|  |  |  |
| --- | --- | --- |
|  | |  | | --- | | [Asp .Net Core MVC]  [Layachi Mohammed Achraf]  [Asp.Net Core MVC] [Aziz] | |

# C’EST QUOI Mvc ?

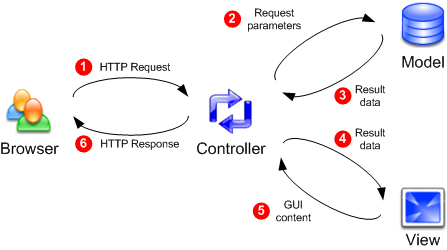
Le «Design Pattern » MVC a été créé par Trygve Reenskaug en 1978. Le nom original est thing model view editor pattern, puis il a été rapidement renommé model-view-controller pattern. MVC a été utilisé la première fois pour créer des interfaces graphiques avec le langage de programmation Smalltalk en 1980.

* **Donc c’est quoi ASP .NET Core MVC ?**

ASP.NET Core est un Framework multiplateforme à hautes performances et open source pour créer des applications cloud modernes et connectées à Internet.

Le modèle d’architecture MODEL-VIEW-CONTROLLER (MVC) sépare une application en trois groupes de composants principaux :

* + M : MODEL : Le MODEL dans une application MVC représente l’état de l’application et de n'importe quelles logiques ou opérations qui doivent être effectuées par celui-ci, autrement dit c’est la manipulation et la gestion de données.
  + V : VIEW : La VIEW est chargée de présenter du contenu via l’interface utilisateur. Elle utilise un [moteur d’affichage](https://docs.microsoft.com/fr-fr/aspnet/core/mvc/overview#razor-view-engine) pour incorporer le code .NET dans le balisage HTML. Donc c’est une sorte de présentation ou bien un IU pour l’utilisateur.
  + C : CONTROLLER : Le CONTROLLER c’est le composant qui gère les interactions avec l’utilisateur, utilise le MODEL et finalement sélectionne une vue à restituer. Dans une application MVC, la VIEW affiche uniquement les informations ; le CONTROLLER gère et répond à la saisie de l’utilisateur et à l’interaction. Dans le modèle MVC, le CONTROLLER est le point d’entrée initial, il est chargé de sélectionner les types de MODEL avec lesquels travailler et la VIEW à restituer (d'où son nom - il contrôle la manière dont l’application répond à une requête donnée).
* **Comment ca fonctionne MVC ?**



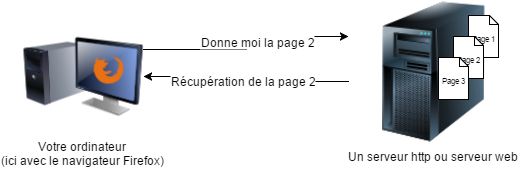
Dans le diagramme on constat que l’utilisateur réalise une requête auprès de navigateur pour que ce dernier envois les requêtes HTTP au Controller pour qu’il demande des données au Model et après la collection de données le Controller envois les données nécessaires au View pour que ce dernier traite les données et les transformer en page html pour que le Controller les envois comme des réponse HTTP au navigateur pour que ce dernier les affiche à l’utilisateur.

* **C’est quoi HTTP ?**

On sais maintenant que le MVC utilise les http pour comuniquer avec l’utilisateur donc c’est quoi au just les http ?

Pour surfer sur le web, vous utilisez un navigateur Internet. En fonction de vos clicks ou de vos saisies, celui-ci va appeler des pages contenues sur des serveurs, les récupérer et les afficher, exactement comme on vien de comprendre avec le MVC en haut.

Voici tout cela résumé par un petit schéma sans prise en compte le MVC:



Comme on peut le constater, c’est très simple. Sur le schéma, et pour simplifier, j’ai représenté 2 étapes. Le navigateur demande la page puis la récupère. *(je pricise encore que cela sans parler du MVC)*

Le serveur attend les connexions et les demandes, il se contente de répondre à la demande en envoyant du contenu. (*le concepte est mieux expliquer dans le MVC*).

* **Parlons sur le .NET de quoi sagi-t-il ?**

Le .Net c’est un Framework pouvant être utilisé par un système d'exploitation [Microsoft Windows](https://fr.wikipedia.org/wiki/Microsoft_Windows) et Microsoft Windows Mobile depuis la version 5.

Il a pour but de faciliter la tâche des développeurs en proposant une approche unifiée à la conception d'applications Windows ou Web, tout en introduisant des facilités pour le développement, le déploiement et la maintenance d'applications. Il a besoin d'être installé sur la machine de l'utilisateur final, rendant les applications créées sous cet environnement impropres à un usage portable.

* **Le .Net Core**

.NET Core est une plateforme de développement généraliste qui est tenue à jour par Microsoft et la communauté .NET sur GitHub. Cette multiplateforme(*Cross-platform*) prend en charge Windows, macOS et Linux. Elle peut être utilisée dans des scénarios d’appareil, de cloud et d’incorporation/IoT.

Voici les caractéristiques qui définissent le mieux .NET Core :

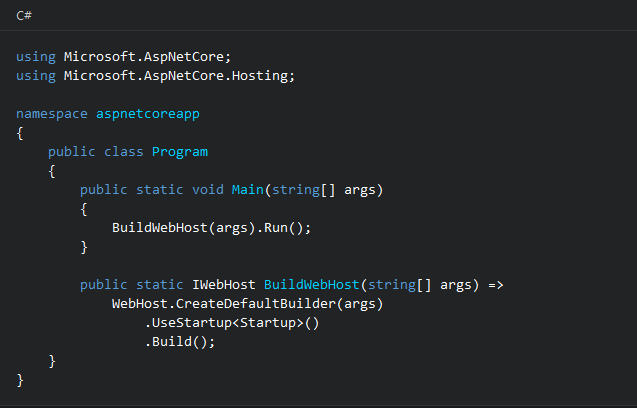
* + **Souplesse de déploiement** **:** peut être inclus dans votre application ou installé côte à côte à l’échelle d’un utilisateur ou de l’ordinateur.
  + **Multiplateforme :** s’exécute sur Windows, macOS et Linux ; peut être porté sur d’autres systèmes d’exploitation. Au fil du temps, les scénarios pris en charge associant [systèmes d’exploitation](https://github.com/dotnet/core/blob/master/roadmap.md), processeurs et applications se multiplieront, qu’ils soient proposés par Microsoft, d’autres sociétés ou des individus.
  + **Outils en ligne de commande :** tous les scénarios de produit peuvent être mis en œuvre au niveau de la commande.
  + **Compatibilité :** .NET Core est compatible avec le .NET Framework, Xamarin et Mono, via [.NET Standard](https://docs.microsoft.com/fr-fr/dotnet/standard/net-standard).
  + **Open Source :** la plateforme .NET Core est open source et utilise des licences MIT et Apache 2. La documentation est concédée sous licence [CC-BY](https://creativecommons.org/licenses/by/4.0/). .NET Core est un projet [.NET Foundation](https://dotnetfoundation.org/).
  + **Support Microsoft** : le support technique de .NET Core est assuré par Microsoft, via le [Support .NET Core](https://www.microsoft.com/net/core/support/).

# Structure d’une application asp .net core MVC

Dans une application Asp.Net Core MVC on trouve plusieurs fichiers, dossiers et classes. Donc de quoi se compose une application Asp.Net Core MVC ?

* **La class Program.cs**

Une application ASP.NET Core est une application console qui crée un serveur web dans sa méthode Main Dans la classe Program.



La méthode Main appelle WebHost.CreateDefaultBuilder, qui suit le modèle du générateur pour créer un hôte d’application web. Le générateur a des méthodes qui définissent le serveur web (par exemple UseKestrel) et la classe de démarrage (UseStartup). L’hôte web d’ASP.NET Core tente de s’exécuter sur IIS, s’il est disponible. D’autres serveurs web, tels que HTTP.sys, peuvent être utilisés via l’appel de la méthode d’extension appropriée.

IWebHostBuilder, le type de retour de l’appel de WebHost.CreateDefaultBuilder, fournit de nombreuses méthodes facultatives. Certaines de ces méthodes incluent UseHttpSys pour l’hébergement de l’application dans HTTP.sys et UseContentRoot pour la spécification du répertoire de contenu racine. Les méthodes Build et Run génèrent l’objet IWebHost, qui héberge l’application et démarre l’écoute des requêtes HTTP.

* + **IIS c’est quoi au juste ?**

Ou bien ***Internet Information Services***, communément appelé **IIS** *(prononcé "2 i s")*, est un serveur Web (FTP, SMTP, HTTP, etc.) des différents systèmes d'exploitation Windows NT.

IIS prend en charge plusieurs techniques Web telles les CGI, les ASP, les ASP .NET et une API spécifique à IIS de nom ISAPI permettant de créer des extensions et des filtres. IIS prend aussi en charge le langage PHP en mode CGI ou ISAPI.

* + **Kestrel Server :**

Kestrel est un serveur web multiplateforme pour ASP.NET Core basé sur libuv, bibliothèque d’E/S asynchrones multiplateforme. Kestrel est le serveur web inclus par défaut dans les modèles de projets ASP.NET Core.

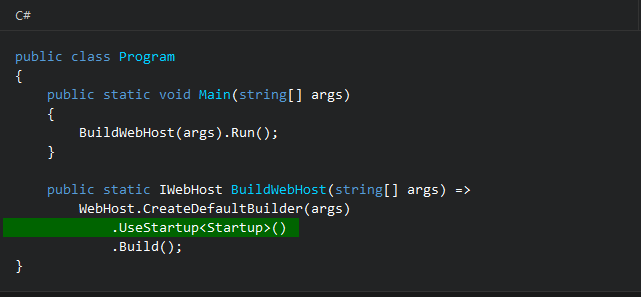
Kestrel prend en charge les fonctionnalités suivantes :

* HTTPS
* Mise à niveau opaque utilisée pour activer les [WebSockets](https://github.com/aspnet/websockets)
* Sockets UNIX pour des performances élevées derrière Nginx

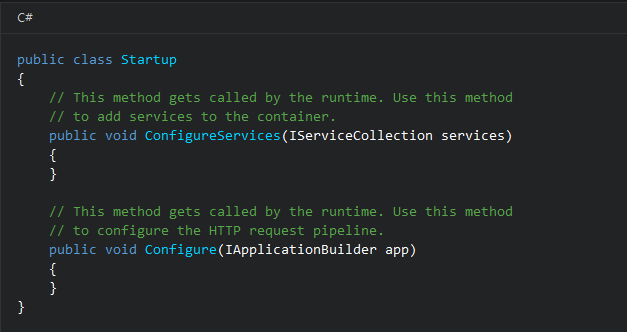
Kestrel est pris en charge sur toutes les plateformes et les versions prises en charge par .NET Core.

* + Démarrage ou bien Startup.cs

La méthode UseStartup sur WebHostBuilder spécifie la classe Startup pour l’application :



La classe Startup est l’emplacement où vous définissez la canalisation(pipeline) de traitement des requêtes et où sont configurés les services nécessaires à l’application. La classe Startup doit être publique et contenir les méthodes suivantes :



ConfigureServices définit les [services](https://docs.microsoft.com/fr-fr/aspnet/core/fundamentals/?tabs=aspnetcore2x#dependency-injection-services) utilisés par votre application (par exemple, ASP.NET Core MVC, Entity Framework Core et Identity). Configure définit les [intergiciels (middleware)](https://docs.microsoft.com/fr-fr/aspnet/core/fundamentals/middleware/index) de la canalisation de requêtes.

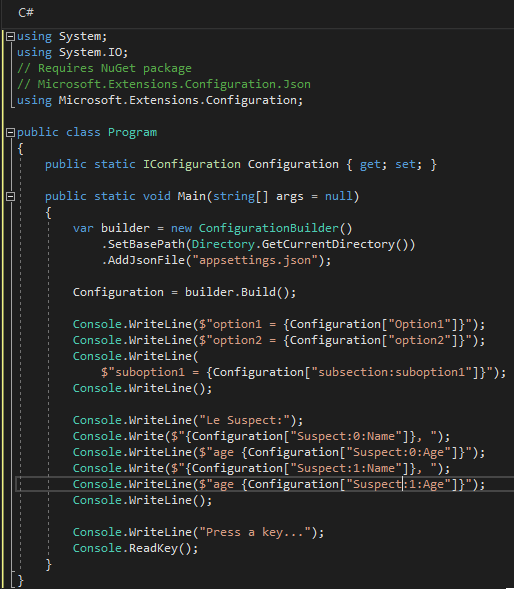
* **Configuration**

La configuration de Asp .Net Core web app est basé sur une liste de paires non/valeur, elle est lue au moment de l’exécution à partir de plusieurs sources. Les paires nom/valeur peuvent être regroupées dans une hiérarchie à plusieurs niveaux.

Il existe des fournisseurs de configuration pour les éléments suivants :

* Formats de fichiers (INI, JSON et XML).
* Arguments de ligne de commande.
* Variables d'environnement.
* Objets .NET en mémoire.
* Stockage [Secret Manager](https://docs.microsoft.com/fr-fr/aspnet/core/security/app-secrets) non chiffré.
* Magasin utilisateur chiffré comme Azure Key Vault.
* Fournisseurs personnalisés (installés ou créés).
* **Exemple d’une configuration JSON :**

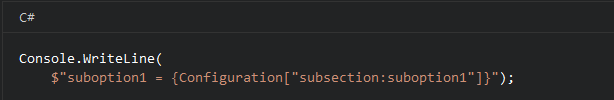
L’application console suivante utilise le fournisseur de configuration JSON :



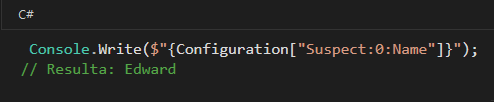
L’application lit et affiche les paramètres de configuration suivants :



La configuration se compose d’une liste hiérarchique des paires nom/valeur dans laquelle les nœuds sont séparés par un signe deux-points. Pour récupérer une valeur, accédez à l’indexeur Configuration avec la clé de l’élément correspondant :



Pour utiliser des tableaux dans des sources de configuration au format JSON, utilisez un index de tableau comme partie d’une chaîne séparée par des signes deux-points. L’exemple suivant obtient le nom du premier élément dans le tableau Suspect précédent :



L’exemple utilise l’indexeur de configuration pour lire des valeurs. Pour accéder à la configuration en dehors de Startup, utilisez le *modèle d’options.*

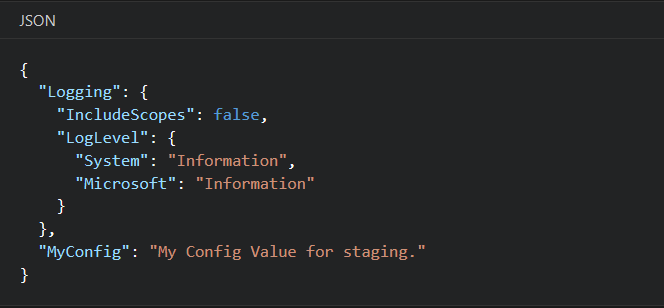
* **Configuration par environnement :**

Il est courant d’avoir des paramètres de configuration différents pour différents environnements, par exemple pour l’environnement de développement, de test et de production. La méthode d’extension CreateDefaultBuilder dans une application ASP.NET Core 2.x (ou l’utilisation de AddJsonFile et de AddEnvironmentVariables directement dans une application ASP.NET Core 1.x) ajoute des fournisseurs de configuration pour la lecture des fichiers JSON et des sources de configuration système :

* appsettings.json
* appsettings.<nom\_environnement>.json
* Variables d’environnement

Les sources de configuration sont lues dans l’ordre où elles sont spécifiées. Dans le code précédent, les variables d’environnement sont lues en dernier. Toutes les valeurs de configuration définies dans l’environnement remplacent celles définies dans les deux fournisseurs précédents.

Considérez le fichier appsettings.Developement.json suivant :



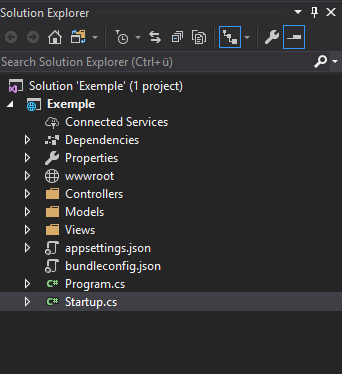
Lorsque l’environnement a la valeur Staging, la méthode Configure suivante lit la valeur de MyConfig :



L’environnement est généralement défini sur Development, Staging ou Production.

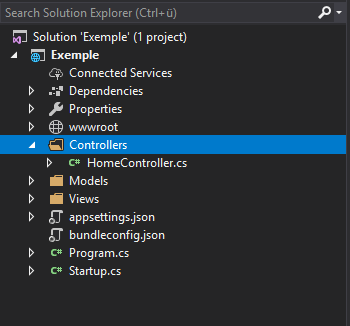
* **Application Asp .Net Core MVC :**

On a beaucoup parler sur l’Asp .Net Core MVC application mais sans un aperçoit a quoi sa ressemble dans Visual Studio, voilà de quoi ce compose cette fameuse MVC :

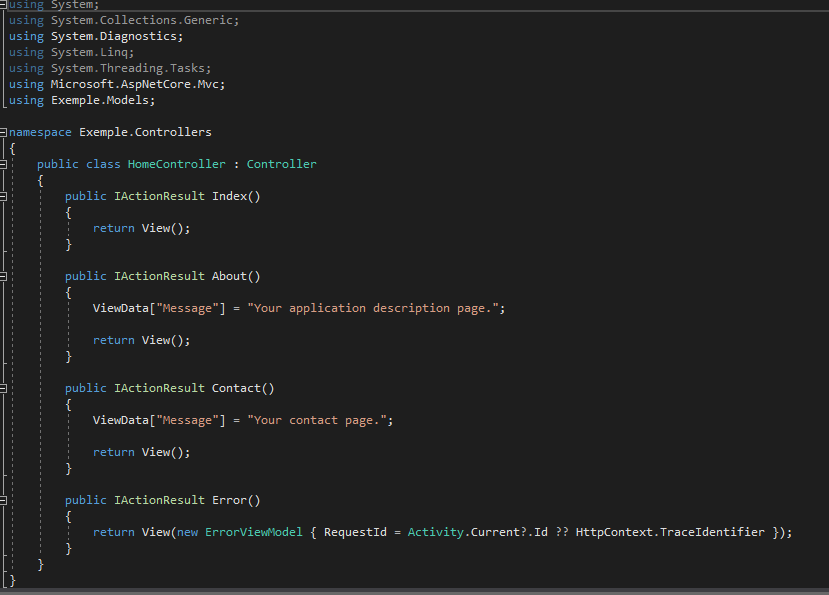


On a parler du Startup.cs et de Program.cs aussi on a parlé sur les fichiers de configuration JSON, donc ce qui attire l’attention c’est les trois dossiers « Controllers » « Model » et « Views ». Faisons un tour dans chaque dossier.

* **Dossier Controllers :**

****

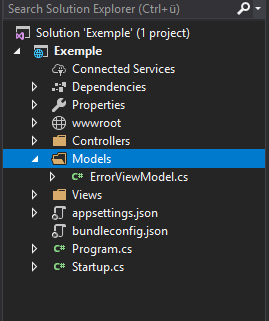
On constate que le dossier Controllers contient un fichier .cs autrement dit une classe, oui les contrôleurs sont des class apparts entiers. Par contre un contrôleur peut exister dans l’application sans le dossier mais c’est mieux organisé comme ca et c’est aussi par défaut a la création d’une application MVC Core.



Comme on peu voir HomeController et une classe avec des méthodes et des fonctions comme n’importe quelle autre classe dans C#.

Passons maintenant au dossier suivant.

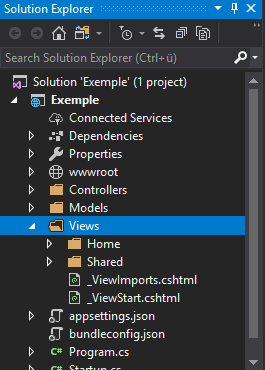
* **Dossier Models :**

****

Aussi le dossier Models contient une classe Model et aussi ca peut exister sans le dossier Models mais encore une fois c’est une question d’organisation et aussi c’est par défaut a la création d’une application MVC.

* **Dossier Views :**

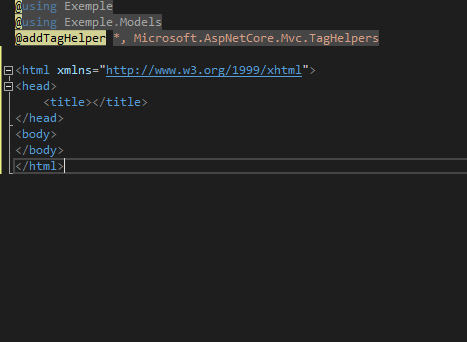
Dans le dossier Views les chose sont un peu différents, voilà à quoi ressemble le dossier Views :



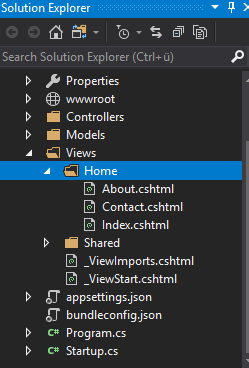
Déjà on constate qu’il y a plusieurs dossiers dedans mais avant parler de ces dossiers voyons c’est quoi le fichier .cshtml.

Avec Asp .Net Core on peut utiliser du html fusionner avec le C# voilà l’exemple :

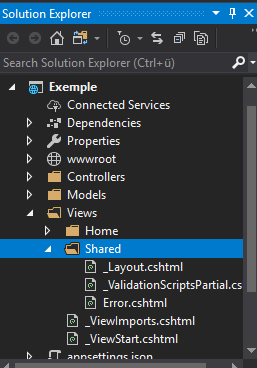
Dans l’image on constate bien qu’il y a des using dans un fichier html, en effet il n’y a pas que ça y a beaucoup plus que ça, on va découvrir tout ça dans les parties qui suivent.



Revenons aux dossiers Home et Shared. Le dossier Home comme son nom l’indique il s’agit de la page Home ou accueil d’un site web et dedans on va trouver aussi des fichier cshtml donc c’est des Views qui ont comme rôle la gestion de la page Home que l’utilisateur va voir et aussi quelque page comme About et Contact.



Pour le dossier Shared, il s’agit d’un ensemble de Views qui sont partager par l’ensemble de l’application par exemple la page erreur et partagée comme ca on facilite la tache et au lieu de créer des dizaines de page erreur une seul page erreur sera assez suffisante pour gérer tous les messages d’erreur dans notre application.



Par contre il faut noter que la View ne peut pas exister dehors le dossier Views, c’est pas le même cas comme les Models et les Controllers.