

ASP.NET MVC

ASP.NET MVC

1. [Introduction](#)
2. [Structure d'une application MVC](#)
3. [Conventions ASP.NET MVC](#)
4. [Route et Navigation](#)
5. [Styles et Layout](#)
6. [Bundling et Minification](#)
7. [Les Contrôleurs ASP.NET MVC](#)
8. [Les Vues ASP.NET MVC](#)
9. [Les Modèles ASP.NET MVC](#)
10. [Razor](#)
11. [Formulaires et Helpers](#)
12. [Custom Helpers](#)
13. [Data Annotation et Validation](#)
14. [Custom Validation Logic](#)
15. [Display and Edit Annotation](#)

ASP.NET MVC










- 14. [Sessions et Application](#)
- 15. [Areas](#)
- 17. [Construire des application responsive ASP.NET MVC](#)
- 18. [Sécurité](#)
- 14. [Testing](#)
- 18. [Déploiement](#)

Introduction

ASP.NET MVC

Introduction

Microsoft fournit les technologies nécessaires pour vous permettre de créer des applications riches web et les publier sur internet ou sur intranet

Developpement	Hébergement	Langage	
		Server	Client
 	  		  

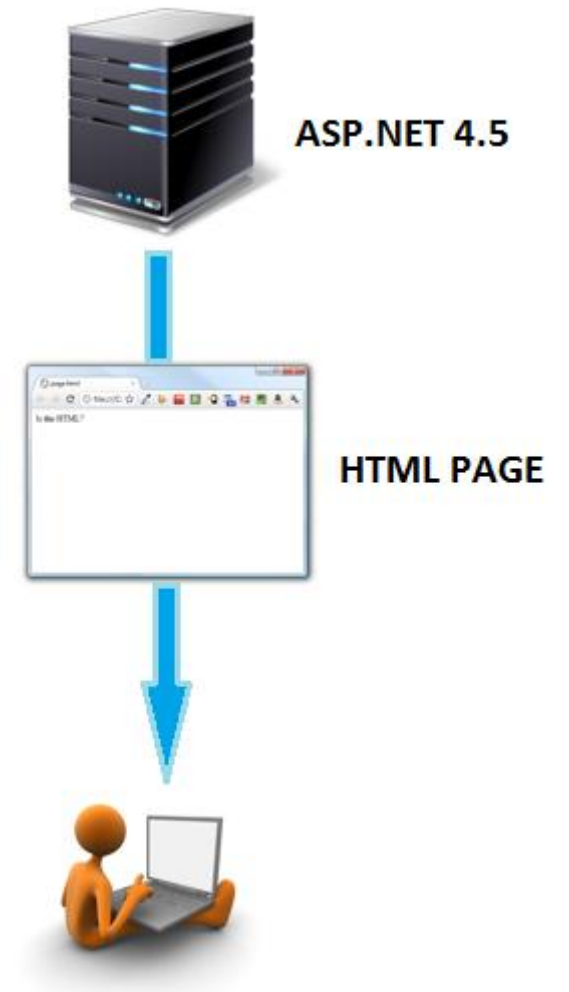
Petit tour de ASP.NET

Asp.net vous permet de développer des applications web tel que des portails, des blogs, des wikis, ...

1. Le model de programmation ASP.NET

- But des webforms

- ✓ Cacher HTTP & HTML en modélisant des interfaces via des contrôles serveurs.
- ✓ Chaque contrôle sauvegarde son propre état grâce au *ViewState* (sérialisation des données via http Request)
- ✓ Chaque contrôle génère son propre HTML permettant l'affichage web de celui-ci et connecte les événements client avec le code serveur correspondant.



Petit tour de ASP.NET

2. Les faiblesses des webforms

- Poids du ViewState

Le mécanisme du *ViewState* qui maintient l'état des contrôles entre les dialogues Client/Server entraine un transfert de larges blocs de données.

➔ Augmente le temps de réponse du server

➔ Augmente le besoin de bande passante

- Illusion de la séparation des couches

Le model *code-behind* permet de séparer la couche HTML du code applicatif.

Cependant, dans la plupart des cas, de part cette pseudo séparation, le développeur est tenté de mixer le côté *présentation* et la *logique applicative*.

- Limitation du contrôle du code HTML généré

L'HTML généré par nos contrôles serveur n'est pas nécessairement l'HTML que l'on souhaite et mettre en page via le css peut s'avérer ardu. De plus, le rendu sur certains navigateur échoue parfois de part le non respect des standards.

- Testing

Il est quasiment impossible de tester le design séparément du code applicatif et mettre en place des tests automatiques est un challenge.

ASP.NET MVC

- **Pattern MVC**

Le pattern MVC n'est pas neuf (1978) mais il a gagné en popularité dans le monde web pour les raisons suivantes :

- **Model**

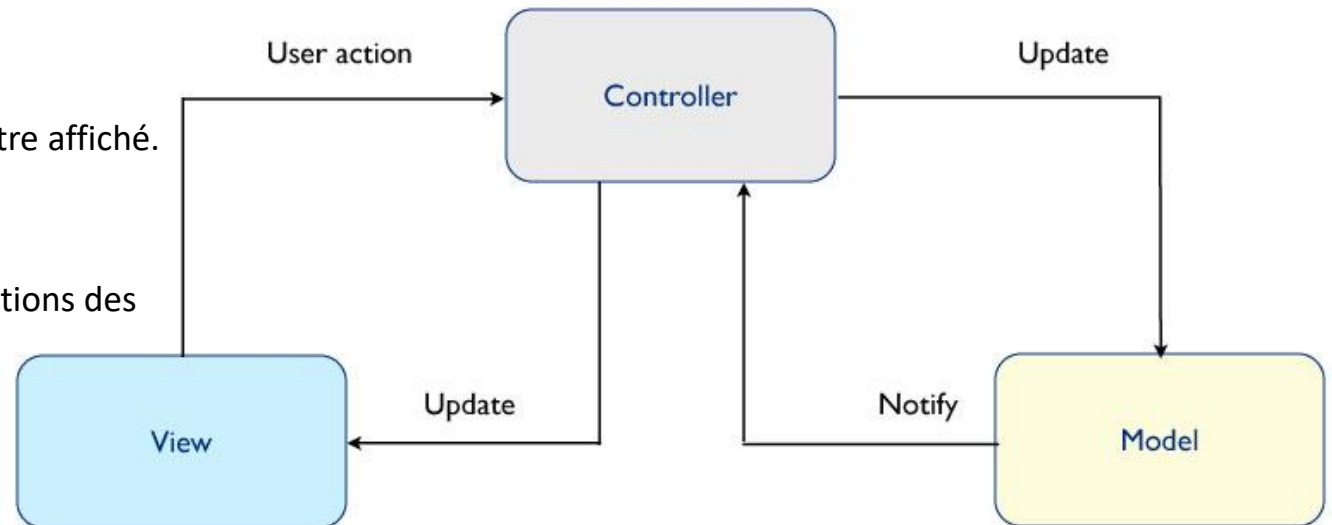
Classes qui décrivent les données sur lesquels nous travaillons. Les règles business qui définissent la façon dont les données doivent être manipulées, modifiées y sont également Implémentées.

- **View**

Définit comment l'interface utilisateur doit être affiché.

- **Controller**

Classes qui interagissent avec les communications des utilisateurs

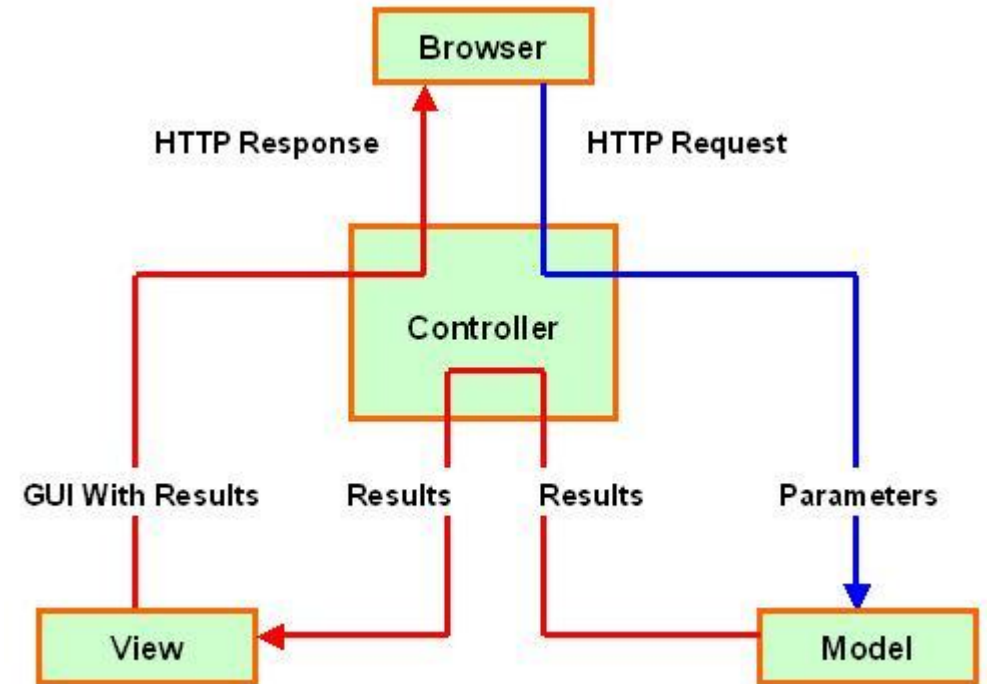


ASP.NET MVC

Le MVC appliqué aux frameworks Web.

ASP.NET MVC Framework implémente une version MVC spécialement repensée pour le monde web.

1. Les interactions utilisateur avec une application MVC un cycle naturel
L'utilisateur effectue une action et l'application répond en présentant une vue mise à jour à celui-ci. Très intéressant pour une application web se basant sur des *request* et *responses*.
2. Les applications web requièrent des combinaisons de plusieurs technologies (DB, HTML, Services,...) généralement décomposées en différentes couches



ASP.NET MVC

ASP.NET MVC contextualise le pattern mvc et traduit grossièrement les différentes couches comme suit :

- **Model**

Classes qui décrivent le domaine de notre application. Ces objets de domaine encapsulent souvent les données stockées en base de données ainsi que le code destiné à les manipuler et les règles business.

En ASP.NET, il fréquent que le modèle soit composé d'une couche de data (DAL) implémentée via EntityFramework ou Nhibernate et de code « custom » représentant les règles business.

- **View**

Il s'agit d'un template permettant la génération automatique d'HTML.

- **Controller**

Il s'agit d'une classe spéciale qui gère les relations entre les vues et le Modèle. Il répond aux entrées utilisateur et , au besoin, décide quel vue doit être générée.

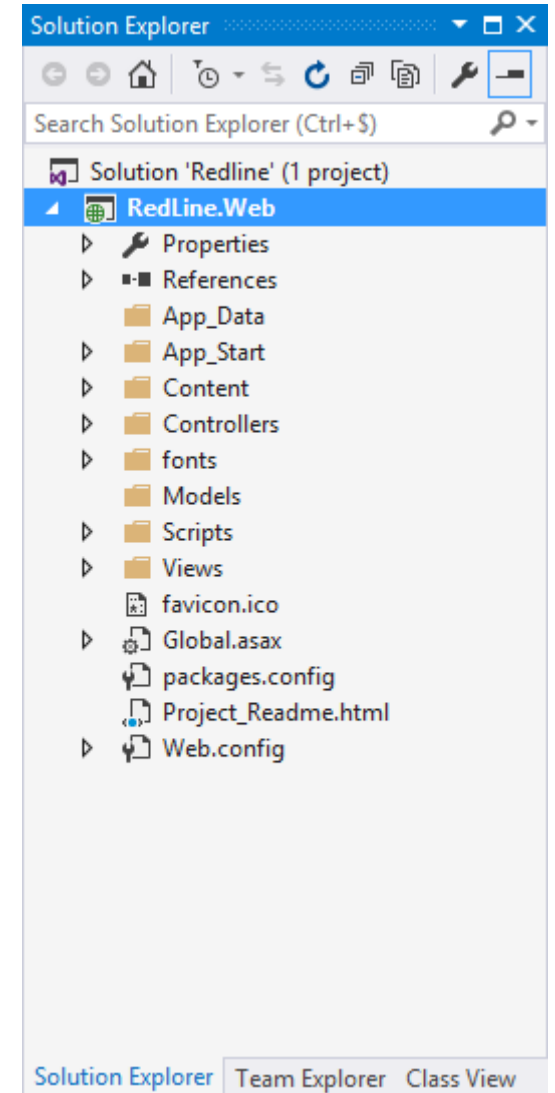
La convention veut que le nom de cette classe se termine par *Controller*.

Structure d'une application MVC

ASP.NET MVC

Structure d'une application MVC

Dossier	But
/Controllers	Dossier où se trouve les contrôleurs qui capturent les requêtes URL
/Models	Les classes qui représentent et manipulent les classes et les objets business
/Views	Les templates UI responsable de la génération des sorties, html
/Scripts	Scripts .js et librairies javascript
/fonts	Les polices (Fonts) de bootstrap
/Content	Pour déposer les css, images et autres contenus (pas les scripts)
/App_Data	Fichiers de données (Read/Write)
/App_Start	Fichiers de configuration pour le routing, bundling et Web API



Conventions ASP.NET MVC

ASP.NET MVC

Conventions ASP.NET MVC

MVC est conçu autour de valeurs par défaut sur la base de conventions qui peuvent être redéfinies au besoin.
Ce concept est communément appelé «convention over configuration »

“We know, by now, how to build a web application. Let’s roll that experience into the framework so we don’t have to configure absolutely everything again.”

Ruby On Rails

*« Nous savons , désormais, comment construire une application web .
Partageons cette expérience de sorte que nous ne devons pas configurer à nouveau absolument tout »*

Conventions ASP.NET MVC

- **Conventions over Configuration**

Ce concept est illustré par la présence des trois dossiers qui font que l'application peut fonctionner :

- Controllers
- Models
- Views

Ces dossiers ne sont pas présent dans notre *Web.config* (fichier de configuration de notre web app) car ils doivent juste être là par *convention* afin que le moteur ASP.MVC puisse retrouver ces éléments.

En d'autre termes, nous ne devons pas spécifier au moteur MVC que nos vues se trouvent dans le dossier *Views* puisque par *convention*, il le sait déjà.

Conventions ASP.NET MVC

Les conventions ASP.NET MVC sont plutôt simple :

- Chaque nom de classe de type Contrôleur, se termine par *Controller* (*HomeController, AccountController, ProductController,...*)
- Il n'a qu'un dossier *Views* pour toutes les vues de l'application
- Les vues utilisées par les contrôleurs doivent se trouver dans un sous-dossier de *Views* qui portera le nom du Contrôleur sans le suffixe. (*ProductController* aura ses vues dans */Views/Product*)
- Les interfaces utilisateurs réutilisables (*partial view,...*) se retrouvent dans le dossier *Views/Shared*

Route et Navigation

ASP.NET MVC

Route et Navigation

1. Uniform Resource Locator

Il est important pour une application web de fournir des URLs respectant les règles suivantes :

- Un nom de domaine facile à retenir et à épeler
- Elle doit être courte
- Elle doit représenter la structure du site
- Elle doit être immuable

Traditionnellement, l'url représente un fichier physique.

Exemple :

<http://example.com/albums/list.aspx> permet de déduire qu'un fichier list.aspx existe dans le dossier albums.

Il existe donc un lien direct entre l'url et les fichiers existants sur le disque.

Ce n'est pas le cas en MVC, où l'url est mappé à une méthode à exécuter plutôt qu'à fichier physique.

Route et Navigation

2. Routing ou Url rewriting?

L'url rewriting consiste à mapper une url sur une autre.

C'est une technique généralement utilisée pour mapper des urls peu explicite vers des urls plus précises.

Exemple :

<http://www.shop.com/Fiche.aspx?id=45&cat=22> → <http://www.shop.com/Sport/Velo/Fiche>

Le Routing consiste quand à lui à mapper une url vers une ressource, pas nécessairement une page web.

Il définit comment la requête est « dispatché » basé sur les caractéristiques de l'URL – il n'y a donc pas de réécriture.

Route et Navigation

3. Les différentes façon de mettre en place le routing

Les informations principales concernant le routing se trouve dans le dossier *App_Start*, dans le fichier *RouteConfig.cs*.

Nous n'avons pas à écrire toutes les combinaisons possibles pour les urls.

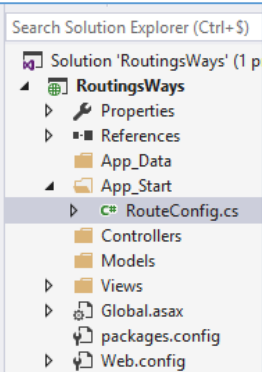
En effet, chaque route contient un Url pattern qui est comparé à l'url demandée.

Si le pattern correspond, il est utilisé pour résoudre l'url.

```
using System.Web.Routing;

namespace RoutingsWays
{
    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 }
            );
        }
    }
}
```



Route et Navigation

Request	Segments
http://mysite.com/Admin/Index	controller = Admin action = Index
http://mysite.com/Index/Admin	controller = Index action = Admin
http://mysite.com/Apples/Oranges	controller = Apples action = Oranges
http://mysite.com/Admin	Erreur - trop peu de segment
http://mysite.com/Admin/Index/Soccer	Erreur - trop de segment

Route et Navigation

Par défaut :

Mvc n'utilise le pattern que pour les urls qui correspondent exactement. Si il y a le bon nombre de segment, il utilise de manière littérale ceux-ci pour trouver le controller et l'action

4. Définir des valeurs par défaut

La valeur par défaut est utilisé lorsque l'url ne match avec aucun pattern

```
routes.MapRoute(  
    "Default", // Nom d'itinéraire  
    "{controller}/{action}/{id}", // URL avec des paramètres  
    new { controller = "Home", action = "Index", id = UrlParameter.Optional } // Paramètres par défaut  
);
```

Route et Navigation

Segment static

Tous les segments ne sont pas nécessairement variables.

Si nous désirons supporter l'url suivante

<http://mydomain.com/Public/Home/Index>

```
routes.MapRoute("", "Public/{controller}/{action}", new { controller = "Home", action = "Index" });
```

Remarque :

Les routes sont appliquées dans l'ordre de leurs déclarations dans le fichier Global.asax

Route et Navigation

5. L'attribut Route

Afin de définir une route, il est possible d'utiliser l'attribut route.

Celui-ci prend en argument un string appelé *Route Template*.

Il définit le mapping nécessaire pour atteindre l'action.

Si nous avons besoin de définir plusieurs route, il suffit de répéter l'attribut.

A ajouter dans le RouteConfig : **Attention à l'ordre des opérations!**

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

```
// Permet de mapper les routes créer avec le RouteAttribute
routes.MapMvcAttributeRoutes();
// a placer avant le MapRoute, maproute obsolète du coups...
```

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

Ne pas oublier d'y ajouter les route par défaut « Controller » / « Action »

```
public class HomeController : Controller
{
    // GET: Home
    [Route("")]
    [Route("home")]
    [Route("home/index")]
    public ActionResult Index()
    {
        return View();
    }
}
```


Route et Navigation

6. Contraintes personnelles

Il est également possible d'ajouter des contraintes dans les paramètres de la route.

Par exemple, le paramètre *Id* définit dans le routeconfig peut revêtir l'aspect d'un int, un double, un string,....

Nous pouvons définir une contrainte afin de forcer, par exemple, le type.

Deux choix s'offre à nous :

- Ajouter la contrainte dans le RouteConfig
- Implémenter IRouteConstraint

```
routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new
    {
        controller = "Home",
        action = "Index",
        id = UrlParameter.Optional
    },
    //Id uniquement numérique
    constraints: new { id = @"\d+" }
);
//Pour les controlleur commençant par H,
//Seul les action INdex et COntract sont valides
constraints: new { controller="^H.*",
                    action = "^Index$|^Contract$" }
```

Route et Navigation

```
public class IntRouteConstraint : IRouteConstraint
{
    public bool Match(HttpContextBase httpContext,
        Route route,
        string parameterName,
        RouteValueDictionary values,
        RouteDirection routeDirection)
    {
        int test = 0;
        return int.TryParse(values[parameterName].ToString(), out test);
    }
}
```

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

ception Fenêtre Commande Fenêtre Exécution Sortie Automatique Variables locales Espion 1