Web Côté Serveur
ASP.Net MVC

Introduction

Lors du développement d'applications, nous avons souvent le choix entre différents frameworks et architectures en fonction du langage dans lequel nous développons.

De nombreux frameworks s'appuient désormais sur l'architecture dite « MVC ».

Le modèle architectural MVC existe depuis longtemps en génie logiciel. La plupart des langages utilisent MVC avec une légère variation, mais sur le plan conceptuel, elles restent les mêmes.

Présentation

Le MVC («Modèle Vue Contrôleur ») est une architecture de développement visant à séparer le code source en modules.

En effet, ce modèle très répandu, consiste à séparer distinctement l'accès aux données (bases de données), la vue affichée à l'utilisateur et la logique métier.

Cette architecture est le plus communément retrouvée au sein d'applications web mais existe également au niveau des applications lourdes.

Architecture MVC

MVC sépare les applications en trois composants: modèle, vue et contrôleur.

Modèle : Le modèle représente la forme des données et la logique métier. Il maintient les données de l'application. Les objets de modèle récupèrent et stockent l'état du modèle dans une base de données.

Le modèle est une logique de données et d'entreprise.

Vue : View est une interface utilisateur. Afficher les données d'affichage en utilisant le modèle pour l'utilisateur et leur permet également de modifier les données.

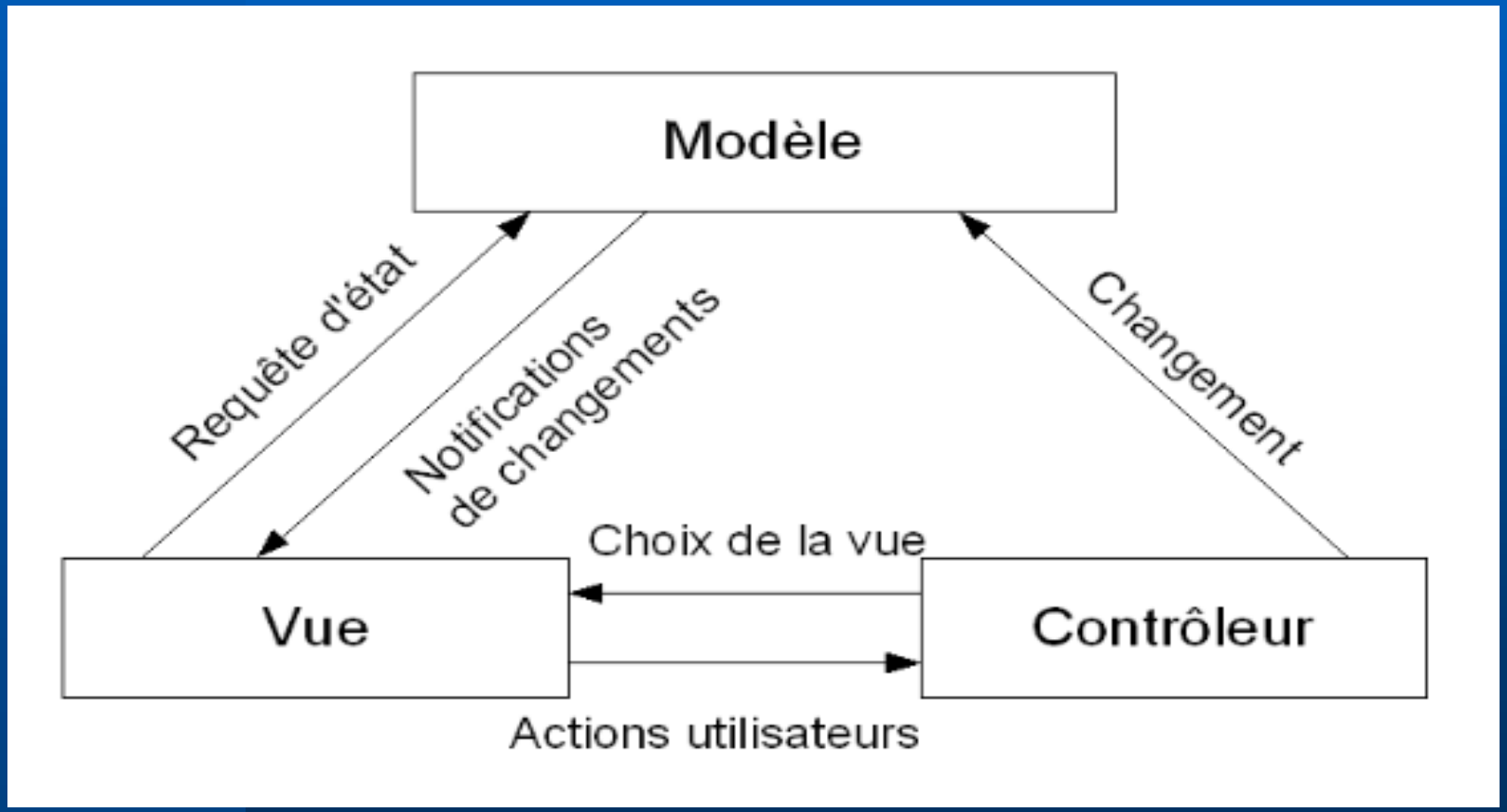
La vue est une interface utilisateur.

Architecture MVC

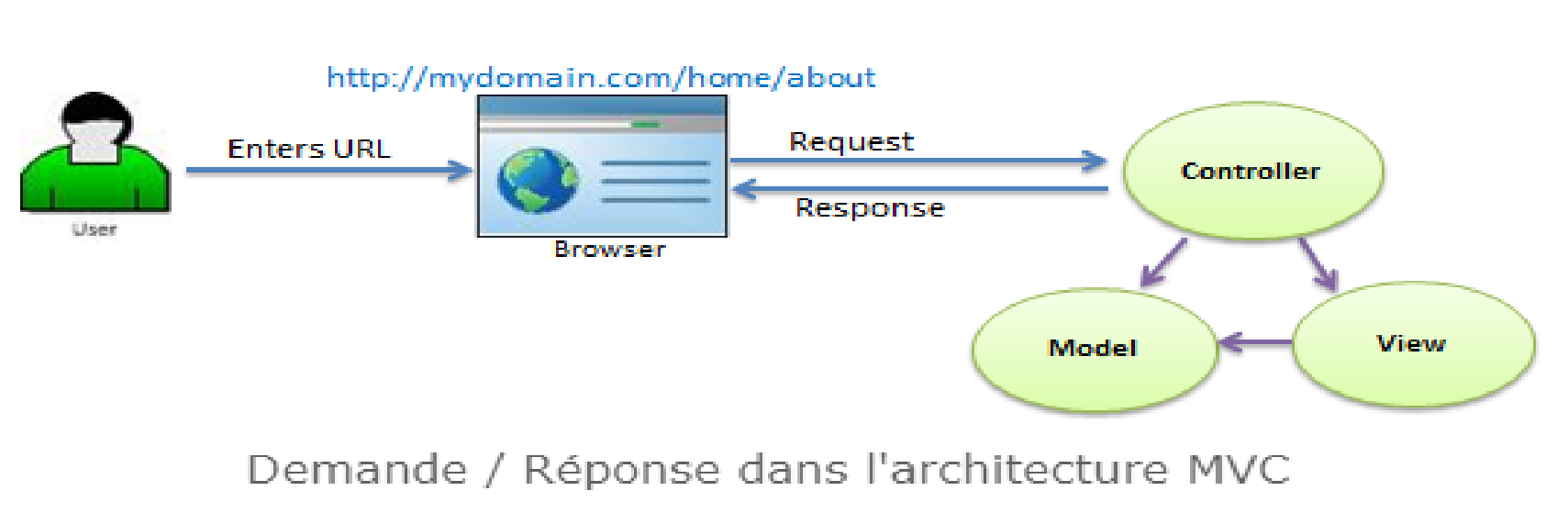
Contrôleur : le contrôleur gère la demande de l'utilisateur. En règle générale, l'utilisateur interagit avec View, ce qui déclenche à son tour la demande d'URL appropriée. Cette demande sera gérée par un contrôleur. Le contrôleur rend la vue appropriée avec les données du modèle en réponse.

Le contrôleur est un gestionnaire de demandes.

Architecture MVC



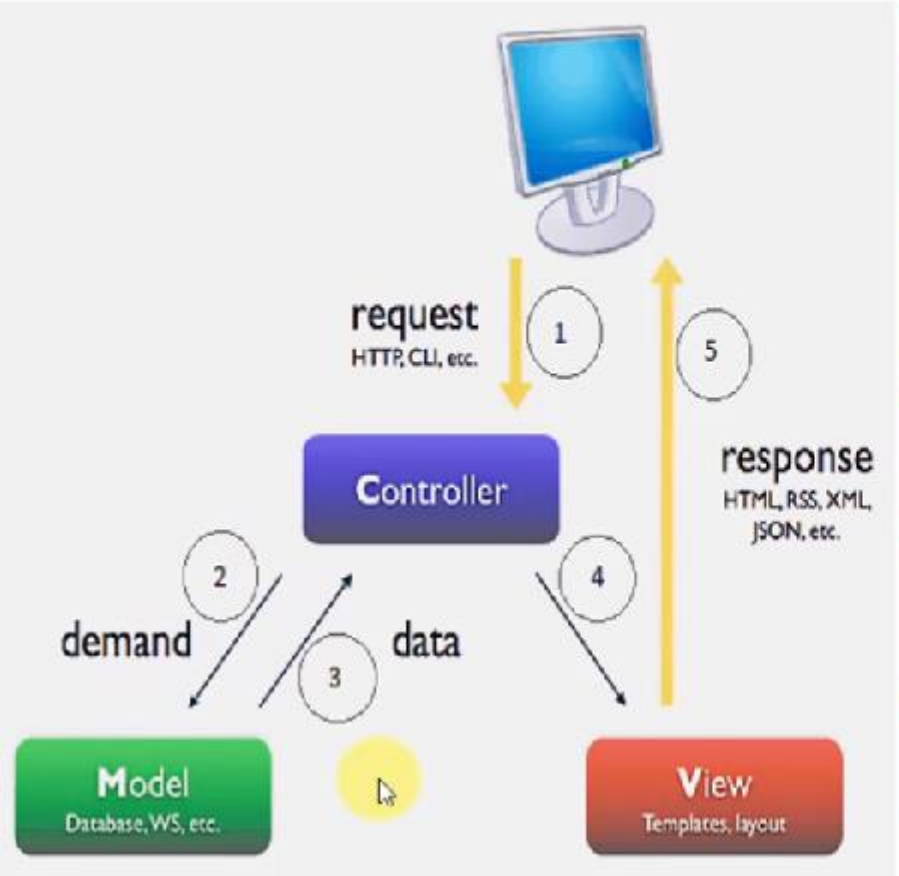
Le principe de fonctionnement



Demande / Réponse dans l'architecture MVC

Conformément à la figure ci-dessus, lorsque l'utilisateur entre une adresse URL dans le navigateur, celui-ci se connecte au serveur et appelle le contrôleur approprié. Ensuite, le contrôleur utilise la vue et le modèle appropriés, crée la réponse et la renvoie à l'utilisateur. Nous verrons les détails de l'interaction dans les prochaines sections..

Le principe de fonctionnement



MVC

01 L'utilisateur envoie une requête HTTP

02 Le contrôleur appelle le modèle, celui-ci va récupérer les données

03 Le modèle retourne les données au contrôleur.

04 Le contrôleur décide de la vue à afficher, va l'appeler.

05 Le code HTML de la vue est envoyé à l'utilisateur pour qu'il puisse naviguer normalement.

Les avantages et inconvénients

Cette architecture apporte plusieurs avantages lors de la création et de la mise en place d'un projet.

Elle facilite la maintenance et les évolutions futures. En effet, étant donné qu'il y a une séparation entre *les différentes couches, il sera plus facile de modifier uniquement la partie vue ou encore uniquement le traitement de la requête dans le contrôleur.*

Dans le cadre de structures avec plusieurs développeurs, il sera plus simple d'avoir un développeur front-end et un développeur back-end travaillant sur le même projet étant donné que les fichiers sont séparés.

Cependant, même si beaucoup de frameworks MVC existent, cette architecture nécessite directement une plus grosse architecture étant donné qu'il y aura 3 fois plus de fichiers. Ainsi, pour les petits projets, cela semble potentiellement inutile.

Quelques exemples frameworks

Nous avons vu la théorie mais passons maintenant à la pratique ! Afin de pouvoir tester cette architecture et l'utiliser dans vos futurs projets, voici une liste non-exhaustive de frameworks MVC pour différents langages :

- *PHP : Laravel, Symfony, CodeIgniter*
- *C# : ASP .NET MVC*
- *Java : Spring MVC, Struts*
- *JavaScript : AngularJS, Ember.js*
- *Python : Django, TurboGears*

Historique de version ASP.NET MVC

Version MVC	Visual Studio	Version .Net	Date de sortie	Caractéristiques
MVC 1.0	VS2008	.Net 3.5	13 mars 2009	•Architecture MVC avec moteur de formulaire Web •Routage •Assistants HTML •Aides Ajax •Reliure automatique
MVC 2.0	VS 2008,	.Net 3.5 / 4.0	10 mars 2010	•Surface •Contrôleur asynchrone •Méthodes d'assistance HTML avec expression lambda •Attributs DataAnnotations •Validation côté client •Modèle personnalisé •Échafaudage
MVC 3.0	VS 2010	.Net 4.0	13 janvier 2011	•Validation javascript discrète •Moteur de vue rasoir •Filtres globaux •Validation à distance •Résolveur de dépendance pour IoC •ViewBag

Historique de version ASP.NET MVC

Version MVC	Visual Studio	Version .Net	Date de sortie	Caractéristiques
MVC 4.0	VS 2010 SP1, VS 2012	.NET 4.0 / 4.5	15 août 2012	•Modèle de projet mobile •Regroupement et minification •Prise en charge de Windows Azure SDK
MVC 5.0	VS 2013	.NET 4.5	17-oct-2013	•Filtres d'authentification •Prise en charge de Bootstrap •Nouveaux articles d'échafaudage •Identité ASP.Net
MVC 5.2 - Courant	VS 2013	.NET 4.5	28 août 2014	•Routage basé sur les attributs •corrections de bugs et fonctionnalités mineures

Conclusion

L'architecture MVC est donc intéressante à utiliser et permet de soigner son architecture logicielle, mais n'est pas forcément adaptée à des petits projets en raison de sa complexité et du nombre de fichiers à créer.

Son utilité peut parfois sembler inexistante au début d'un projet mais son besoin se montre rapidement lorsque celui-ci gagne en taille.

C'est donc une bonne idée de prendre l'habitude de l'utiliser lors du développement de projets !

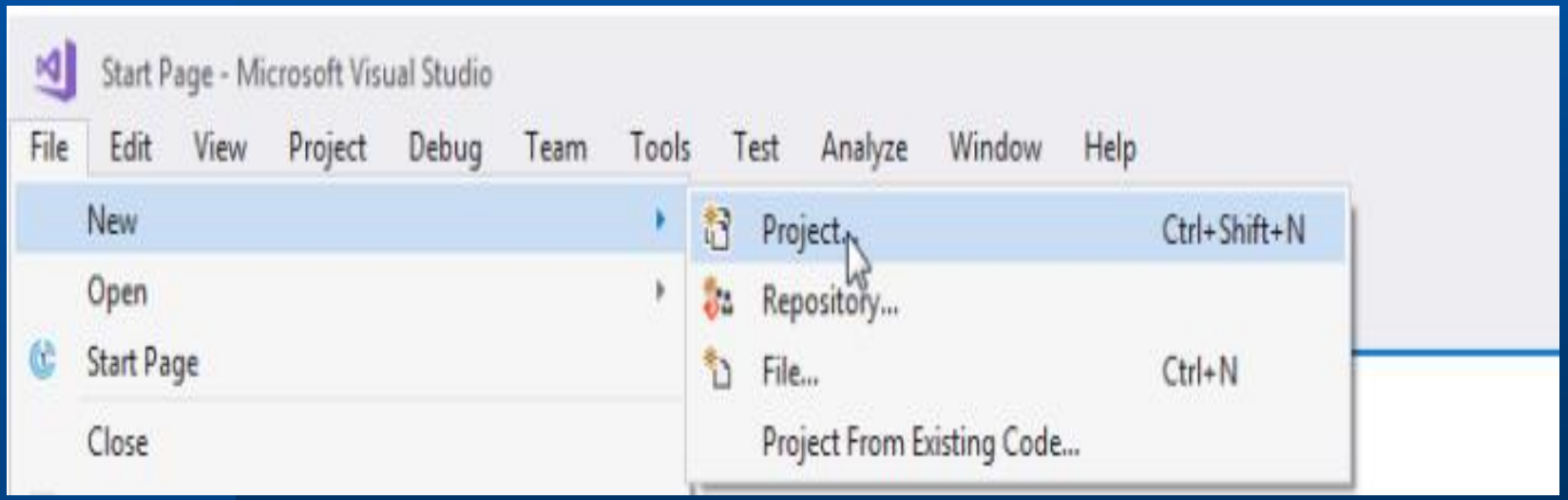
Création d'application ASP.NET MVC

Configuration de l'environnement de développement

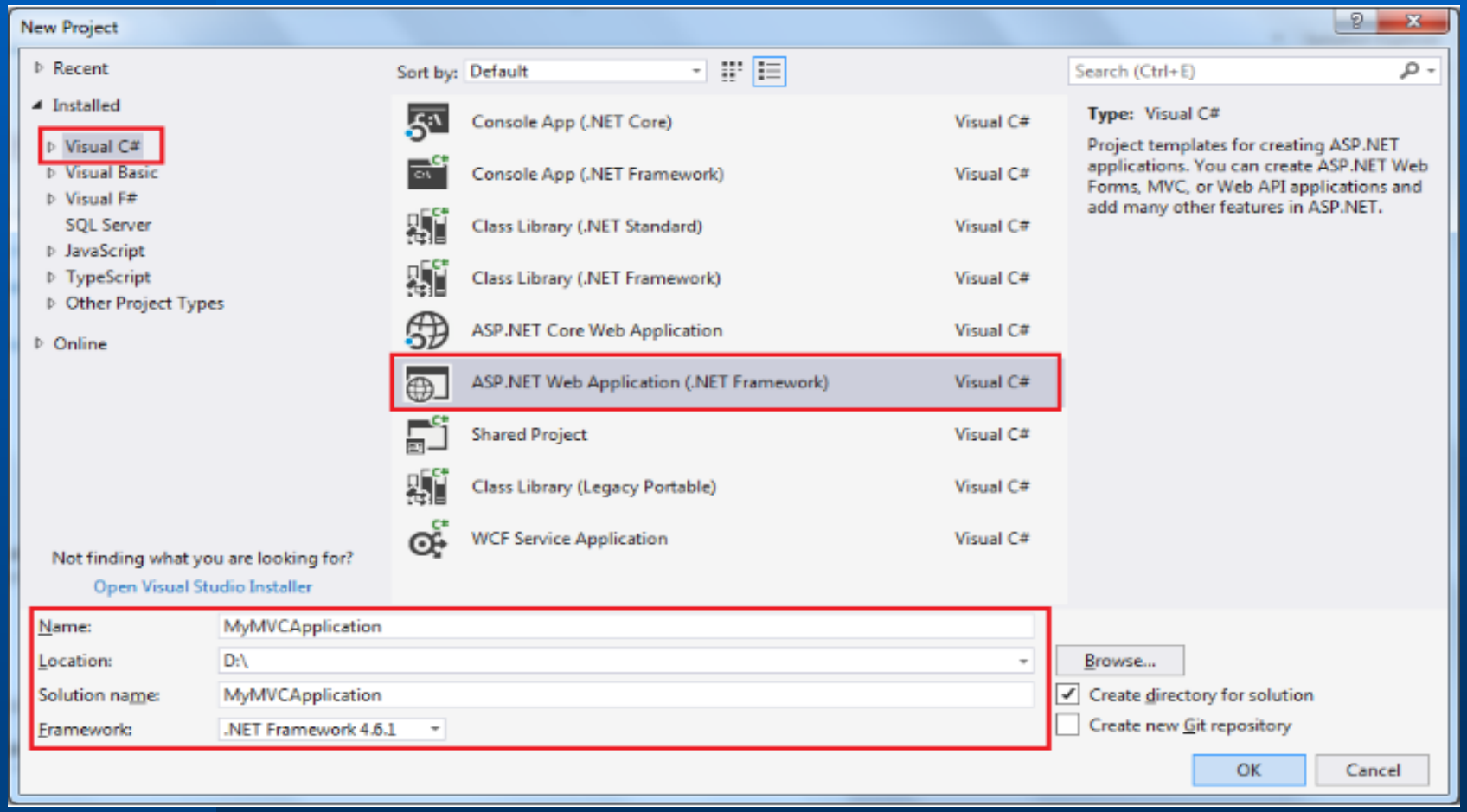
On peut développer l'application ASP.NET MVC avec la version appropriée de Visual Studio et du framework .NET, comme vu dans l'historique des versions.
Ici, nous utiliserons MVC v5.2, l'édition de Visual Studio Community et le .NET Framework 4.6 pour créer notre première application MVC.

Création d'application ASP.NET MVC

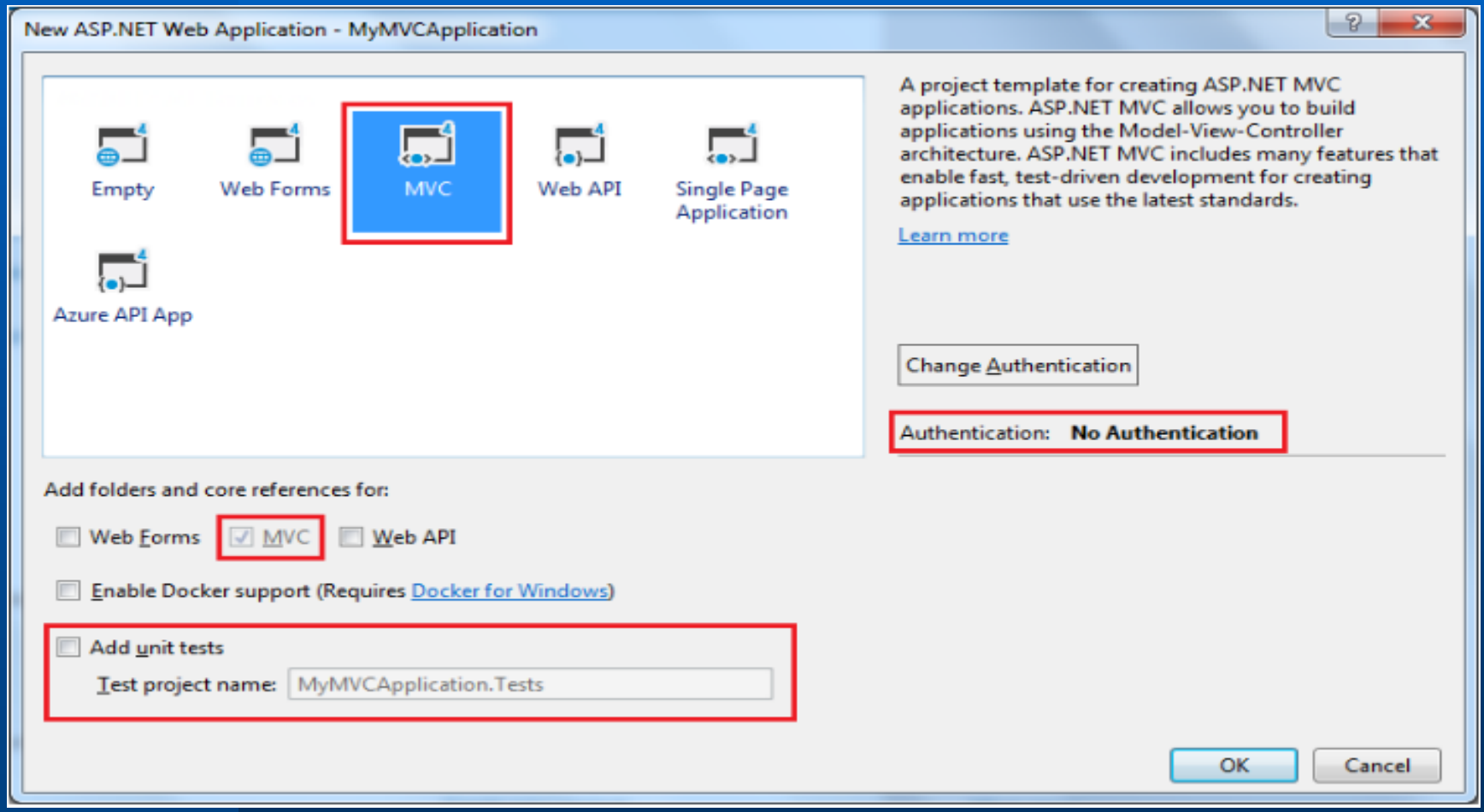
Tout d'abord, ouvrez une édition de Visual Studio Community et sélectionnez le menu Fichier -> Nouveau -> Projet comme indiqué ci-dessous.



Création d'application ASP.NET MVC

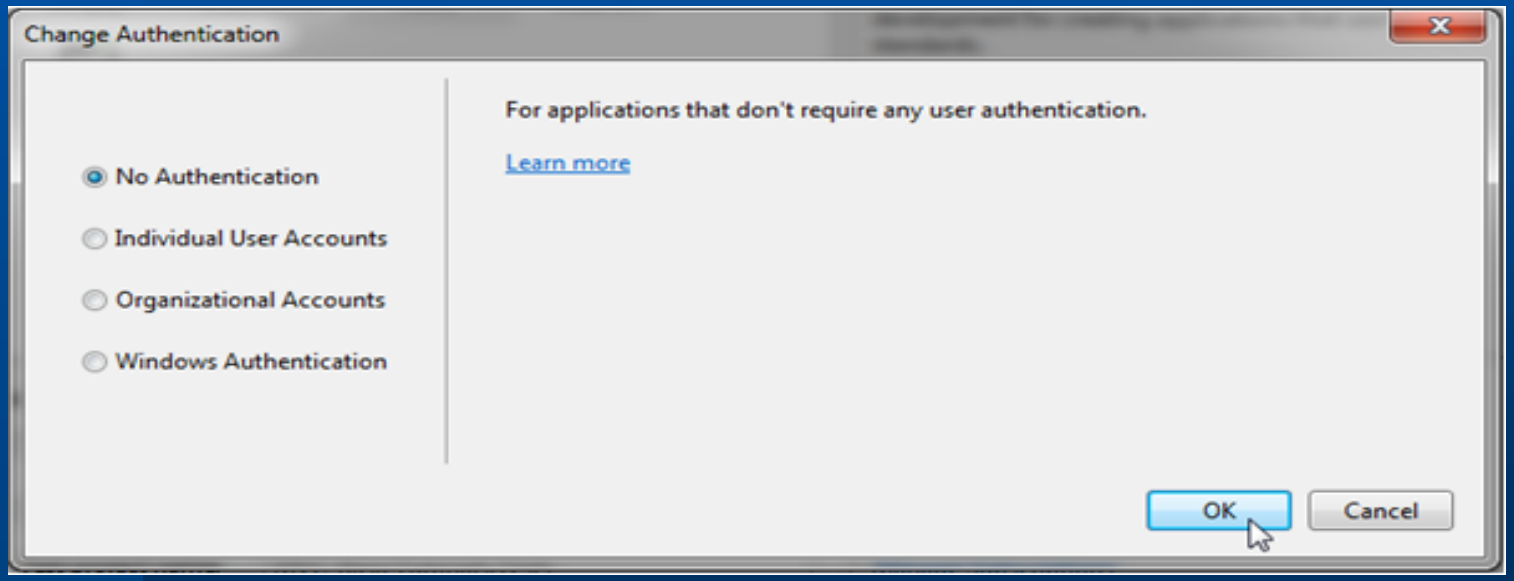


Création d'application ASP.NET MVC

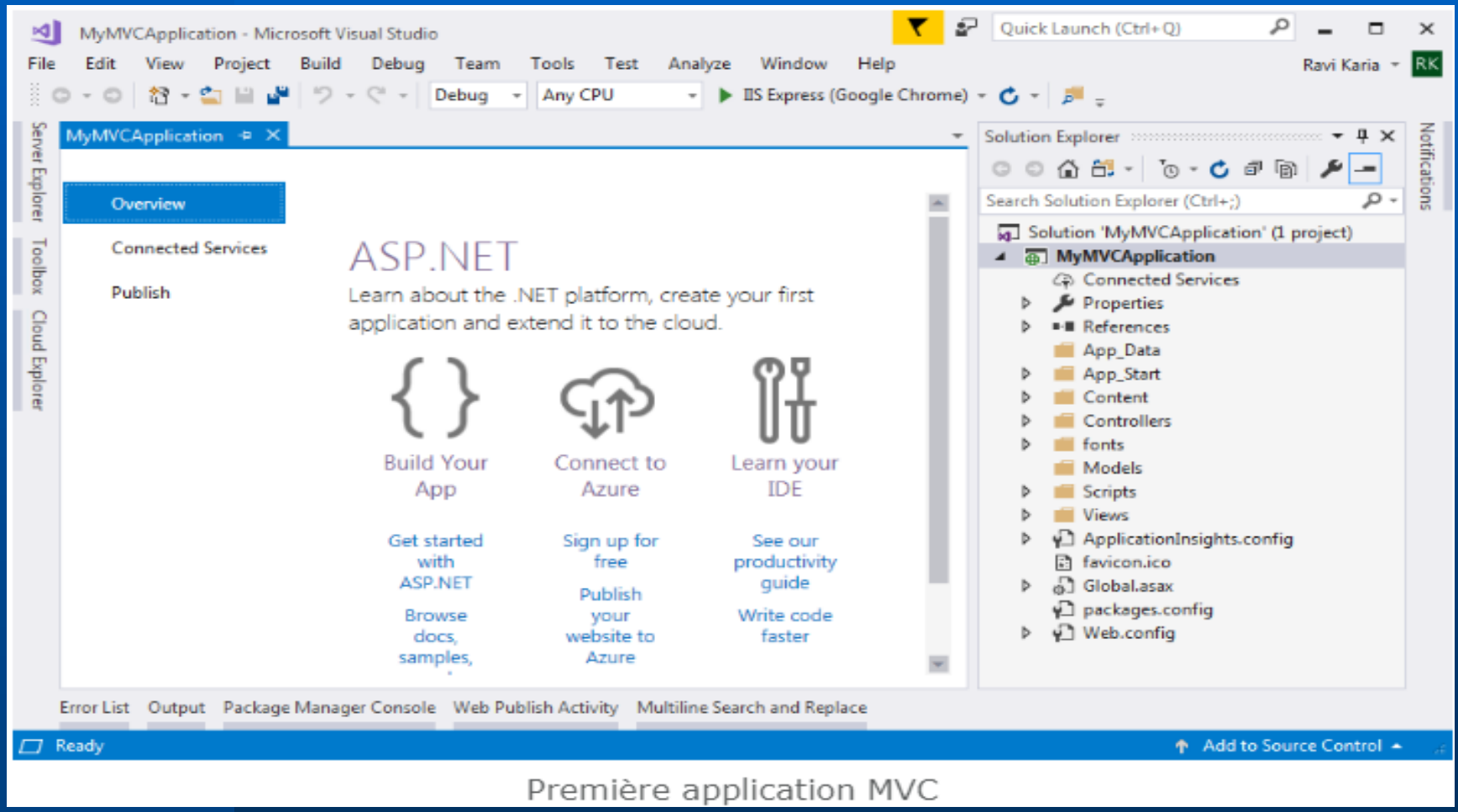


Création d'application ASP.NET MVC

On peut également modifier l'authentification en cliquant sur le bouton Modifier l'authentification . Vous pouvez sélectionner le mode d'authentification approprié pour votre application, comme indiqué ci-dessous.



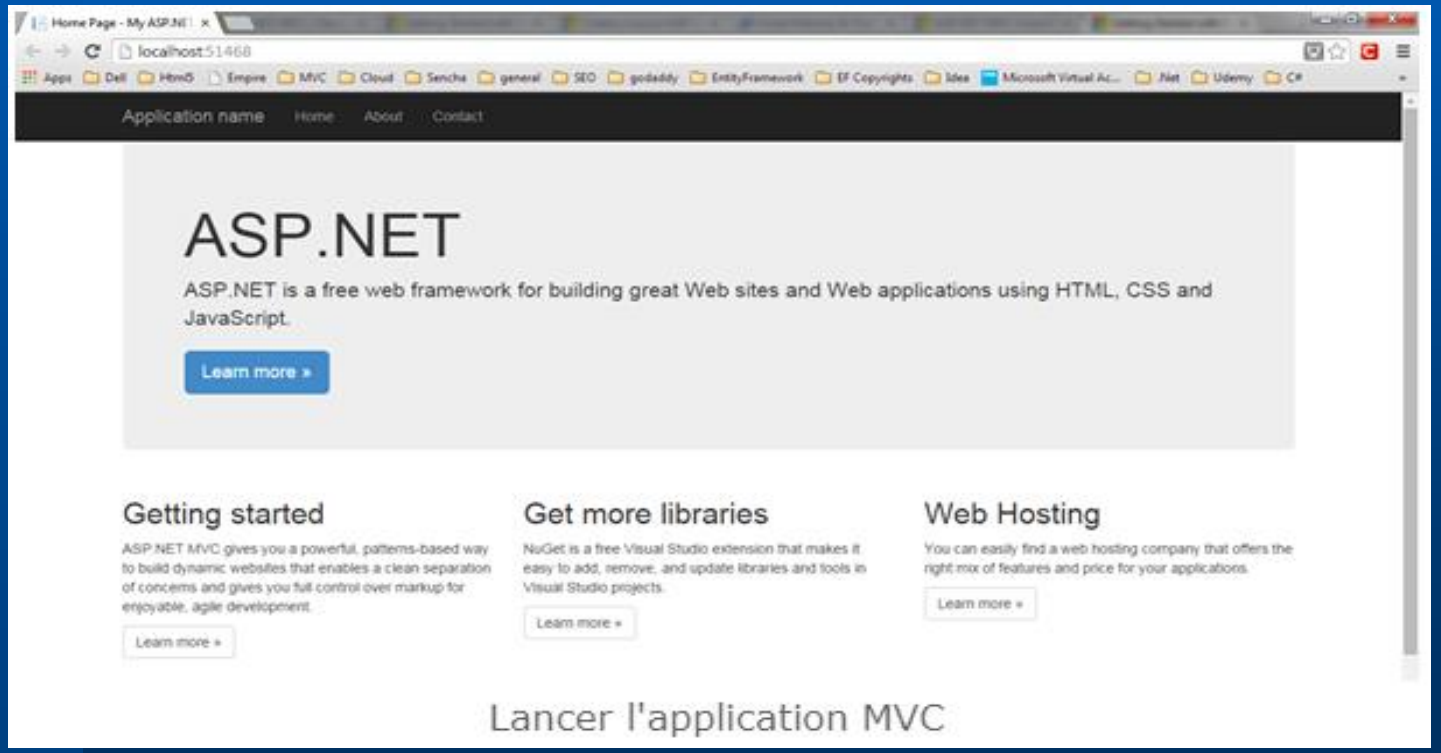
Création d'application ASP.NET MVC



Première application MVC

Création d'application ASP.NET MVC

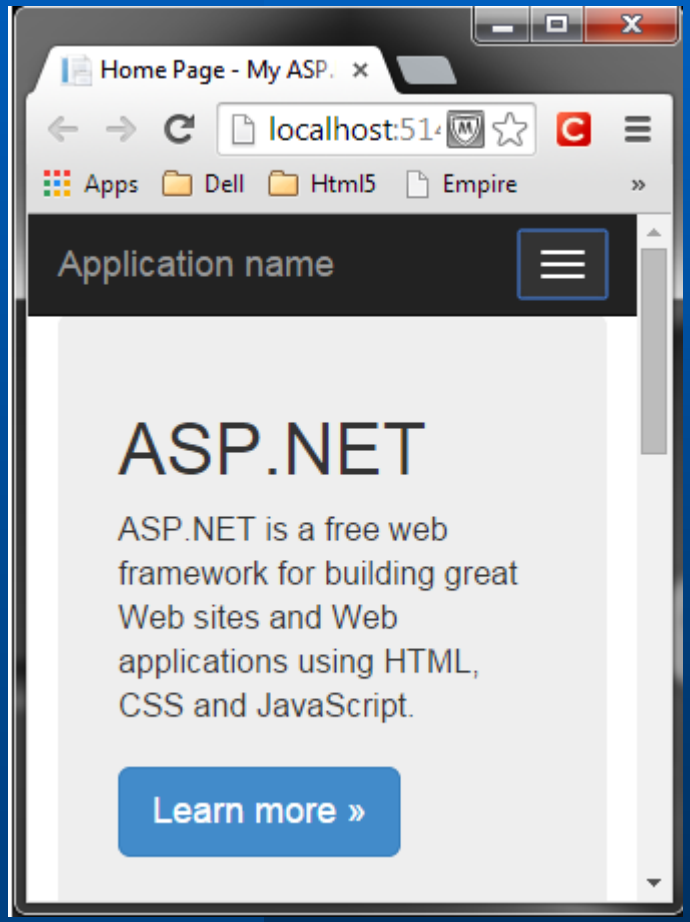
Exécuter le projet en mode débogage ou sans débogage. Il ouvrira la page d'accueil dans le navigateur, comme indiqué ci-dessous.

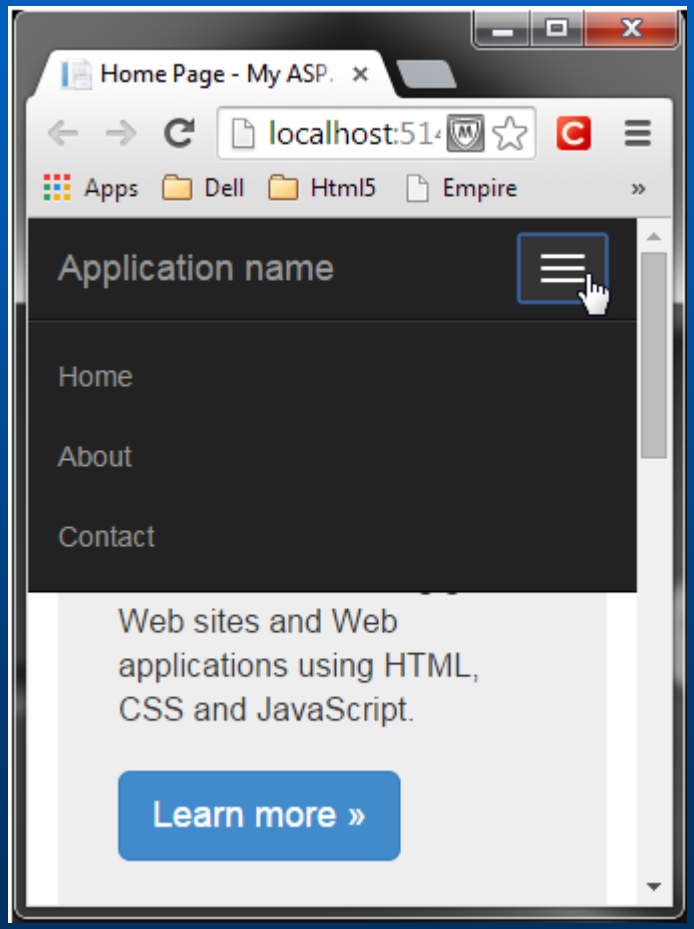


Création d'application ASP.NET MVC

Le projet MVC comprend les fichiers JavaScript et CSS de bootstrap . Vous pouvez ainsi créer des pages Web réactives. Cette interface utilisateur réactive modifiera son apparence en fonction de la taille de l'écran des différents périphériques. Par exemple, la barre de menus supérieure sera modifiée dans les appareils mobiles, comme indiqué ci-dessous.

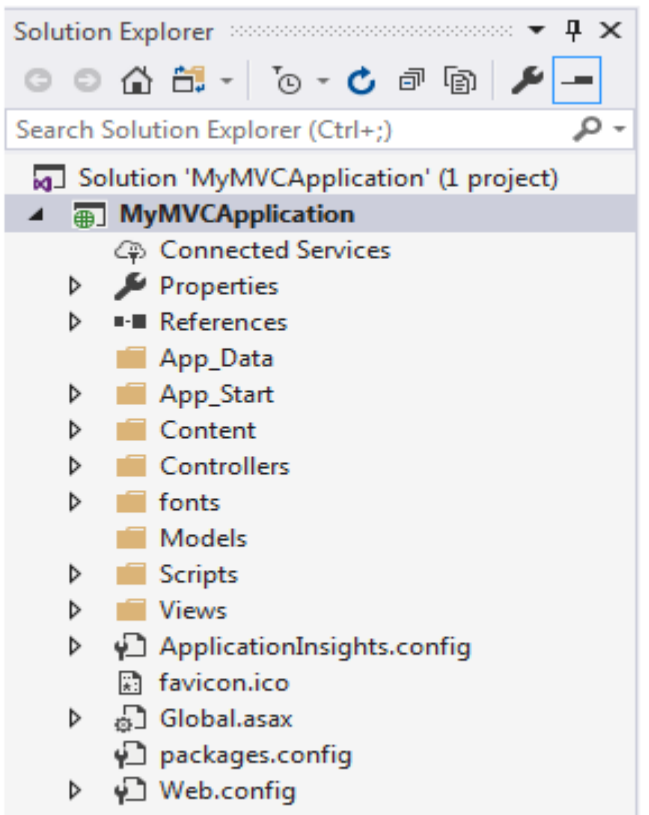
Création d'application ASP.NET MVC





Structure de dossiers ASP.NET MVC

Visual Studio crée par défaut la structure de dossiers suivante pour l'application MVC.



Structure de dossiers MVC

Structure de dossiers ASP.NET MVC

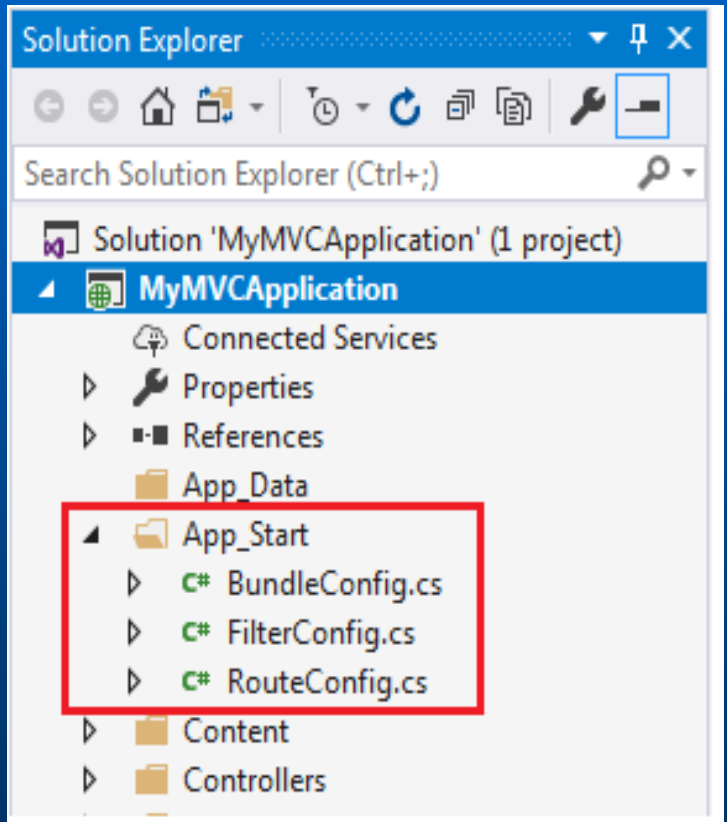
Données d'application

Le dossier App_Data peut contenir des fichiers de données d'application tels que LocalDB, des fichiers .mdf, des fichiers xml et d'autres fichiers liés aux données. IIS ne servira jamais les fichiers du dossier App_Data.

Structure de dossiers ASP.NET MVC

App_Start :

Le dossier App_Start peut contenir des fichiers de classe qui seront exécutés au démarrage de l'application. En règle générale, il s'agit de fichiers de configuration tels que AuthConfig.cs, BundleConfig.cs, FilterConfig.cs, RouteConfig.cs, etc. MVC5 inclut par défaut BundleConfig.cs, FilterConfig.cs et RouteConfig.cs.

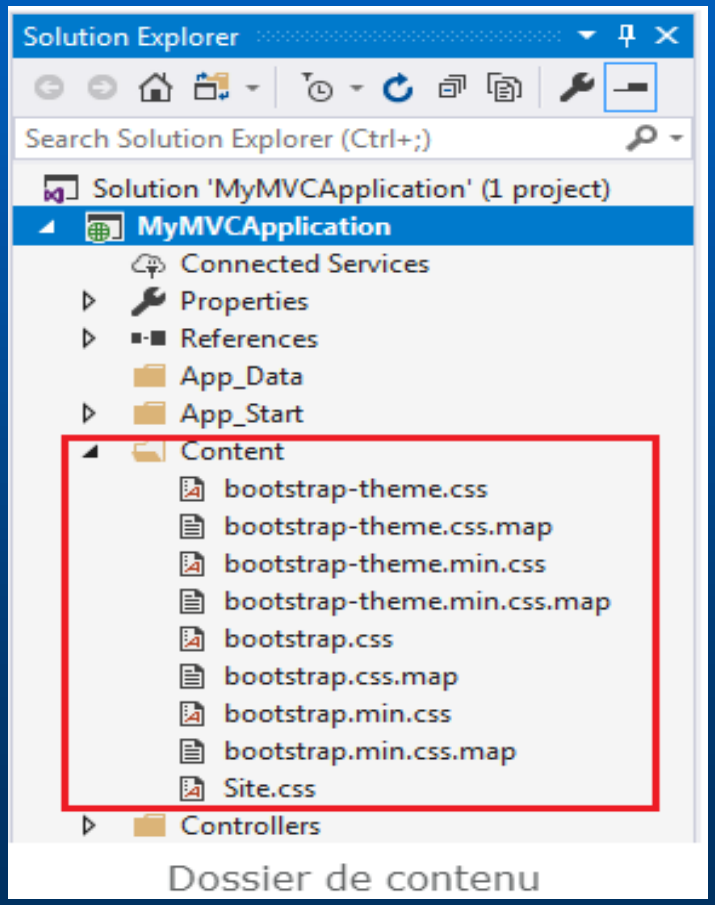


Dossier App_Start

Structure de dossiers ASP.NET MVC

Contenu:

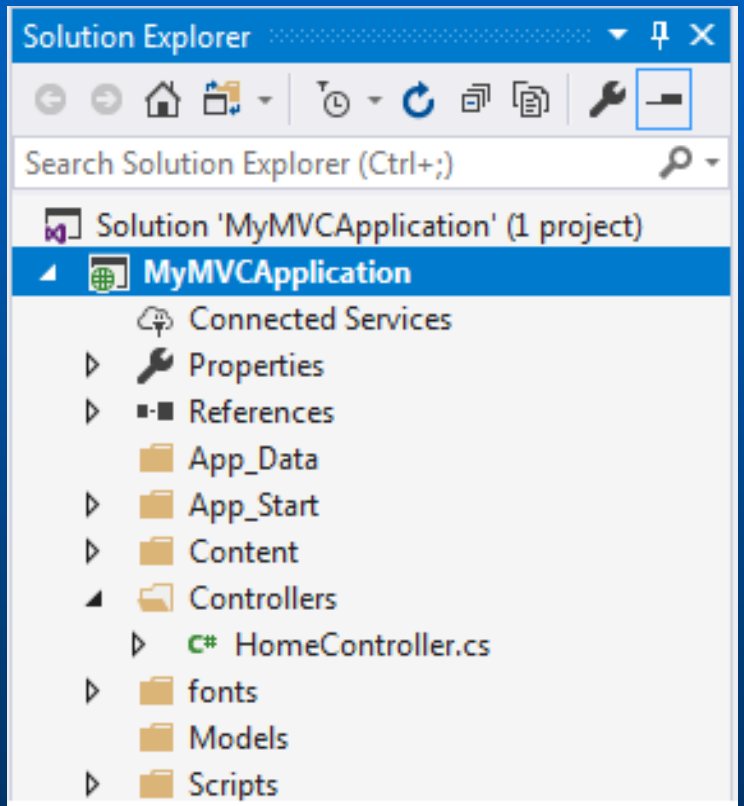
Le dossier de contenu contient des fichiers statiques tels que des fichiers CSS, des images et des fichiers d'icônes. L'application MVC 5 inclut par défaut bootstrap.css, bootstrap.min.css et Site.css.



Structure de dossiers ASP.NET MVC

Contrôleurs:

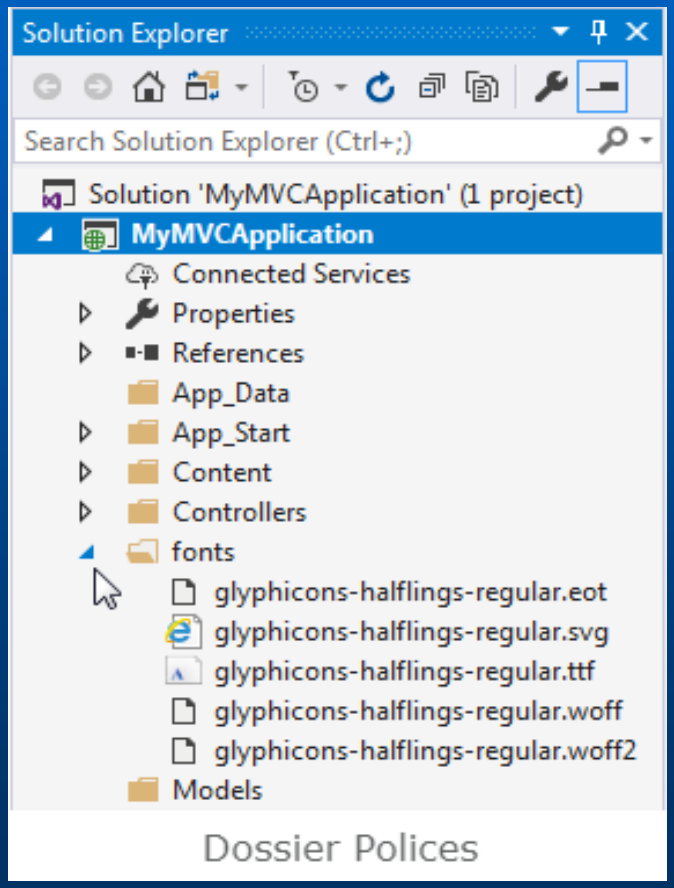
Le dossier Controllers contient des fichiers de classe pour les contrôleurs. Les contrôleurs traitent la demande des utilisateurs et retournent une réponse. MVC exige que le nom de tous les fichiers de contrôleur se termine par "Controller". Vous en apprendrez plus sur le contrôleur dans la section suivante.



Structure de dossiers ASP.NET MVC

Les polices:

Le dossier Polices contient des fichiers de polices personnalisés pour votre application.



Structure de dossiers ASP.NET MVC

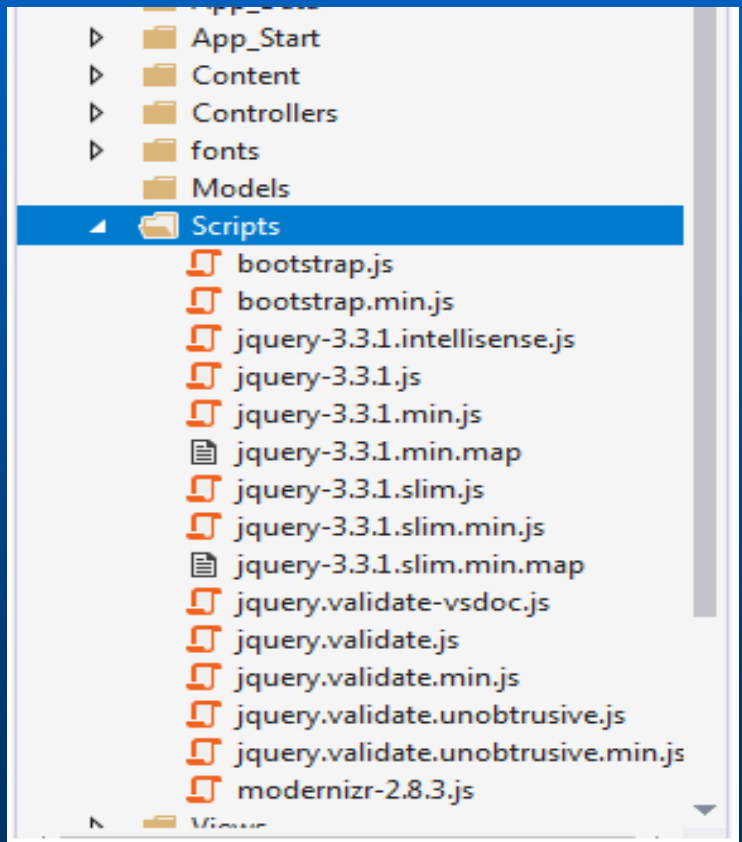
Des modèles:

Le dossier Modèles contient des fichiers de classe de modèle. En règle générale, la classe de modèle inclut des propriétés publiques, qui seront utilisées par l'application pour stocker et manipuler les données de l'application.

Structure de dossiers ASP.NET MVC

Les scripts:

Le dossier Scripts contient des fichiers JavaScript ou VBScript pour l'application. MVC 5 inclut les fichiers javascript pour bootstrap, jquery 1.10 et modernizer par défaut.



Dossier Scripts

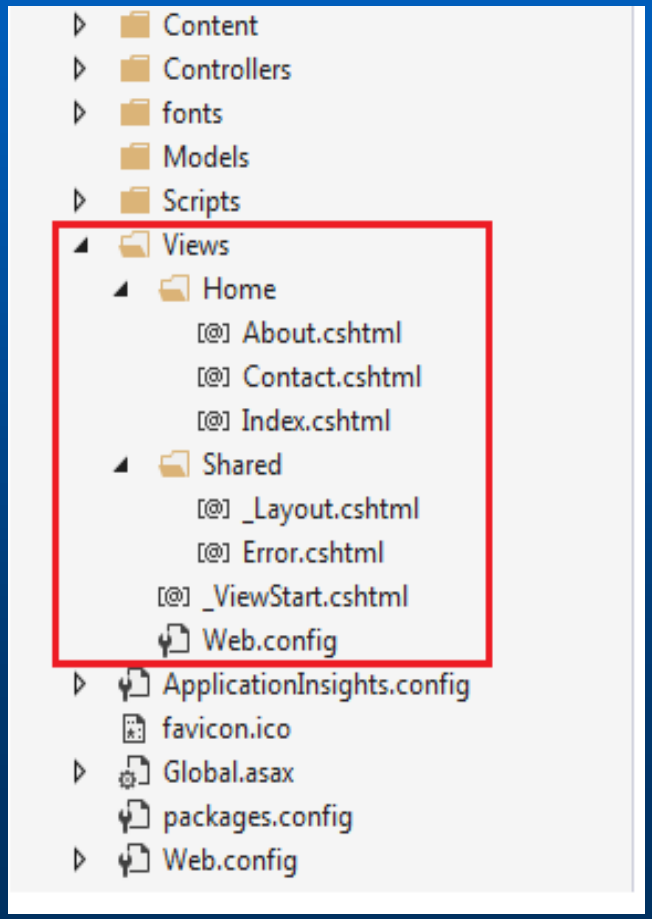
Structure de dossiers ASP.NET MVC

Des vues:

Le dossier Views contient des fichiers HTML pour l'application. En règle générale, view file est un fichier .cshtml dans lequel vous écrivez du code HTML et C # ou VB.NET.

Le dossier Views comprend un dossier distinct pour chaque contrôleur. Par exemple, tous les fichiers .cshtml, qui seront rendus par HomeController, seront dans le dossier View> Home.

Dossier partagé sous le dossier Voir contient toutes les vues qui seront partagées entre différents contrôleurs, par exemple des fichiers de présentation.



Structure de dossiers ASP.NET MVC

Global.asax:

Global.asax vous permet d'écrire du code qui s'exécute en réponse à des événements au niveau de l'application, tels que Application_BeginRequest, application_start, application_error, session_start, session_end, etc.

Packages.config:

Le fichier Packages.config est géré par NuGet pour assurer le suivi des packages et des versions que vous avez installés dans l'application.

Web.config:

Le fichier Web.config contient des configurations au niveau de l'application.

Exemple

Model

de démarrage | Global.asax.cs | Index.aspx | SimpleMathController.cs | SimpleMathInput.cs

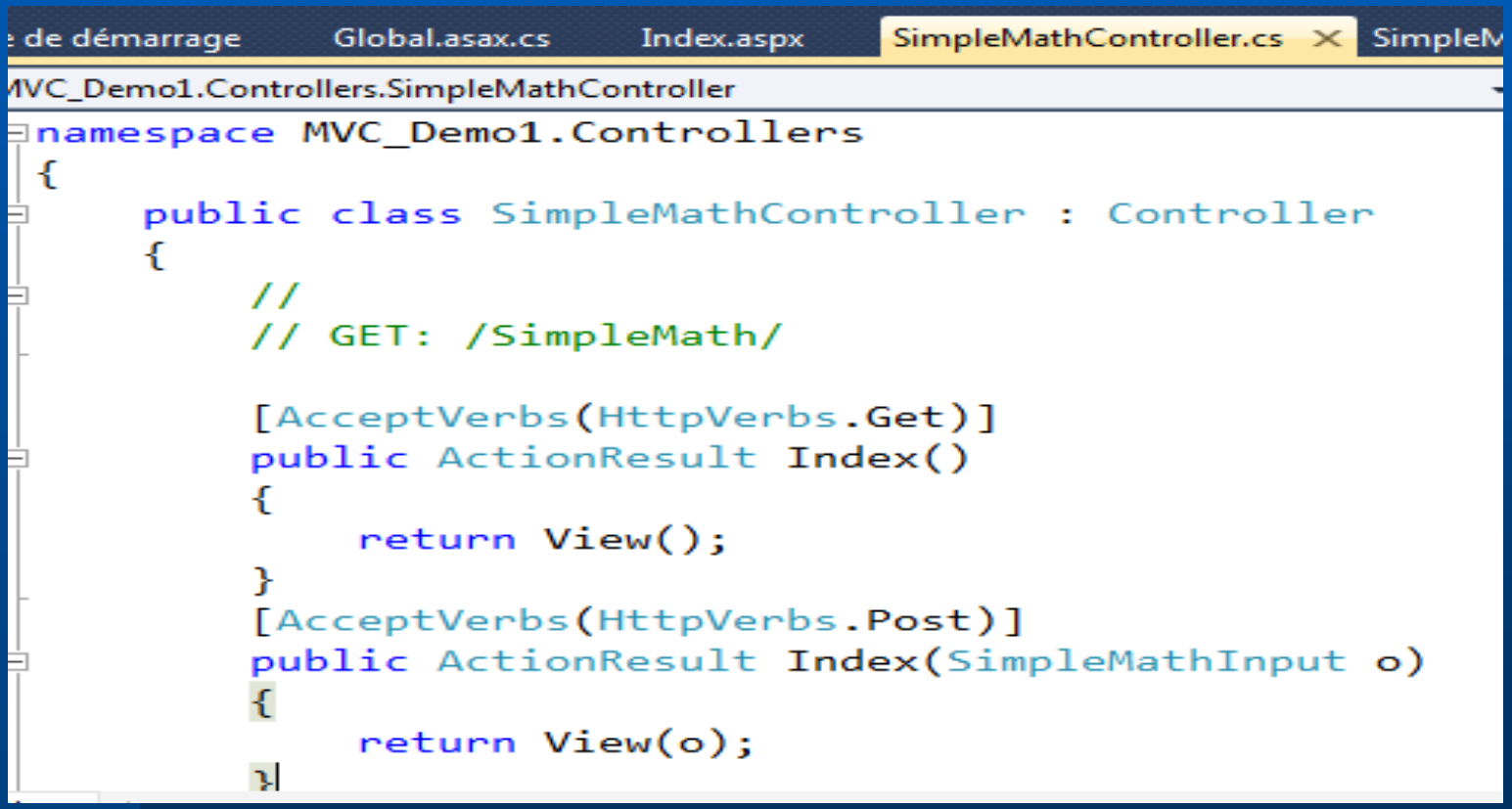
MVC_Demo1.Models.SimpleMathInput | GetResultat()

```
namespace MVC_Demo1.Models
{
    public class SimpleMathInput
    {
        public int? Num1 { get; set; }
        public int? Num2 { get; set; }
        public String GetResultat()
        {    string Resultat=String.Empty;
            {
            Resultat=String.Format("sum={0}", Num1.Value+Num2.Value);
            }

        return Resultat;
        }
    }
}
```

Exemple

Controleur:



Exemple

Vue :

x.aspx | Page de démarrage | Global.asax.cs | Index.aspx | SimpleMathController.cs | SimpleMathInp

Objets et événements serveur | (Aucun événemer

```
<%@ Page Language="C#" Inherits="System.Web.Mvc.ViewPage<MVC_Demo01.M

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Index</title>
</head>
<body>
    <div>
    <%using (Html.BeginForm())
      { %>
      Entrer Num1 :
      <%=Html.TextBox("Num1")%><br/>
      Entrer Num2 :
      <%=Html.TextBox("Num2")%><br/>
      <input type="submit" />
      <br />
      <%=(Model != null) ? Model.GetResultat() : String.Empty%>
      <%} %>

    </div>
```

Exemple

Global.asax:

Page de démarrage | Global.asax.cs | Index.aspx | SimpleMathController.cs | SimpleMathInput.cs

o1.MvcApplication | RegisterRoutes(RouteCollection routes)

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

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

    }

    protected void Application_Start()
    {
        AreaRegistration.RegisterAllAreas();

        RegisterRoutes(RouteTable.Routes);
    }
}
```