



**Code
pour les
Débutant**

Razor ASP.NET MVC 3

**Developper des application web en utilisant les
structure MVC avec Razor et C#**



**Dassaud Jeremy
Dachez Lucas**

Sommaire

Sommaire	-----	2
Introduction	-----	3
Qu'est-ce que c'est ?	-----	3
Pourquoi s'en servir ?	-----	3
Les Prérequis	-----	4
Création d'une application (sans MVC)	-----	4
Création d'une application (avec MVC)	-----	6
Elément de base d'une application sans MVC	-----	7
Créer votre première page	-----	7
Bloc de code	-----	7
Elément de base d'une application avec MVC	-----	8
Principe des Contrôleurs	-----	8
Principe des vues	-----	9
Création de modèles	-----	9
Elément avancé d'une application	-----	10
Validation de Formulaire	-----	10
Accès aux données d'une base de donnée	-----	11
Elément de cybersécurité	-----	13
Principe des vues partiel	-----	7
Source pour aller plus loin	-----	7

Introduction

Qu'est-ce que c'est ?

Le Razor permet d'utiliser des templates afin de créer des pages web qui sont dynamique. Ce langage va fonctionner en ASPNET. Il va nous permettre par exemple de combiner du code en C# et du HTML afin de créer des pages web en la rendant plus facile à créer mais aussi à la maintenir et la mettre à jour. Ce type d'application est le plus souvent utilisé pour des applications web qui va nécessiter des fonctionnalités avancées comme par exemple faire de la validation de formulaire, de l'authentification ou encore faire de la gestion d'autorisation. Il est même possible de faire de la gestion de données sur une base de données.

Pourquoi s'en servir ?

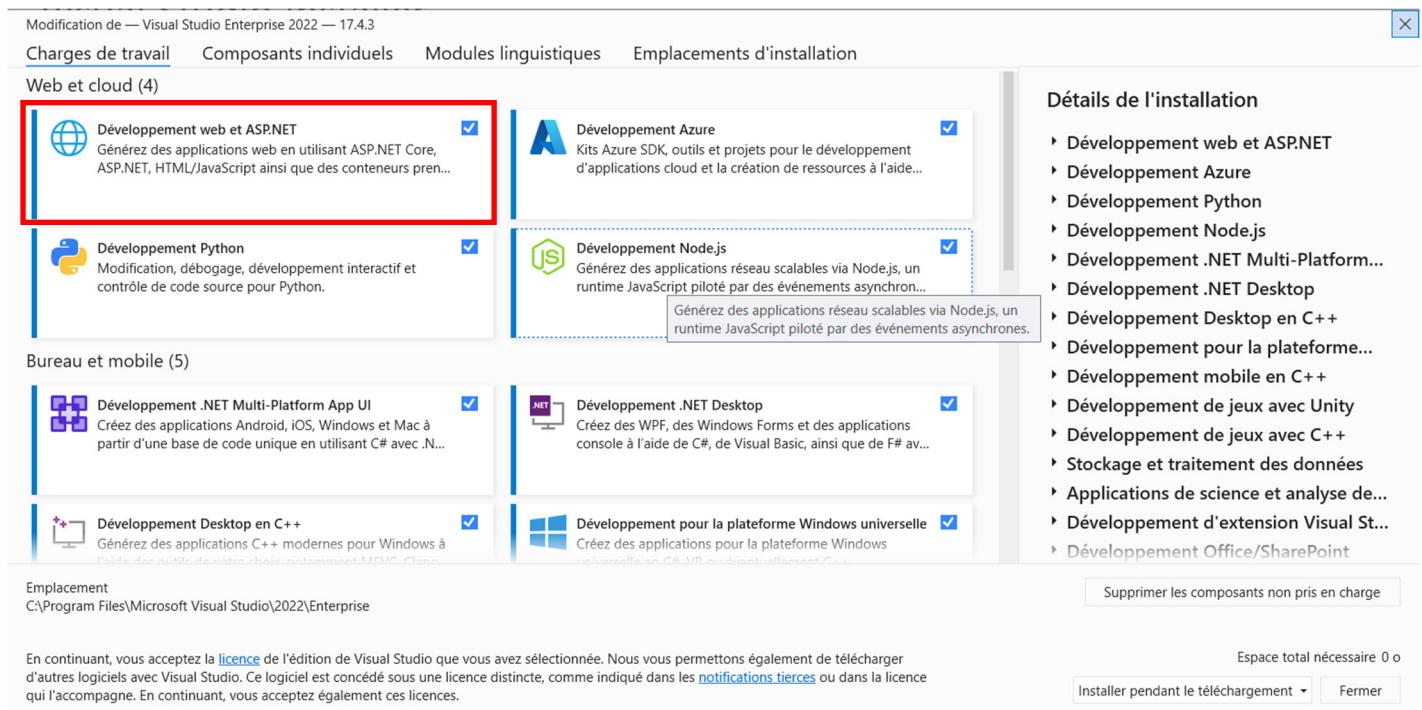
Nous allons utiliser Razor afin de vous montrer qu'il va nous permettre de simplifier la syntaxe notamment dans les vues ASP.NET MVC, cela nous permettra de séparer le code HTML et C# donc de mettre des éléments sans les mélanger à d'autres. De plus, le code est plus lisible et plus agréable à lire mais il est aussi facile à maintenir. Ce qui va nous permettre d'ajouter des fonctionnalités dans ses vues telles que des boucles et des conditions à la vue sans la lier à un fichier contenant du code C# et donc de tout mélanger.%

Ensuite, il va nous permettre de créer des vues partir de modèles près remplissent ce qui peut intéresser est un gain de temps notamment pour les choses les plus basiques comme créer, supprimer ou modifier une liste d'objet. Mais nous pouvons créer aussi notre propre modèle. De plus, l'éditeur va nous trier automatiquement nos fichiers en les mettant dans leurs couches respectivement les contrôleurs dans le dossier contrôleur Des vues triées par contrôleurs, etc.%

Pour finir, lors de la création de notre projet nous auront le choix de créer sur un projet qui va nous donner une interface de base avec MVC en nous créer quelque élément tel qu'une page d'accueil. Mais l'on peut aussi partir de zéro avec uniquement l'architecture de base MVC avec tous les dossiers de chaque couche.

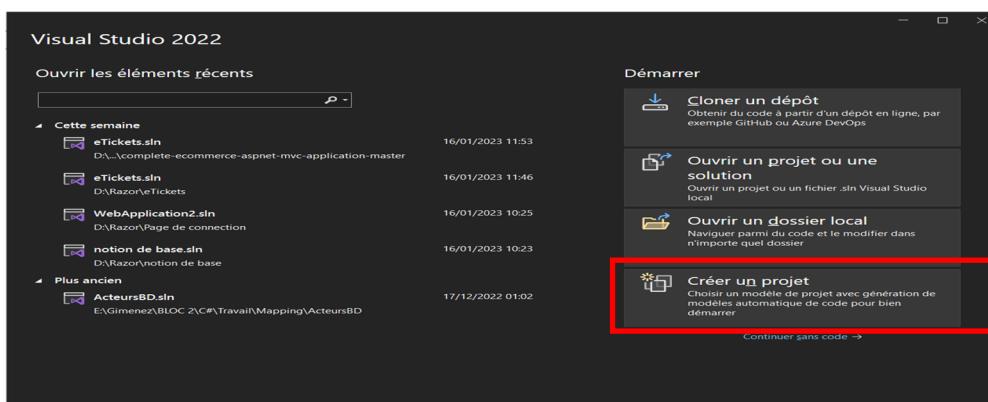
Les prérequis

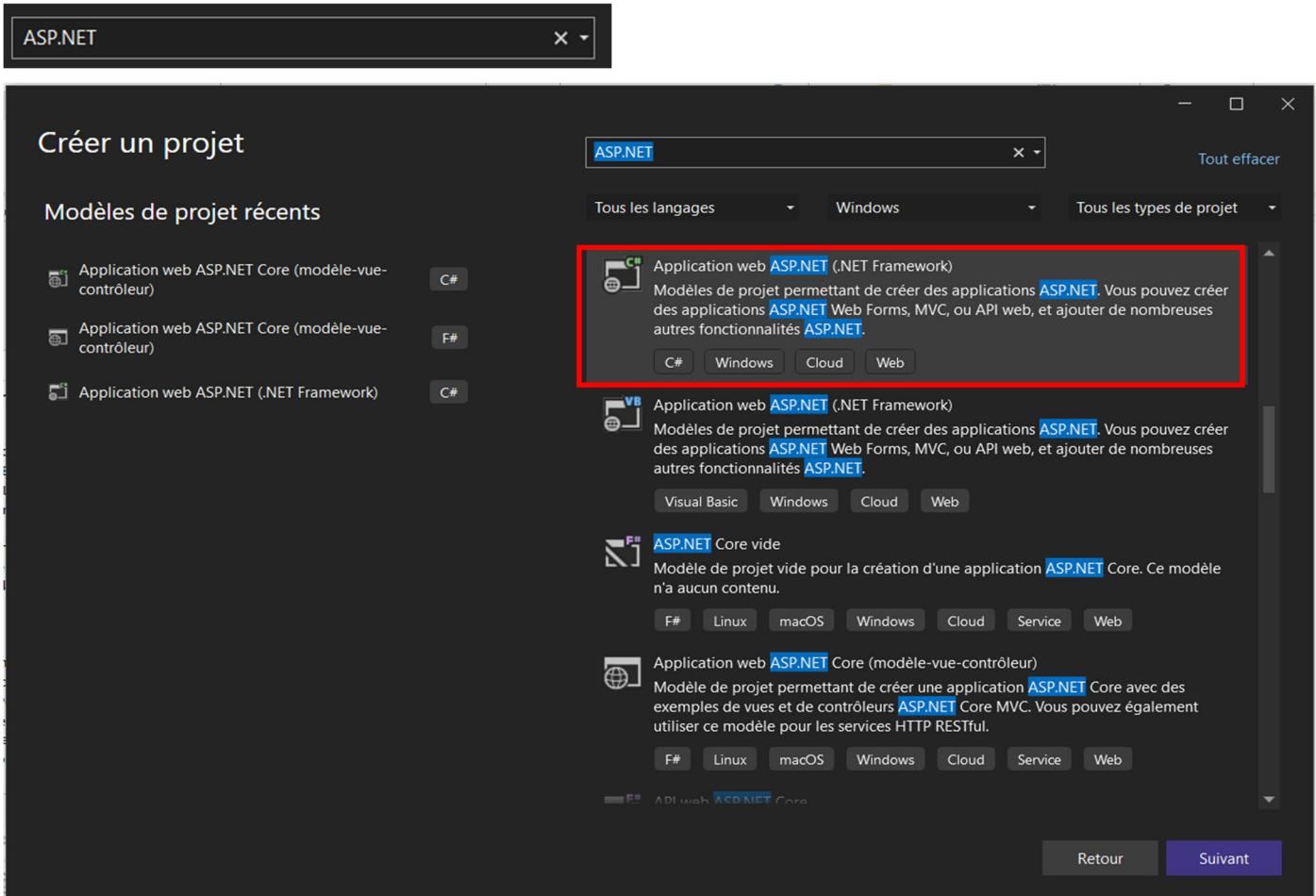
Afin de pouvoir coder en Razor ASP.NET il faut installer Visual studio ou Visual studio code. Pour Visual studio il faut la charge de travail développement web et ASP.NET dans l'installateur de Visual studio. Pour Visual studio code il faudra installer au préalable C# pour Visual studio code et SDK .NET 6.0 ou dernière version. Mais ici nous ne nous intéresserons que de Visual studio mais Visual studio code marche à peu près de la même façon que Visual studio.



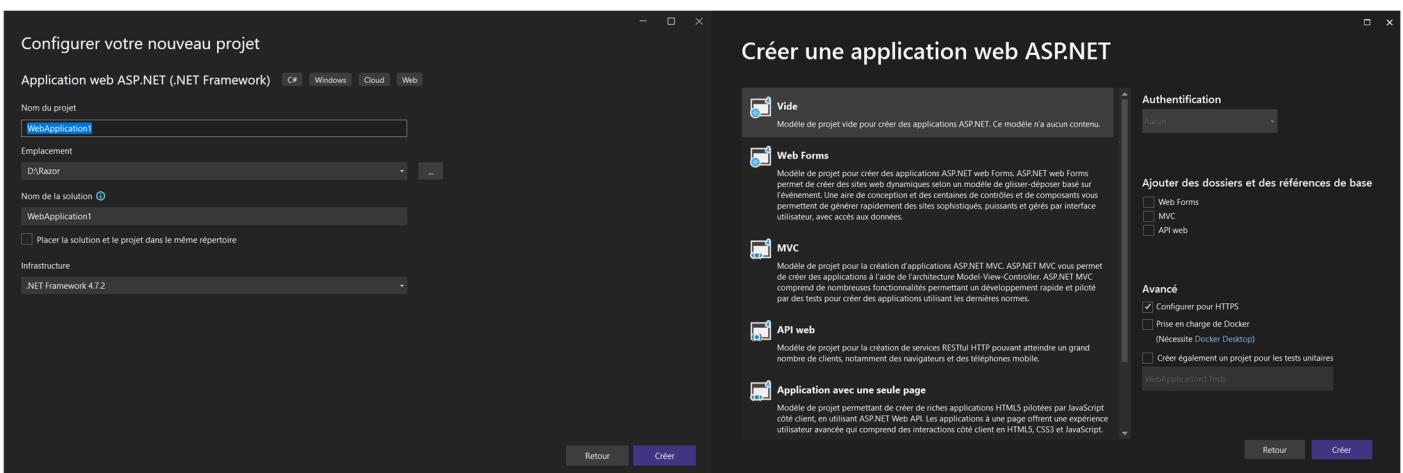
Création d'une application Razor (sans mvc)

Afin de créer un nouveau projet pour du Razor nous allons ouvrir Visual studio puis créé un nouveau projet. Dans la barre de recherche nous allons rechercher ASP.NET. Cela aura pour but de nous afficher toutes les applications ASP.NET. Nous allons choisir l'application ASP. NET (Framework) en C#. Attention il faut regarder que l'application soit bien en C# car celle-ci existe dans d'autres langages comme F#.



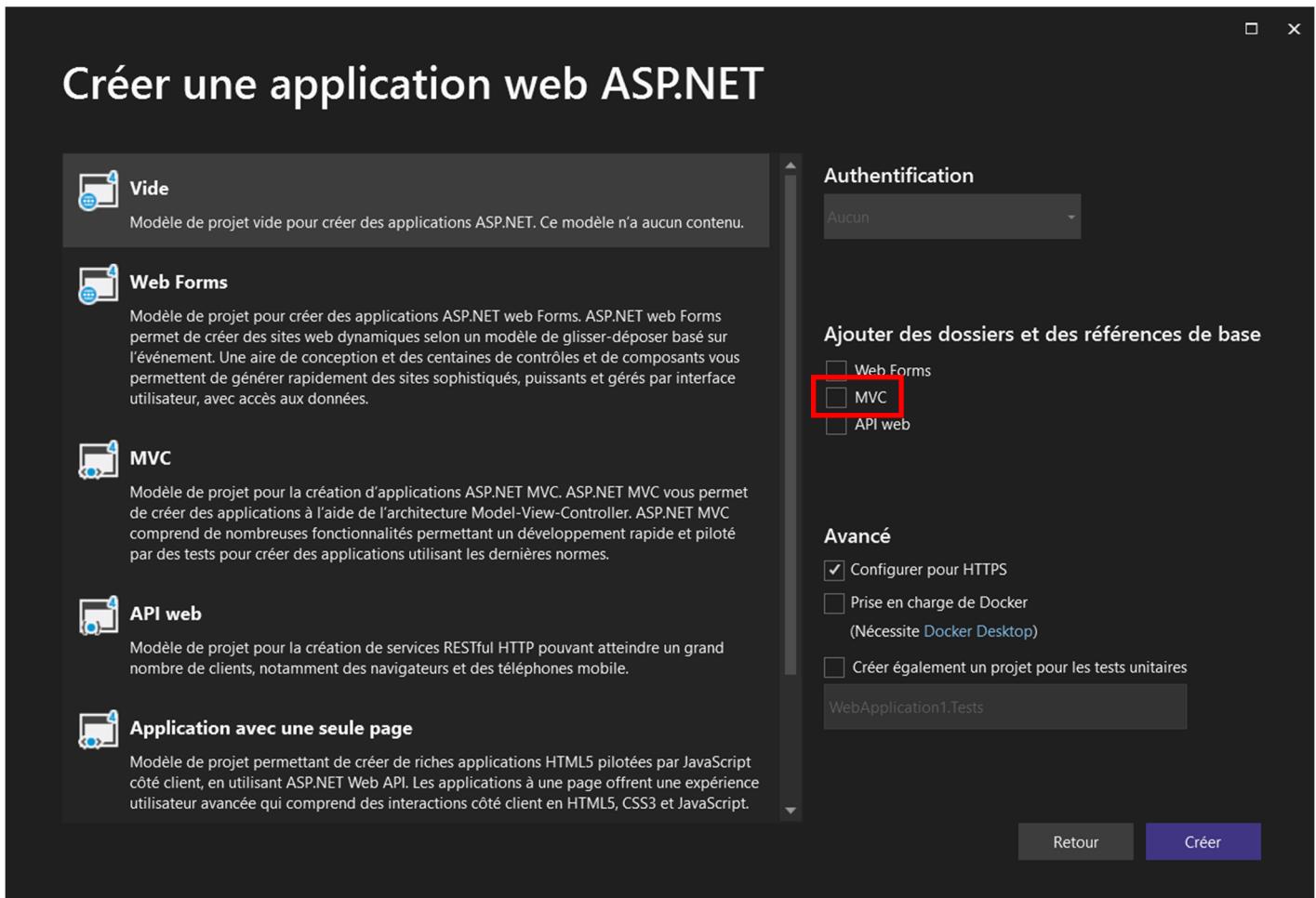


Nommer ensuite votre application comme vous le désirez après l'avoir nommé et choisissez l'infrastructure que vous désirer mais il est conseillé de prendre la dernière version. Après cela Visual studio vous demandera de choisir un modèle mais pour l'instant n'en prenait pas compte décochez juste configuration https et cliquer sur créer. Votre application est désormais créée.

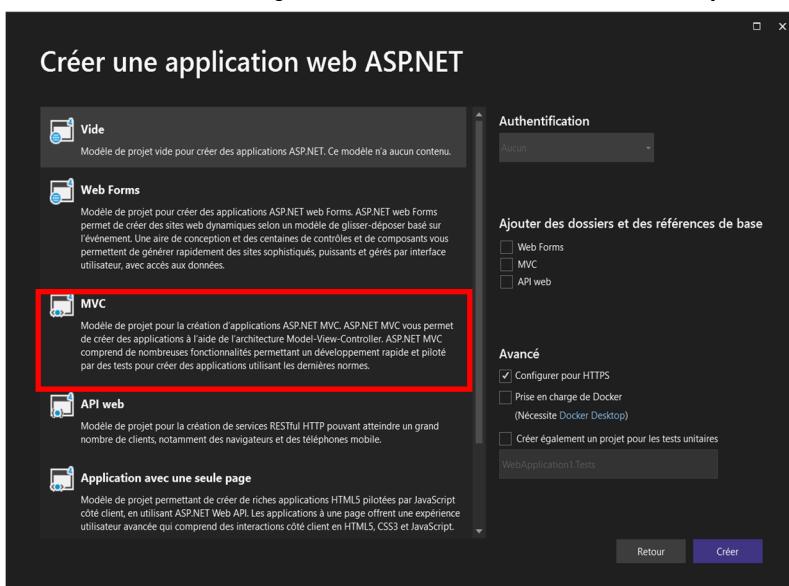


Création d'une application Razor (avec MVC)

Pour créer une application Razor avec une architecture il y a deux manières de procéder qui sont après avoir nommé votre application et choisissez votre infrastructure. Lorsque vous choisissez votre modèle. DÉCOchez configuration pour https et cliquez une fois sur vide puis MVC dans la section ajoutée des dossiers et des références de base.



La deuxième façon et de choisir tout simplement MVC.



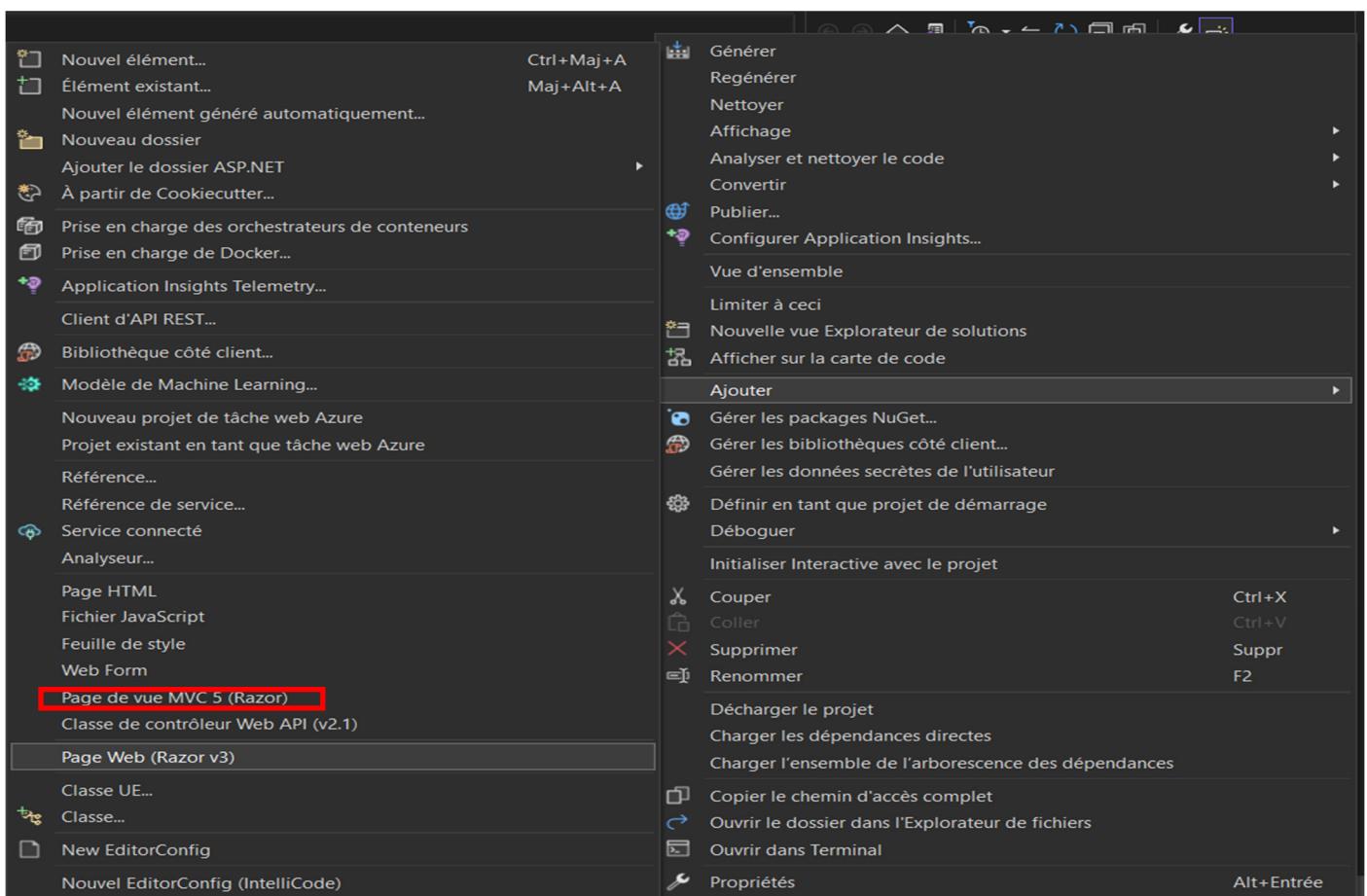
La différence entre les deux et qu'en allant sur la deuxième option il vous générera directement une interface de test et l'architecture de dossiers alors que dans l'autre il ne va créer que l'architecture de dossier vide.

Elément de base d'une application sans MVC

Créer votre première page

Après avoir créé votre application vous allez arriver sur une page de vue d'ensemble ne vous en occupez pas. Pour pouvoir créer votre première page il faut faire un clique droit sur l'espace d'application et aller sur ajouter puis dans ajouter cliqué sur page web (Razor V3), cela va vous créer une nouvelle page avec du code HTML. Mais si vous prenez attention à l'extension du fichier vous verrez qu'en effet ce n'est pas une page .html mais .cshtml ce qui confirme la combinaison C# et HTML.

%



Bloc de code

Razor utilise la syntaxe de balisage comme HTML. Le bloc de code va nous permettre d'incorporer du code .NET sur les pages web. Ici cela va être composé de balisage C# et HTML. Afin de passer de l'un à l'autre vous pouvez utiliser le @ avec une accolade ouverte. Et ainsi vous pourrez écrire du code en C#.

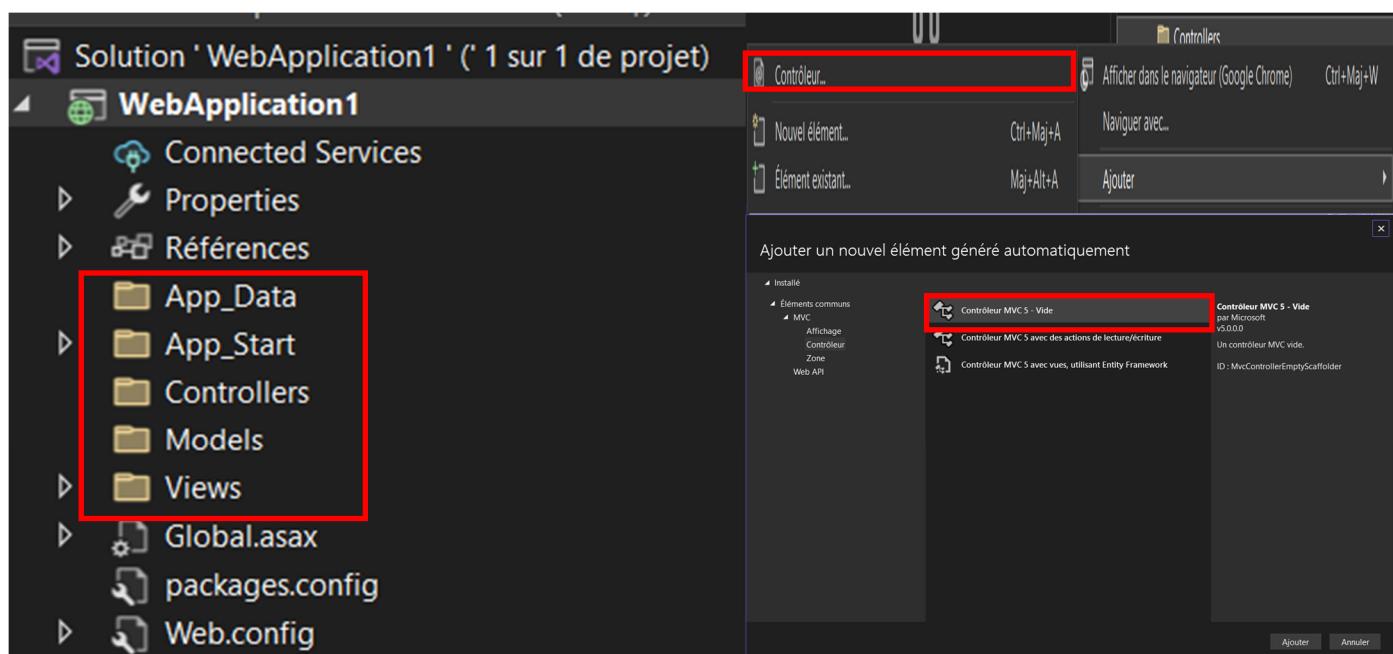
```
@{  
}  
}
```

Il est conseillé de mettre ses blocs de code au début de votre code. Dans les blocs de code nous pouvons absolument tout faire que l'on pouvait faire avec C# à quelque exception près mais sur une page HTML. On peut donc instancier des variables et les utiliser dans du code HTML avec toujours @ suivie de votre nom de variable

Elément de base d'une application avec MVC

Principe des contrôleurs

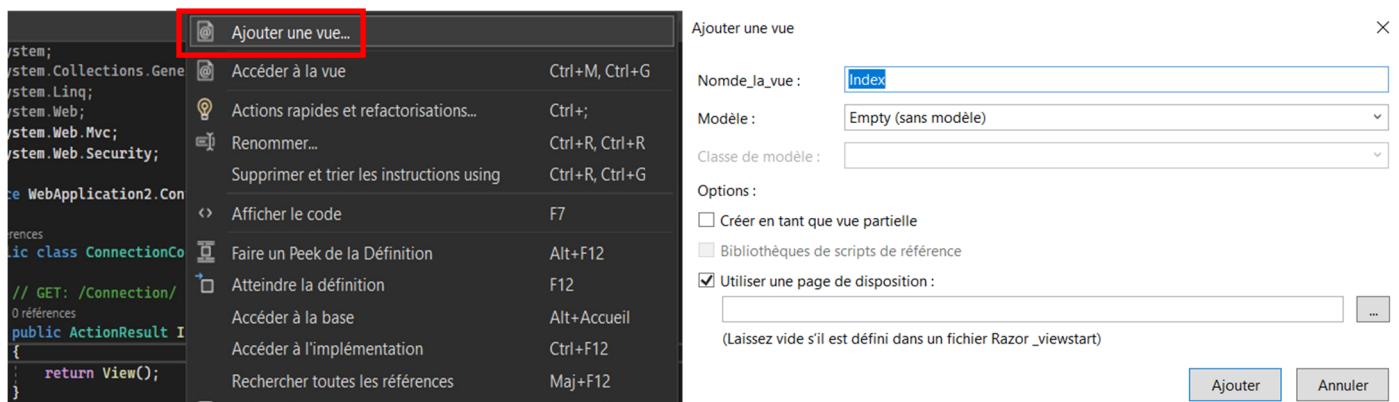
Un contrôleur est une classe ASP.NET qui gère toutes les interactions entre l'utilisateur et l'application web en utilisant des actions pour gérer des requêtes HTTP et les réponses dans le but de générer une vue. Après avoir créé une application avec un modèle vide est en cochant MVC ou avec le modèle MVC vous verrez l'architecture suivante et nous porterons votre attention sur les différents dossiers qui ont été créés car pour coder avec Razor sur une structure il suffit de faire clique droit sur le dossier contrôleurs et en allant d'en ajouter vous verrez « Contrôleur... ». Choisissez un contrôleur MVC 5 vide, les autres serviront à exécuter diverses actions telles que la lecture et l'écriture par exemple. Mais pour l'instant nous ne nous intéresserons uniquement au modèle vide.



Après l'avoir créé et nommé vous remarquerez qu'il l'a directement placé dans le dossier contrôleur. Vous pouvez donc maintenant coder.

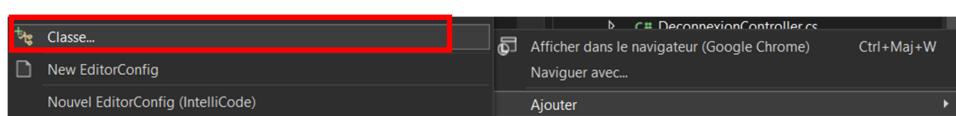
Principe des vues

Maintenant nous allons parler des vues qui sont des fichiers de template définissent la structure de la page web qui va être affichée. Elle va fonctionner de la même manière qu'une page web (Razor V3) car elle va se servir du langage template basé sur HTML et C# afin de combiner du code et contenu afin de générer des pages web dynamiques. On va utiliser les contrôleurs pour créer les vues afin de générer une page web en fonction de la requête utilisateur. Pour créer une vue vous pouvez dans votre contrôleur créer une autre instance `public ActionResult` ou bien utiliser celle qui est de base pour se faire aller dans l'espace et faites un clique droit et vous verrez en premier ajouter une vue. . Une fenêtre apparaît où l'on va configurer notre vue. Dans les modèles on peut choisir le modèle « Empty » afin de choisir une classe de modèle que l'on verra après. Aussi, remarquer que dans les modèles nous retrouvons différentes actions telles que `create` ou `delete` cela sert pour créer des vues déjà complétées afin de créer un élément. Vous verrez après avoir créé la vue que celle-ci est un fichier `.cshtml` confirmant une fois de plus la combinaison des deux langages. Pour finir avec les vues vous verrez que les vues sont classées avec des sous-dossiers représentant les contrôleurs3



Création de modèle

Le modèle est un objet qui va contenir les données qui vont être utilisées pour générer la vue. Il est souvent utilisé pour utiliser ou passer des informations via le contrôleur vers la vue. Ces modèles peuvent contenir des propriétés et méthodes. Pour créer un modèle il faut faire un clique droit sur le dossier `models` afin de créer une classe C# d'en ajouter classes.



Après l'avoir créé vous pouvez maintenant créer vos méthodes et propriétés. Après avoir fait cela nous allons créer une vue avec ce modèle pour ce faire suivait comme précédemment dans les principes des vues la création de celle-ci via le contrôleur après avoir fait cela dans la fenêtre au lieu d'utiliser le modèle Empty (sans modèle), nous prendrons Empty et vous pouvez voir que classent de modèles c'est dégriser et maintenant nous pouvons choisir notre modèle dans la liste déroulante et l'on peut voir qu'il y a déjà des modèles prédéfinis qui est RouteConfig. Après cela vous pouvez créer la vue et remarquerez en haut qu'en haut l'on peut avoir écrit @model WebApplication1.Models.le_modèles ce qui signifie que vous avez réussi à créer une vue via un modèle.

Ajouter une vue X

Nom de la vue :	Index
Modèle :	Empty
Classe de modèle :	
Options :	<ul style="list-style-type: none">le_modèles (WebApplication1.Models)RouteConfig (WebApplication1)
<input type="checkbox"/> Crée en tant que vue partielle	
<input checked="" type="checkbox"/> Bibliothèques de scripts de référence	
<input checked="" type="checkbox"/> Utiliser une page de disposition :	
(Laissez vide s'il est défini dans un fichier Razor _viewstart)	
Ajouter Annuler	

Eléments avancés d'une application

Après avoir vu les élément et principe de base sur la création d'une application avec et sans la structure MVC nous allons voir les éléments plus techniques.

Validations de formulaire

La validation et le processus de vérification des données saisie par l'utilisateur avant l'envoi au serveur, cela permet de voir si les données sont valides et conformes aux règles voici un exemple avec un code avec MVC :

```

public class LoginModel
{
    [Required]
    [StringLength(10)]
    public string Utilisateur { get; set; }

    [Required]
    public string Motdepasse { get; set; }
}

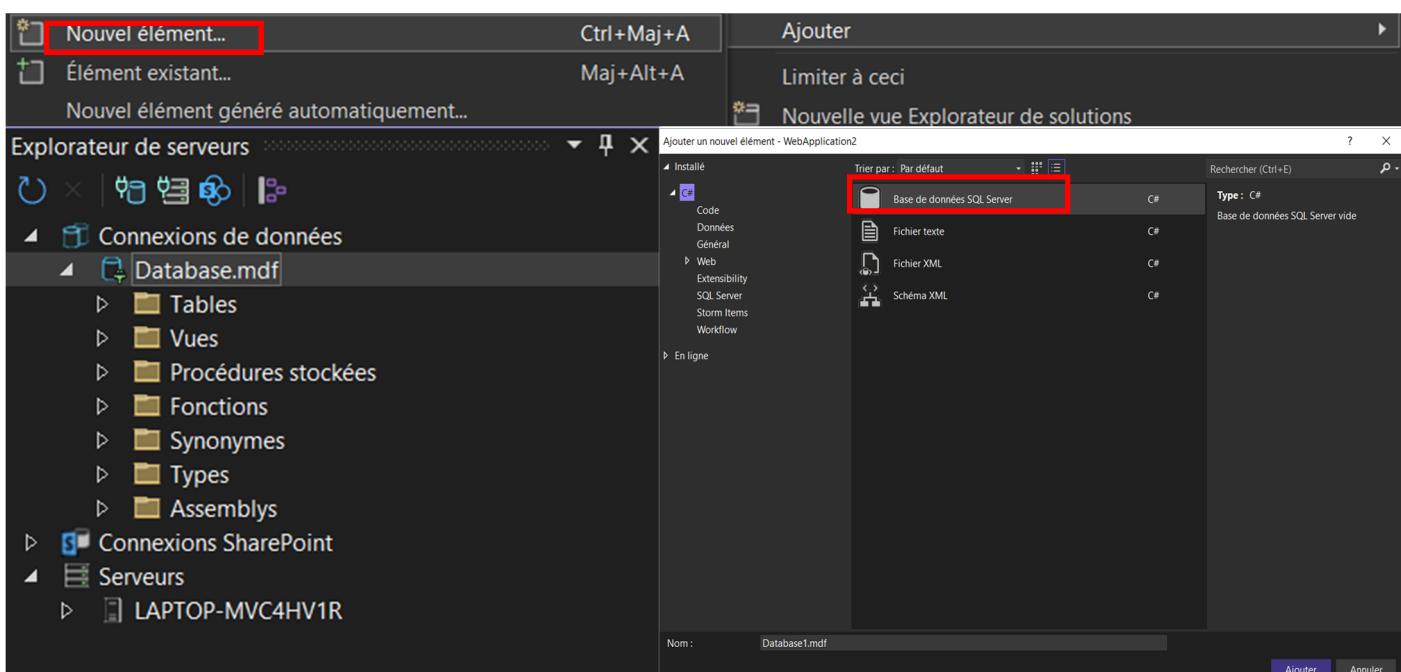
@using(Html.BeginForm())
{
    @Html.LabelFor(m => m.Utilisateur)
    @Html.TextBoxFor(m => m.Utilisateur)
    @Html.ValidationMessageFor(m => m.Utilisateur)
    @Html.LabelFor(m => m.Motdepasse)
    @Html.PasswordFor(m => m.Motdepasse)
    <input type="submit" />
    @Html.ValidationSummary()
}

```

On peut voir que dans le code de droite qui est le code de la vue on crée une form on va créer différents champs qui portent les mêmes noms que les champs de windowsForm où l'on va mettre les champs que l'on a créés dans la classe qui est l'exemple de gauche. On peut voir que dans la classe il y a des éléments entre crochets ceux-ci vont servir à définir des vérifications ici on ne peut qu'utilisateur doit être requis et qu'il doit faire moins ou 10 caractères maximaux.

Accès au données d'une base de données

L'accès au données a une base de données va nous permettre de gérer et vérifier des informations avec ses données, nous allons voir ici le cas où l'on va vérifier des données. Pour créer votre base de données nous allons faire un clic droit sur le dossier App_data et aller dans ajouter > nouvel élément... et choisissez le premier. Vous verrez alors qu'une fenêtre s'ouvre à gauche avec votre serveur.



Suite à cela nous allons créer une table en faisant apparaître via la petite flèche de notre serveur tous ses dossiers, ensuite faisons un clique droit sur tables et ajouter une nouvelle table alors une fenêtre va apparaître et vous pourrez alors gérer votre nouvelle table graphiquement et par code. Si vous faites la table en mode graphique pensait à mettre dès que vous avez fini un point virgules à la fin de votre requête sur la fenêtre du bas. De plus vous pouvez choisir les contraintes de la même manière que les paramètres dans WindowsForm.

Nom	Type de données	Autoriser les valeurs NULL	Par défaut
Id_Utilisateur	int	<input type="checkbox"/>	
Utilisateur	nvarchar(50)	<input checked="" type="checkbox"/>	
Mot_de_Passe	nvarchar(50)	<input checked="" type="checkbox"/>	

Clés (1)
 <sans nom> (Clé primaire, Clustered: Id_Utilisateur)
Contraintes de validation (0)
Index (0)
Cles étrangères (0)
Déclencheurs (0)

Pour créer des éléments dans la table faire un clique droits sur votre table dans la liste des tables et cliquez sur afficher les données de la table. Attention, si votre table n'apparait pas pensez à rafraîchir après l'avoir créé et regardé si vous n'avez pas oublié votre point virgules à la fin de votre requête. Vous pourrez alors ajouter d'autres éléments à votre table.

	Id_Utilisateur	Utilisateur	Mot_de_Passe
▶	3	Jeremy	P@ssw0rd
✖	NULL	NULL	NULL

Pour traiter les données de celle-ci vous devrait taper la commande afin d'accéder à celle-ci `DataAccess.DAL`. Pour ce qui est de la méthode c'est comme avec ODBC il y a juste à créer la connexion et la requête avec nos champs que l'on a créés précédemment cela va avoir pour conséquence de vérifier si l'utilisateur est dans la base de données ou pas.

```

if(ModelState.IsValid)
{
    //if(model.Utilisateur == "Jeremy" && model.Motdepasse == "P@ssw0rd") // Simule
    if(DataAccess.DAL.Utilisateur_Valide(model.Utilisateur, model.Motdepasse))
    {
        FormsAuthentication.SetAuthCookie(model.Utilisateur, true);
        return RedirectToAction("index", "Accueil");
    }
    else
    {
        ModelState.AddModelError("", "Nom d'utilisateur ou mot de passe invalide");
    }
}

bool authentifier = false;

string query = string.Format("SELECT * FROM [User] WHERE Utilisateur = '{0}' And Mot_de_Passe = '{1}'", username, mot_de_passe);

SqlCommand cmd = new SqlCommand(query, connection);
connection.Open();
SqlDataReader sdr = cmd.ExecuteReader();
authentifier = sdr.HasRows;
connection.Close();

return authentifier;
}

```

Eléments de cybersécurité

Maintenant nous allons voir quelques éléments de cybersécurité que nous pouvons réaliser sur Razor. Tout d'abord, nous pouvons hacher un mot de passe de la même manière hors Razor en instanciant un grain de sel dans un tableau de bit et un buffer ensuite on va utiliser l'algorithme Rfc2898DeriveBytes pour hacher notre mot de passe entré dans une zone de texte et le sauvegarder sur une base de données

```

1 référence
public static string HashPassword(string password)
{
    byte[] salt;
    byte[] buffer2;
    if (password == null)
    {
        throw new ArgumentNullException("password");
    }
    using (Rfc2898DeriveBytes bytes = new Rfc2898DeriveBytes(password, 0x10, 0x3e8))
    {
        salt = bytes.Salt;
        buffer2 = bytes.GetBytes(0x20);
    }
    byte[] dst = new byte[0x31];
    Buffer.BlockCopy(salt, 0, dst, 1, 0x10);
    Buffer.BlockCopy(buffer2, 0, dst, 0x11, 0x20);
    return Convert.ToBase64String(dst);
}

```

```

using (var db = new LoginContext())
{
    var user = new LoginModel
    {
        Username = model.Username,
        Password = hashedPassword,
        ConfirmPassword = hashedPassword,
        Email = model.Email,
        TermsAndConditions = model.TermsAndConditions
    };
    db.Login.Add(user);
    db.SaveChanges();
}

```

Ensute on va créer un fichier de log qui va selon nos informations que l'on va saisir nous montrer les éléments que l'on a entré et ajouté. Pour ce faire nous utilisera la méthode writeLog qui va créer un fichier .log avec la date et l'heure en fonction du moment où l'on a effectué l'action.

```
4 références
private void WriteLog(string message)
{
    using (var streamWriter = new StreamWriter(LogFilePath, true))
    {
        streamWriter.WriteLine($"{DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss")}{message}");
    }
}

2023-02-02.log - Bloc-notes
Fichier Edition Format Affichage Aide
2023-02-02 10:31:18 Test958
2023-02-02 10:31:18 test.test@test.test89
2023-02-02 10:31:18 A0gCefLrEv7FN+UMI5qwgIn6gpxRX0ovT299a3f1J/QHaHfH5AQv00jw9uCV+YMwCA==
2023-02-02 10:31:18
2023-02-02 10:48:31 azerty
2023-02-02 10:48:31 azreqfsfd@azefqds
2023-02-02 10:48:31 AFll0MpulBLtBBXlhBp7TD3V35U1uAfrxgE8W0yo/dizWcgUCN0/2EhiPU1BMOSHQ==
2023-02-02 10:48:31
```

```
WriteLog(model.Username);
WriteLog(model.Email);
WriteLog(hashedPassword);
WriteLog(" ");
```

Puis nous allons effectuer un contrôle de saisi via un modèle. Pour ce faire il nous suffit d'écrire entre crochets la méthode que l'on veut par exemple required pour requis ou encore string lenght pour déterminé une longueur maximale dans le champ de texte.

```
[Required]
[StringLength(100, ErrorMessage = "Le {0} doit contenir au moins {2} caractères.", MinimumLength = 8)]
[DataType(DataType.Password)]
[Display(Name = "Mot de passe")]
5 références
public string Password { get; set; }
```

Pour finir avec les éléments de cybersécurité nous allons voir un élément unique au Razor qui est les cookies sécurisés pour ce faire nous allons tout d'abord faire une fonction de hachage normale afin d'hacher nos cookies. Ensuite on va pouvoir créer une variable avec nos cookies hachés. Nous pouvons aussi mettre des paramètres tels que si on veut qu'il soit sécurisé et en httpsonly. La dernière fonction va servir à afficher la valeur des cookies.

```
1 référence
private string HashString(string input)
{
    using (SHA256 sha256Hash = SHA256.Create())
    {
        byte[] bytes = sha256Hash.ComputeHash(Encoding.UTF8.GetBytes(input));
        StringBuilder builder = new StringBuilder();
        for (int i = 0; i < bytes.Length; i++)
        {
            builder.Append(bytes[i].ToString("x2"));
        }
        return builder.ToString();
    }
}
```

```

[HttpPost]
0 références
public ActionResult SetCookie(string cookieValue)
{
    // Hash the cookie value
    string hashedCookieValue = HashString(cookieValue);

    // Create a new cookie with the hashed value
    HttpCookie cookie = new HttpCookie("SecureCookie", hashedCookieValue);
    cookie.HttpOnly = true;
    cookie.Secure = true;

    // Set the cookie in the response
    Response.Cookies.Add(cookie);

    return RedirectToAction("Index");
}

0 références
public ActionResult GetCookie()
{
    HttpCookie cookie = Request.Cookies["SecureCookie"];
    if (cookie == null)
    {
        return Content("Cookie not found");
    }

    return Content("Cookie value: " + cookie.Value);
}

```

Vues Partielles

Pour finir nous allons voir les vues partielles qui sont le fait d'afficher une par exemple un texte sur une autre page en le faisant apparaître et disparaître avec un bouton. En rajoutant cette méthode cela va créer notre vu partiel. Sur la vue qui est dans le chemin d'accès de la surcharge de la méthode. On va donc ajouter cette méthode dans toutes les pages dans lesquelles on veut que la vue partielle s'affiche.

```

@Html.Partial("~/Views/Shared/_Navbar.cshtml")

```

Pour Aller plus Loin et source

Voici quelque lien qui est nos sources mais qui vous permettront d'approfondir et d'aller plus loin dans le Razor :

https://www.w3schools.com/asp/razor_intro.asp

<https://learn.microsoft.com/en-us/aspnet/web-pages/overview/getting-started/introducing-razor-syntax-c>

<https://www.learnrazorpageds.com/>

