



Conception & Développement Informatique

ASP.NET MVC : HelloWorld

C#, Csharp, .NET Framework, Visual Studio

Mickaël DEVOLDÈRE

MD v1.0.1

23/04/2018



<http://www.arfp.asso.fr>





ASP.NET MVC : HelloWorld

C#, Csharp, .NET Framework, Visual Studio

Technologies et langages

	HTML
	CSS
	Javascript
	ASP.Net
	Microsoft SQLServer

Légende des icônes

	Information complémentaire
	Point d'attention particulier
	Intervention du formateur possible
	Lien vers une ressource externe

Sommaire

ASP.NET Kesako ?	3
ASP.NET MVC	3
Le pattern MVC : « Modèle View Controller »	4
La couche « Modèle »	5
La couche « Vue »	5
La couche « Contrôleur »	5
ASP.NET MVC	6
Création d'un projet ASP.NET MVC sous Visual Studio	6
Sélection du modèle d'application.....	7
Tester le projet vide	8
Mon 1 ^{er} Contrôleur : HomeController	9
La route	12
Ajouter du code HTML	16
Les vues	17
Historique du document	21
Crédits	21

ASP.NET Kesako ?

ASP.NET est la plateforme de développement de Microsoft permettant la réalisation d'applications web.

Une plateforme de développement est un ensemble de composants permettant de construire une application (ici, une application web) qui comprend :

- un outil de développement (exemple : Visual Studio)
- une boîte à fonctionnalités pour développer (exemple : Le framework .NET)
- une logique de développement (exemple : respect de la structure d'une application web)
- un langage de programmation (exemple : C#)

ASP propose 2 manières de concevoir une application Web :

- ASP WebForms
- ASP MVC

ASP.NET est le socle de la plateforme de développement de Microsoft pour réaliser des applications web. WebForms et MVC sont deux logiques différentes de développement ASP.NET.

Ce document traite **ASP.NET MVC**.

ASP.NET MVC

ASP.NET MVC a fait son apparition en 2009. La logique de réalisation des applications web s'appuie sur un patron de conception très célèbre, MVC : Modèle Vue Contrôleur.

Là où les WebForms offrent des composants « préfabriqués » afin de simplifier votre vie de développeur, ASP.NET MVC propose un cadre pour la conception d'applications web.

L'utilisation du design pattern « MVC » impose de suivre une ligne directrice qui guidera le développeur dans la réalisation d'une application web.

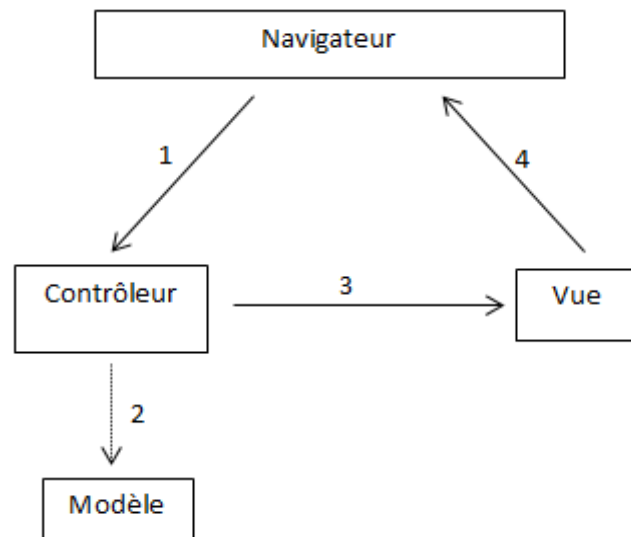
ASP.NET MVC donne également plus de liberté dans le rendu du HTML en permettant notamment d'utiliser des composants externes (Javascript, CSS, appels AJAX).

Le pattern MVC : « Modèle View Controller »

MVC est un patron de conception (Design Pattern) très répandu pour la conception de sites web. L'objectif de ce pattern est de séparer l'affichage, le contrôle et l'accès aux données.

MVC n'est associé à aucun langage de programmation et peut donc être utilisé avec ASP.NET, Java, PHP etc...

MVC permet de simplifier la maintenance du code et la mise à jour des fonctionnalités.



1. Action utilisateur via une requête HTTP
2. Consultation et/ou mise à jour du modèle (facultatif)
3. Le contrôleur décide de la vue à afficher
4. La vue renvoi le HTML au navigateur

La couche « Modèle »

Le M de MVC signifie « Modèle » (Model en Anglais). Il s'agit des données et de l'état de votre application web (L'utilisateur actuellement connecté, le panier de produits à acheter sur un site marchand, la liste des produits en vente...).

Ces données sont représentées sous forme de classe où chaque instance de ladite classe représente une occurrence parmi une liste (liste qui peut provenir d'une base de données ou d'un fichier sérialisé, par exemple).

Exemple :

Pour représenter chaque client enregistré en base de données, nous aurons une classe « Client » (et donc une instance de Client par occurrence) qui contiendra les attributs du client (nom, prénom, N° client ...), des méthodes pour obtenir telle ou telle information du client et des méthodes pour le modifier (changement d'adresse email par exemple).

La couche « Vue »

Le V de MVC signifie « Vue » (View en Anglais). Une vue est responsable de l'affichage des données issues des modèles (juste les afficher, sans interpréter leur contenu). De plus, une vue peut être composée d'une ou plusieurs « sous vues » lesquelles peuvent également agréger d'autres vues etc...

La couche « Vue » est la partie la plus simple à développer, surtout si vous maîtrisez HTML/CSS.

La couche « Contrôleur »

Le C de MVC signifie « Contrôleur » (controller en Anglais). Le contrôleur établit le lien entre la vue et le modèle. Le contrôleur est responsable des requêtes effectuées. Il interprète et contrôle les données en provenance de l'utilisateur et invoque la vue et les modèles nécessaires.

Comme on peut le voir dans le schéma précédent, le contrôleur est le point d'entrée de l'application web.

ASP.NET MVC : Contrôleur et Vue

Création d'un projet ASP.NET MVC sous Visual Studio

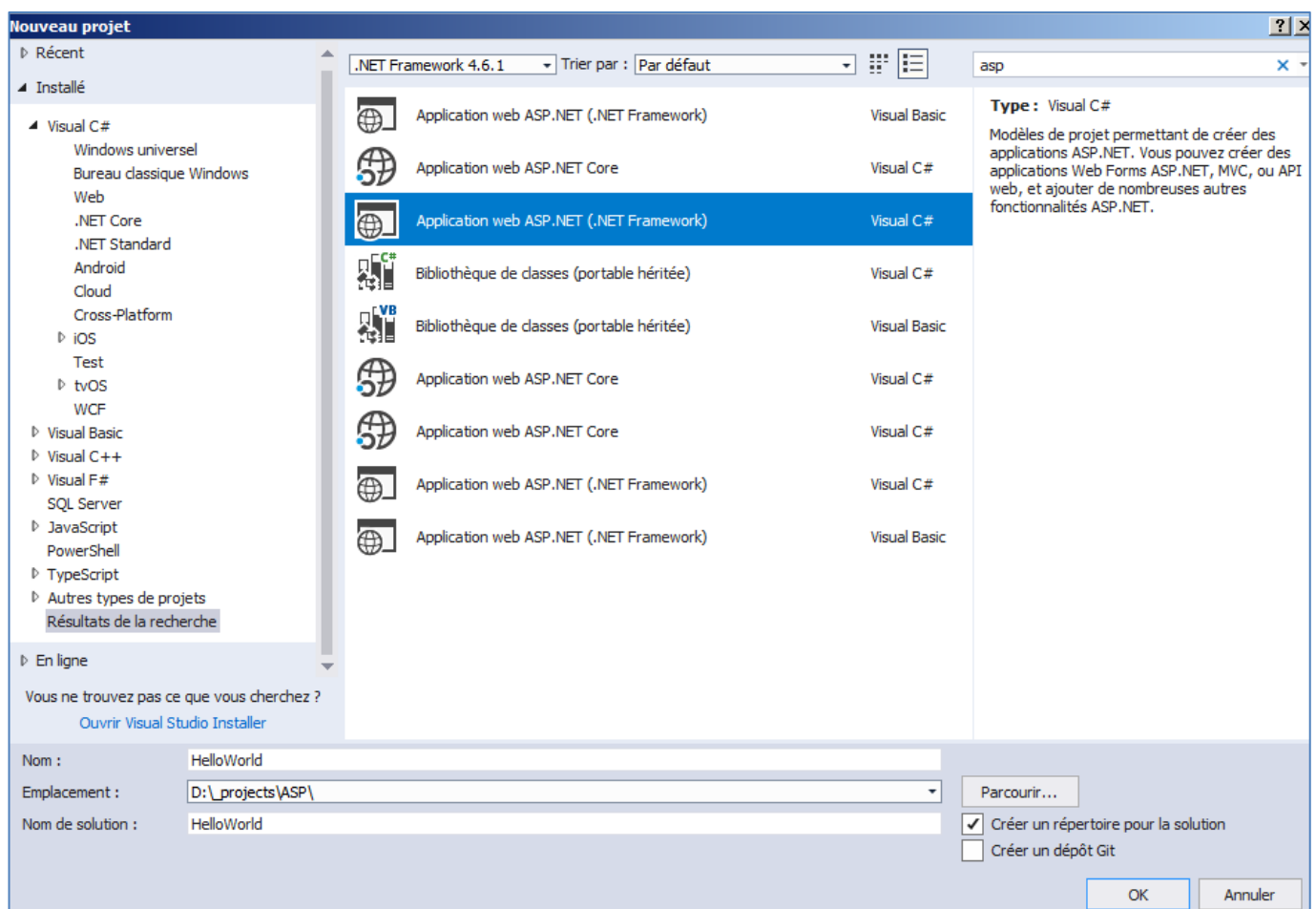
Dans Visual Studio, créer un nouveau Projet, puis dans les modèles, choisissez un projet Visual C#/Web de type « Application Web ASP.NET (NET Framework) ».



Veillez à bien sélectionner un projet « Visual C# » et non pas « Visual Basic »



Visual Studio 2017 a ajouté le type de projet « Application Web ASP.NET (NET Core) ». NET Core est une version « allégée » du NET Framework et l'approche est différente. Ce document traite bien du type de projet « Application Web ASP.NET (NET Framework) »

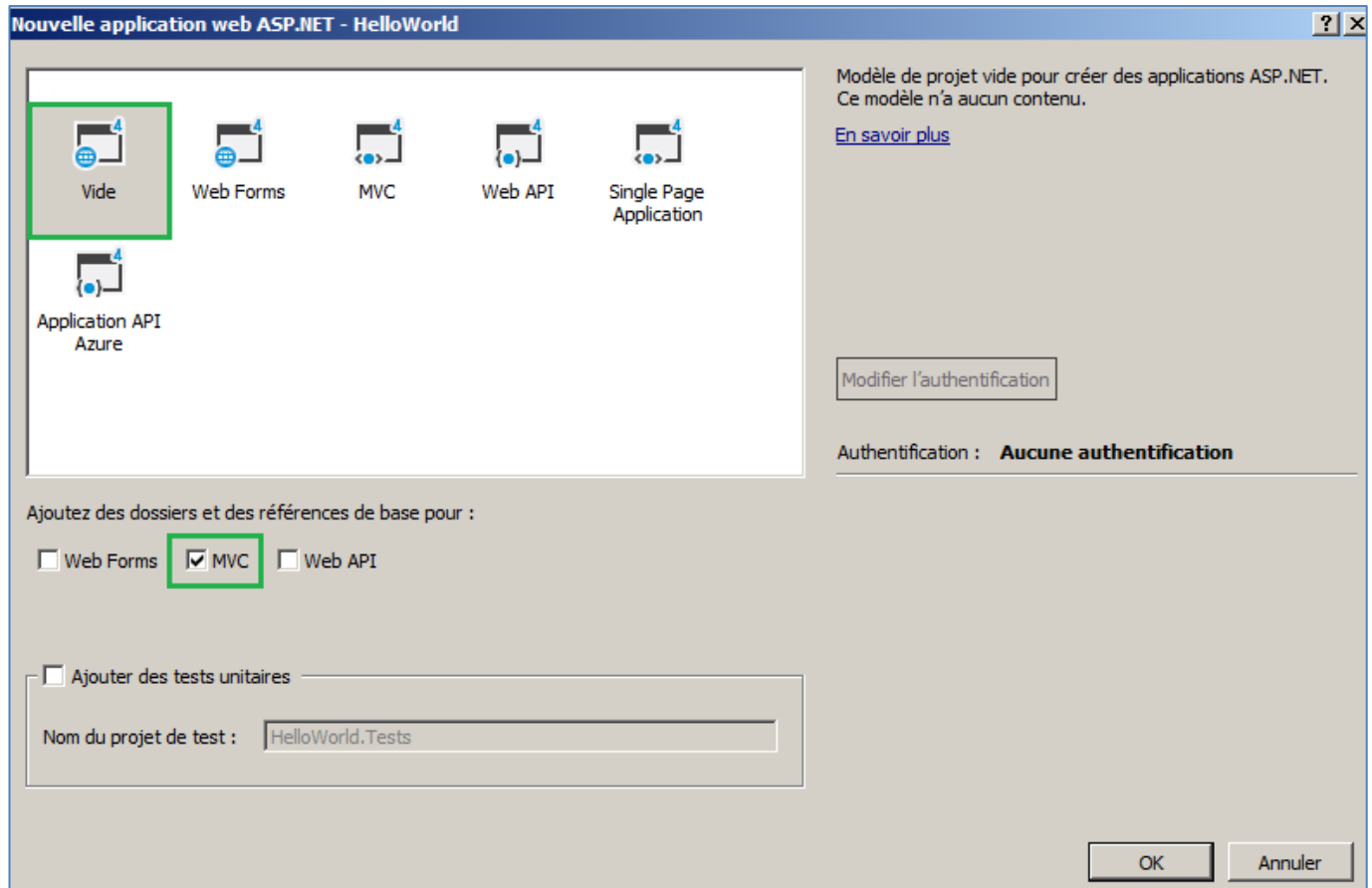


Nommer le projet « HelloWorld » puis cliquer sur le bouton « OK ».

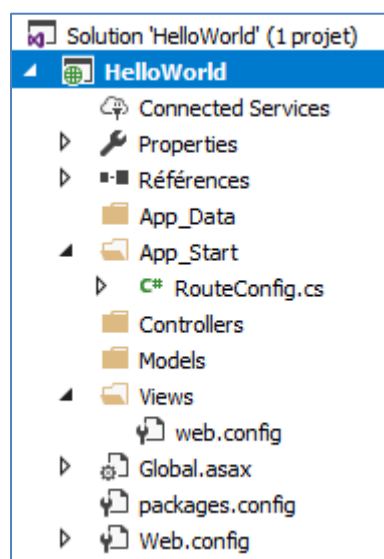
Sélection du modèle d'application

La fenêtre suivante vous invite à sélectionner un modèle.

Afin de bien assimiler le fonctionnement de ASP.NET MVC, sélectionner le modèle « empty » (vide) puis cocher la case « MVC ».



Le projet se crée et va générer quelques fichiers et dossiers. Vous pouvez dès à présent repérer 3 dossiers reprenant les noms des composantes du pattern MVC (Models, Views, Controllers)



La structure de fichiers et dossier générée correspond à la structure par défaut d'une application ASP.NET MVC.

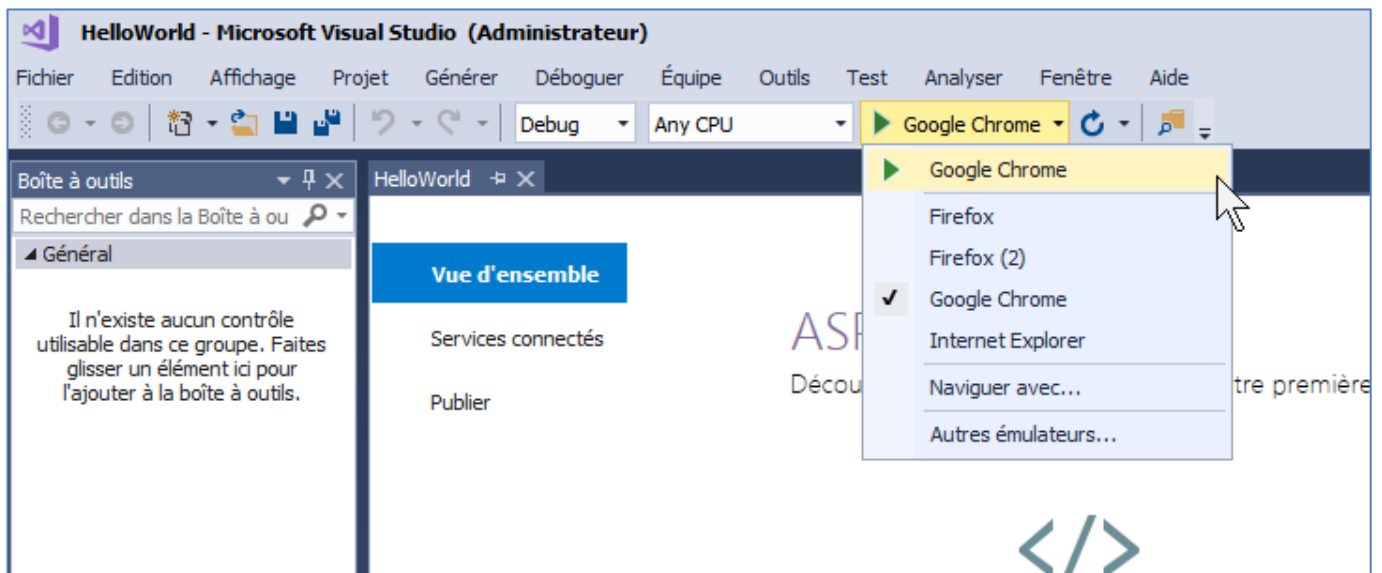
Répertoire « App_Data »	Répertoire dédié au stockage de données (fichiers de sérialisation, bases de données locales)
Répertoire « App_Start »	Contient le fichier Routeconfig.cs qui permet de configurer l'application
Répertoire « Models »	Stockage des classes « Modèle » de l'application
Répertoire « Views »	Stockage des vues de l'application et du fichier « web.config »
Répertoire « Controllers »	Stockage des contrôleurs de l'application
Fichier « Global.asax »	Fichier de configuration exécuté au démarrage de l'application
Fichier « packages.config »	Fichier de configuration des packages « NuGet »
Fichier « web.config »	Fichier à la racine du projet qui permet (encore) de définir des paramètres de configuration

Le projet ainsi créé, nous allons maintenant l'exécuter.

Tester le projet vide

En lieu et place du bouton « Démarrer » lorsque nous sommes dans un projet Windows Forms ou WPF se trouve un bouton indique le nom d'un navigateur Web (Chrome, Firefox...).

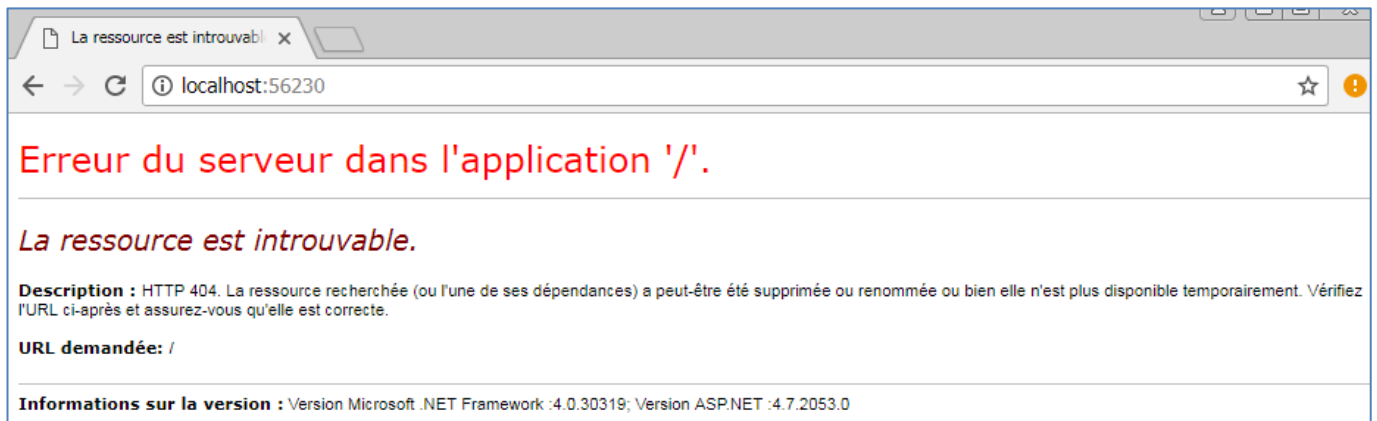
Cliquer sur la petite flèche sur la droite du bouton permet de sélectionner le navigateur que vous souhaitez utiliser pour tester votre application ASP.NET MVC.



Sélectionner un navigateur, puis lancer le projet.

Que remarquez vous ?

L'application affiche un « joli » message d'erreur ressemblant à ceci :



Que nous indique ce message d'erreur ?

Le message d'erreur nous indique que la ressource « / » est introuvable. Ceci est normal et est dû au fait que nous n'avons encore rien développé dans notre application web.

Mon 1^{er} Contrôleur : HomeController

Dans le projet Visual Studio, faire un clic droit sur le répertoire « Controllers » puis ajouter un nouveau contrôleur (Sélection dossier Controllers → clic droit → Ajouter → Contrôleur)

Une fenêtre s'ouvre alors pour vous proposer différents modèles de contrôleur.

Sélectionner le modèle « Contrôleur MVC 5 – Vide » puis le nommer « HomeController » (veillez à bien respecter la casse).

Le contrôleur est alors généré puis s'ouvre dans l'éditeur de code. Le fichier devrait ressembler à ceci :

```
7 namespace HelloWorld.Controllers
8 {
9     public class HomeController : Controller
10    {
11        // GET: Home
12        public ActionResult Index()
13        {
14            return View();
15        }
16    }
17 }
```

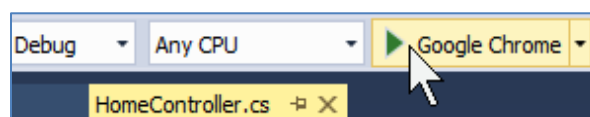


La création d'un contrôleur va automatiquement générer un répertoire portant le nom du contrôleur à l'intérieur du dossier « Views ». Dans notre cas, le dossier « /Views/House/ » est généré.

Analyse du contenu du fichier HomeController généré

Ligne	Code	Description
7	namespace HelloWorld.Controllers	Déclaration de l'espace de noms
9	public class HomeController : Controller	Déclaration de la classe « HomeController ». Notez bien que cette classe dérive de la classe « Controller »
12	public ActionResult Index()	Déclaration de la méthode « Index » qui renvoie un objet ActionResult par défaut
14	return View()	La méthode Index retourne le résultat de la méthode « View() »

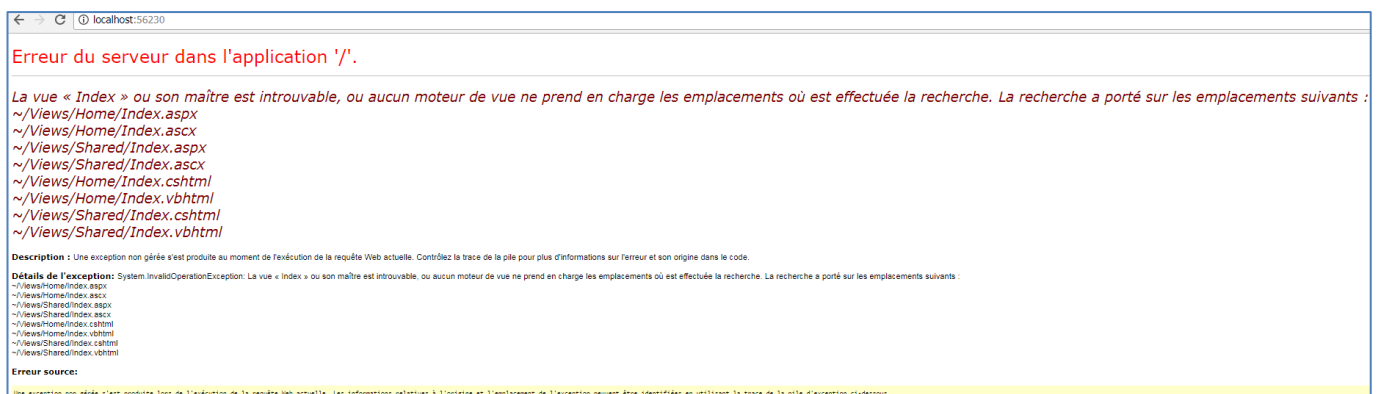
Une fois le contrôleur généré, lancer à nouveau l'application web.



Que remarquez vous ?

Hé oui, l'application a encore planté !

Le message d'erreur est par contre différent du précédent :



Cette fois-ci, le message nous indique que la vue « Index » est introuvable. La vue... du modèle MVC, bien sur ! Encore une fois, ceci est normal : nous n'avons pas encore créé de vue !

En analysant le message d'erreur, on voit apparaître une liste de fichiers. Ce sont les vues que ASP.NET MVC a tenté de charger (en vain), nous y reviendrons un peu plus loin.

Nous allons modifier le contrôleur HomeController et plus précisément la fonction Index().

Dans le fichier HomeController.cs, remplacez la fonction Index() actuelle

```
public ActionResult Index()  
{  
    return View();  
}
```

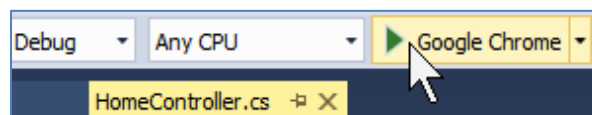
par le code suivant :

```
public string Index()  
{  
    return "Hello world";  
}
```

Notez que l'on a modifié :

- le type de retour de « ActionResult » à « string »
- la valeur retournée n'est plus un appel à la méthode View() mais une chaîne de caractère codée en dur

Une fois le contrôleur modifié, lancer à nouveau l'application web.



Hourra ! L'application web ne plante plus !

Que voyez vous à l'écran ?

La chaîne de caractère « Hello World » apparaît bien dans la fenêtre du navigateur. Afficher le code source de la page permet de vérifier que seule la chaîne de caractère a été renvoyée (aucun code HTML).

Que s'est-il passé ?

Garder la page web ouverte, puis ajouter à la fin de l'url « Home/Index » pour former un url ressemblant à <http://localhost:50230/Home/Index> (le port sur votre machine sera probablement différent), puis valider.

Que remarquez vous ?

L'affichage reste identique au précédent.

Nous avons simplement indiqué au framework MVC quel contrôleur exécuter (Home) et quelle action invoquer (Index).

La partie « /Home/Index » est ce qu'on appelle « une route ».

La route

Le routage est le mécanisme qui permet au framework MVC de déterminer le contrôleur à appeler et la méthode à exécuter.

Pour bien comprendre ce mécanisme, ouvrir le fichier « RouteConfig.cs » situé dans le répertoire « App_Start ».

Le contenu du fichier devrait ressembler à ceci :

```

10 public class RouteConfig
11 {
12     public static void RegisterRoutes(RouteCollection routes)
13     {
14         routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
15
16         routes.MapRoute(
17             name: "Default",
18             url: "{controller}/{action}/{id}",
19             defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
20         );
21     }
22 }

```

La partie du code qui nous intéresse ici commence à la ligne 16. Il s'agit de la définition du routage par défaut. Le framework MVC va utiliser les informations contenues dans l'attribut « url » pour déterminer les fichiers à exécuter.

On remarque que le framework attend des URL formatés où chaque élément est séparé par un « / ».

{controller}	Nom du contrôleur
{action}	Nom de la méthode dans le contrôleur
{id}	Paramètre optionnel récupérable dans le contrôleur

Les urls ressembleront à :

http://localhost:49819/	Appelle le contrôleur et l'action par défaut.
http://localhost:49819/Home/	Appelle le contrôleur « HomeController » puis l'action par défaut.
http://localhost:49819/Home/Liste	Appelle le contrôleur « HomeController » puis l'action « Liste() » dudit contrôleur.
http://localhost:49819/User/Details/3	Appelle le contrôleur « UserController » puis l'action « Details() » et enverra la valeur 3 en tant que paramètre (ID)

Si aucun élément n'est fourni, c'est la route définie à la ligne 19 qui est appelée.

De ce fait, les URL suivantes :

- http://localhost:49926/
- http://localhost:49926/Home/
- http://localhost:49926/Home/Index/

sont identiques.

Décomposition de l'URL

Soit l'url : <http://localhost:49926/Home/Index> découpée en trois parties:

http://localhost:49926/Home/Index

http://localhost:49926	Nom de domaine + port	Adresse pour accéder à l'application Web.
Home {controller}	Nom du contrôleur à appeler (insensible à la casse)	Le framework ajoute « Controller » et recherche le fichier « HomeController.cs » dans le répertoire « Controllers ». Si le contrôleur correspondant n'est pas trouvé, une erreur 404 est soulevée.
Index {action}	Nom de l'action à invoquer dans le contrôleur appelé (insensible à la casse)	Une fois le contrôleur chargé, le framework va y exécuter la méthode portant le même nom que l'action invoquée. Si la méthode est introuvable, une erreur 404 est soulevée.
{id}	Paramètre optionnel (sensible à la casse)	Ce paramètre peut être récupéré dans la méthode invoquée.

1 = Nom du contrôleur à charger
2 = Nom de la méthode à exécuter DANS le contrôleur

```

7 namespace HelloWorld.Controllers
8 {
9
10     public class HomeController : Controller
11     {
12         // GET: Home
13         public string Index()
14         {
15             return "Hello world";
16         }
17     }
18 }

```

Dans l'url, si vous remplacez « Index » par autre chose, l'application soulèvera une erreur car aucune autre méthode n'est (pour le moment) implémentée dans HomeController.

Exercices #1

Ajoutez maintenant une méthode qui affichera « Bonjour les développeurs » et qui correspondra à l'url suivant :

- <http://localhost:49926/Home/Bonjour>

Implémentez cette méthode de la même manière que HomeController.Index().

Testez !

Ajoutez ensuite une méthode qui affichera « Salut les concepteurs » et qui correspondra à l'url suivant :

- <http://localhost:49926/House/Salut>

Implémentez cette méthode de la même manière que HomeController.Index().

Testez !

Le paramètre optionnel « id »

Il est possible d'envoyer un paramètre via l'url (voir : paramètres d'URL dans le document « ASP Introduction »).

Le système de routage du framework MVC permet de passer ce paramètre dans l'url sans devoir le construire avec la paire clé=valeur (type `index.php?id=1`)

Reprendre le code de la méthode `HomeController.Salut()` et la modifier comme ci-dessous :

```

17 public string Salut(string id)
18 {
19     if(String.IsNullOrEmpty(id))
20     {
21         id = "les concepteurs";
22     }
23
24     return "Salut " + id;
25 }

```

Nous avons ajouté un paramètre à la méthode. Ce paramètre correspond à la troisième partie du système de routage : `http://localhost:50241/Contrôleur/Action/Paramètre`

Démarrer l'application puis se rendre sur l'url correspondant :

- <http://localhost:50632/House/Salut>

Puis ajouter un prénom à la fin de l'url :

- <http://localhost:50632/House/Salut/Mickaël>

Le paramètre a été récupéré à partir de l'url et évalué dans la méthode `HomeController.Salut()`.

1 = Nom du contrôleur à charger
2 = Nom de la méthode à exécuter DANS le contrôleur
3 = Paramètre URL {id} récupéré en tant que paramètre de méthode

```

9 public class HomeController : Controller
10 {
11     // GET: House
12     public ActionResult Index() ...
18
19     // GET: House/Salut/{id}
20     public string Salut(string id)
21     {
22         if(String.IsNullOrEmpty(id))
23         {
24             id = "les concepteurs";
25         }
26
27         return "Salut " + id;
28     }

```


Ajouter du code HTML

Afficher des chaînes de caractères, c'est bien (sisi, c'est bien) mais l'objectif premier d'une application web est généralement de renvoyer du code HTML qui sera interprété par le navigateur des utilisateurs.

Dans le contrôleur « HouseController », ajouter la méthode « Liste() » qui retourne une chaîne de caractère (de la même manière que les autres méthodes créées précédemment) à la différence que nous allons renvoyer du code HTML.

Puis, implémenter la méthode comme ci-dessous :

```
22 public string Liste()  
23 {  
24     return @"<html>  
25         <head>  
26             <meta charset=""UTF-8"">  
27             <title>Liste des courses</title>  
28         </head>  
29         <body>  
30             <h1>Liste de courses du " + DateTime.Now.ToLongDateString() + @"</h1>  
31             <ul>  
32                 <li>Eau</li>  
33                 <li>Lait</li>  
34                 <li>Dernier album de Nanamoukouri</li>  
35                 <li>Chevilles &Oslash; 8</li>  
36                 <li>Matériel camping</li>  
37             </ul>  
38         </body>";  
39     }  
40 }
```

Démarrer le projet puis se rendre à l'adresse correspondante sachant que nous sommes dans le contrôleur « House » et que l'action est le nom de la méthode ci-dessus.

Remarquez que le code HTML a bien été interprété par le navigateur (click droit dans la page puis Afficher la source).

Séparation du code C# et du code HTML : Les vues

Générer du code HTML en dur directement dans le contrôleur fonctionne mais ne respecte pas le principe de séparation imposé par le patron de conception MVC.

Pour respecter le principe de séparation des couches, le code HTML destiné à l'affichage doit être exporté dans un fichier à part : la vue.

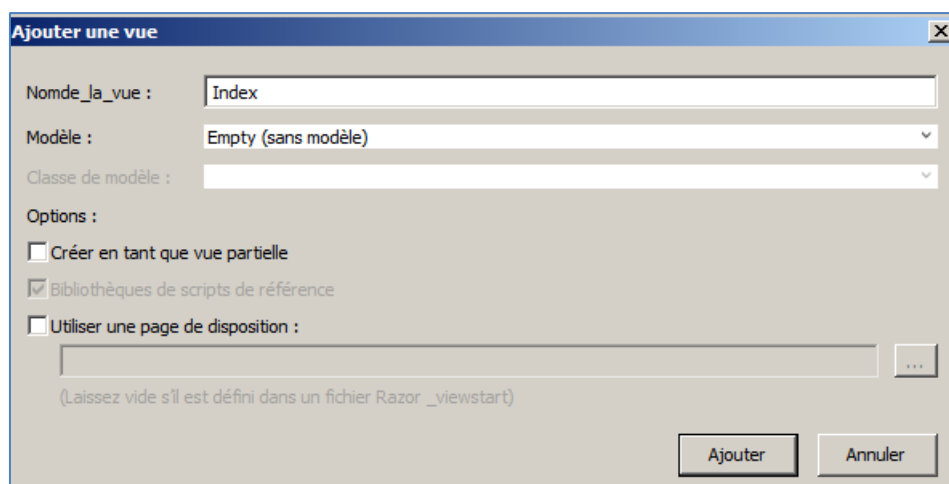
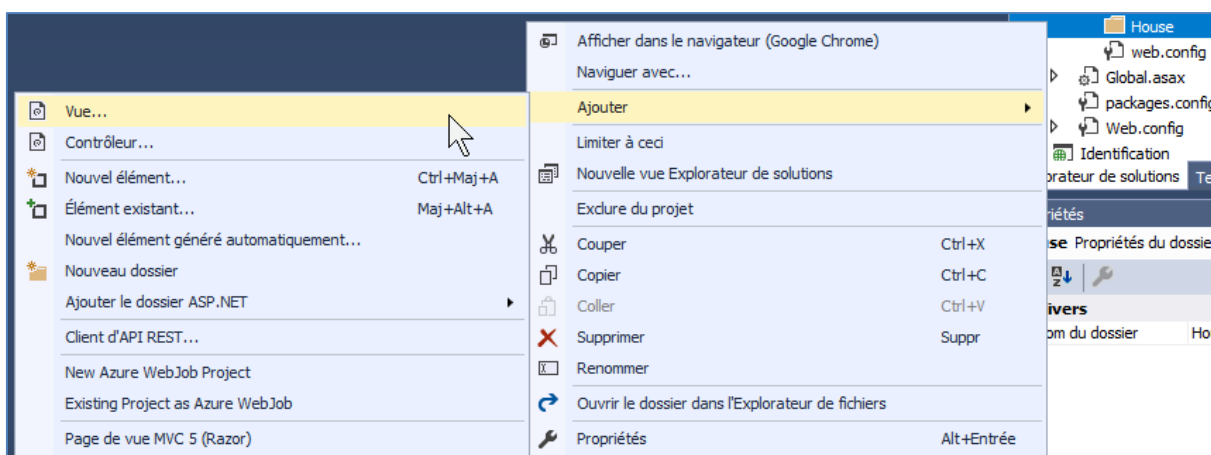
Dans un projet ASP.NET MVC, les vues peuvent être stockées à plusieurs endroits :

- Dans le répertoire ~/Views/Shared/ pour les vues utilisables par plusieurs contrôleurs.
- Dans le répertoire ~/Views/Nom_Du_Contrôleur/ pour les vues associées à 1 contrôleur.

Par exemple, les vues associées au contrôleur « HomeController », seront stockées dans « ~/Views/Home/ ».

Les vues sont des fichiers portant une extension particulière : « .cshtml ».

Dans le répertoire « ~/Views/Home/ », créer une nouvelle vue nommée « Index »



VisualStudio va alors générer un fichier « Index.cshtml » qui contient une base HTML que nous pouvons (et devons) alimenter.

```
1  @{
2      Layout = null;
3  }
4
5  <!DOCTYPE html>
6
7  <html>
8  <head>
9      <meta name="viewport" content="width=device-width" />
10     <title>Index</title>
11 </head>
12 <body>
13     <div>
14     </div>
15 </body>
16 </html>
17
```

Pour le moment, c'est la partie « HTML » qui nous intéresse. Ne vous occupez pas du code en ligne 1 à 4, nous y reviendrons plus tard.

Dans le fichier « Views/House/Index.cshtml », ajouter un titre de niveau 1 « Index du contrôleur House ».

Démarrer le projet et se rendre à l'url correspondant à l'action « Index » du contrôleur « House » .

Que s'est-il passé ?

Le code HTML du fichier Index.cshtml est affiché ! Afficher le code source de la page nous permet de vérifier que le code HTML bien été interprété par le navigateur.

Reprenons maintenant la méthode « Index() » du contrôleur « HomeController » qui, pour rappel, ressemble à ceci :

```
12 public ActionResult Index()  
13 {  
14     return View();  
15 }
```

La méthode « View() » permet de charger une vue et de lui transmettre des données.

L'appel de la méthode « View() » sans paramètre va dynamiquement rechercher une vue dont le nom correspond au nom de la méthode. Cette recherche s'effectue dans 2 répertoires distincts et dans cet ordre: « /Views/Nom_Du_Contrôleur/ » et « /Views/Shared/ » et porte sur plusieurs extensions de fichiers dans un ordre défini.

Dans notre cas, l'appel à la méthode View() sans paramètre dans « HomeController.Index() » implique la recherche suivante :

1. ~/Views/House/index.aspx
2. ~/Views/House/ index.ascx
3. ~/Views/Shared/ index.aspx
4. ~/Views/Shared/ index.ascx
5. ~/Views/House/ index.cshtml
6. ~/Views/House/ index.vbhtml
7. ~/Views/Shared/ index.cshtml
8. ~/Views/Shared/ index.vbhtml

Nous ne nous étendrons pas sur les différentes extensions mais sachez qu'un projet ASP.NET MVC est également utilisable avec les fichiers « .aspx » de ASP WebForms et peut également être développé avec le langage Visual Basic (les vues portent alors l'extension .vbhtml).

Si nous avions voulu créer une vue liée à la méthode « HomeController.Bonjour() », le fichier à ajouter serait nommé « ~/Views/House/Bonjour.cshtml ».

Bravo, vous avez réalisé une application Web avec ASP.NET MVC.

Nous avons implémenté 2 parties du modèles MVC : Le contrôleur et la vue.

Dans le cours suivant, nous aborderons les points suivants :

- La couche « Modèle »
- La page de disposition
- Les vues partielles
- La gestion des formulaires

--- Fin du document ---

Historique du document

Auteur	Date	Observations
Mickaël DEVOLDÈRE	12/04/2018	Création du document
Mickaël DEVOLDÈRE	24/04/2018	Ajout section « Requêtes http »
Mickaël DEVOLDÈRE	16/05/2018	Ajout schémas ASP.NET MVC

Crédits

Ce document est largement inspiré :

- du MOOC d'openclassrooms « Apprendre ASP.NET MVC »
 - o <https://openclassrooms.com/courses/apprendre-asp-net-mvc>
- du MOOC d'openclassrooms « Créez votre site Web avec ASP.NET »
 - o <https://openclassrooms.com/courses/creez-votre-site-web-en-asp-net>

Consultez les liens ci-dessus pour voir la version originale (plus détaillée sur certains points) de ce document.