ASP.NET MVC

Mourad MAHRANE

Objectifs

- Maîtriser l'utilisation du framework ASP.Net MVC 5
- Apprendre la syntaxe du moteur de vue Razor
- Construire des applications sécurisées.

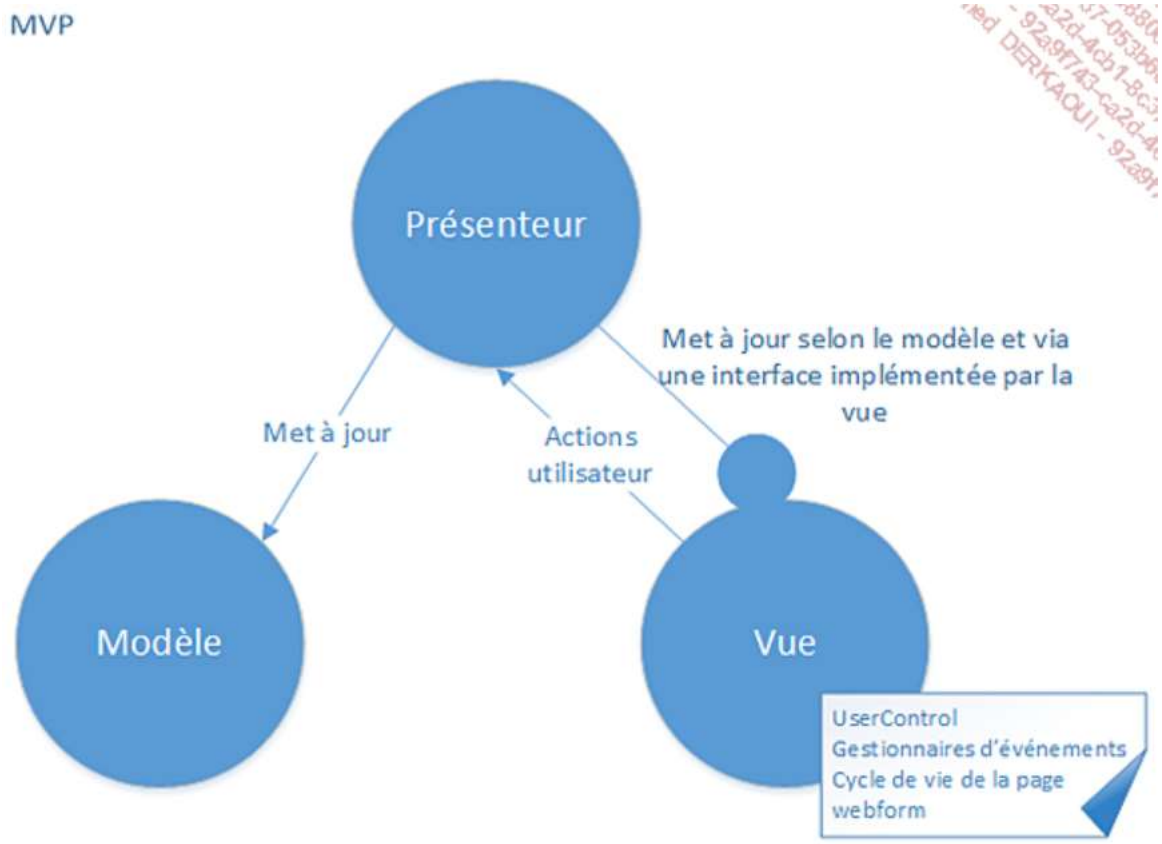
Plan

- Introduction
- Vue d'ensemble
- Routage
- Conception de vues
- Gestion des pages d'erreur
- Utilisation avancée des contrôleurs
- Gestion du cache
- Sécurité

Introduction

Modèle traditionnel (ASP.Net WebForms)

- Les WebForms s'appuyent sur le modèle MVP (Model-View-Presenter)

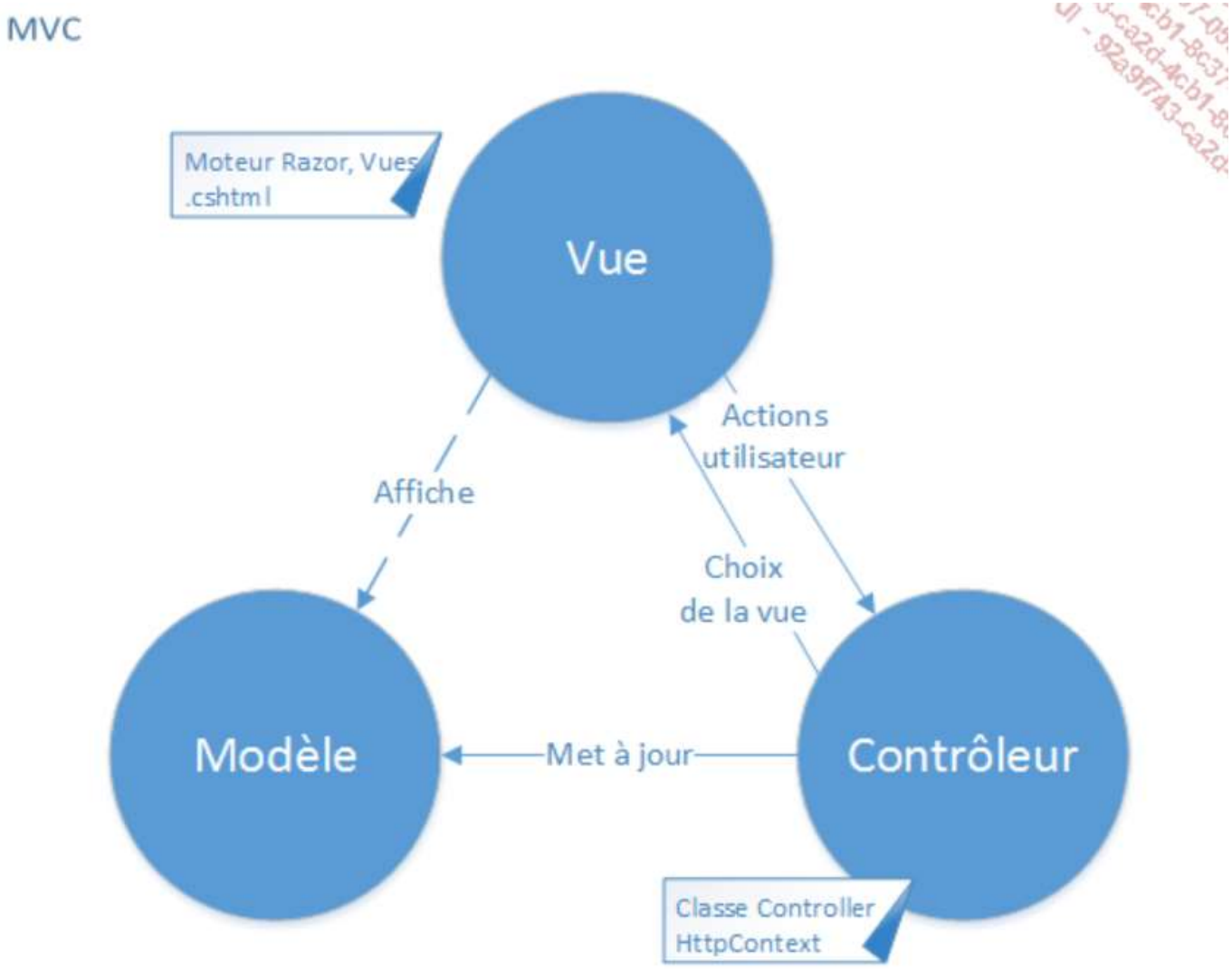


ASP.NET MVC

Framework applicatif de présentation basé sur le modèle MVC pour séparer les responsabilités entre les différents acteurs de l'application (http://www.asp.net/mvc/overview/getting-started/) :

- **Le modèle :** il s'agit de la brique centrale du paradigme. Le modèle est une classe hydratée par le contrôleur et ensuite passée à la vue qui est générée en fonction de l'état de ce modèle.
- **La vue :** page qui ne contient que des éléments d'affichage, principalement du HTML. Ceux-ci sont remplis à partir d'une instance du modèle.
- **Le contrôleur :** c'est lui qui réagit aux entrées utilisateurs et qui s'en sert pour manipuler des données.

ASP.Net MVC (2)

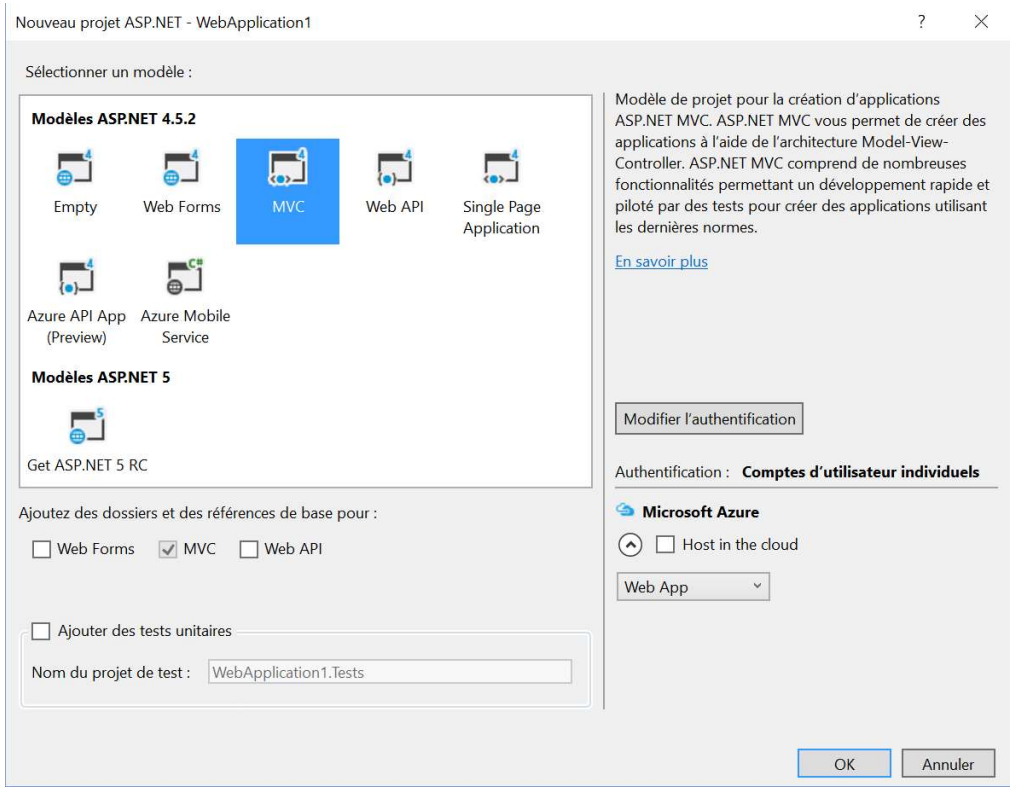

MVC
Moteur Razor, Vues
.cshtml
Vue
Actions
utilisateur
Affiche
Choix
de la vue
Modèle
Met à jour
Contrôleur
Classe Controller
HttpContext

Versions

- **ASP.NET MVC 1 (2008)**
- **ASP.NET MVC 2 (2010)** : création de vues partielles, regroupement de contrôleurs en zones, support de l'asynchrone, validation du modèle par attributs, helpers HTML, filtres d'actions.
- **ASP.NET MVC 3 (2011)** : màj majeure, nouveau moteur Razor, support d'annotations, résolutiion de dépendances, filtres d'actions à portée globale.
- **ASP.NET MVC 4 (2012)** : nouvelle charte graphique, support des dernières versions de jQuery, jQuery UI et Modernizr, ASP.NET Web API pour concevoir des services HTTP, regroupement de ressources en bundle.
- **ASP.NET MVC 5 (2013)** : ASP.NET Identity, routage par attributs, paramètres optionnels, filtres d'authentification, Bootstrap, génération de vues à partir du modèle.
- **ASP.NET MVC 6 RC1** (Novembre 2015)

Environnement de développement

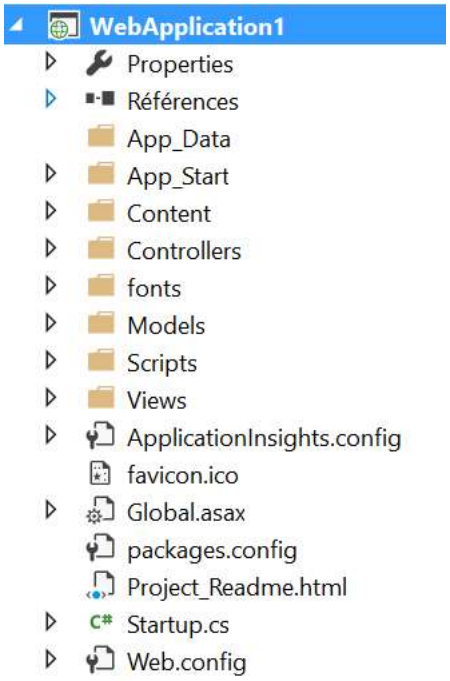
- ASP.Net MVC 5 : Visual Studio 2013 min.
- Si Visual Studio 2012 est installé, ASP.NET MVC 4 est déjà disponible et il n'est pas nécessaire d'exécuter d'autres installeurs.
- Un modèle de projet MVC est disponible.



Vue d'ensemble

Structure d'un projet MVC

- **App_Data :** stockage des bases de données embarquées, fichiers XML ou toute autre source de données (dossier invisible de l'extérieur et inaccessible des utilisateurs de l'application).
- **App_Start :** classes qui correspondent à l'éclatement du contenu du fichier Global.asax
- **Content :** fichiers CSS et autres ressources qui ne seraient ni des fichiers JavaScript ni des images.
- **Controllers :** stockage des contrôleurs présents par défaut.
- **Models :** classes métier de l'application
- **Scripts :** fichiers JavaScript de l'application.
- **Views :** vues de l'application, classées dans des répertoires du nom du contrôleur qui les utilise. Dans certains contextes, des vues peuvent également être partagées entre plusieurs contrôleurs, elles seraient alors placées dans le sous-répertoire Shared.



Le modèle

- Classes métier de l'application qui peuvent être décorées pour définir des contraintes ou un format.

```
public class Contact
{
        public int Id { get; set; }

        [Required(ErrorMessage = "Champ obligatoire")]
        [Display(Name ="Nom :")]
        public string Name { get; set; }

        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:dd/MM/yyyy}", ApplyFormatInEditMode = true)]
        [Display(Name = "Date de naissance :")]
        public DateTime BirthDate { get;  set; }
}
```

La vue

Le dossier Views regroupe les vues du projet

- **Shared** : dossier contenant les éléments partagés. On y trouve :
  - Un fichier de mise en page global « _Layout.cshtml » : équivalent de MasterPage dans les WebForms, ce fichier est référencé dans une vue Razor avec :

```
@{
        Layout = "~/Views/Shared/_Layout.cshtml";
}
```

  Il n'est pas nécessaire de répéter ce code pour chaque vue car, par défaut, une vue utilise automatiquement le fichier de mise en page défini dans un autre fichier : Views/_ViewStart.cshtml.

  - Une vue partielle partagée : lorsqu'une même vue partielle doit être utilisée par plusieurs contrôleurs, il est possible de la placer dans ce répertoire.

  - Un modèle d'affichage et un modèle d'édition : leur vocation fonctionnelle se rapproche de celle des contrôles utilisateurs d'ASP.NET WebForms. Ils permettent de spécifier un affichage bien précis pour un modèle donné, pour l'affichage et pour l'édition.

_Layout.cshtml

```
<!DOCTYPE html>
<html><head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - Mon application ASP.NET</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
</head>
<body>
    <div class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
                    …
                </button>
                @Html.ActionLink("Nom de l'application", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li>@Html.ActionLink("Accueil", "Index", "Home")</li>
                    <li>@Html.ActionLink("À propos de", "About", "Home")</li>
                </ul>
            </div>
        </div>
    </div>
    <div class="container body-content">
        @RenderBody()
        <hr />
        <footer>
            <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
        </footer>
    </div>
    @Scripts.Render("~/bundles/jquery")
    @Scripts.Render("~/bundles/bootstrap")
    @RenderSection("scripts", required: false)
</body></html>
```

Détail d'une vue Razor

- Structure HTML5, intégration de Bootstrap (RWD).
- Certains éléments sont récupérés dynamiquement :
  - **ViewData** ou **ViewBag** correspondent à un même dictionnaire clé-valeur, accessible depuis le contrôleur ou la vue et utile pour partager quelques données simples. ViewBag constituant la version dynamique de ViewData, la propriété est accessible comme suit :
  ViewBag.Title = "Bienvenue";
  ou ViewBag["Title"]="Bienvenue";

  - @Styles.Render et @Scripts.Render : instructions permettant d'inclure automatiquement un bundle dans une page.

  - **@Html.Partial** permet de rendre une vue partielle (pour décomposer la structure globale de la page en sous-éléments réutilisables, et donc plus simples à maintenir).

Détail d'une vue Razor (2)

- **@Html.ActionLink :** un HTML Helper permettant de générer facilement une vue et du contenu HTML en se basant sur certains paramètres, souvent, une classe du modèle spécifique.

- **@RenderBody :** correspond au rendu de la vue en elle-même (emplacement où le contenu de la vue est inséré).

- **@RenderSection** : donne la possibilité de réserver un espace dans la structure globale pour qu'une vue écrive du contenu spécifique. Par exemple, prévoir une section nommée scripts est un scénario assez courant qui permet à une vue de faire des inclusions de fichiers JavaScript tout en bas de la page. Un paramètre required permet de préciser si la vue doit obligatoirement définir cette section.

Syntaxe Razor

- @{ …….} : bloc de code à exécuter et qu'il ne s'agit pas de texte à écrire directement dans le flux de sortie.

```
@foreach (var c in ViewBag.ContactList) { … }
@if (condition) { … }
```

- @using NS1 : import d'un namespace
  @model NS1.NomClasse : import d'une classe métier.
- @Html.methode(...) : helper HTML ciblant des propriétés
- @section nomSection { …….} : définition d'une section nommée. Une vue peut définir précisément le contenu d'un bloc de code qui est pourtant hors de sa portée directe, à un endroit différent du @RenderBody.
- Objets implicites : Request, Response, Service, Session, User ou encore UICulture. Il existe même certaines propriétés utilitaires pour informer plus précisément le développeur du contexte d'exécution courant, par exemple les propriétés IsAjax ou IsPost.

Exemple de formulaire

```
@using (Html.BeginForm("Save", "ContactManagement"))
    { //@using (Html.BeginForm("NomAction", "Controleur", FormMethod.Post, new { enctype = "multipart/form-data" }))
        @Html.AntiForgeryToken()
        <div class="form-horizontal">
            @Html.ValidationSummary(true, "", new { @class = "text-danger" })
            @Html.HiddenFor(model => model.Id)
            <div class="form-group">
                @Html.LabelFor(model => model.Name, htmlAttributes: new { @class = "control-label col-md-2" })
                <div class="col-md-10">
                    @Html.EditorFor(model => model.Name, new { htmlAttributes =
                                                            new { @class = "form-control", placeholder="write a name" } })
                    @Html.ValidationMessageFor(model => model.Name, "", new { @class = "text-danger" })
                </div>
            </div>
            <div class="form-group">
                @Html.LabelFor(model => model.BirthDate, htmlAttributes: new { @class = "control-label col-md-2" })
                <div class="col-md-10">
          @Html.EditorFor(model => model.BirthDate, new { htmlAttributes = new { @class = "form-control datepicker" } })
                    @Html.ValidationMessageFor(model => model.BirthDate, "", new { @class = "text-danger" })
                </div>
            </div>
            <div class="form-group">
                <div class="col-md-offset-2 col-md-10">
                    <input type="submit" value="Save" class="btn btn-default" />
                </div>
            </div>
        </div>
    }
```

Le contrôleur

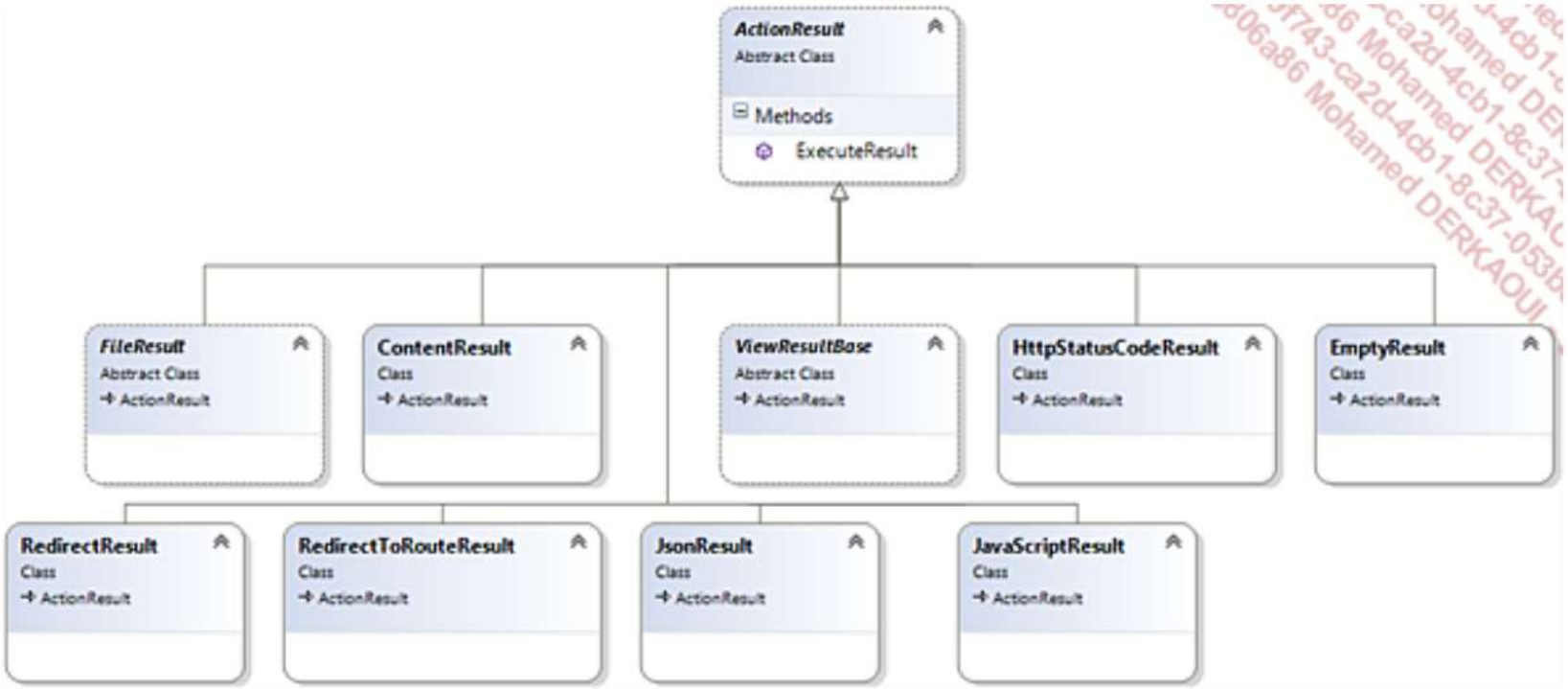
- Classe qui dérive **Controller**.
  Elle doit contenir le terme Controller dans son nom.
  On parle alors d'un contrôleur Home et non de HomeController.

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

    public ActionResult About()
    {
        ViewBag.Message = "Your application description page.";

        //On peut définir des variables librement dans le ViewBag : simples, complexes
        ViewBag.MyContact = new Contact { Name = "Moh. DERKAOUI" };
        return View();
    }

    public ActionResult Contact()
    {
        ViewBag.Message = "Your contact page.";

        return View();
    }
}
```

Retour des actions du contrôleur

- Les actions ont pour type de retour ActionResult ou une de ses dérivées. Il est également possible de créer une action dont le type de retour est void ou un type primitif.
- ViewBag est une propriété présente sur Controller. Elle est de type dynamic et est automatiquement transmise à la vue appelée à la suite de l'exécution d'une action.
- La méthode View(...) est une méthode retournant un résultat de type ViewResult, un type dérivé d'ActionResult. Ce type de résultat doit contenir le nom de la vue à invoquer auprès du moteur de vue. Par défaut, si aucun nom n'est spécifié, c'est automatiquement le nom de l'action en exécution qui est recherché, dans le répertoire du contrôleur.
- On peut également rediriger vers une autre action avec la méthode RedirectToAction(...)

Retour des actions du contrôleur (2)



Accès aux actions du contrôleur

- Les actions d'un contrôleur sont accessibles par des requêtes basées sur des URLs :
  - http://monsite/home/index : contrôleur Home et l'action Index.
  - http://monsite/home/about : contrôleur Home et l'action About.
  - http://monsite/home : contrôleur Home et l'action Index.

- Ce comportement provient de la configuration par défaut des routes présente à la création du projet. Si aucune action n'est spécifiée dans l'URL, c'est l'action Index qui est recherchée par défaut. **App_Start/RouteConfig.cs** :

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

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

Routage

Configuration des routes

- Les routes sont configurées dans la classe RouteConfig :

```
routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

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

- Définition de routes :

```
routes.MapRoute(name: "ContactManagement",
                url: "{controller}/{action}/{name}/{id}"
);
```

- Le paramètre de la route est accessible dans le contrôleur via :

```
var monParametre = RouteData.Values["monParametre"];
```

- Introduction de segments :

```
routes.MapRoute(name: "Default",
    url: "articles/{parametreA}-{parametreB}-{parametreC}",
    defaults: new { controller = "Home", action = "Index" }
);
```

Routage par attributs

- ASP.NET MVC 5 autorise le routage par attributs. Il faut l'activer dans le RouteConfig avec : **`routes.MapMvcAttributeRoutes();`**
  (On peut combiner routage classique et routage par attributs)

- Routage simple :

```
// ex: /1/Ordinateur
[Route("{productId:int}/{productTitle}")]
public ActionResult Show(int productId)
```

- Paramètres optionnels et valeurs par défaut :

```
// ex: /books ou  /books/1430210079
[Route("books/{isbn?}")]
public ActionResult View(string isbn) {
     if (!String.IsNullOrEmpty(isbn))
        return View("OneBook", GetBook(isbn));

     return View("AllBooks", GetBooks());
}

// /books/lang ou /books/lang/en ou /books/lang/fr
[Route("books/lang/{lang=en}")]
public ActionResult ViewByLanguage(string lang) {
     return View("OneBook", GetBooksByLanguage(lang));
}
```

Routage par attributs
Préfixes

- On peut préfixer d'un chemin :

```
[RoutePrefix("reviews")]
public class ReviewsController : Controller
{
    // ex.: /reviews
    [Route]
    public ActionResult Index() { … }

    // ex.: /reviews/5
    [Route("{reviewId}")]
    public ActionResult Show(int reviewId) { … }

    // ex.: /reviews/5/edit
    [Route("{reviewId}/edit")]
    public ActionResult Edit(int reviewId) { … }

    // On peut redéfinir le préfixe en utilisant le tilde ~
    // ex.: /spotlight-review
    [Route("~/spotlight-review")]
    public ActionResult ShowSpotlight() { … }
}
```

Routage par attributs
Route par défaut

- On peut appliquer l'attribut [Route] au contrôleur :

```
[RoutePrefix("promotions")]
[Route("{action=index}")] //action par défaut = index
public class ReviewsController : Controller
{
    // ex.: /promotions
    public ActionResult Index() { … }

    // ex.: /promotions/archive
    public ActionResult Archive() { … }

    // ex.: /promotions/new
    public ActionResult New() { … }

    // ex.: /promotions/edit/5
    [Route("edit/{promoId:int}")]
    public ActionResult Edit(int promoId) { … }
}
```

Routage par attributs Contraintes

- Contrainte = appliquer des restrictions sur les params

```
// ex: /users/5
[Route("users/{id:int}"]
public ActionResult GetUserById(int id) { … }

// ex: users/ken
[Route("users/{name}"]
public ActionResult GetUserByName(string name) { … }
```

Routage par attributs Contraintes (2)

Constraint	Description	Example
alpha	Matches uppercase or lowercase Latin alphabet characters (a-z, A-Z)	{x:alpha}
bool	Matches a Boolean value.	{x:bool}
datetime	Matches a **DateTime** value.	{x:datetime}
decimal	Matches a decimal value.	{x:decimal}
double	Matches a 64-bit floating-point value.	{x:double}
float	Matches a 32-bit floating-point value.	{x:float}
guid	Matches a GUID value.	{x:guid}
int	Matches a 32-bit integer value.	{x:int}
length	Matches a string with the specified length or within a specified range of lengths.	{x:length(6)} {x:length(1,20)}
long	Matches a 64-bit integer value.	{x:long}
max	Matches an integer with a maximum value.	{x:max(10)}
maxlength	Matches a string with a maximum length.	{x:maxlength(10)}
min	Matches an integer with a minimum value.	{x:min(10)}
minlength	Matches a string with a minimum length.	{x:minlength(10)}
range	Matches an integer within a range of values.	{x:range(10,50)}
regex	Matches a regular expression.	{x:regex(^\d{3}-\d{3}-\d{4}$)}

Routage par attributs Contraintes (3)

- On peut appliquer plusieurs contraintes sur un paramètre :

```
// ex: /users/5

// but not /users/10000000000 because it is larger than int.MaxValue,
// and not /users/0 because of the min(1) constraint.

[Route("users/{id:int:min(1)}")]
public ActionResult GetUserById(int id) { … }
```

- Si le paramètre est optionnel, le ? doit être spécifié après la contrainte :

```
// ex: /greetings/bye

// and /greetings because of the Optional modifier,
// but not /greetings/see-you-tomorrow because of the maxlength(3) constraint.
[Route("greetings/{message:maxlength(3)?}")]
public ActionResult Greet(string message) { … }
```

Routage par attributs
Contraintes personnalisées

- Création d'une classe qui implémente IRouteConstraint puis enregistrement dans la classe RouteConfig avec un ContraintsResolver :

```
public class MyValuesConstraint : IRouteConstraint
{
    private readonly string[] validOptions;

    public MyValuesConstraint(string options)
    {
        validOptions = options.Split('|');
    }

    public bool Match(HttpContextBase httpContext, Route route, string parameterName,
                      RouteValueDictionary values, RouteDirection routeDirection)
    {
        object value;
        if (values.TryGetValue(parameterName, out value) && value != null)
        {
          return validOptions.Contains(value.ToString(), StringComparer.OrdinalIgnoreCase);
        }
        return false;
    }
}
```

Routage par attributs
Contraintes personnalisées (2)

- Enregistrement (RouteConfig) :

```
var constraintsResolver = new DefaultInlineConstraintResolver();
constraintsResolver.ConstraintMap.Add("myvalues", typeof(MyValuesConstraint));
routes.MapMvcAttributeRoutes(constraintsResolver);
```

- Utilisation de la contrainte personnalisée :

```
// ex: temp/celsius and /temp/fahrenheit but not /temp/kelvin
[Route("temp/{scale:myvalues(celsius|fahrenheit)}")]
public ActionResult ShowTemp(string scale)
```

Routage par attributs
Routes nommées

- On peut nommer une route pour faciliter la génération d'URI :

```
[Route("menu", Name = "mainmenu")]
public ActionResult MainMenu() { … }
```

- Utilisation dans la vue :

```
<a href="@Url.RouteUrl("mainmenu")">Main menu</a>
```

Routage par attributs Areas

- Définition de zones de routage :

```
[RouteArea("admin")]
//si préfixe différent du nom : [RouteArea("admin", AreaPrefix="espace-admin")]
[RoutePrefix("menu")]
[Route("{action}")]
public class MenuController : Controller
{
    // ex: /admin/menu/login
    public ActionResult Login() { … }

    // ex: /admin/menu/show-options
    [Route("show-options")]
    public ActionResult Options() { … }

    // ex: /stats
    [Route("~/stats")]
    public ActionResult Stats() { … }
}
```

- Enregistrement des zones dans le RouteConfig :

```
AreaRegistration.RegisterAllAreas();
```

Debug de routage

- Il existe plusieurs extensions permettant de debugger les routes (packages NuGet) : Route Debugger, Glimpse
- Exemple d'utilisation de Route Debugger

  - ajout du package (Editeur Phil Haack)
  - activation dans le Web.Config :

```
<add key="RouteDebugger:Enabled" value="true" />
```

Conception de vues

Conception de formulaires

- Vue :

```
@using (Html.BeginForm("NomAction", "Controleur", FormMethod.Post,
                    new { enctype = "multipart/form-data" })) {

          Champs du formulaire (utilisation des HtmlHelpers)
}
```

- Contrôleur : méthode décorée avec [HttpGet] ou [HttpPost]

```
[HttpPost]
public ActionResult Save(FormCollection collection) { … }
```

- On peut récupérer les paramètres passés en Get avec :

```
Request.QueryString["nom du param"]
ou Request.Form["nom du param"]
```

- Token de validation du formulaire (contre les injections) :
  - ajouter dans la vue : `@Html.AntiForgeryToken()`
  - ajouter dans l'action du contrôleur : `[ValidateAntiForgeryToken]`

Conception de formulaires (2)

```
@model ProjMVCTraining.Models.Contact

<h2>Ajout / Modif d'un contact</h2>
@using (Html.BeginForm("Save", "ContactManagement",FormMethod.Post)) {
    @Html.AntiForgeryToken()
    <div class="form-horizontal">
        @Html.HiddenFor(model => model.Id)
        <div class="form-group">
            @Html.LabelFor(model => model.Name, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Name,
                            new { htmlAttributes = new { @class = "form-control", placeholder="write a name" } })
            </div>
        </div>
        <div class="form-group">
            @Html.LabelFor(model => model.BirthDate, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.BirthDate,
                                new { htmlAttributes = new { @class = "form-control datepicker" } })
            </div>
        </div>
        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Save" class="btn btn-default" />
            </div>
        </div>
    </div>
}
```

Attributs d'affichage

Il est possible de donner des renseignements sur la manière dont un modèle doit être utilisé (System.ComponentModel.DataAnnotations) :

```
[Display(Name ="Nom :")]
public string Name { get; set; }

[DisplayFormat(DataFormatString = "{0:dd/MM/yyyy}", ApplyFormatInEditMode = true)]
[Display(Name = "Date de naissance :")]
public DateTime BirthDate { get;  set; }

[DisplayFormat(DataFormatString = "{0:C}")]
public decimal Prix { get; set; }
```

Par défaut, le format spécifié ne sera pas utilisé dans les champs de type input. Pour qu'il continue à s'appliquer dans les scénarios d'édition, il est nécessaire de passer la valeur de la propriété ApplyFormatInEditMode à true :

```
[DisplayFormat(DataFormatString = "{0:C}", ApplyFormatInEditMode= true)]
public decimal Prix { get; set; }
(Attention à ne pas inclure des éléments qui perturberont la récupération, comme le symbole monétaire, ce
qui donnera une injection d'une chaine de caractères au lieu d'une valeur décimale).
```

Attributs d'affichage (2)

- Valeurs substitutives :

```
[DisplayFormat(NullDisplayText = "Pas de description")]
public string Description { get; set; }
```

- Champ caché :

```
[HiddenInput]
public int Id { get; set; }
```

- Type de données :

```
[DataType(DataType.MultilineText)]
public string Description { get; set; }

[DataType(DataType.Date)]
[DisplayFormat(DataFormatString = "{0:dd/MM/yyyy}")]
[Display(Name = "Date de naissance :")]
public DateTime BirthDate { get;  set; }
```

- **Affichage dans la vue (syntaxe Razor) :**

```
@Html.LabelFor(model => model.NomChamp)
@Html.EditorFor(model => model.Name)
```

Attributs de validation

- Champ obligatoire :

```
[Required(ErrorMessage = "Champ obligatoire")]
```

- Contrainte de taille (le max est obligatoire) :

```
[StringLength(100, MinimumLength = 5,
     ErrorMessage = "Le champ {0} doit comprendre au moins {2} caractères et ne
                     doit pas faire plus de {1} caractères.")]
```

- Range :

```
[Range(0, double.MaxValue, ErrorMessage = "Le prix ne peut pas être négatif")]
public double Prix { get; set; }
```

- Expression régulière :

```
[RegularExpression(@"^([0-9a-zA-Z]([-.\w]*[0-9a-zA-Z])*@([0-9a-zA-Z][-\w]*
                        [0-9a-zA-Z]\.)+[a-zA-Z]{2,9})$")]
public string Email { get; set; }
```

- **Affichage dans la vue :**

```
@Html.ValidationMessageFor(model => model.Champ, "", new {@class="text-danger"})
@Html.ValidationSummary(true, "", new { @class = "text-danger" })
```

Attributs de validation (2)

- Comparaison avec un autre champ :

```
[System.ComponentModel.DataAnnotations.Compare("Email")]
public string ConfirmationEmail { get; set; }
```

- Validation métier :
  (nécessite d'une méthode qui retourne un JsonResult)

```
[Remote("ValiderPrix", "Index", AdditionalFields = "IdMagasin")]
public decimal PrixHt { get; set; }

public JsonResult ValiderPrix(decimal Prix, int IdMagasin) {
      var magasin = new Magasin(IdMagasin);
      return Json(magasin.ValiderPrix(Prix), JsonRequestBehavior.AllowGet);
}
```

Localisation des messages de validation

- Création de fichier de ressources et définition de paires clé/valeur (public).
- Référence de la clé du message dans l'attribut de validation :

```
[Required(ErrorMessageResourceName = "nameError",
          ErrorMessageResourceType =typeof(MyResources1))]
```

- La propriété NullDisplayText de l'attribut DisplayFormat ne supporte pas la localisation ; pour résoudre cette limitation, il faut créer une classe qui dérive DisplayFormatAttribute :

```
public class DisplayFormatAvecValeurDeRemplacementAttribute : DisplayFormatAttribute
{
  public DisplayFormatAvecValeurDeRemplacementAttribute(string cle, Type typeResource) : base()
  {
        var resourceManager = new ResourceManager(typeResource);
        NullDisplayText = resourceManager.GetString(cle);
  }
}
```

-
```
[DisplayFormatAvecValeurDeRemplacementAttribute(
                "Produit_Description_Remplacement", typeof(Resource1))]
  public string Description { get; set; }
```

Attributs de validation personnalisés

- Dériver de la classe ValidationAttribute et redéfinir sa méthode IsValid
- La validation est uniquement côté serveur.
  Pour que l'attribut ait aussi une influence sur le HTML généré et que la validation fonctionne de même côté client, il doit implémenter l'interface IClientValidatable et sa méthode GetClientValidationRules.
- Cette implémentation a ensuite besoin d'être combinée à du code côté client.

Validations côté client

- Exemple d'implémentation de **IClientValidatable** :

```
public class PlusGrandQueAttribute : ValidationAttribute, IClientValidatable {
   private readonly int _valeurMinimum;

   public PlusGrandQueAttribute(int valeurMinimum) { _valeurMinimum = valeurMinimum; }

   public IEnumerable<ModelClientValidationRule> GetClientValidationRules(ModelMetadata metadata,
                     ControllerContext context) {
       var rule = new ModelClientValidationRule {
           ErrorMessage = FormatErrorMessage(_valeurMinimum.ToString(CultureInfo.CurrentCulture)),
           ValidationType = "plusgrandque"
       };
       rule.ValidationParameters.Add("valeurminimum", _valeurMinimum);
       yield return rule; //yield => itérateur
   }

}
```

- Modèle :

```
[PlusGrandQue(8, ErrorMessage="l'age minimal est {0}")]
public int Age { get; set; }
```

- Il faut ensuite s'occuper des implémentations JavaScript de la règle. Cela se fait en deux temps : il faut ajouter d'une part la règle et le message sur l'adaptateur de jQuery Validate Unobtrusive, en spécifiant bien en paramètre le nom de la règle et les paramètres qu'elle prend

Validations côté client (2)

- Implémentation js (à mettre dans le dossier Scripts) :

```
jQuery.validator.unobtrusive.adapters.add('plusgrandque', ['valeurminimum'],
      function (options) {
          options.rules.plusgrandque = options.params;
          options.messages.plusgrandque = options.message;
       });

jQuery.validator.addMethod('plusgrandque', function (value, element, params) {
     var valeurMini = params.valeurminimum;
     if (parseInt(value) < parseInt(valeurMini))
          return false;
     return true;
});
```

- Référence au script dans la vue :

```
@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
    @Scripts.Render("~/Scripts/myClientValidation.js")
}
```

- Champ et capture du message de validation :

```
@Html.EditorFor(model => model.Age, new { htmlAttributes = new { @class="form-control" } })
@Html.ValidationMessageFor(model => model.Age, "", new { @class="text-danger" })
```

Traitement de la validation dans le contrôleur

- Les validateurs définis sur les différentes propriétés sont récupérés et exécutés par l'intermédiaire d'une instance de la classe DataAnnotationsModelValidator.
- À l'issue du processus de validation et de liaison de données, les éventuelles erreurs sont placées dans la propriété **ModelState**, disponible sur tous les contrôleurs (ModelState = instance de la classe ModelStateDictionary)
- Deux cas de figure se présentent :
  - Le modèle est valide, l'action est effectuée sur la couche de service puis la redirection.
  - Le modèle n'est pas valide, on affiche à nouveau l'écran d'insertion ou d'édition, avec les valeurs que l'utilisateur vient de saisir. Il suffit juste de refaire appel à la vue, en lui passant le modèle incorrect reçu en paramètre de l'action plutôt qu'en instanciant un nouveau modèle vierge.

```
if (ModelState.IsValid) { … }
```

Syntaxe Razor

- Instruction simple : Bonjour, nous sommes le @DateTime.Now
- Commentaire : @* Si l'utilisateur est authentifié *@
- Bloc de code :

```
@{
    var monContenu = "Bonjour !";
}
```

- Conditions et boucles :

```
@if(a > 10)  {  <p>Condition remplie</p>  }
@foreach(var element in MaCollection) { <li>@element</li> }
```

- Echaper à la ligne courante :

```
<script type="text/javascript">
   @if (Request.IsAuthenticated) {
        @:console.log("mon contenu");
   }
</script>
```

HtmlHelpers

- Chaque vue MVC hérite en fait d'une classe générique WebViewPage qui possède la propriété Html :
  public HtmlHelper<TModel> Html { get; set; }
- Le namespace System.Web.Mvc.Html apporte de nombreuses méthodes d'extension à HtmlHelper :
  - La classe **DisplayExtensions** permet de générer le HTML pour des scénarios d'affichage avec la méthode Display.
  - La classe **InputExtensions** permet de générer de nombreux types de saisie : cases à cocher, champs de texte, champs de type mot de passe, boutons radio, etc.
  - La classe **LinkExtensions** participe de tout ce qui est génération de liens : un lien menant vers une action, un lien menant vers une route, etc.
  - La classe **ChildActionExtensions** permet d'écrire directement le résultat d'actions dans le flux de la réponse grâce aux méthodes Action et RenderAction.

HtmlHelpers (2)

- Spécification d'attributs HTML :

```
@Html.LabelFor(model => model.Name,
                htmlAttributes: new { @class = "control-label col-md-2" })

@Html.EditorFor(model => model.Name,
                new { htmlAttributes =
                      new { @class = "form-control", placeholder="write a name" } })
```

- Création de méthodes personnalisées :

```
@helper EcrireListe(params string[] liste)
{
    <ul>
        @foreach (var e in liste) { <li>@e</li> }
    </ul>
}
<h2>Titre</h2>
@EcrireListe("toto", "tata", "titi")
```

- **On peut externaliser les helpers dans un .cshtml placé dans le dossier App_Code:**

```
@MesMethodesHelper.EcrireListe("toto","tata", "titi")
```

Génération d'une vue depuis le modèle

- Génération avec l'éditeur de Visual Studio
- Génération avec @Html.EditorForModel().
- **Création d'une vue personnalisée associée à une classe :**
  une vue partielle du nom exact de la classe modèle, placée au bon endroit, peut remplacer la génération automatique de la vue. Pour la suite de l'explication, EditorForModel est pris pour exemple, mais il en est exactement de même pour DisplayForModel.

  La classe HtmlHelper EditorForModel cherche une vue du nom exact du modèle située dans un dossier nommé **EditorTemplates**. Dans le cadre de Razor, le dossier peut alors se situer à deux niveaux :

  - Dans le dossier Views\Nom du contrôleur\EditorTemplates.
    La vue est alors accessible du contrôleur uniquement.

  - Dans le dossier Views\Shared\EditorTemplates.
    La vue est accessible de tous les contrôleurs.

Modes d'affichage

- Possibilité de servir un autre fichier en fonction du terminal (User-Agent) : NomVue.Mobile.cshtml
- Les modes d'affichage ne sont pas limités au distinguo mobile/PC car il est possible de définir des modes d'affichage personnalisés, au démarrage de l'application web.

  - Le développeur décide alors du suffixe à ajouter à la vue et des conditions qui font que le User-Agent est déclaré éligible à ce mode de vue.
  - Il peut enlever les modes par défaut et recréer des modes personnalisés, qui auraient par exemple une granularité plus fine. Par exemple, pour distinguer PC, tablette et mobile, des modes d'affichage pourraient être déclarés dans le fichier Global.asax lors du démarrage de l'application. C'est la classe singleton **DisplayModeProvider** qui contient tous les modes d'affichage supportés par ASP.NET MVC et c'est là que les modes personnalisés peuvent être rajoutés.

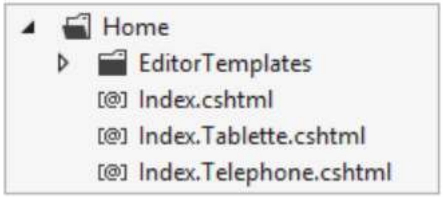
Modes d'affichage (2)

- **Global.asax :**

```
public class MvcApplication : System.Web.HttpApplication {
        protected void Application_Start() {
                ……
                MyDisplayModesConfig.RegisterDisplayModes(DisplayModeProvider.Instance.Modes);
        }
    }
```

- **Notre implémentation des modes :**

```
public class MyDisplayModesConfig {
    public static void RegisterDisplayModes(IList<IDisplayMode> displayModes) {
        displayModes.Clear();
        var modeTablette = new DefaultDisplayMode("Tablette") {
           ContextCondition = (c =>
           {
              var ua = c.Request.UserAgent.ToLower();
              return ua.Contains("gt-p1000") || ua.Contains("nexus 7") || ua.Contains("ipad");
           })
        };
        var modeMobile = new DefaultDisplayMode("Telephone") {
              ContextCondition = (c => c.Request.UserAgent.ToLower().Contains("mobile"))
        };
        var modeBureau = new DefaultDisplayMode(string.Empty) {
              ContextCondition = (_ => true)
        };

        displayModes.Add(modeTablette);
        displayModes.Add(modeMobile);
        displayModes.Add(modeBureau);
    }
}
```



Gestion du download

- Vue :

```
@Html.ActionLink("Texte à afficher", "NomAction")
```

- Action du contrôleur : on doit retourner un résultat de type : FileResult, FileContentResult ou FileStreamResult

```
public FileContentResult NomAction()
{
    StringBuilder sb = new StringBuilder();
    ContactDao.FindAll().ForEach(c => sb.AppendLine(c.Id + ";" + c.Name));
    return new FileContentResult(Encoding.UTF8.GetBytes(sb.ToString()),
                                 "text/csv");
}
```

Gestion de l'upload

- Vue :

```
@using (Html.BeginForm("Upload", "ContactManagement", FormMethod.Post,
                        new { enctype = "multipart/form-data" })) {
    @Html.AntiForgeryToken()
    <fieldset>
        <legend>Upload a file</legend>
        <div class="form-group">
            @Html.TextBox("file", "", new { type = "file" })
        </div>
        <div class="form-group">
            <input type="submit" value="Upload" class="btn btn-default" />
        </div>
    </fieldset>
}
```

- Contrôleur :

```
[HttpPost]
 public ActionResult Upload(HttpPostedFileBase file) {
       try {
           if (file.ContentLength > 0) {
               var fileName = Path.GetFileName(file.FileName);
               var path = Path.Combine(Server.MapPath("~/App_Data"), fileName);
               file.SaveAs(path);
           }
           ViewBag.Message = "Upload successful";
       } catch { ViewBag.Message = "Upload failed"; }
       ViewBag.ContactList = ContactDao.FindAll();
       return View("Index");
 }
```

Accès aux objets du contexte

- Vue : **@HttpContext.Current**
  ou directement par @Request, @Response, @Session, @ApplicationInstance,…

- Contrôleur : **Classe HttpContext**
  Request, Response, Session, ...

Vues partielles

- Définition de code réutilisable :

  - .cshtml en tant que vu partielle sans disposition
  - inclusion dans les vues avec :

```
@Html.Partial("NomVuePartielle", Model)
```

i18n

- Définition de fichiers de ressources / langue

  NomBase.codeLangueISO639-1.resx

- Redéfinition de la méthode BeginExecuteCore de Controller :

```
protected override IAsyncResult BeginExecuteCore(AsyncCallback callback, object state)
```

- La locale est persistée dans un cookie

- Changement de la locale :

```
Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo(cultureName);
Thread.CurrentThread.CurrentUICulture = Thread.CurrentThread.CurrentCulture;
```

Gestion des pages d'erreur

Solution 1 : capturer l'exception dans l'action

```
public ActionResult UneActionAvecTraitementEx()
{
      try
      {
            //.......
      }
      catch (Exception ex)
      {
            return View("Error");
      }

      return View();
}
```

Solution 2 : capturer l'exception dans le contrôleur

- Redéfinir la méthode OnException ; on peut définir notre propre classe BaseController pour factoriser le traitement.

```
public class MyBaseController : Controller
{
    protected override void OnException(ExceptionContext filterContext)
    {
        Exception ex = filterContext.Exception;
        filterContext.ExceptionHandled = true;

        var model = new HandleErrorInfo(ex, "Controller", "Action");
        filterContext.Result = new ViewResult()
        {
            ViewName = "Error",
            ViewData = new ViewDataDictionary(model)
        };
    }
}
```

- Activer les CustomErrors dans le Web.Config :

```
<customErrors mode="On" defaultRedirect="~/Error"></customErrors>
```

et définir le ErrorController. La vue est également partagée (Shared)

Solution 2 : capturer l'exception dans le contrôleur

- Vue :

```
@{
    ViewBag.Title = "Erreur";
    if (Model != null)
    {
        Exception ex = Model.Exception;
        HttpException httpEx = (HttpException)ex;

        <h1>Code : @httpEx.GetHttpCode()</h1>
        <h4>Message : @ex.Message</h4>
    }
    else
    {
        <h1>Code : 404</h1>
        <h4>Page introuvable : @Request.Params["aspxerrorpath"]</h4>
    }
}
```

Solution 3 : utiliser l'attribut [handleError]

- Activer l'attribut dans le FilterConfig
- Décorer l'action ou le contrôler :

```
public class HomeController : Controller
 {
        [HandleError()]
        public ActionResult SomeError()
        {
            throw new Exception("test");
        }
}
```

- On peut spécifier le type d'exception à capturer et/ou la vue à utiliser (par défaut : Error.cshtml)

```
public class HomeController : Controller
{
        [HandleError(ExceptionType=typeof(ArithmeticException),View="Arthimetic")]
        [HandleError(ExceptionType = typeof(NotImplementedException),View ="Error1")]
        public ActionResult SomeError()
        {
              //… …
        }
}
```

Solution 4 : capturer l'exception dans Global.asax

- Traitement dans Application_Error

```
protected void Application_Error()
{
        var ex = HttpContext.Current.Server.GetLastError();
        //Redirection vers la page souhaitée
}
```

Solution non optimale

Utilisation avancée des contrôleurs

Gestion de l'état de la session

- Pour augmenter les performances, on peut définir un comportement particulier pour la gestion de l'état de la session lié à la requête de l'utilisateur : soit en le désactivant totalement ou en le limitant à de la lecture seule (**au niveau du contrôleur et non de l'action**).
- Il faut implémenter l'interface IControllerFactory et redéfinir la méthode :

```
public SessionStateBehavior GetControllerSessionBehavior(
                        RequestContext requestContext, string controllerName)
```

- Le retour peut prendre les valeurs suivantes :
  - Disabled : le support de l'état de session est totalement désactivé.
  - Readonly : les données de session sont utilisables uniquement en lecture.
  - Required : les données de session sont utilisables en lecture ainsi qu'en écriture.
  - Default : correspond à la logique par défaut utilisée par ASP.NET WebForms (comportement au niveau de l'instance de IHttpHandler. Avec ASP.NET WebForms, les objets de type Page implémentent l'interface IHttpHandler et exposent une propriété nommée EnableSessionState.

Libération des ressources

- L'interface IControllerFactory possède une autre méthode qui contiendra l'implémentation des ressources à libérer (un service, un accès à une base de données ou un fichier en écriture) :

```
public void ReleaseController(IController controller)
```

- Autre possibilité : implémenter l'interface IDisposable dans le contrôleur

Retour d'exécution d'une action

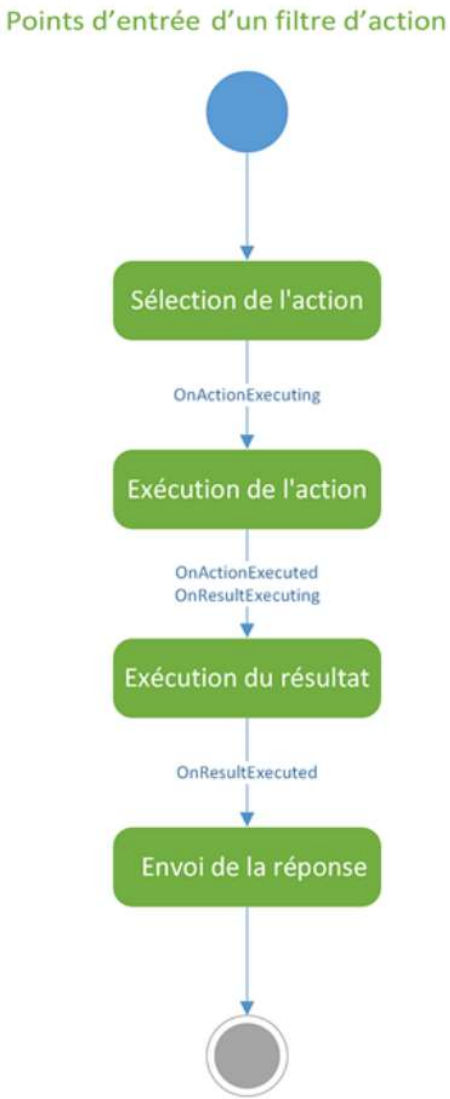
- **EmptyResult :**
  return new EmptyResult();
- **HttpStatusCodeResult :**
  return new HttpStatusCodeResult(HttpStatusCode.Forbidden);
- **RedirectResult :**
  return new RedirectPermanent("http://www.dawan.fr", true);//redir. 301
  return new Redirect("http://www.dawan.fr");//redirection temporaire 302
- **RedirectToRouteResult :**
  return new RedirectToRouteResult("RouteParDefaut",
  new RouteValueDictionary() { { "id", 15 } });
- **JavascriptResult :**
  return JavaScript("alert('bonjour !')");
- **JsonResult :**
  var monInstance = new MaClasse() { MaPropriete = "Ma valeur" };
  return Json(monInstance, JsonRequestBehavior.AllowGet);
- **FileResult** ou dérivées
- On peut définir d'autres types en héritant de « ActionResult »

Filtres d'actions (intercepteurs)

- Plusieurs interfaces disponibles à implémenter :
  - IActionFilter : 2 événements du cycle de vie d'une action, le premier avant son exécution, le second après son exécution.
  - IResultFilter : avant et après la génération du résultat.
  - IAuthorizationFilter : autoriser ou non l'exécution d'une action pour l'utilisateur courant.
  - IExceptionFilter : exécuter un comportement spécifique dès lors que l'exécution d'une action se termine avec une exception.

- Les filtres sont recherchés dans les **attributs** à différents niveaux :
  - Sur une action, le comportement du filtre est alors applicable uniquement à l'action qu'il décore.
  - Sur un contrôleur, le comportement du filtre est alors applicable à toutes les actions du contrôleur qu'il décore.
  - Pour toute l'application, on parle alors de filtre global. Ceux-ci sont définis au lancement de l'application en étant enregistrés dans une collection de filtres.

Filtres d'actions (2)



Filtres d'action (3)

- La classe abstraite **ActionFilterAttribute** permet de créer des filtres d'exécution personnalisés. En effet, elle implémente les interfaces **IActionFilter** et **IResultFilter** et permet de réagir facilement à l'un des événements exposés en redéfinissant la méthode adéquate :

```
public class MyLogFilter : ActionFilterAttribute
{
    public override void OnActionExecuting(ActionExecutingContext filterContext)
    {
        Trace.TraceInformation("OnActionExecuting", filterContext.RouteData);
    }
}
```

- Application à différents niveaux :

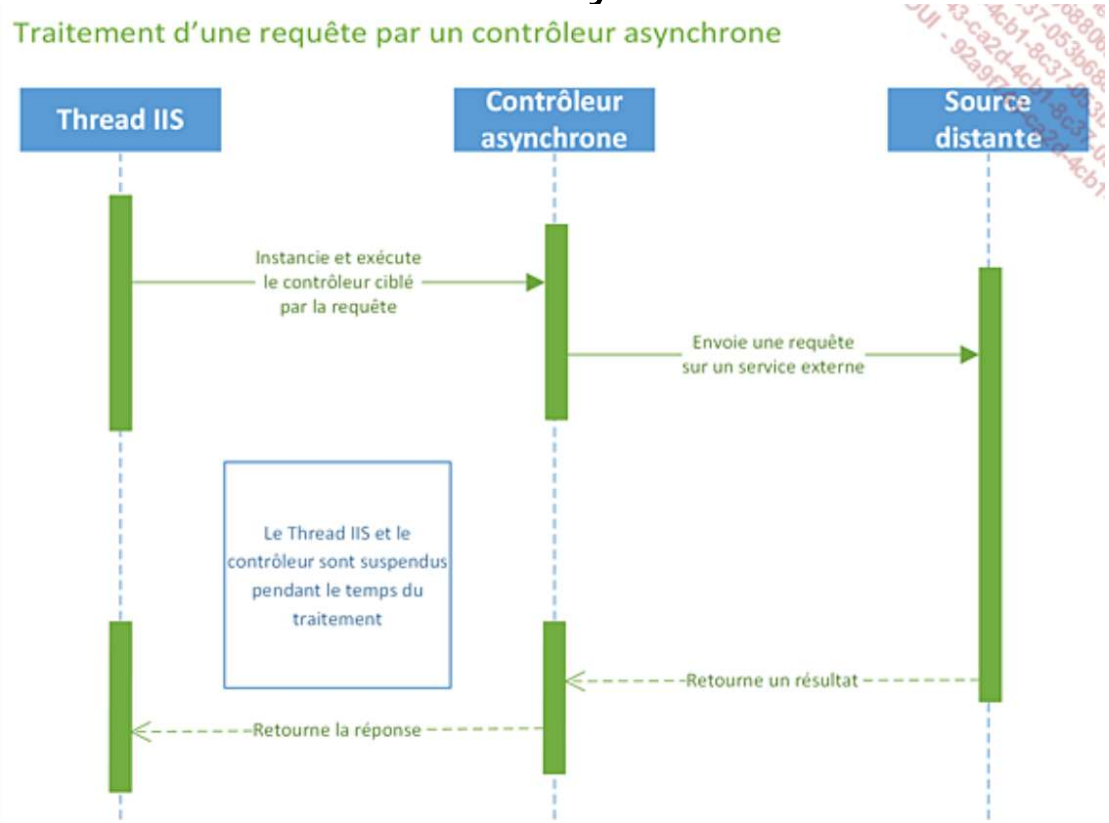
  - contrôleur : `[MyLogFilter]` `public class MyController : Controller`

  - action du contrôleur : `[MyLogFilter] sur l'action`

  - ensemble des contrôleurs : ajout dans le constructeur dans la classe FilterConfig : `filters.Add(new MyLogFilter());`

Contrôleurs asynchrones

- Les contrôleurs asynchrones ne sont pas tant des optimisations pour le client qui sont visées, que des gains et une économie de ressources pour le serveur.
- Fonctionnement des contrôleurs asynchrones :



Contrôleurs asynchrones Implémentation

- ASP.NET MVC < 4 : AsyncController
- ASP.NET MVC >=4 : il n'est plus nécessaire de créer un contrôleur qui dérive de la classe AsyncController. L'action doit simplement être marquée comme **async** (et utiliser en interne un appel avec le mot-clé **await**, selon les spécifications du langage C#) et retourner une Task ou une **Task<T>**, où T correspondant au type de la réponse que le client recevra :

```
[HttpGet]
public async Task<ActionResult> Contrats()
{
    var contrats = await _serviceDistant.TelechargerContrats();
    return View("Contrats", contrats);
}
```

Notions avancées

JavaScript et jQuery

- Utilisation libre de JavaScript, jQuery
- Helpers Ajax disponibles : System.Web.Mvc.AjaxHelper

```
@Ajax.ActionLink(
"Liste des produits", //texte affiché

"DisplayProduits", //action

"Ajax", //controleur

 new AjaxOptions

  {

   UpdateTargetId="ObjectsList", //Id de l'element DOM à maj

   InsertionMode = InsertionMode.Replace, //type de rempl/ de contenu

   HttpMethod = "GET"

  }

)
```

- Soumission asynchrone de formulaire :
  @using (Ajax.BeginForm(...)){ … }
- Ne pas oublier d'ajouter la librairie jquery.unobstrusive.ajax

Utilisation du cache

- **Cache de niveau requête** (similaire à ASP.NET) **:**
  Cacher des données dans la portée de la requête seulement permet aux modules HTTP, aux gestionnaires HTTP, aux actions ou aux filtres d'actions de partager des données sans avoir besoin d'intermédiaire ni de refaire le traitement nécessaire pour trouver la donnée :
  - Mise en cache : `HttpContext.Items["Code"] = 32142;`
  - Cet élément est accessible dans un filtre à l'aide de :
  `filterContext.HttpContext.Items["Code"] as Int32;`
- Utilisation des autres espaces de persistance : Session, Application
- **System.Web.Caching.Cache :**
  Cet objet offre en outre des fonctionnalités d'invalidation et d'expiration du cache assez avancées :
  `HttpContext.Cache["NomApplication"] = "MonApplication";`

Utilisation du cache l'objet Cache

Fonctionnalités :

- Donner des priorités aux objets ajoutés au cache (ceux avec une priorité moindre sont d'abord supprimés lorsque le processus IIS fait de la place), il est possible d'assigner une priorité NonRemovable, où la valeur n'est jamais supprimée.
- Spécifier une dépendance de l'objet mis en cache, par exemple un fichier : lorsque le fichier spécifié en dépendance est modifié, l'objet en cache est invalidé.
- Spécifier une méthode de rappel : lors de la suppression du cache de la valeur, la méthode spécifiée est alors exécutée.
- Donner un temps d'expiration glissante : le temps maximal entre deux accès qui peut s'écouler avant que l'objet ne soit supprimé du cache.
- Donner un temps d'expiration absolue : la durée de vie maximale de la valeur, quelles que soient les fréquences d'accès.
- Un seul mode d'expiration peut être spécifié : glissant ou absolu, par objet caché.

Utilisation du cache l'objet Cache (2)

```
public ActionResult ApplicationCacheDependency(string nomapplication)
{
            if (string.IsNullOrEmpty(nomapplication))
            {
                HttpContext.Cache.Add("NomApplication", "MonApplication", null,
                                      Cache.NoAbsoluteExpiration,
                                      new TimeSpan(1, 0, 0), CacheItemPriority.Default, null);

                HttpContext.Cache.Add("NomPage", "Mon Application > Ma page",
                                      new CacheDependency(null, new[] {"NomApplication"}),
                                      Cache.NoAbsoluteExpiration,
                                      new TimeSpan(1, 0, 0),CacheItemPriority.Default, null);
            }
            else
            {
                HttpContext.Cache.Add("NomApplication", nomapplication, null,
                                      Cache.NoAbsoluteExpiration, new TimeSpan(1, 0, 0),
                                      CacheItemPriority.Default, null);
            }

            ViewBag.NomPage = HttpContext.Cache.Get("NomPage");
            return View();
}
```

Mise en cache du HTML côté serveur

- Possibilité de mettre en cache le code html généré par la vue :

```
[OutputCache(Duration = 60)]
public ActionResult CacheSimple()
{
    ViewBag.Date = DateTime.Now.ToString("dd:MM:yyyy hh:mm:ss");
    return View();
}

[OutputCache(Duration = 3600, VaryByParam = "nom")]
public ActionResult VarierParParametre(string nom)
{
    ViewBag.Nom = nom;
    ViewBag.Date = DateTime.Now.ToString("dd:MM:yyyy hh:mm:ss");
    return View();
}

[OutputCache(Location = OutputCacheLocation.Client)]
public ActionResult CacheLocation(string nom)
{
    ViewBag.Nom = nom;
    return View();
}
```

Profil de cache

- Si un certain nombre d'actions doivent avoir l'attribut **OutputCache** configuré de la même manière à chaque fois, il est conseillé de factoriser l'ensemble.
  Cela peut se faire avec l'attribut apposé sur le contrôleur lui-même.

- **Un profil** peut également être défini dans le fichier Web.config.
  Ce profil doit alors simplement être référencé pour chaque action concernée. Le profil est créé dans la section web du fichier Web.config :

```
<caching>
  <outputCacheSettings>
    <outputCacheProfiles>
      <add name="MonProfil" duration="15" />
    </outputCacheProfiles>
  </outputCacheSettings>
</caching>
```

- Faire référence au profil :

```
[OutputCache(CacheProfile = "MonProfil")]
public ActionResult CacheProfile() {
    ViewBag.date = ViewBag.Date = DateTime.Now.ToString("dd:MM:yyyy hh:mm:ss");
    return View();
}
```

Sécurité

- Gestion de l'authentification :

  Assemblage WebMatrix.WebData
  - SimpleMembershipProvider et SimpleRoleProvider
  - La classe helper WebSecurity

- Connexion à des systèmes d'authentification

Plus d'informations sur http://www.dawan.fr
Contactez notre service commercial au **09.72.37.73.73**(prix d'un appel local)

DAWAN Paris, 11 rue Antoine Bourdelle, 75015 PARIS
DAWAN Nantes, 28 rue de Strasbourg, 44000 Nantes
DAWAN Lyon, Bâtiment de la Banque Rhône Alpes, 235 cours Lafayette, 69006 Lyon
DAWAN Lille, 16 Place du Général de Gaulle, 59800 Lille
formation@dawan.fr