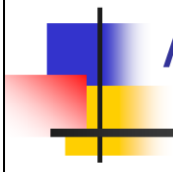


ASP.Net MVC

Les bases d'Asp.Net MVC
One Asp – Modèles - Contrôleurs

Web.config

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.5.1" />
    <httpRuntime targetFramework="4.5.1" />
  </system.web>
  <appSettings>
    <add key="webpages:Version" value="3.0.0.0" />
    <add key="webpages:Enabled" value="false" />
    <add key="ClientValidationEnabled" value="true" />
    <add key="UnobtrusiveJavaScriptEnabled" value="true" />
  </appSettings>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="System.Web.Helpers" publicKeyToken="31bf3856ad364e35" />
        <bindingRedirect oldVersion="1.0.0.0-3.0.0.0" newVersion="3.0.0.0" />
      </dependentAssembly>
      <dependentAssembly>
        <assemblyIdentity name="System.Web.WebPages" publicKeyToken="31bf3856ad364e35" />
        <bindingRedirect oldVersion="1.0.0.0-3.0.0.0" newVersion="3.0.0.0" />
      </dependentAssembly>
      <dependentAssembly>
        <assemblyIdentity name="System.Web.Mvc" publicKeyToken="31bf3856ad364e35" />
        <bindingRedirect oldVersion="1.0.0.0-5.2.2.0" newVersion="5.2.2.0" />
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
  . . .
</configuration>
```

Web.config

- Paramètres classiques

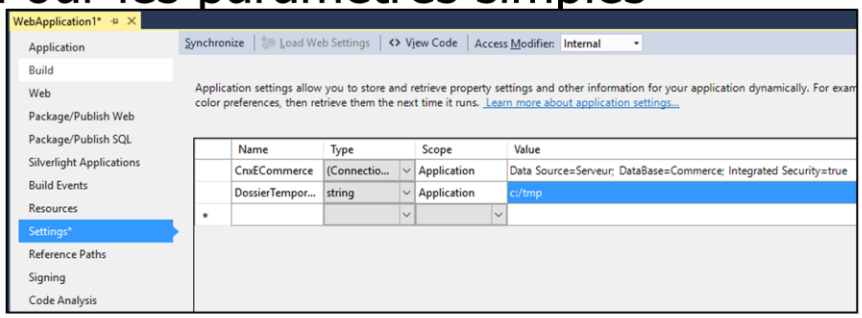
```
<appSettings>
  <add key="webpages:Enabled" value="false" />
  <add key="CheminFichierNbDeVisiteurs" value="c:/temp/NbDeVisiteurs.txt" />
  <add key="DossierTemporaire" value="c:/tmp" />
</appSettings>
<connectionStrings>
  <add name="CnxCommerce" providerName="System.Data.SqlClient"
       connectionString="Data Source=Serveur; DataBase=Commerce; Integrated Security=true" />
</connectionStrings>
```

- Pour une utilisation souple & typée

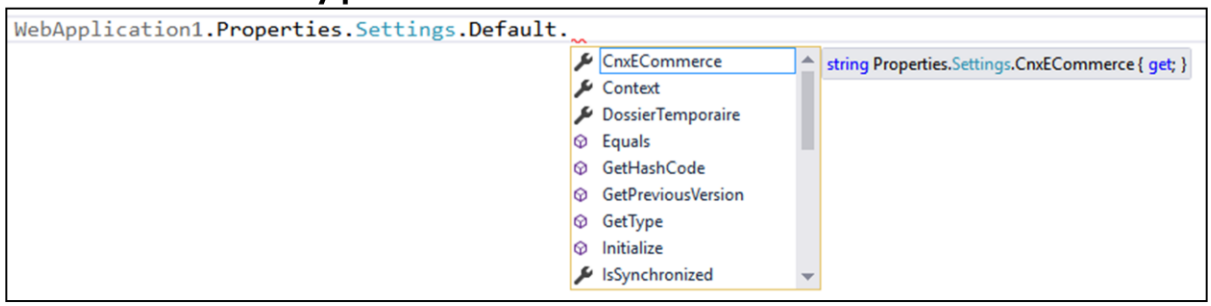
```
public static class ConfigSpecifique {
  public static readonly string DossierTemporaire;
  public static readonly System.Configuration.ConnectionStringSettings CnxECommerce;
  static ConfigSpecifique() {
    DossierTemporaire = System.Configuration.ConfigurationManager.AppSettings["DossierTemporaire"];
    CnxECommerce = ConfigurationManager.ConnectionStrings["DossierTemporaire"];
  }
  public static DbConnection CreerConnexionECommerce() {
    if( CnxECommerce.ProviderName == "System.Data.SqlClient" )
      return new SqlConnection( CnxECommerce.ConnectionString );
    if( CnxECommerce.ProviderName == "System.Data.OleDb" )
      return new OleDbConnection( CnxECommerce.ConnectionString );
    throw new NotSupportedException( CnxECommerce.ProviderName );
  }
}
```

Web.config

- Pour les paramètres simples



- Utilisation typée



Les "settings" sont également écrits dans Web.config, à la racine du projet.
Les propriétés sont également stockées dans <appSettings> et <connectionStrings>

Les noms des chaines de connexion sont préfixées "WebApplication1.Properties.Settings."

Outils Asp.Net - Request

- Principaux membres de `Controller.Request`

Méthode	Info	Exemple
QueryString	Collection, décortique la requête présente dans l'URL,	var idDeLObjetAtripoter = int.Parse(Request.QueryString["id"]);
Form	Collection, permet de lire directement les zones de saisie envoyées par le navigateur en décortiquant les entêtes HTTP,	string saisie = Request.Form["NameDUnControleHtml"];
Cookies	Collection, permet de lire les cookies envoyés par le navigateur,	HttpCookie c = Request.Cookies["CouleurPreferee"]; string couleur = c == null ? String.Empty : c.Value;
Url	Propriété, permet de décortiquer l'URL demandée par le navigateur,	string p = System.IO.Path.GetFileNameWithoutExtension(Request.Url.AbsolutePath);

HttpMethod	Propriété, permet de connaître le verbe utilisé par le client	if(Request.HttpMethod == "POST")
		  cEstQuoi = "Une soumission de formulaire";
		else if(Request.HttpMethod == "GET")
		  cEstQuoi = "Une navigation";

  - Rechercher un cookie absent dans `Request.Cookies` retourne `null`

Outils Asp.Net - Response

- Principaux membres de Controller.Response

Méthode	Info	Exemple
Cache	Propriété, gère le comportement du cache navigateur.	Response.Cache.SetNoStore();
Cookies	Collection, permet d'envoyer des cookies au navigateur. Sans date d'expiration, les cookies sont dits "de session", càd perdu à la fermeture du navigateur.	HttpCookie c = new HttpCookie("CouleurPreferee", TbCouleur.Text); c.Expires = DateTime.Now.AddDays(3); Response.Cookies.Add(c);

Redirect	Méthode populaire en WebForm. Elle envoie un entête HTTP au navigateur pour qu'il requête une autre page (statut 302).
	En Asp.Net MVC nous préfèrerons l'aiguillage vers une autre vue lorsque c'est possible ou Controller.Redirect(uneURL)
		if(resultat > 5)
		  Response.Redirect("../Depassement/Index");
		else if(resultat < 0)
		  this.Redirect("../TropPetit/Index");
		else if(resultat == 0)
		  return RedirectToAction("Index", "TEsNul"); // Sans suffixe "Controller"
		else
		  return View("SpecifiqueNumero" + resultat);

Write	Méthode, écrit directement dans le flux http vers le navigateur.	Response.Write("Devinez où ça va apparaître dans le HTML !");

- Rechercher un cookie absent dans `Response.Cookies` retourne un cookie de valeur `String.Empty`
  - ☹ Ce cookie a été ajouté à votre insu dans la collection et sera donc envoyé au navigateur !
  - ☺ On peut (heureusement) tester `Cookies.AllKeys`

Outils Asp.Net - Déconnexion HTTP

- Le protocole HTTP n'oblige pas à maintenir la connexion entre deux requêtes
  - Objet contrôleur à usage unique
  - Variables d'instances perdues
- Pour survivre au "post" ou à la navigation
  - Session
  - Cookies
  - Variables de classe
  - Stockage permanent (BD, fichiers XML, registre, …)
  - Champs cachés (`ViewState` réservé aux WebForms !)

Outils Asp.Net – Champs cachés

- Utilisation des champs cachés
  - Simple `<input type="hidden" />`
    - L'attribut name permet d'en envoyer la valeur dans les entêtes
    - La valeur peut être fixée par le serveur (persistance d'état)
    - La valeur peut être fixée en JavaScript (communication code client / code serveur)
  - Affectation par la vue à partir du modèle fournit par le contrôleur
  - Lecture, pour le moment, par `Request.Form`
    - ⚠ Sérialisation / dé-sérialisation manuelle des données
    - ⚠ Faire un HTTP POST "nécessite" un formulaire

Outils Asp.Net - Champs cachés

- Exemple - Contrôleur : ExempleChampCacheController.cs

```
public class ExempleChampCacheController : Controller {
  static string[] Questions = { "Un sport de combat", "Un produit de 1ère nécessité", . . . };
  public ActionResult Index() {                          ← Action = méthode qui retourne un ActionResult
    ViewBag.Precedent = DateTime.Now.ToString( "G" );
    ViewBag.Questions = ExempleChampCacheController.Questions;
    if( Request.HttpMethod == "POST"  ) {
      string r = Request.Form["r"]; ← Le ControllerActionInvoker permettra de le recevoir en paramètre
      DateTime precedent = DateTime.Parse( Request.Form["precedent"] );
      var delai = (DateTime.Now - precedent).TotalSeconds;
      var ok = Request.Form["r"]?.Split( ',' ).Length == Questions.Length ? "correctement"
                                                                          : "de travers";
      ViewBag.Message =
                $"Il vous a fallu {delai:0.00} seconde(s) pour répondre {ok} !\nVous pouvez rejouer.";
    }
    else {
      ViewBag.Message = "Répondez vite, répondez bien !";
    }
    return View();                                       ← View() retrouve la vue associée à l'action
  }
```

Parenthèse Razor

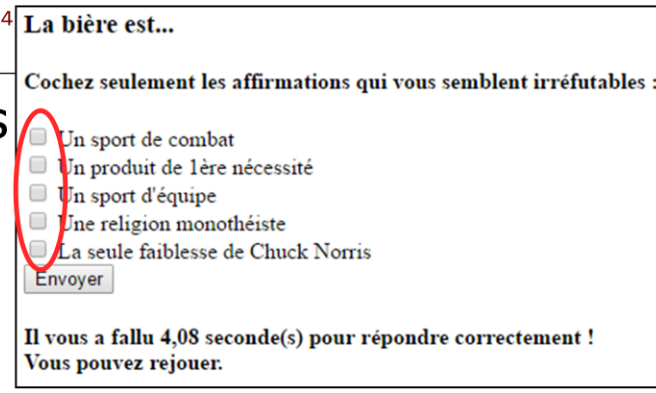
- Razor "*devine*" ce qui est code ou HTML
  - @ précède du code serveur
  - @@ déspécialise le caractère @
  - @: précise explicitement qu'il s'agit d'une ligne de HTML
  - <text> précise explicitement qu'il s'agit d'un bloc HTML
  - @(i + j) et pas @i + j
  - ☺ Le texte fabriqué par Razor est toujours "*HtmlEncodé*"
    - Utiliser Html.Raw() pour l'éviter
- Razor permet de créer des blocs de code
  - Code injecté dans une méthode de rendu, au milieux de "`Response.Write`"
    - Déclarer des variables locales, faire des boucles, …
    - ☹ ~~Faire le boulot du contrôleur~~
  - `@functions { . . . }` pour créer des méthodes de présentation
- Razor utilise C# 5 avec VS 2015, pour mettre à jour
  - PM> Install-Package Microsoft.CodeDom.Providers.DotNetCompilerPlatform

Outils Asp.Net - Champs cachés

- Exemple - Vue : Index.cshtml

```
<form method="post" action="/ExempleChampCache/Index">
  <h3>La bière est...</h3>
  <h4>Cochez seulement les affirmations qui vous semblent irréfutables :</h4>
  @for( int i = 0, len = ViewBag.Questions.Length; i < len; i++ ) {
    <input type="checkbox" name="r" value="@i" id="r@(i)" />
    <label for="r@(i)">@ViewBag.Questions[i]</label><br />
  }
  <input type="hidden" name="precedent" value="@ViewBag.Precedent" />
  <input type="submit" value="Envoyer" />
  <h4 style="white-space: pre">@ViewBag.Message</h4
</form>
```

☹ Comment conserver les cases cochées ?



Outils Asp.Net - Champs cachés

- Exemple : correction du contrôleur

```
int[] reponses = Request.Form["r"]?.Split( ',' ).Select( str => int.Parse( str ) ).ToArray();
reponses = reponses ?? new int[0];
ViewBag.Selectionner = reponses;
ViewBag.Questions = ExempleChampCacheController.Questions;
```

- Correction de la vue

```
<h4>Cochez seulement les affirmations qui vous semblent irréfutables :</h4>
@{ var selectionner = (int[])ViewBag.Selectionner; }
@for( int i = 0, len = ViewBag.Questions.Length; i < len; i++ ) {
  <input type="checkbox" name="r"
         value="@i" id="r@(i)"
         @(selectionner.Contains( i ) ? "checked" : "") />
<label for="r@(i)">@ViewBag.Questions[i]</label><br />
}
```

La bière est...

Cochez seulement les affirmations qui vous semblent irréfutables :

- [ ] Un sport de combat
- [ ] Un produit de 1ère nécessité
- [x] Un sport d'équipe
- [x] Une religion monothéiste
- [ ] La seule faiblesse de Chuck Norris

Envoyer

Il vous a fallu 0,19 seconde(s) pour répondre de travers !
Vous pouvez rejouer.

Outils Asp.Net - `Controller.Session`

- Web.config, dans <configuration><system.web>

```
<sessionState
    mode="[Off|InProc|StateServer|SQLServer|Custom]"
    timeout="20"
    cookieName="session identifier cookie name"
    cookieless="[true|false|AutoDetect|UseCookies|UseUri|UseDeviceProfile]"
    sqlConnectionString="sql connection string"
    stateConnectionString="tcpip=server:port"
    compressionEnabled="[True|False]">
</sessionState>
```

⚠ Tag allégé !

- API

Méthode	Info	Exemple

IsNewSession	Propriété, utile pour détecter une perte de session	if(IsNewSession)
		  return View("CaddieVide");
		else { ... }

SessionID	Propriété, valeur du cookie de session	
Timeout	Propriété, réglable finement pour chaque session	Session.TimeOut = 120;
Abandon	Méthode, la prochaine requête déclenchera Session_Start	Session.Abandon(); return View("CaddieVide");
Add	☺	Session.Add("clé", objetValeur);
Remove	☺	Session.Remove("clé");
this[string]	Indexeur, pour accéder en lecture ou en écriture aux données de session	Session["DerniereVisite"] = DateTime.Now;

Outil Session, patterns d'utilisation

- Utiliser Global.asax/Global.asax.cs
  - `Session_Start` : mettre une valeur par défaut dans le champ de session
  - `Session_End` : sauver les données de session dans un stockage permanent
- Utiliser des propriétés de classe
  - `get` → lire une valeur en session
  - `set` → écrire une valeur en session
  - `Session` étant une propriété d'instance de `Controller` ➔ utiliser `HttpContext`
    - Bien sûr, `HttpContext` inutilisable lorsque `Session_End` !

Outil Session, patterns d'utilisation

- Nombreux handlers dans global.asax.cs

Global.asax.cs

```
public class MvcApplication : System.Web.HttpApplication {
  protected void Application_Start( object sender, EventArgs e ) { }
  protected void Application_BeginRequest( object s, EventArgs a ) { }      ← Attention : pas encore de Session !
  protected void Application_PostAcquireRequestState( object s, EventArgs a ) { }        ← OK, Session présente
  protected void Application_Error( object sender, EventArgs e ) { }
  protected void Session_Start( object sender, EventArgs e ) {
    Preferences.Pagination = 10;
  }
  protected void Session_End( object sender, EventArgs e ) {
    // Attention : ici pas de requête HTTP, donc pas de HttpContext !
  }
}
```

Preferences.cs

```
static class Preferences {
  public static int Pagination {
    get {
      return (int)HttpContext.Current.Session["Pagination"];
    }
    set {
      HttpContext.Current.Session["Pagination"] = value;
    }
  }
}
```

Outils Asp.Net - `Application`

- Grandement obsolète…
  - Permettait de stocker des valeurs uniques
    - Les variables de classe sont désormais plus fiables
  - Seules les verrous sont encore (un peu) d'actualité
    - L'API .Net de programmation multithread reste plus fine
- API

Méthode	Info	Exemple
this[string]	Indexeur, obsolète, permettait de stocker des valeurs globales en Asp classique. **Préférer les variables de classes en ASP.Net.**	Application.Lock(); MaClasseStatique.UtilisateursConnectes.Add(User.Identity.Name); Application.UnLock();
Lock()	Méthode permettant au thread courant d'entrer en exclusion mutuelle	
UnLock()	Méthode permettant au thread courant de quitter l'exclusion mutuelle	

Outils Asp.Net - Autres outils

- `Controller.Server` / `HttpUtility`

Méthode	Info	Exemple
HtmlEncode / HtmlDecode	Remplace les caractères spéciaux HTML par leur version encodée / décodée ("<" → "<") Préférer la variante **HttpUtility.HtmlEncode** / **HttpUtility.HtmlDecode**	System.Net.Mail.MailMessage m = new System.Net.Mail.MailMessage(); m.IsBodyHtml = true; var info = " du HTML "; m.Body = $" Info <pre>{HttpUtility.HtmlEncode(info)}</pre>";
UrlEncode / UrlDecode	Remplace les caractères spéciaux des URL par leur version encodée / décodée (" " → "+") Préférer la variante **HttpUtility.UrlEncode** / **HttpUtility.UrlDecode**	ICI

MapPath	Retrouve le chemin disque d'une URL relative. Préférer **HostingEnvironment.MapPath** qui est utilisable en dehors d'un contrôleur.	var binaire = GenererLePDF();
		var path = string.Format("{0}/toto.pdf", Server.MapPath("~/uploads"));
		using(var f = new System.IO.FileStream(path, System.IO.FileMode.Create)) {
		f. Write(binaire, 0, binaire.Length)
		}

- `HttpContext`
  - Hors d'un objet contrôleur, ni `this.Session`, ni `this.Request`, …
  - Utiliser la propriété de classe `HttpContext.Current`

HttpContext.Current se repose sur un TLS.

Un Thread Local Storage est une variable globale (variable de classe), mais stockée dans la pile.

Chaque thread ayant sa propre pile, chaque thread verra une valeur qui lui est propre dans le TLS.

La variable de classe reste visible depuis n'importe quel point du code, comme n'importe quelle variable de classe.

Exercice - Compteurs

- Réaliser une interface affichant
  - Le nombre d'interactions (post back) effectués sur la page par le client
  - Le nombre de visites du navigateur sur la page
  - Le nombre de visiteurs à être venus sur le site
  - Le nombre de visites de l'utilisateur sur le site
- Questions essentielles
  - Où stocker ?
  - Quand incrémenter ?
  - Quand et à quelle valeur initialiser ?

Design MVC

- Modèle = données métier
  - Identifier les types et leur composition
  - Identifier les relations et contraintes métier
  - Choisir un stockage et un moyen d'accès
    - XML
    - Sérialisation binaire
    - BD
      - ADO.Net
      - TableAdapters
      - Linq To Sql
      - Entity Framework
        - Database First
        - Model First
        - Code First

Design MVC

- Contrôleur = accès au modèle
  - Identifier les contrôleurs
    - Un par type du modèle ?
      - ArticleController + CaddieController ?
    - Un par type d'opération ?
      - CommerceController + AideController + SecuriteRessourcesController ?
  - Prévoir les actions
    - Centraliser les opérations ayant un lien entre-elles dans un même contrôleur

Très délicat.

ArticleController : remonter le détail d'un article

CaddieController : choisir une adresse de livraison

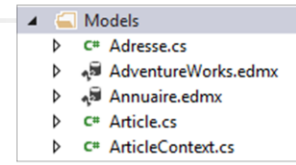
Comment choisir qui ajoute un article au caddie ?

CommerceController va être bien gros…

Design MVC

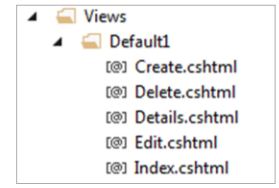
- Vue = affichage d'informations et proposition d'actions
  - 1 contrôleur ⇔ 1 vue serait bien limité !
    - Le contrôleur doit pouvoir choisir la vue à afficher en fonction de l'état du modèle
  - Bonne pratique : Le contrôleur offre un modèle de vue (view model) et non directement son modèle à la vue
    - Adapte les informations à afficher à l'état du modèle / de l'application / des droits
      - Limite les risques de divulgation d'information
      - Lutte contre l'angoisse de la page blanche
  - Notions de…
    - Disposition
    - Vue partielle

Conventions



- Simplifie la configuration
  - Simplifie la reprise d'un projet
  - Un dossier pour les classes du modèle propres au site
    - Namespace automatiquement MonAppliMVC.Models
  - Un dossier pour les contrôleurs
  - Un dossier pour les vues
    - Un sous dossier pour regrouper les vues d'un contrôleur particulier

- Chaque vue affiche le résultat d'une action effectuée par le contrôleur
  - n vues ⇔ n Actions
  - L'action effectuée par un contrôleur est déterminée par l'URL de la requête
    - Les conventions de routage sont importantes



Un contrôleur peut offrir plusieurs vues. OK.

Si si : une vue peut aussi être partagée par plusieurs contrôleurs.

Ça n'est pas courant, mais ça peut être bien pratique !

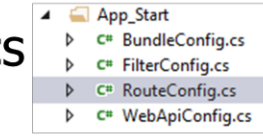
Nous en reparlerons.

Routage ASP.Net MVC

- Les URL dictent les actions

http://localhost:36271/Fiche/Edit/3

- Configuration : /App_Start/RootConfig.cs




```
namespace WebApplication1 {
  public class RouteConfig {
    public static void RegisterRoutes( RouteCollection routes ) {
      routes.IgnoreRoute( "{resource}.axd/{*pathInfo}" );
      routes.MapRoute(                                                    ← Route par défaut
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { action = "Index", id = UrlParameter.Optional }
            );
      routes.MapRoute(                                                    ← Ajout pour supporter une invocation REST
                name: "PersoAMoi",
                url: "CtrlrSpecial/ActionSpecialeREST/{unParam}/{unAutre}",
                defaults: new { controller = "CtrlrSpecial", action = "ActionSpecialeREST" }
            );
    }
  }
}
```

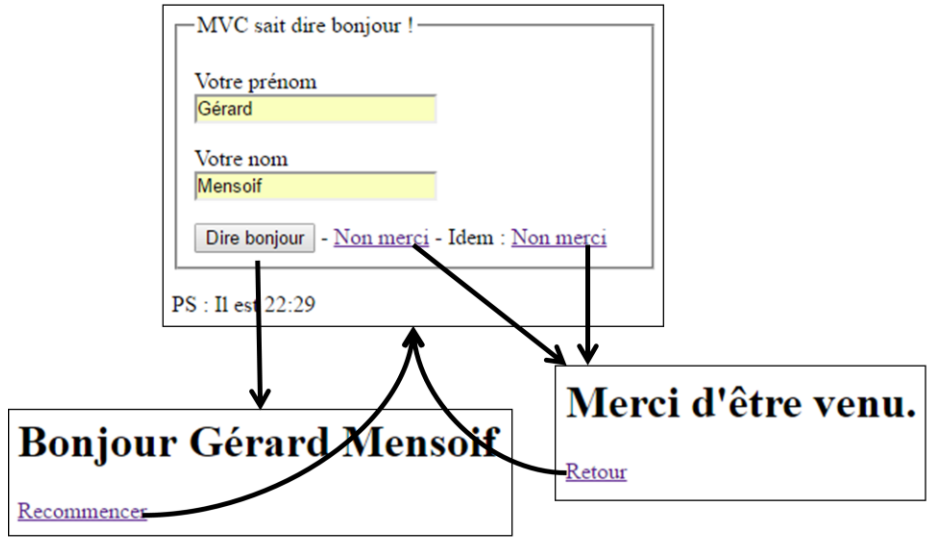

Je doute que vous ayez souvent besoin de modifier le routage par défaut.

Moteur MVC

Étape	Détails
Seulement sur réception de la 1ère requête de l'application	Global.asax.cs déclenche `Application_Start` qui ajoute des objets Route à la `RouteTable` (par l'invocation de `RouteConfig.RegisterRoutes( RouteTable.Routes );` Dans App_Start/RouteConfig.cs
Exécution du routage	Si le routage n'est pas ignoré, le module `UrlRoutingModule` utilise la première route compatible de la RouteTable et crée un objet RouteData. Celui-ci permet de créer un objet `RequestContext` (`HttpContext` + `RouteData`.) Si le routage est activé pour le format de la requête et qu'aucune route n'est compatible, une exception détaillant les routes testées est soulevée.
Instanciation du gestionnaire de requêtes MVC	Le `MvcRouteHandler` crée un `MvcHandler` et lui donne l'objet `RequestContext`.
Instanciation du contrôleur	Le `MvcHandler` utilise l'objet `RequestContext` pour identifier la mise en œuvre de `IControllerFactory` à instancier. Il s'agit habituellement de la classe `DefaultControllerFactory`. La fabrique instancie un contrôleur.
Exécution du contrôleur	Le `MvcHandler` invoque la méthode `Execute` du contrôleur.
Invocation de l'action	Les contrôleurs héritent généralement de `ControllerBase`, le `ControllerActionInvoker` associé au contrôleur détermine alors l'action à invoquer et appelle cette méthode. Le `ControllerActionInvoker` utilise la réflexion pour affecter les paramètres avec les valeurs envoyées par la requête. Les paramètres peuvent être de multiples types simples et/ou des objets complexes (model binding par le `DefaultModelBinder`.)
Production du résultat	La méthode d'action reçoit les saisies de l'utilisateur, effectue les traitements requis, prépare les données à afficher dans la réponse et finalement retourne un résultat de type : `ViewResult` (type le plus utilisé car il permet de réaliser le rendu d'une vue), `RedirectToRouteResult`, `RedirectResult`, `ContentResult`, `JsonResult`, `FileResult`, et `EmptyResult`.

Bonjour MVC

- Modèle, contrôleur et vues



Bonjour MVC

- /Views/Bonjour/AuRevoir.cshtml
  - La vue affichée suite à une action peut n'être qu'un message de confirmation…

```
@{ Layout = null; }
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>Au revoir</title>
</head>
<body>
  <div>
    <h1>Merci d'être venu.</h1>
    <a href="/Bonjour">Retour</a>
  </div>
</body>
</html>
```

Merci d'être venu.

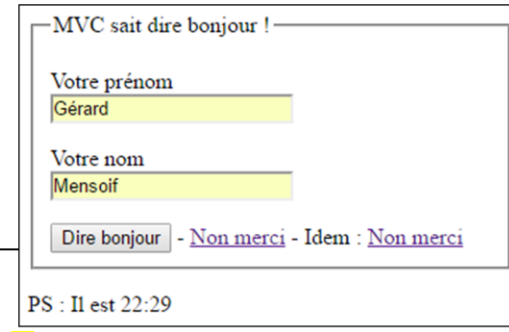
Retour

← Prenez garde à respecter le routage

Des "helpers" nous permettront de respecter les règles de routage. Nous en reparlerons, souvent.

Bonjour MVC

- Views/Bonjour/Index.cshtml
  - Html Helper
    - Effet de bord : génère du HTML
    - Peut retourner un objet




```
@{ Layout = null; }
<!DOCTYPE html><html><head><title>Index</title></head>
<body><div>
  @* Alternative à un <form method="post" action="Bonjour/Bonjour">*@
  @using( Html.BeginForm( "Bonjour1", "Bonjour" ) ) {     ← Action, ctrl, respecte les routes → <form action=…>
    <fieldset>
      <legend>MVC sait dire bonjour !</legend>
      <p>Votre prénom<br /><input type="text" name="prenom"></p>
      <p>Votre nom<br /><input type="text" name="nom"></p>
      <input type="submit" name="BtDireBonjour" value="Dire bonjour" />
      - @Html.ActionLink( "Non merci", "AuRevoir" )
      - Idem : <a href="/Bonjour/AuRevoir">Non merci</a>
    </fieldset>
    <p>PS : Il est @(((DateTime)ViewBag.Maintenant).ToShortTimeString())</p>    ← Le ctrl peut fournir un modèle
  }
</div></body></html>
```

⚠Mais où est le </form> ?

Bonjour MVC

- Les variantes du contrôleur

```
public class BonjourController : Controller {
  public ActionResult Index() {
    ViewBag.Maintenant = DateTime.Now;
    return View();
  }
  // Très looser : dès ASP.Net MVC 1
  public ActionResult Bonjour1() {
    ViewData["Personne"] = new Individu() { Nom = Request.Form["nom"], Prenom = Request.Form["prenom"] };
    return View( "Bonjour1" );
  }
  // Looser : dès ASP.Net MVC 3
  public ActionResult Bonjour2() {
    ViewBag.Personne = new Individu() { Nom = Request.Form["nom"], Prenom = Request.Form["prenom"] };
    return View( "Bonjour2" );
  }
  // Winner MVC 3 : Utiliser un paramètre et un "View Model" typés
  public ActionResult Bonjour( Individu personne ) { // Vive le model binding et le ControlerActionInvoker  !
    return View( "Bonjour", personne ); // personne sera accessible par la vue dans la variable "Model"
  }
  [ActionName( "AuRevoir" )] ← Une action peut être nommée autrement que sa méthode : attribut ActionName
  public ActionResult ByeBye() {
    return View( "AuRevoir" );
  }
}
```

Bonjour MVC

- Les variantes de la vue : Le dictionnaire `ViewData` d'ASP.Net MVC 1
  - String / object
  - ☹ Non fiable
  - ☹ Pas d'intellisense
  - ☺ Dès MVC 1
- Typage possible
  - Préférer le typage dans une variable intermédiaire

Bonjour1.cshtml

```
@{ Layout = null; }
@using WebApplication1.Models @* Pour faire un using... *@
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>Bonjour looser</title>
</head>
<body>
  <div>
    <h1>
      Bonjour
      @(((Individu)ViewData["Personne"]).Prenom)
      @{ var p = (Individu)ViewData["Personne"]; }
      @p.Nom
    </h1>
    @Html.ActionLink( "Recommencer", "Index" )← Helper routage
  </div>
</body>
</html>
```

Bonjour MVC

- Les variantes de la vue : Le `dynamic` `ViewBag` d'ASP.Net MVC 3
  - Type `dynamic`
  - Accès à `ViewData`
  - ☹ Non fiable
  - ☹ Pas d'intellisense
  - ☺ Dès MVC 3
  - ☺ Mise en œuvre rapide
  - ☺ Plusieurs objets OK, avantage par rapport à `Model` (à suivre)

Bonjour2.cshtml

```
@{ Layout = null; }
@using WebApplication1.Models
<!DOCTYPE html> <html><head>…</head>
<body>
 <div>
  <h1>
    Bonjour
    @ViewBag.Personne.Prenom    ← dynamic → sans intellisense
    @{ var personne = (Individu)ViewBag.Personne; }    ← Typer
    @personne.Nom ← Variable de type Personne → intellisense
  </h1>
  @Html.ActionLink( "Recommencer", "Index" )
 </div>
</body>
</html>
```

Bonjour MVC

- Démystifier ViewBag

```
class Program {
  static dynamic MonSac = new EtendableInsensibleALaCasse();        ← Un dynamic est utilisé via la réflexion
  static void Main( string[] args ) {
    MonSac.TRUC = "Oh... J'ai un gros truc dans mon sac !";    ← Une écriture recherche une propriété publique
    Console.WriteLine( MonSac.truc );                          ← Une lecture  recherche une propriété publique
  }                                                            ← Le type réel est testé par réflexion
}
public class EtendableInsensibleALaCasse : System.Dynamic.DynamicObject {  ← Si c'est DynamicObjet...
  public Dictionary<string, object> DicoDObjets;
  public EtendableInsensibleALaCasse() {
    DicoDObjets = new Dictionary<string, object>();
  }
  public override bool TryGetMember( GetMemberBinder binder, out object resultat ) {
    object valeur;                                                  ← Ces méthodes sont utilisées
    if( DicoDObjets.TryGetValue( binder.Name.ToLower(), out valeur ) ) { ← à la place de Type.GetProperty
      resultat = valeur;                                            ← et de Type.GetField
      return true;
    }
    resultat = null;
    return false;
  }
  public override bool TrySetMember( SetMemberBinder binder, object valeur ) { ← Ces méthodes sont utilisées
    DicoDObjets[binder.Name.ToLower()] = valeur;                    ← à la place de
    return true;                                                    ← Type.GetProperty
  }                                                                 ← et de Type.GetField
}
```

Bonjour MVC

- Les variantes de la vue : Le `dynamic` `Model` MVC 3
  - Type `dynamic`
  - ☹ Non fiable…
    - ☺ Mais souple
  - ☹ Pas d'intellisense
  - ☺ Dès MVC 3
  - ☺ Mise en œuvre rapide
  - ☺ `@model` pour typer
    - ☹ Mais un seul `Model`

Bonjour.cshtml

```
@{ Layout = null; }
@using WebApplication1.Models
@{var personne = (Individu)Model; // Model est de type dynamic
}
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>DireBonjour</title>
</head>
<body>
  <div>
    <h1>
      Bonjour
      @Model.Prenom @* dynamic => sans intellisense *@
      @personne.Nom @* Type Personne => avec intellisense *@
    </h1>
    @Html.ActionLink( "Recommencer", "Index" )
  </div>
</body>
</html>
```

@model Individu

Votre modèle est typé ?
Utiliser @model Individu
La variable Model n'est plus dynamic mais typée !

Vues et espace de noms

- Le code Razor est injecté dans une méthode de rendu
  - Directive `@using`

```
@using WebApplication1.Models
```

  - Idem : la directive `@functions`

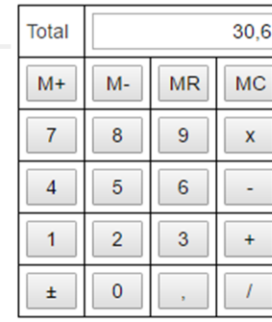
```
@functions {
  int Incrémenter( int n ) { return ++n; }
}
```

  - ~/**Views**/Web.config pour les "`using`" communs

```
<configuration>
  <system.web.webPages.razor>
    <pages pageBaseType="System.Web.Mvc.WebViewPage">
      <namespaces>
        <add namespace="System.Web.Mvc" />
        [ . . . ]
        <add namespace="WebApplication1" />
      </namespaces>
    </pages>
  </system.web.webPages.razor>
[. . .]
```


/!\ Chaque zone ("area", nous en reparlerons) a ses propres using par défaut, dans ~/Areas/NomDeLaZone/Views/Web.config

Exercice



- Une calculatrice ringarde
  - Au choix du développeur, l'opérande est saisie dans une zone de texte ou avec des touches
  - Un bouton par opération :
    - Addition, soustraction, …, Mémoire+, Mémoire- et Retour mémoire.
    - La valeur saisie est "opérateurée" au total
    - Le total est affiché dans la zone où l'opérande a été saisie
- Réaliser un contrôleur de calcul
  - Une seule action, qui testera l'opération à effectuer
- Une seule vue car une seule présentation du résultat
- Mémoire, affichage, opérateur, résultat…
  - Faites le bon choix : communiquer avec plusieurs valeurs ou avec un modèle typé ?

Prévoir un peu de JavaScript pour la gestion des touches numériques.
Si vous ne voulez pas de JavaScript utilisez une saisie classique dans la zone de texte.

Les écueils des bases d'Asp.Net MVC

- Une application simple
  - Un modèle simple : articles
  - Un stockage simple : dico de classe
  - Un contrôleur simple
    - Lister le stock et le caddie
    - Ajouter au caddie
    - Vider le caddie
  - Une vue simple unique :
    - Afficher le stock + ajouter au caddie
    - Afficher le caddie
- Problèmes de navigation entre les actions

Modèle

- Modèle + accès

```
public class Article {
  public Guid Id { get; set; }
  public Decimal PrixAchat { get; set; }
  public Decimal PrixVente { get; set; }
  public int Quantite { get; set; }
  public string Nom { get; set; }
  public Article() {
    Id = Guid.NewGuid();
  }
  public static readonly Dictionary<Guid, Article> Stock;
  public static Dictionary<Guid, Article> CaddieUtilisateur {
    get {
      var caddie = (Dictionary<Guid, Article>)HttpContext.Current.Session["CaddieUtilisateur"];
      if( caddie == null )
        HttpContext.Current.Session["CaddieUtilisateur"] = caddie = new Dictionary<Guid, Article>();
      return caddie;
    }
  }
  static Article() {                                          ← Constructeur DE CLASSE
    Stock = new Dictionary<Guid, Article>();
    Article a;
    a = new Article() { Nom = "Kharkhorin", PrixAchat = 2.5m, PrixVente = 5m, Quantite = 12345 };
    Stock.Add( a.Id, a );
    // . . .
  }
}
```

Contrôleur

- Simple… un peu trop

```
public class CommerceSimpleController : Controller {
  public ActionResult Index() {
    ViewBag.Caddie = Article.CaddieUtilisateur;
    ViewBag.Stock = Article.Stock;
    return View();
  }
  public ActionResult Ajouter( Guid id ) {
    Article a;
    if( Article.CaddieUtilisateur.TryGetValue( id, out a ) ) {
      a.Quantite++;
    }
    else {
      var articleEnStock = Article.Stock[id];
      a = new Article() { Id = articleEnStock.Id, Nom = articleEnStock.Nom, . . ., Quantite = 1 };
      Article.CaddieUtilisateur.Add( a.Id, a );
    }
    ViewBag.ArticleAjoute = a;                    ← La vue doit afficher une confirmation
    return RedirectToAction( "Index" );           ← Je ne veux pas fournir encore Caddie et Stock au ViewBag
  }
  public ActionResult Vider() {
    Article.CaddieUtilisateur.Clear();
    return RedirectToAction( "Index" );           ← Je ne veux pas fournir encore Caddie et Stock au ViewBag
  }
}
```

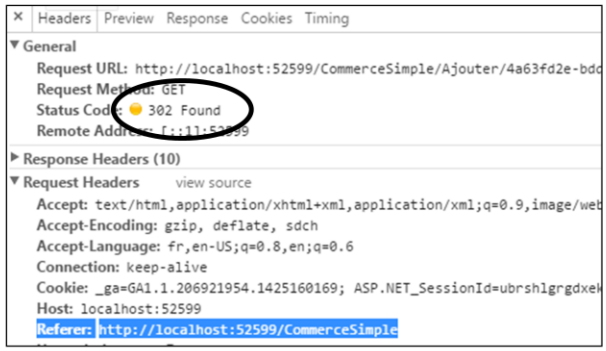
Vue

- Simple... un peu trop

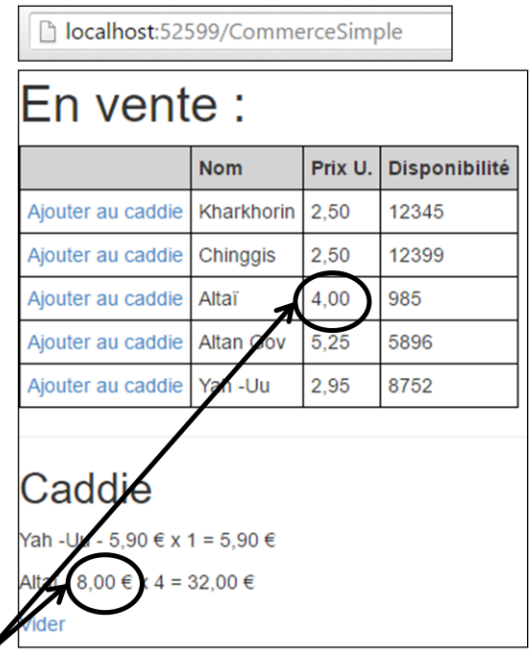
```
@{Layout = null; var caddie = (Dictionary<Guid, Article>)ViewBag.Caddie;
  var stock = (Dictionary <Guid, Article>)ViewBag.Stock; var aa = (Article)ViewBag.ArticleAjoute; }
[ . . .]
<body> <div class="body-content"> <h1>En vente :</h1>
    <table class="std">
      <tr> <th> </th> <th>Nom</th> <th>Prix U.</th> <th>Disponibilité</th> </tr>
      @foreach( var a in stock.Values ) {
        <tr>
          <td>
            @Html.ActionLink( "Ajouter au caddie", "Ajouter", new { id = a.Id } )
          </td> <td>@a.Nom</td> <td>@a.PrixAchat.ToString( "0.00" )</td> <td>@a.Quantite
        </tr>
      }
    </table>
    <hr /> <h2>Caddie</h2>
      @foreach( var a in caddie.Values ) {
        <p>@($"{a.Nom} - {a.PrixVente:0.00} € x {a.Quantite} = {a.PrixVente * a.Quantite:0.00} €")</p>
      }
    @Html.ActionLink( "Vider", "Vider" )
  </div>
  @if( aa != null ) {
    <script type="text/javascript">
      alert('@( Html.Raw( $"Dans le caddie : {aa.Quantite} x {aa.Nom}".Replace( "'", "\\'" ) ) )');
    </script>
  }
</body>
</html>
```

Écran

- Rien ne vous choque ?
  - `RedirectToAction` à *bien* empêché que l'URL ne reste à "/Ajouter/unID", mais...



  - ☹ Pas d'alerte sur ajout :
    - ☹ `RedirectToAction` fait un `Response.Redirect`
  - Et bug d'affichage : le prix d'achat



Le dictionnaire TempData permet de partager des données entre le contrôleur d'origine et le contrôleur vers lequel on redirige.

Attention : TempData utilise la Session !

Bonnes pratiques

- ☹ ActionLink crée un lien ➔ renseigne l'url
  - L'action peut être reproduite à votre insu
  - Un HTTP GET ne doit **JAMAIS** modifier l'état de l'application
    - Utiliser un formulaire
    - Décorer la méthode d'action `[HttpPost]`
    - ☹ Ne permet pas de poser un marque-page
    - ☹ `RedirectToAction` permet de nettoyer l'URL mais "redirige"
  - Éviter de doubler les requêtes HTTP
    - À la façon d'un `Server.Transfer` des WebForms
      - ☺ Performances
      - ☺ Permet d'enrichir le `ViewBag`
      - ☹ L'URL reste renseignée avec le nom de l'action ➔ il faut ruser avec des annulables
- ☹ L'accès direct au modèle par la vue ouvre la porte à la divulgation d'informations

Parfois l'action Index doit déclencher des méthodes différentes mais de même signature sur POST et sur GET.

Utiliser alors des méthodes de nom différents mais en décorer une à l'aide de `[ActionName( "Index" )]` permet d'éviter de tester le protocole explicitement.

Vue

- Moins simple

```
[…]   @foreach( var a in stock.Values ) {
        <tr>
          <td>
            @*@Html.ActionLink( "Ajouter au caddie", "Ajouter", new { id = a.Id } )*@
            @using( Html.BeginForm("Ajouter", "CommerceSimple") ) {                    ← Action, Contrôleur
              <input type="hidden" name="id" value="@a.Id" />                  ← Paramètre dans les en-têtes
              <input type="submit" value="+" />
            }
          </td>
          <td>@a.Nom</td> <td>@a.PrixAchat.ToString( "0.00" )</td> <td>@a.Quantite</td> ← Divulgation d'info
        </tr>                                                                          ← toujours possible
      }
    </table> <hr /> <h2>Caddie</h2>
      @foreach( var a in caddie.Values ) {
        <p>@($"{a.Nom} - {a.PrixVente:0.00} € x {a.Quantite} = {a.PrixVente * a.Quantite:0.00} €")</p>
      }
    @*@Html.ActionLink( "Vider", "Vider" )*@
    @using( Html.BeginForm( "Vider", "CommerceSimple" ) ) {                            ← Action, Contrôleur
      <input type="submit" value="Vider" />                                ← Paramètre dans les en-têtes
    }
  </div>
  @if( aa != null ) {
    <script type="text/javascript">
      alert('@( Html.Raw( $"Dans le caddie : {aa.Quantite} x {aa.Nom}".Replace( "'", "\\'" ) ) )');
    </script>
  }
[…]
```

Contrôleur

- Moins simple

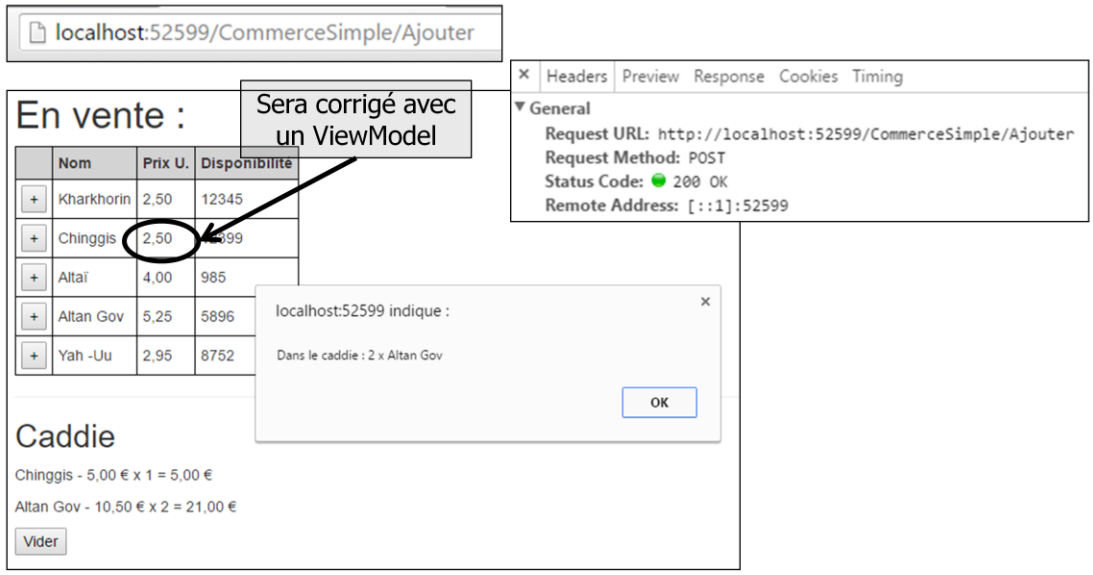
```
public ActionResult Index() {
  ViewBag.Caddie = Article.CaddieUtilisateur; ViewBag.Stock = Article.Stock;
  return View( "Index" );      ← Remplace return View() car utilise la requête pour trouver la vue par défaut
}
//[HttpPost]       ← Restriction au verbe POST, interdit l'utilisation du Html.ActionLink ou d'un marque-page
//public ActionResult Ajouter( Guid id ) {        ← L'id n'est plus fourni sur l'utilisation d'un marque-page
public ActionResult Ajouter( Guid? id ) {         ← Un annulable corrige le problème
  if( Request.HttpMethod != "POST" )     ← Si le get est permis, on peut ainsi éviter l'exécution de l'action
    return RedirectToAction( "Index" );  ← Double requête HTTP, oui, mais pas bien souvent
  Article a;
  if( Article.CaddieUtilisateur.TryGetValue( /* id */ id.Value, out a ) )
    a.Quantite++;
  else {
    var articleEnStock = Article.Stock[/*id*/ id.Value];
    a = new Article() { Id = articleEnStock.Id, Nom = articleEnStock.Nom, [. . .], Quantite = 1 };
    Article.CaddieUtilisateur.Add( a.Id, a );
  }
  ViewBag.ArticleAjoute = a; ← On garde le même ViewBag car simple exécution de méthode Index(),
  return Index();            ← on reste sur la même instance, contrairement à RedirectToAction( "Index" )
}
//[HttpPost] ← Pour accepter seulement un post, interdit l'utilisation du Html.ActionLink ou d'un marque-page
public ActionResult Vider() {
  if( Request.HttpMethod != "POST" )                                    ← Action ssi POST, comme ci-dessus
    return RedirectToAction( "Index" );
  Article.CaddieUtilisateur.Clear();
  return Index(); // Simple exécution de méthode, remplace return RedirectToAction( "Index" );
}
```

Contrôleur

- Alternative

```
[HttpGet]                                          ← Le verbe GET est autorisé => on supporte le marque page
public ActionResult Ajouter() {                    ← L'id n'est alors pas fourni
  return RedirectToAction( "Index" );              ← Et on redirige, mais pas bien souvent
}
[HttpPost]                                ← La méthode avec paramètre obligatoire n'est accessible que sur POST
public ActionResult Ajouter( Guid id ) {
Article a;
  if( Article.CaddieUtilisateur.TryGetValue( id, out a ) ) {
    a.Quantite++;
  }
  else {
    var articleEnStock = Article.Stock[id];
    a = new Article() { Id = articleEnStock.Id, Nom = articleEnStock.Nom,
                  PrixAchat = articleEnStock.PrixAchat, PrixVente = articleEnStock.PrixVente, Quantite = 1 };
    Article.CaddieUtilisateur.Add( a.Id, a );
  }
  ViewBag.ArticleAjoute = a;
  return Index();
}
[HttpPost]    ← Si on veut n'exécuter que sur un post, interdit Html.ActionLink (tant mieux !) ou marque-page
public ActionResult Vider () {
  Article.CaddieUtilisateur.Clear();  return Index();              ← Simple exécution de méthode : optimisé
}
[HttpGet, ActionName( "Vider" )]    ← Get possible et on redirige (rarement !) on supporte les marque-pages !
public ActionResult ViderGet() {
  return RedirectToAction( "Index" );
}
```

Écran


localhost:52599/CommerceSimple/Ajouter
Sera corrigé avec un ViewModel
Headers Preview Response Cookies Timing
General
Request URL: http://localhost:52599/CommerceSimple/Ajouter
Request Method: POST
Status Code: 200 OK
Remote Address: [::1]:52599
En vente :
Nom Prix U. Disponibilité
Kharkhorin 2,50 12345
Chinggis 2,50
Altaï 4,00 985
Altan Gov 5,25 5896
Yah -Uu 2,95 8752
localhost:52599 indique :
Dans le caddie : 2 x Altan Gov
OK
Caddie
Chinggis - 5,00 € x 1 = 5,00 €
Altan Gov - 10,50 € x 2 = 21,00 €
Vider

Retours des actions

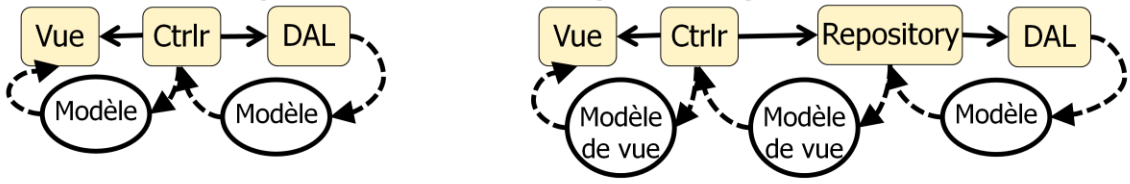
Retour	Détails
string	La chaine brute sera envoyée au client. Vous pouvez *jouer* avec Response.ContentType. Response.ContentType = "application/json"; return "{chaine:'toto',nombre:1}";
View()	Crée un objet ViewResult qui restitue une vue dans la réponse. Le nom de la vue et l'objet modèle peuvent être passés en argument.
Redirect(url) / RedirectPermanent(url)	Envoie un code 302 ou 301 au navigateur. Tous les "Redirect…" offrent les deux options. Sur le code 301, la redirection est mise en cache par le navigateur.
RedirectToAction()	Envoie un code 302 et l'url de l'action est calculée à partir du routage. Paramètres : action, contrôleur, un objet pour les paramètres
RedirectToRoute()	Permet de préciser le nom d'une route à utiliser ou les paramètres de routage
PartialView()	Crée une vue partielle (nous en reparlerons plus tard)
JavaScript()	Retourne un format JavaScript, permet de générer dynamiquement les scripts à inclure.
Json(objet)	Retourne la version sérialisée en JSON de l'objet. Permet de faire de l'AJAX. Nous en reparlerons plus tard.
File(octets, contentType, nom)	Permet au client de télécharger un fichier binaire. Nous en reparlerons plus tard.

Modèles, entrepôts et dépendances

- Modèles
  - Données
  - Logique métier → Data Annotations
  - Libellés → Data Annotations
- Accès au modèle
  - Par le contrôleur ?
    - ☺ Mise en œuvre simple & rapide
    - ☹ Risque de réécrire un même accès dans 2 contrôleurs
    - ☹ Le contrôleur devient dépendant du stockage
  - Par un entrepôt (repository), avec injection de dépendance ?
    - ☺ Permet de centraliser & réutiliser
    - ☺ Découple la mise en œuvre du stockage
    - ☺ L'injection de dépendance ne permet pas à la vue d'accéder à la couche de données
    - ☹ Mise en œuvre moins rapide & moins simple

Modèles et entrepôts

- MVC classique / Repository

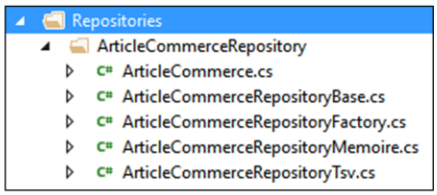


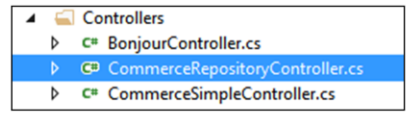
- Repository = Indirection supplémentaire
  - Contrôleur indépendant du stockage
  - Accès seulement à un modèle de vue
  - N'empêche pas la vue de connaître le modèle issu de la DAL
- Repository + Dependency Injection
  - Envoyer une dépendance à l'objet dépendant au lieu de permettre à l'objet dépendant d'accéder à sa dépendance
  - Utilise des interfaces, le fichier de configuration et la réflexion

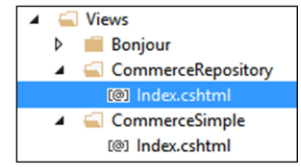
DAL = Data Acces Layer, Couche d'Accès aux Données

Exemple de d'entrepôt

- Architecture
  - Un View Model réduit Article aux propriétés utiles au contrôleur de commerce
    - Un autre pourra être écrit pour le contrôleur de gestion de magasin
  - Une interface / classe abstraite décrit les opérations réalisables par le repository
  - Plusieurs mises en œuvre sont possibles
  - Une factory permet d'instancier le repository voulu
    - Modification web.config suffit pour choisir le repository







Exemple de d'entrepôt

- View model

```
namespace WebApplication1.Repositories.ArticleCommerceRepository {
  public class ArticleCommerce {
    public Guid Id { get; }
    public Decimal PrixVente { get; }
    public int Quantite { get; set; }
    public string Nom { get; }
    public ArticleCommerce( Guid id, Decimal prixVente, int quantite, string nom ) {
      Id = id;
      PrixVente = prixVente;
      Quantite = quantite;
      Nom = nom;
    }
  }
}
```

- Article peut éventuellement devenir `internal`
  - Ssi le repository est dans l'appli web, la vue n'a alors plus accès à Article

```
public internal class Article {
```


Les vues n'ont pas accès aux types "internal" du projet ?

Les vues sont a priori compilée dans un autre assembly que celui des contrôleurs.

C'est grâce à cela qu'on peut utiliser une vue modifiée sans avoir à recompiler l'application Web.

Exemple de d'entrepôt

- Repository de base pour commercer

```
namespace WebApplication1.Repositories.ArticleCommerceRepository {
  public abstract class ArticleCommerceRepositoryBase : IDisposable {
    public abstract IList<ArticleCommerce> ListerStock();
    public abstract IList<ArticleCommerce> ListerCaddie();
    public abstract ArticleCommerce AjouterAuCaddie( Guid articleCommerceId );
    public abstract void ViderCaddie();

    // Pour les codeurs "de bonne disposition" :
    private bool Disposed = false;
    protected virtual void Dispose( bool jeterLesObjetsGeresEgalement ) {
      if( Disposed )
        return;
      if( jeterLesObjetsGeresEgalement ) {
        // Cas TRÈS particulier : rien à jeter dans ce repository de base !
      }
      // Libération des non-gérés : Rien
      Disposed = true;
    }
    public void Dispose() { // Mise en œuvre de IDisposable
      Dispose( true );
      GC.SuppressFinalize( this ); // Les types dérivés n'auront qu'à overrider Dispose(bool)
    }
  }
}
```

Exemple de d'entrepôt

- Repository stockage en mémoire

```
public class ArticleCommerceRepositoryMemoire : ArticleCommerceRepositoryBase {
  private static readonly Dictionary<Guid, Article> Stock = Article.Stock;
  private static Dictionary<Guid, ArticleCommerce> Caddie {
    get {
      var caddie = (Dictionary<Guid, ArticleCommerce>)HttpContext.Current.Session["Caddie"];
      if(caddie==null) HttpContext.Current.Session["Caddie"] =caddie =new Dictionary<Guid,ArticleCommerce>();
      return caddie;
    }
  }
  public override ArticleCommerce AjouterAuCaddie( Guid articleCommerceId ) {
    ArticleCommerce ac;
    if( Caddie.TryGetValue( articleCommerceId, out ac ) )
      ac.Quantite++;
    else {
      var articleEnStock = Article.Stock[articleCommerceId];
      ac = new ArticleCommerce( articleEnStock.Id, articleEnStock.PrixVente, 1, articleEnStock.Nom );
      Caddie.Add( ac.Id, ac );
    }
    return ac;
  }
  public override void ViderCaddie() { Caddie.Clear(); }
  public override IList<ArticleCommerce> ListerCaddie() { return Caddie.Values.ToList(); }
  public override IList<ArticleCommerce> ListerStock() {
    lock( Stock ) {
      return Stock.Values.Select( a => new ArticleCommerce( a.Id, a.PrixAchat, a.Quantite, a.Nom ) ).ToList();
    }
  }
}
```

Exemple de d'entrepôt

- Une fabrique pour choisir le repository à employer

```
public class ArticleCommerceRepositoryFactory {
  private static System.Reflection.ConstructorInfo ArticleCommerceRepositoryCtor;
  public static ArticleCommerceRepositoryBase Instancier() {
    if( ArticleCommerceRepositoryCtor == null ) {
      string nomDuType = ConfigurationManager.AppSettings["ArticleCommerceRepository"];
      ArticleCommerceRepositoryCtor =
        AppDomain.CurrentDomain
        .GetAssemblies()
        .SelectMany( asm => asm.GetTypes() )
        .Where( t => t.FullName == nomDuType )
        .Select( t => t.GetConstructor( new Type[0] ) )
        .First();
      }
    return (ArticleCommerceRepositoryBase)ArticleCommerceRepositoryCtor.Invoke( null );
  }
}
```

  - Configuration

```
<appSettings>
  <!--<add key="ArticleCommerceRepository"
       value="WebApplication1.Repositories.ArticleCommerceRepository.ArticleCommerceRepositoryMemoire" />-->
  <add key="ArticleCommerceRepository"
       value="WebApplication1.Repositories.ArticleCommerceRepository.ArticleCommerceRepositoryTsv" />
</appSettings>
```

Exemple de d'entrepôt

- Repository alternatif - début

```
public class ArticleCommerceRepositoryTsv : ArticleCommerceRepositoryBase {

 static string FichierCaddie => $"{HttpRuntime.AppDomainAppPath}Repositories/ArticleCommerceRepository/Caddie_
{HttpContext.Current.Session.SessionID}.tsv";

 static string FichierStock=$"{HttpRuntime.AppDomainAppPath}Repositories/ArticleCommerceRepository/Stock.tsv";

 public override void ViderCaddie() {
   if( File.Exists( FichierCaddie ) )
     File.Delete( FichierCaddie );
 }
 private static List<ArticleCommerce> ListerDesArticles( string fichier ) {
   if( !File.Exists( fichier ) )
     return new List<ArticleCommerce>();
   using( var r = new StreamReader( fichier ) ) {
     return r.ReadToEnd().Split( '\n' ).Where( s => !string.IsNullOrWhiteSpace( s ) ).Select( s => s.Split(
'\t' ) ).Select( t => new ArticleCommerce( Guid.Parse( t[0] ), decimal.Parse( t[1] ), int.Parse( t[2] ), t[3]
) ).ToList();
   }
 }
 public override IList<ArticleCommerce> ListerStock() => ListerDesArticles( FichierStock );

 public override IList<ArticleCommerce> ListerCaddie() => ListerDesArticles( FichierCaddie );

// =>
```

Exemple de d'entrepôt

- Repository alternatif – suite et fin

```
public override ArticleCommerce AjouterAuCaddie( Guid articleCommerceId ) {
  var caddie = ListerCaddie();
  ArticleCommerce ac = caddie.FirstOrDefault( a => a.Id == articleCommerceId );
  if( ac != null ) {
    ac.Quantite++;
  }
  else {
    var articleEnStock = ListerStock().First( a => a.Id == articleCommerceId );
    ac = new ArticleCommerce( articleEnStock.Id, articleEnStock.PrixVente, 1, articleEnStock.Nom );
    caddie.Add( ac );
  }
  using( var writer = new StreamWriter( FichierCaddie ) ) {
    foreach( var a in caddie )
      writer.Write( $"{a.Id}\t{a.PrixVente}\t{a.Quantite}\t{a.Nom}\n" );
  }
  return ac;
 }
}
```

Exemple de d'entrepôt

- Contrôleur

```
public class CommerceRepositoryController : Controller {
  private ArticleCommerceRepositoryBase _Repository;                    ← Pattern classique
  private ArticleCommerceRepositoryBase Repository {
    get {                                                  ← Instanciation en cas de besoin
      if( _Repository == null ) _Repository = ArticleCommerceRepositoryFactory.Instancier();
      return _Repository;
    }
  }
  public ActionResult Index() {
    ViewBag.Caddie = Repository.ListerCaddie(); ViewBag.Stock = Repository.ListerStock();
    return View( "Index" );
  }
  public ActionResult Ajouter( Guid? id ) {
    if( Request.HttpMethod != "POST" ) return RedirectToAction( "Index" );
    ArticleCommerce a = Repository.AjouterAuCaddie( id.Value );
    ViewBag.ArticleAjoute = a;
    return Index();
  }
  public ActionResult Vider() {
    if( Request.HttpMethod != "POST" ) return RedirectToAction( "Index" );
    Repository.ViderCaddie();
    return Index(); }
  protected override void Dispose( bool disposing ) {          ← Dispose en cas de besoin
    if( disposing && _Repository != null )
      _Repository.Dispose();
    base.Dispose( disposing );
  }
}
```

Exemple de d'entrepôt

- Vue

```
@using WebApplication1.Repositories.ArticleCommerceRepository
@{
  Layout = null;
  var caddie = (IList<ArticleCommerce>)ViewBag.Caddie;
  var stock = (IList<ArticleCommerce>)ViewBag.Stock;
  var aa = (ArticleCommerce)ViewBag.ArticleAjoute;
}

@* La suite ? Inchangée !... Ou presque : on ne peut heureusement plus afficher le PrixAchat  *@
```

View Model & Over posting

- Over posting, définition
  - Envoyer à l'action des données non-éditées
- View Model
  - ☺La vue ne pas peut afficher les données sensibles
  - ☺L'action ne peut pas subir l'over posting
  - ☹Il faut écrire une classe view model
    - ☹Pour les données à afficher
    - ☹Parfois une seconde pour les données postées vers l'action
- Alternative : attribut anti-over posting

```
public ActionResult ModifierCaddie( [Bind( Include = "Id, Quantite" )] Article article ) { … }
```

  - ☺Pour éviter d'écrire une classe spécifique au post

CSRF hack

- Kéchécha ?
  - Cross Site Request Forgery
  - L'utilisateur du site ciblé dispose d'un cookie d'identification
  - Il visite un site non-fiable qui le fait poster vers une action sensible du site ciblé
  - L'action est autorisée car le cookie en authentifie l'auteur

```
<form id="blackHat" method="post" action="http://www.pauvresite.com/CompteReglement/Edit/78958847521">
  <input type="hidden" name="id" value="78958847521" />
  <input type="hidden" name="rib" value="30002 00550 0000157845Z 02" />
</form>
<script type="text/javascript">
  getElementById("blackHat").submit();
</script>
```

CSRF hack

- AntiForgeryToken
  - Le site se protège en ajoutant un champ caché et un cookie au formulaire
  - Impossible de soumettre le formulaire sans passer au préalable par son affichage

```
@using( Html.BeginForm() ) {
  @Html.AntiForgeryToken()
  . . .
}
```

```
<input name="__RequestVerificationToken" type="hidden" value="zaTFdpkKN0BYa . . . FVmelvzwRZpAME" /> + COOKIE
```

```
public class CompteReglementController : Controller {
  [HttpPost]
  [Authorize( Roles = "Admins" )]
  [ValidateAntiForgeryToken()]
  public ActionResult Edit( string id, string rib ) {
    . . .
  }
```

Éditer plusieurs objets

- ☺ Le model binding supporte !
  - L'action reçoit une liste dans son paramètre "lesTrucs"
  - Utiliser des champs de nommés lesTrucs[i].Champ
  - Optionnel : La répétition d'un champ nommé lesTrucs.Index permettra de changer l'emplacement des objets créés par model binding dans la collection

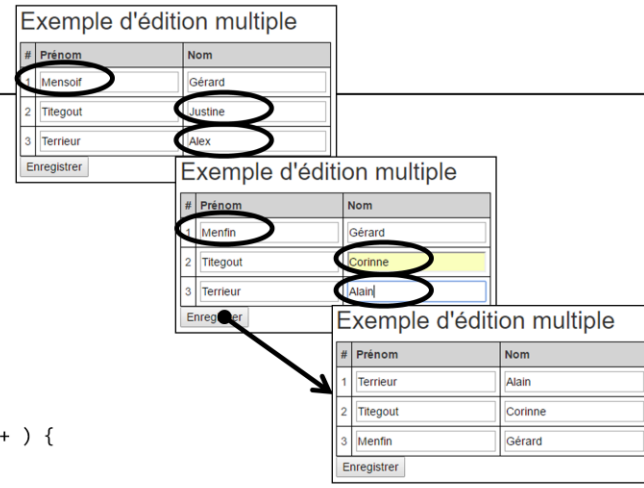
Éditer plusieurs objets

- Exemple - Contrôleur

```
public class ExempleEditionMultipleController : Controller {
  private static List<Individu> MesGens = new List<Individu>() {
                                          new Individu { Prenom = "Gérard", Nom = "Mensoif" },
                                          new Individu { Prenom = "Justine", Nom = "Titegout" },
                                          new Individu { Prenom = "Alex", Nom = "Terrieur" } };
  public ActionResult Index() {
    lock( MesGens ) {
      return View( MesGens.ToArray() );
    }
  }
  [HttpPost]
  public ActionResult Index( Individu[] edition ) {
    lock( MesGens ) {
      for( int i = 0, len = edition.Length; i < len; i++ ) {
        MesGens[i].Prenom = edition[i].Prenom;
        MesGens[i].Nom = edition[i].Nom;
      }
      return View( MesGens.ToArray() );
    }
  }
}
```

Éditer plusieurs objets

- Exemple - Vue




```
@using WebApplication1.Models;
@model Individu[]
@{ ViewBag.Title = "Exemple d'édition multiples"; }

<h2> @ViewBag.Title </h2>

@using( Html.BeginForm() ) {
  <table class="std">
    <tr>
      <th>#</th>
      <th>Prénom</th>
      <th>Nom</th>
    </tr>
    @for( int i = 0, len = Model.Length; i < len; i++ ) {
      <tr>
        <td>
          @(i + 1)
          <input type="hidden" name="edition.Index" value="@(len - i - 1)" /> ← Pour de rire, je retourne tout !
        </td>
        <td><input type="text" name="edition[@i].Nom" value="@Model[i].Nom" /></td>
        <td><input type="text" name="edition[@i].Prenom" value="@Model[i].Prenom" /></td>
      </tr>
    }
  </table>
  <input type="submit" value="Enregistrer" />
}
```

Exercice

- Une messagerie
  - Un utilisateur doit fournir un pseudo avant d'y accéder
    - Il voit alors la liste des utilisateurs qui se sont connectés
  - Il peut en suite
    - Envoyer un message à un autre utilisateur, en cliquant sur son nom
    - Consulter ses propres messages, en cliquant sur son propre nom
- Réalisation
  - Modèles
    - Message : émetteur + date + texte
    - Un dictionnaire en mémoire : nom / liste de messages
      - Le couple reste tant que l'utilisateur est connecté
      - Comment gérer la déconnexion d'un utilisateur ?
  - Contrôleur : un seul, une action par… action
  - Vues : login, liste des utilisateurs connectés, mes messages, envoyer un message
- En cas de crise de courage, réaliser un repository et un stockage alternatif

Exercice


Login messagerie
localhost:52599/Messagerie
Entrez votre pseudo
Pseudo
rikou
Aller causer
Messagerie
localhost:52599/Messagerie/Loc
Messagerie
Mes messages;
Cliquez sur un utilisateur pour lui envoyer un message.
rikou (pour test...)
Écrire
localhost:52599/Messagerie/Ecrire?destinataire=rikou
Message: Coucou
Envoyer
Messagerie
localhost:52599/Messagerie/Loc
Messagerie
es messages;
Cliquez sur un utilisateur pour lui envoyer un message.
rikou (pour test...)
Mes messages
localhost:52599/Messagerie/Envoyer
Info
Message envoyé
Mes messages
Émission
Émetteur
Message
11/04/2016 23:38
SYSTÈME
Bienvenue sur la messagerie
11/04/2016 23:43
rikou
Coucou
etour
Mes messages
localhost:52599/Messagerie/Consulter
Mes messages
Émission
Émetteur
Message
11/04/2016 23:38
SYSTÈME
Bienvenue sur la messagerie
11/04/2016 23:43
rikou
Coucou
Retour

Filtres d'action

- Attributs des actions ou contrôleurs
  - Influencent les exécutions d'action
  - Quelques filtres existants

Filtre	Utilisation
OutputCache	Permet de conserver en cache le résultat d'une action (nombreux paramètres)
HandleError	Vous permet de gérer les erreurs
Authorize	Vous permet de préciser les conditions d'accès

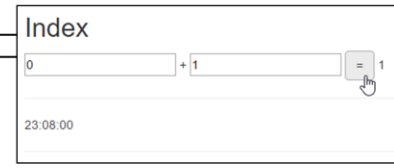
  - Types de filtre

Type	Interface	Attribut
Authorisation	IAuthorizationFilter, exécuté avant l'action et avant les autres filtres	AuthorizeAttribute
Action	IActionFilter, exécuté avant et après l'action	ActionFilterAttribute
Exception	IExceptionFilter, exécuté en cas d'exception dans l'action ou un autre filtre	HandleErrorAttribute
Result	IResultFilter, exécuté avant et après l'action	ResultFilterAttribute

Filtres d'action

- Exemple d'utilisation

```
[OutputCache( Duration = 100, Location = System.Web.UI.OutputCacheLocation.Server )]
public ActionResult Index( int? a, int? b ) {
  ViewBag.A = a;
  ViewBag.B = b;
  return View( (a ?? 0) + (b ?? 0) );
}
```




```
@model int
@{ViewBag.Title = "Index";
  int a = ViewBag.A ?? 0;
  int b = ViewBag.B ?? 0;}
<h2>@ViewBag.Title</h2>
@using( (Html.BeginForm()) ) {
  <div>
    <input type="number" name="a" value="@a" /> + <input type="number" name="b" value="@b" />
    <input type="submit" value="=" class="btn btn-default" />
    <span>@Model</span>
    <hr />
    @DateTime.Now.ToString( "T" )
  </div>
}
```

Filtres d'action

- Filtre perso : horaires d'accès

```
namespace WebApplication1.ActionFilters {
  public class HorairesDAccesAttribute : FilterAttribute, IAuthorizationFilter {
    public TimeSpan Debut { get; }
    public TimeSpan Fin { get; }
    public HorairesDAccesAttribute( string debut, string fin ) {
      Debut = TimeSpan.Parse( debut );
      Fin = TimeSpan.Parse( fin );
    }
    public void OnAuthorization( AuthorizationContext filterContext ) { ← Donne accès, notamment, au Contrôleur
      var now = DateTime.Now.TimeOfDay;
      if( !(Debut <= now && now <= Fin) )
        filterContext.Result = new ContentResult() {
                                        Content = $"Utilisable seulement dans la plage horaire {Debut}-{Fin}" };
    }
  }
}
```

```
public class ExempleFiltreController : Controller {
  [OutputCache( Duration = 600, Location = System.Web.UI.OutputCacheLocation.Server )]
  [HorairesDAcces( "23:48", "23:50" )]
  public ActionResult Index( int? a, int? b ) {
    ViewBag.A = a;
    ViewBag.B = b;
    return View( (a ?? 0) + (b ?? 0) );
  }
}
```