        dragons = new List<Dragon>();
    }

    dragons.Add(DB.Dragons.Where(d => d.Id == Id).Single());
    HttpContext.Session.Set<List<Dragon>>("dragons", dragons);



    return View(dragons);
}
```

- On récupère la liste de dragons déjà dans le **panier**
- Si elle n'existait pas, on la crée !
- On y ajoute le **dragon** dont l'**Id** correspond au bouton cliqué (Reçu en paramètre avec **int Id**)
- On met à jour la liste de dragons dans le **panier** (Dans la **variable de session** avec la clé "dragons")
- On retourne la vue du **Panier** en lui fournissant la liste de dragons.

## ◆ Méthodes Set<T> et Get<T>

- « Ajouter un article au panier » (Suite)
- Le panier est une vue qui affiche la **List<Dragon>** reçue... ou qui affiche « Votre panier est vide » si la liste est « null » !

```
@model List<Ex_Cours11.Models.Dragon>
```

Votre panier		
Id	Name	Image
4	Spyro	
3	Elizabeth	

```
@if (Model == null)
{
    <p>Votre panier est vide !</p>
}
else
{
    <table class="table">
        <thead>
            <tr><th>Id</th><th>Name</th><th>Image</th></tr>
        </thead>
        <tbody>
            @foreach (var item in Model)
            {
                <tr>
                    <td>@Html.DisplayFor(modelItem => item.Id)</td>
                    <td>@Html.DisplayFor(modelItem => item.Name)</td>
                    <td></td>
                </tr>
            }
        </tbody>
    </table>
}
```



## ◆ Méthodes Set<T> et Get<T>

- « Ajouter un article au panier » (Suite)
- Ne pas oublier que l'action de type « GET » pour le panier doit aussi récupérer la liste de dragons dans les **variables de session** pour l'envoyer à la vue.

```
public IActionResult Cart()  
{  
    List<Dragon> dragons = HttpContext.Session.Get<List<Dragon>>("dragons");  
    return View(dragons);  
}
```





## ◆ Rediriger vers une **vue**

- Ces méthodes retournent des **vues**. (.cshtml)
- Elles sont à utiliser dans des **actions** du **contrôleur**.

Méthode	Description
<code>return View();</code>	Retourne la <b>vue</b> du même nom que le nom de l' <b>action</b> .
<code>return View("Nom_de_la_vue");</code>	Retourne la <b>vue</b> qui possède le nom définie en <b>paramètre</b> entre guillemets.

- Si jamais la **vue** retournée n'existe pas ou ne peut pas être trouvée dans le dossier du contrôleur (Ex : **Views/Home**) ou dans le dossier **Shared**... on a une erreur !



## ◆ Rediriger vers une **action**

- Ces méthodes appellent une autre **action**.
- Elles sont à utiliser dans des **actions** du **contrôleur**.

Méthode	Description
<code>return RedirectToAction("Action");</code>	Redirige vers l' <b>action</b> qui porte le nom fournit en paramètre. Elle doit être dans le même contrôleur.
<code>return RedirectToAction("Action", "Contrôleur");</code>	Redirige vers l' <b>action</b> qui porte le nom fournit en 1 <sup>er</sup> paramètre et qui fait partie du <b>contrôleur</b> fournit en 2 <sup>e</sup> paramètre.
<code>return RedirectToAction("Action", "Contrôleur", new { id = 3 });</code>	Redirige vers le <b>contrôleur</b> et l' <b>action</b> spécifiés tout en fournissant un ou plusieurs <b>paramètres</b> dans un objet anonyme.

## ◆ Rediriger vers une **action**

- Exemple pour `RedirectToAction("Action", "Contrôleur", new { id = 3 });`

```
0 references
public IActionResult Action1(int poof)
{
    return View(poof);
}

[HttpPost]
0 references
public IActionResult Action2()
{
    // ...
    int x = 3;
    return RedirectToAction("Action1", "Home", new { poof = x });
}
```

- Ici, l'**Action2** redirige vers l'**Action1** tout en fournissant la valeur « 3 » pour le paramètre « poof ».
- Notons que si nous avons tout simplement mis « `Return View("Action1");` » à la place d'une redirection, la vue **Action1.cshtml** n'aurait jamais reçu le paramètre **poof** et le code de l'**Action1** ne se serait pas exécuté.

## ◆ Ajouter et supprimer une entité enfant

- Pas si simple ! D:
- Tous les **enfants** sont dans ...
  - La liste d'enfants de la BD
  - La liste d'enfants du parent

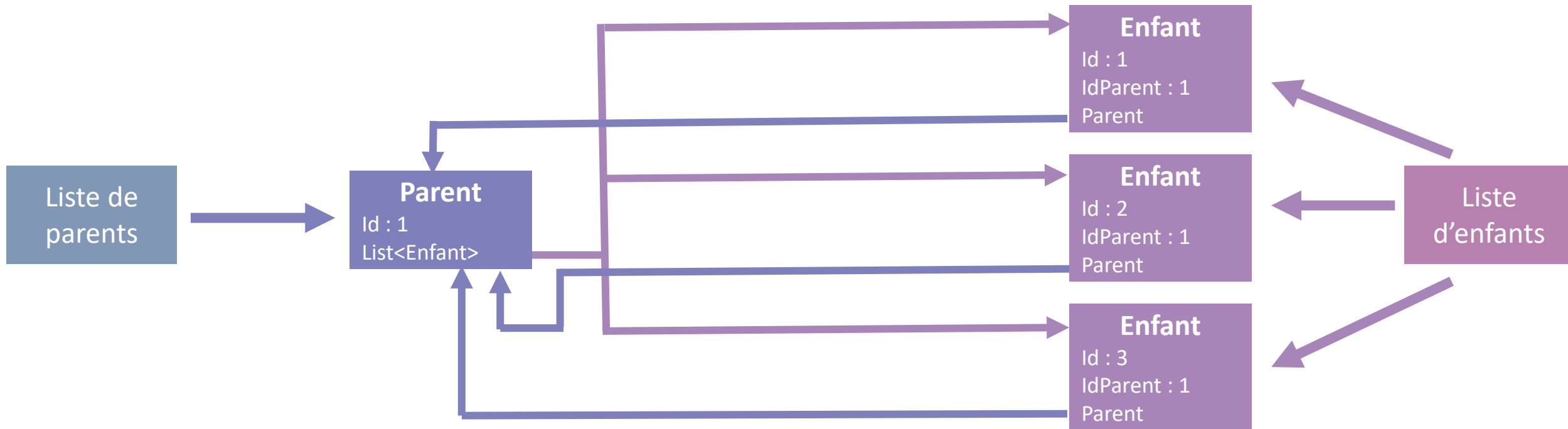
```
Enfants = new List<Enfant>();
Parents = new List<Parent>();

Parents.Add(new Parent() { Id = 1, Nom = "Le clan des démons", ImageU
Parents.Add(new Parent() { Id = 2, Nom = "Le clan des déesses", ImageU
Parents.Add(new Parent() { Id = 3, Nom = "Le clan des fées", ImageUR
Enfants.Add(new Enfant() { Id = 1, Nom = "Élaine", IdParent = 3, Imag
Enfants.Add(new Enfant() { Id = 2, Nom = "Gloxinia", IdParent = 3, Irr
Enfants.Add(new Enfant() { Id = 3, Nom = "Glaylord", IdParent = 1, Irr
Enfants.Add(new Enfant() { Id = 4, Nom = "Helbram", IdParent = 3, Ima
Enfants.Add(new Enfant() { Id = 5, Nom = "King", IdParent = 3, ImageU
Enfants.Add(new Enfant() { Id = 6, Nom = "Ludeciel", IdParent = 2, Irr
Enfants.Add(new Enfant() { Id = 7, Nom = "Melascula", IdParent = 1, I
Enfants.Add(new Enfant() { Id = 8, Nom = "Monspiet", IdParent = 1, Irr
Enfants.Add(new Enfant() { Id = 9, Nom = "Sariel", IdParent = 2, Imag
Enfants.Add(new Enfant() { Id = 10, Nom = "Tarmiel", IdParent = 2, Irr
Enfants.Add(new Enfant() { Id = 11, Nom = "Zaneri", IdParent = 2, Ima
Enfants.Add(new Enfant() { Id = 12, Nom = "Zeldris", IdParent = 1, Irr

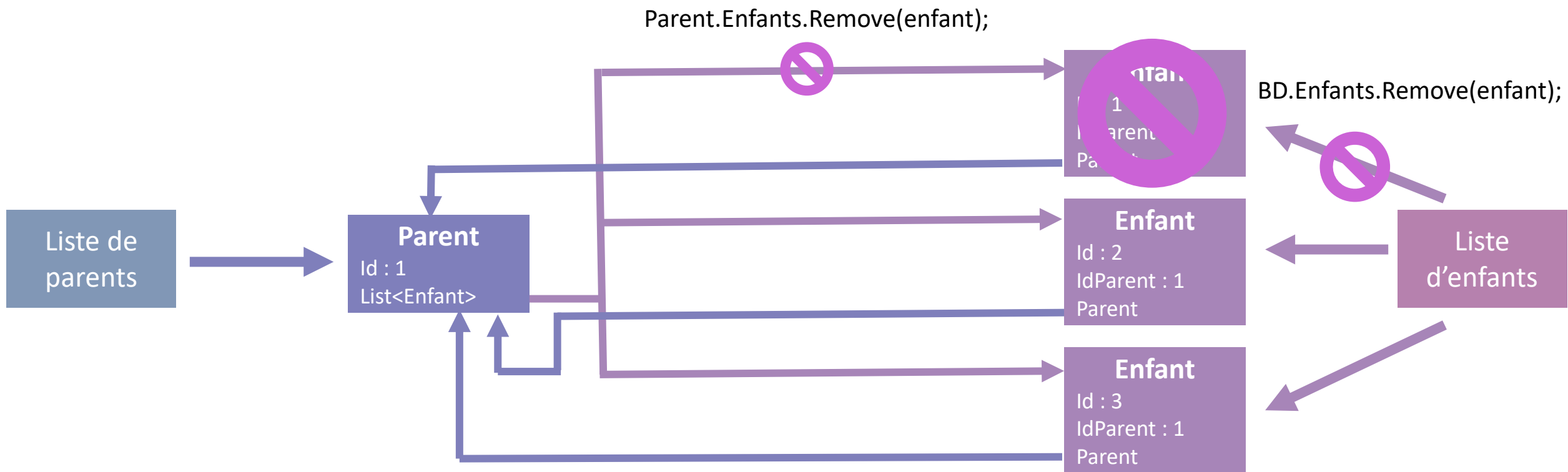
//Lier les objets enfants aux différents parents
foreach (var p in Parents)
{
    p.Enfants = new List<Enfant>();
    var enfants = Enfants.Where(e => e.IdParent == p.Id).ToList();
    p.Enfants.AddRange(enfants);
}

//Lier les objets parents aux différents enfants
foreach(var e in Enfants)
{
    e.Parent = Parents.Where(p => p.Id == e.IdParent).Single();
}
```





## ◆ Supprimer un enfant



## ◆ Ajouter un enfant

