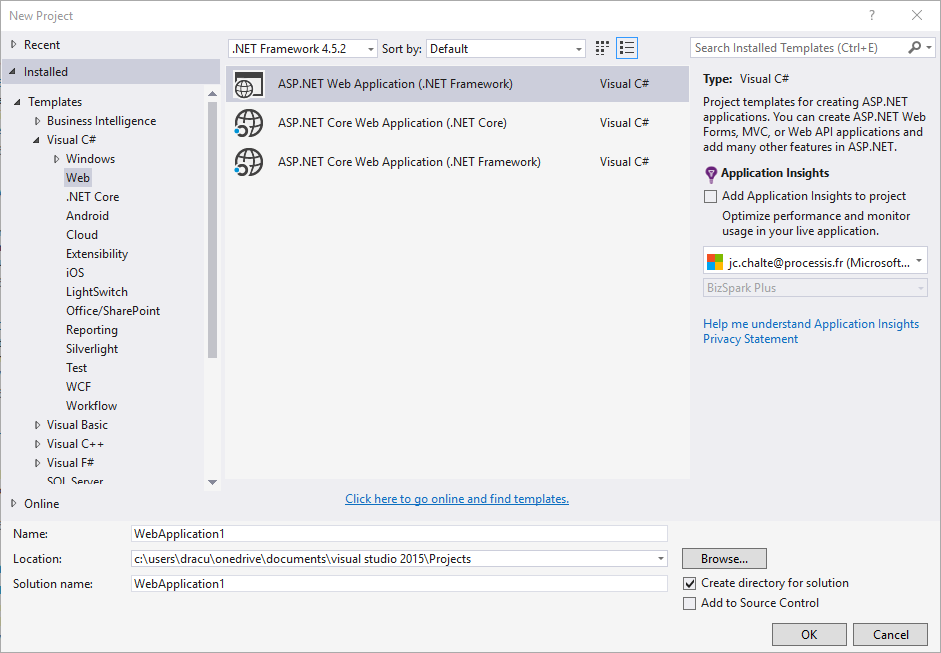
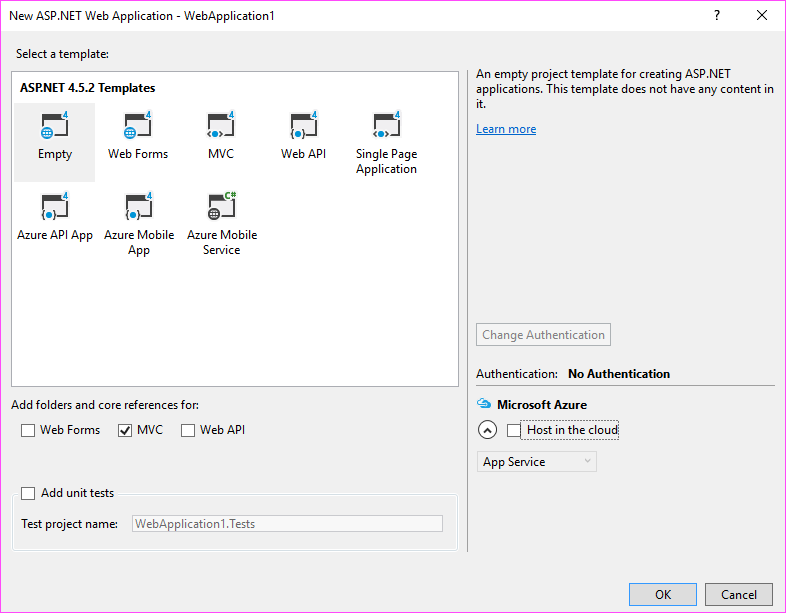
Exercices ASP.Net MVC

# Exercice I - Création d'une application asp.net MVC

Dans Visual Studio, crééez une nouvelle solution de type "Application web ASP.Net (.NET Framework)



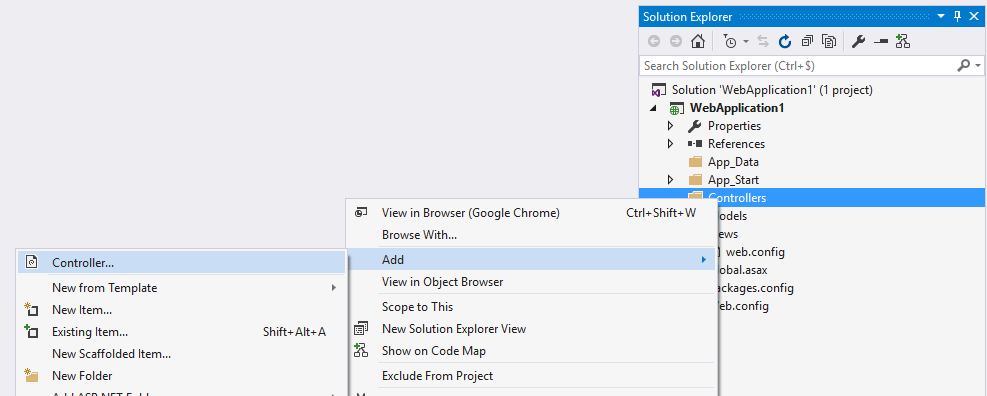
Puis dans l'écran suivant, sélectionnez le patron "Vide", en cochant la référence vers "MVC", et en décochant "Azure".

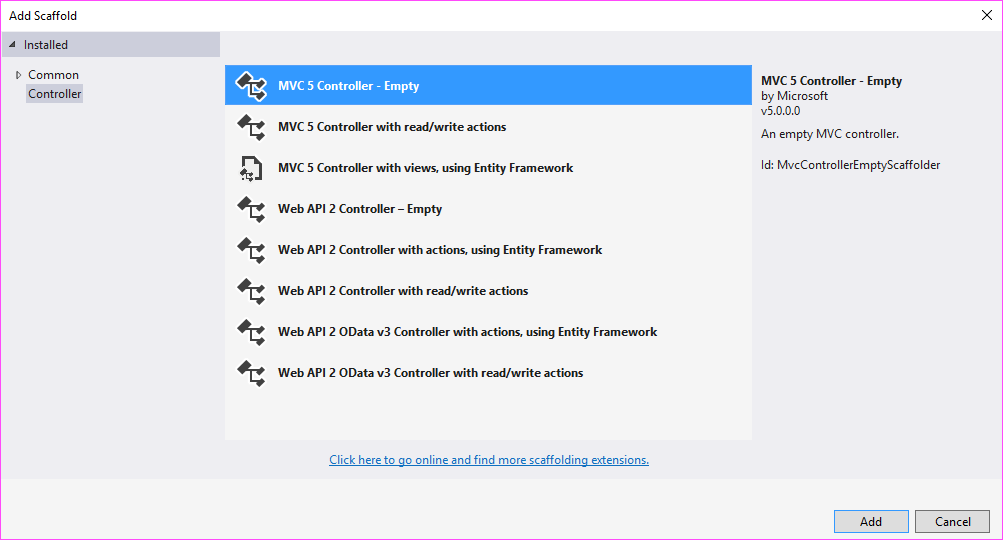


Puis, une fois le projet créé, démarrez la solution. L'application sera lancée, un navigateur démarrera, il tentera de se connecter à l'application, mais cette dernière crashera rapidement car il n'y a aucun code écrit qui permettrait de savoir comment réagir à l'appel au site web effectué via le navigateur.

*En attendant, notez bien le port réseau utilisé pour se connecter à l'application. Ce port peut changer selon la configuration du PC car Visual Studio utilise le premier port non utilisé du PC lors du premier démarrage de la solution. Par exemple, si votre navigateur tente de se connecter à* [*http://localhost:8831*](http://localhost:8831)*, le port est "8831". Dans les exemples de liens plus tard dans cet exercice, il faudra remplacer le port proposé par celui qui correspond à votre environnement.*

Nous allons maintenant rajouter un premier contrôleur pour pouvoir faire nos premières requêtes au site web. Pour ce faire, créez le contrôleur "HomeController"



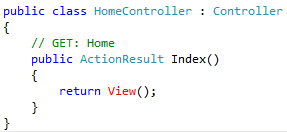


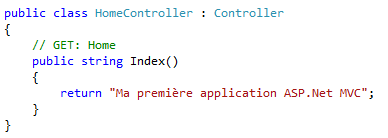
Puis, notez que…

* Une classe "HomeController" a été créée.
* Une méthode "Index" a été créée automatiquement lors de la création du contrôleur.
* La méthode Index fait appel à une méthode appelée "View" (Le V de MVC).

Comme nous n'avons pas encore développé la vue, et en attendant la description de ce concept plus tard dans ce cours, nous allons modifier la fonction Index pour ne pas avoir le besoin d'appeler cette méthode "View" temporairement.

Pour ce faire, modifiez la déclaration de la méthode "Index" pour indiquer qu'elle doit retourner une chaîne de caractère (plutôt qu'un ActionResult comme c'est le cas par défaut), et modifiez le corps de cette méthode pour retourner la chaine de caractère de votre choix.





Relancez l'application puis essayez d'accéder aux adresses suivantes :

* <http://localhost:8831/Home/Index>
* [http://localhost:8831/Home](http://localhost:8831/Home/Index)
* [http://localhost:8831/](http://localhost:8831/Home/Index)
* [http://localhost:8831/UneAutreAdresse](http://localhost:8831/UneAutreAdresse/)

Certains accès doivent fonctionner, d'autres non. Essayez d'en trouver une explication possible.

# Exercice II - Paramètres d'actions

Pour tester l'usage de paramètres, nous allons réutiliser le concept des opérations pour proposer une première version simplifiée d'une calculatrice en ligne.

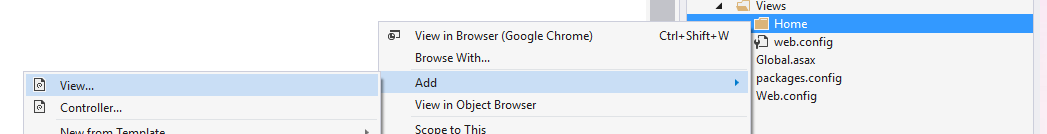
Pour ce faire, rajoutez un nouveau contrôleur "OperationController", contenant une action "Addition" prenant en paramètre l'opérande gauche et l'opérande droite, puis en réutilisant la classe "Operation" que l'on a définit dans les cours passés (qu'il faudra reprendre -- positionnez là dans un répertoire Models -- ), calcul le résultat de cette addition, puis l'affiche (de la même manière qu'Index affiche une chaîne en dur, Addition affichera une chaîne calculée). Conservez aussi le comportement qui sauvegarde l'opération en base de données lorsque le calcul est demandé (on réutilisera ce comportement plus tard).

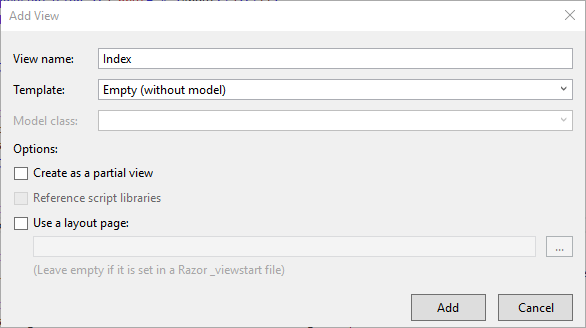
Vérifiez le bon fonctionnement de cette action en l'appelant avec la bonne adresse.

Une fois cela fait, rajoutez les différentes autres opérations : "Soustraction", "Multiplication", etc.

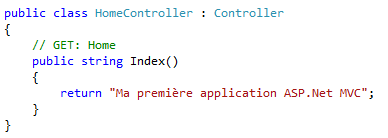
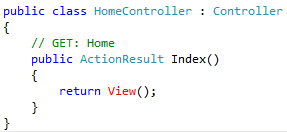
# Exercice III - Vue

Dans le projet, dans le dossier Views/Home, rajouter une vue appelée "Index.cshtml", sans template, et en décochant le "layout" (page de disposition) coché par défaut.





Maintenant que nous avons la vue, nous allons pouvoir "défaire" ce que nous avions fait à l'époque Dans le contrôleur HomeController, modifiez la déclaration de la méthode "Index" pour indiquer qu'elle retourne un "ActionResult", puis remplacez le code de l'action pour renvoyer le retour de l'appel à la méthode "View".



Lancez l'application. La page affichée sera vide, mais notez que le titre de la page a été modifié en "Index". Si vous regardez le code HTML contenu dans Index.cshtml, vous verrez que celui-ci ne contient rien d'autre que l'affectation d'un titre et la construction d'une hiérarchie HTML vierge.

Remplacez le code HTML de ce fichier par celui fourni dans Index.html sur le GIT, puis relancez l'application.

Pour information, le code proposé utilise Bootstrap qui est un ensemble de composants graphiques facilement réutilisables qui permettent de rapidement avoir un design simple pour un prototype d'application (cf. [www.getbootstrap.com](http://www.getbootstrap.com) )

# Exercice IV - Passage de données à la vue - Model

## Addition & soustraction

Nous allons maintenant modifier la méthode "Addition" pour qu'elle renvoie une vrai page HTML qui affiche le calcul effectué.

Pour cela :

* Dans le dossier Views/Operation, rajoutez une vue "Addition.cshtml"
* Dans ce fichier, rajoutez une clause "@model" spécifiant que le type du modèle de la vue est une "Operation" (ex. : " @model MonApplicationWeb.Models.Operation ")
* Dans l'action "Addition" :
  + Modifiez la signature pour spécifiquer que l'action retourne un ActionResult
  + Une fois l'instance de la classe "Operation" créée, envoyez là à la vue via la méthode "View" et retournez le résultat de cet appel (ex. "return View(monOperation);" )
* A partir de là, vous aurez accès dans la vue à l'instance d'opération qui vient d'être calculée dans le contrôleur via la variable nommée "Model" (avec un M majuscule).
* Modifiez la vue pour que celle-ci affiche le calcul complet, et non seulement le résultat.

Une fois cela fait, effectué, vous devriez pouvoir appeler l'addition à partir de votre navigateur pour en voir le résultat.

Refaites ces manipulations pour pouvoir afficher la soustraction.

## Refactoring - une seule vue

Une fois cela fait, la suite logique serait d'en faire de même avec les autres opérations, mais on se rend bien compte à ce moment-là que si la vue a bien été réalisée, le code est identique pour chaque action. Il n'est donc pas logique d'avoir à "copier-coller" le code cshtml. Par défaut, l'appel à la méthode "View" dans le contrôleur va chercher la vue qui possède le même nom que l'action. Or dans notre cas, ce n'est pas bon. Mais cette même méthode "View" peut-être appelée avec un second paramètre optionnel de type "string" qui, s'il est renseigné, sera utilisé en lieu et place du nom de l'action lors de la recherche du fichier cshtml correspondant.

Renommez donc "Addition.cshtml" en "Operation.cshtml" (par exemple), puis modifiez l'appel à "View" pour envoyer 2 paramètres : l'opération en cours comme c'est déjà le cas, et la chaine de caractère "Operation". Ceci devrait afficher le fichier "Operation.cshtml".

Vous pouvez à ce moment là supprimer la vue "Soustraction", modifier le code de l'action Soustraction pour utiliser elle aussi la vue "Operation", puis finaliser les autres appels de calcul.

## Historique d'une opération

Ajoutez une action "Historique" qui permettra d'afficher un calcul que l'on avait effectué précédemment. Cette action prendra en paramètre l'identifiant du calcul que l'utilisateur souhaite afficher (sa clé primaire en BDD). Le corps de cette action ira charger les informations nécessaires en BDD, construira une instance d'Operation et l'affichera.

## Historique de toutes les opérations

Ajouter une action "Toutes" qui permettra d'afficher toutes les opérations enregistrées en BDD. Cette action n'a pas besoin de paramètres. Elle ira charger en BDD toutes les informations dont elle aura besoin pour reconstruire une liste d'opérations qu'elle enverra à la vue. Une nouvelle vue sera donc à réaliser. Celle-ci aura comme modèle une liste d'opérations (List<Models.Operation>), puis affichera cette liste dans une table HTML.

Dans un second temps, vous pourrez même rajouter un hyperlien sur la ligne de la grille qui naviguera vers l'affichage du calcul sélectionné.