

Programmation avancée en c#.NET

Ing.Meryem OUARRACHI

Plan du module

Programmation WEB

- ☐ Généralités outils Web
- ☐ **ASP MVC**
- ☐ ASP.Net core
- ☐ Angular en ASP.Net Core

Génie logiciel en .Net

BI en Self Service

Programmation distribuée avancée

- ☐ Web API
- ☐ GraphQL

CHAPITRE 2:

ASP MVC

Plan du chapitre

1. Structure du projet MVC
2. Gestion d'Etat
3. Mise en forme
4. Entity Framework
5. Razor
6. Les helpers HTML
7. Les validateurs
8. Exemple de l'utilisation de quelques composants
9. Ajax
10. Internationalisation en ASP MVC
11. Sécurité du projet ASP MVC

Entity Framework

Entity Framework

- C'est un ORM (mapping objet-relationnel) : technique de programmation informatique qui crée une base de données orientée objet à partir d'une base de données relationnelle en définissant des correspondances entre cette base de données et les objets du langage utilisé.
- Entity Framework équivalent de Hibernate(J2EE).
- Le principe est proche de linq to sql .

Entity Framework

- Différence entre Linq to sql et Entity Framework

Linq to sql	Entity Framework
Cela ne fonctionne qu'avec la base de données SQL Server.	Il peut fonctionner avec diverses bases de données comme Oracle, MYSQL, SQL Server etc.
Il génère un .dbml pour maintenir la relation	Il génère initialement des fichiers .edmx.
Il ne peut pas générer de base de données à partir du modèle.	Il peut générer une base de données à partir du modèle.

Entity Framework

EF propose trois approches pour créer une base de données:

- L'approche **Database First**: Comme son nom l'indique, cette méthode implique que vous créiez votre base de données de A à Z en SQL puis que vous l'importiez dans votre code.

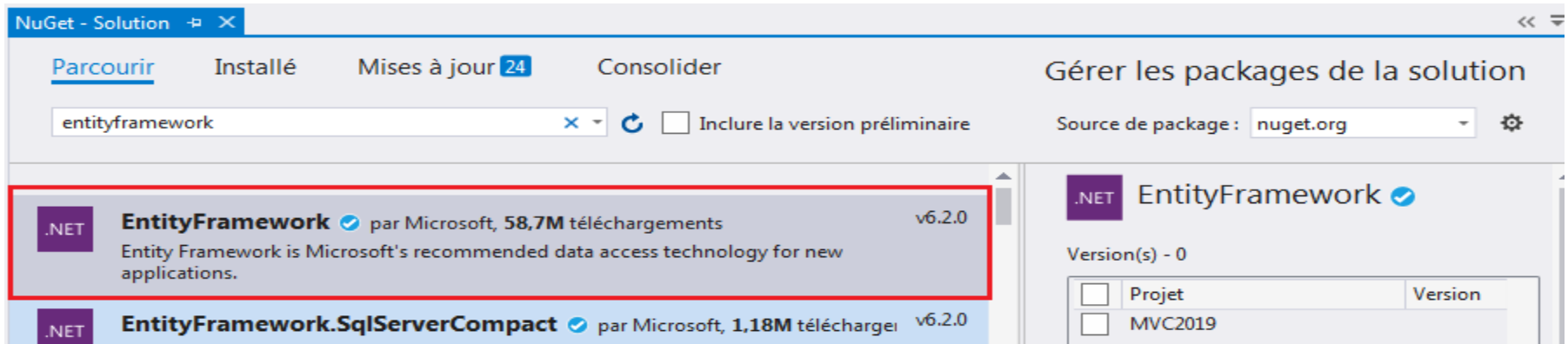
- L'approche **Code First**: consiste à coder des classes(classes POCO) et générer la BD à partir de ces classes.

- L'approche **Model First**: permet de générer la BD à partir du modèle Logique de Données

Entity Framework

Pour utiliser entity framework dans les projets.Net, on doit l'installer dans le projet concerné via Nuget soit:

- Par Console: **Install-Package EntityFramework**
- Ou Par Gestionnaire de package pour la solution:



Entity Framework

Une fois installé, il s'ajoute dans le fichier Web.config

```
<entityFramework>  
  <providers>  
    <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices,  
      EntityFramework.SqlServer" />  
  </providers>  
</entityFramework>
```

Opérations Entity Framework

ClientDataContext cl = new ClientDataContext();

Sélectionner plusieurs lignes	var x =cl.clients.ToList();
Sélectionner une seule ligne	Client x=cl.clients.Find(id=2);
Sélectionner selon une condition	var x = cl.clients.Where(p => p.id > 5 && p.id < 10)

Opérations Entity Framework

ClientDataContext cl = new ClientDataContext();

Requête d'insertion

```
Client nouveauClient = new  
Client();  
nouveauClient.Id=12;  
nouveauClient.nom="aaa" ;  
cl.clients.Add(nouveauClient );  
cl.SaveChanges();
```

Opérations Entity Framework

ClientDataContext cl = new ClientDataContext();

Requête de suppression	<ol style="list-style-type: none">1. On extrait les données à supprimer (stocké le résultat dans une variable x)2. cl.clients. Remove(x);3. cl.SaveChanges();
Requête de modification	<ol style="list-style-type: none">1. On extrait les données à modifier (stocké le résultat dans une variable x)2. x.nom= "rrr";3. cl.SaveChanges(); // valider la modification effectuée

Entity Framework

- **Code First:**

- Technique pour créer une base de donnée à partir du code.

- Il se base sur les annotations

Model → ajouter une classe(dans laquelle on va définir la structure de notre table)

```
public class User
{
    public int id { get; set; }
    public string nom { get; set; }
    public string prenom { get; set; }
}
```

Entity Framework

- **Code First:**

- Points à prendre en considération:

- N'ajouter pas un « s » à la fin de l'identificateur de classe parce que le programme va générer une table ,nommé par le nom de classe+s

- La variable qui porte le nom Id sera par la suite la clé de table crée sinon il faut ajouter l'annotation Key

```
[Column("ID")]  
[Key]  
[DatabaseGenerated(DatabaseGeneratedOption.Identity)]  
public int moncle { get; set; }
```

Entity Framework

-Modifier le nom affiché des colonnes:

Pour cela on utilise cet attribut dans le modèle avant le nom de colonne .

```
[Display(Name = "FisrtName")]  
public string nom { get; set; }
```

-Il faut importer :

```
using System.ComponentModel.DataAnnotations;
```


Entity Framework -Code First-

Objectif: Créer un modèle et générer ses tables via codeFirst, ensuite créer des opérations CRUD sur cette table

Il y'a deux méthodes:

-Méthode1: Code First Migration

-Méthode2: Utiliser le Scaffolding

EF Code First -Code First Migration-

1.Création du modèle

Dans le modèle créer la classe qui représente la structure de notre future table

```
public class User
{
    public int id { get; set; }
    public string nom { get; set; }
    public string prenom { get; set; }
}
```

EF Code First -Code First Migration-

1.Création du modèle

-Relation One-To-many



[Pour Configurer les relations entre les tables, On utilise les annotations et on ajoute des références: une ou plusieurs références vers l'autre classe selon la cardinalité]

-Pour une relation One-To-many: la relation du côté plusieurs qui reçoit comme clé étrangère la clé primaire de la relation du côté 1

EF Code First -Code First Migration-

1.Création du modèle

-Relation One-To-many

```
public partial class etudiant
{
    public int id { get; set; }
    public string nom { get; set; }
    public string prenom { get; set; }
    public string sexe { get; set; }
    public Nullable<System.DateTime> date_naiss { get; set; }
    [ForeignKey("Filiere")]
    public Nullable<int> id_fil { get; set; }
    public string tof { get; set; }

    public virtual Filiere Filiere { get; set; }
}
```

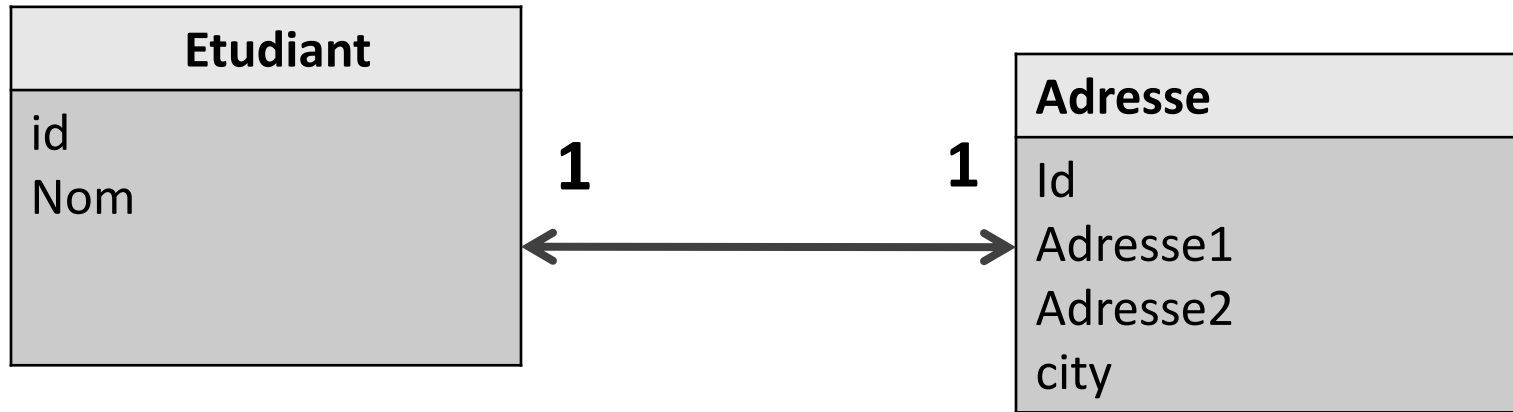
```
public partial class Filiere
{
    public Filiere()
    {
        this.etudiants = new HashSet<etudiant>();
    }
    [Key]
    public int Id_filiere { get; set; }
    public string Nom_filiere { get; set; }

    public virtual ICollection<etudiant> etudiants { get; set; }
}
```

EF Code First -Code First Migration-

1.Création du modèle

-Relation One-To-One



- la clé primaire de l'une des relations doit figurer comme clé étrangère dans l'autre relation.

EF Code First -Code First Migration-

1.Création du modèle

-Relation One-To-One

```
public class Etudiant1
{
    [Key]
    public int EtudiantId { get; set; }
    public string EtudiantName { get; set; }

    public virtual EtudiantAddress Address { get; set; }
}
```

```
public class EtudiantAddress
{
    [ForeignKey("Etudiant1")]
    public int EtudiantAddressId { get; set; }

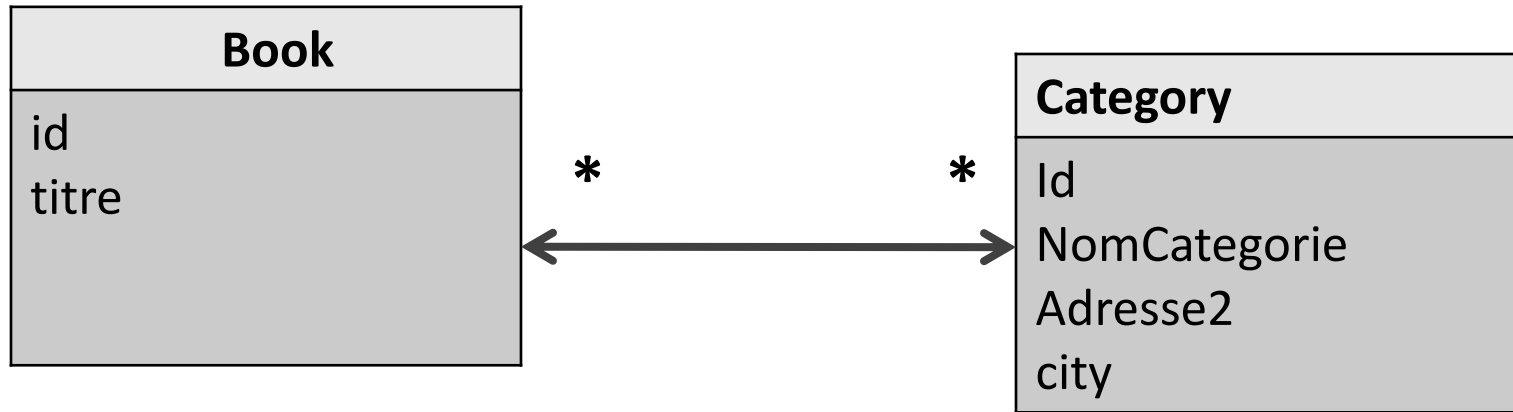
    public string Address1 { get; set; }
    public string Address2 { get; set; }
    public string City { get; set; }
    public string Country { get; set; }

    public virtual Etudiant1 Etudiant1 { get; set; }
}
```

EF Code First -Code First Migration-

1.Création du modèle

-Relation many-To-many



-Dans la BD il faut introduire une nouvelle relation dont les attributs sont les clés primaires des relations en association et dont la clé primaire est la concaténation de ces deux attributs.

EF Code First -Code First Migration-

1.Création du modèle

-Relation many-To-many

```
public class Book
{
    public int BookId { get; set; }
    public string Title { get; set; }

    public ICollection<Category> Categories { get; set; }
}
```

```
public class Category
{
    public int CategoryId { get; set; }
    public string CategoryName { get; set; }

    public ICollection<Book> Books { get; set; }
}
```


EF Code First -Code First Migration-

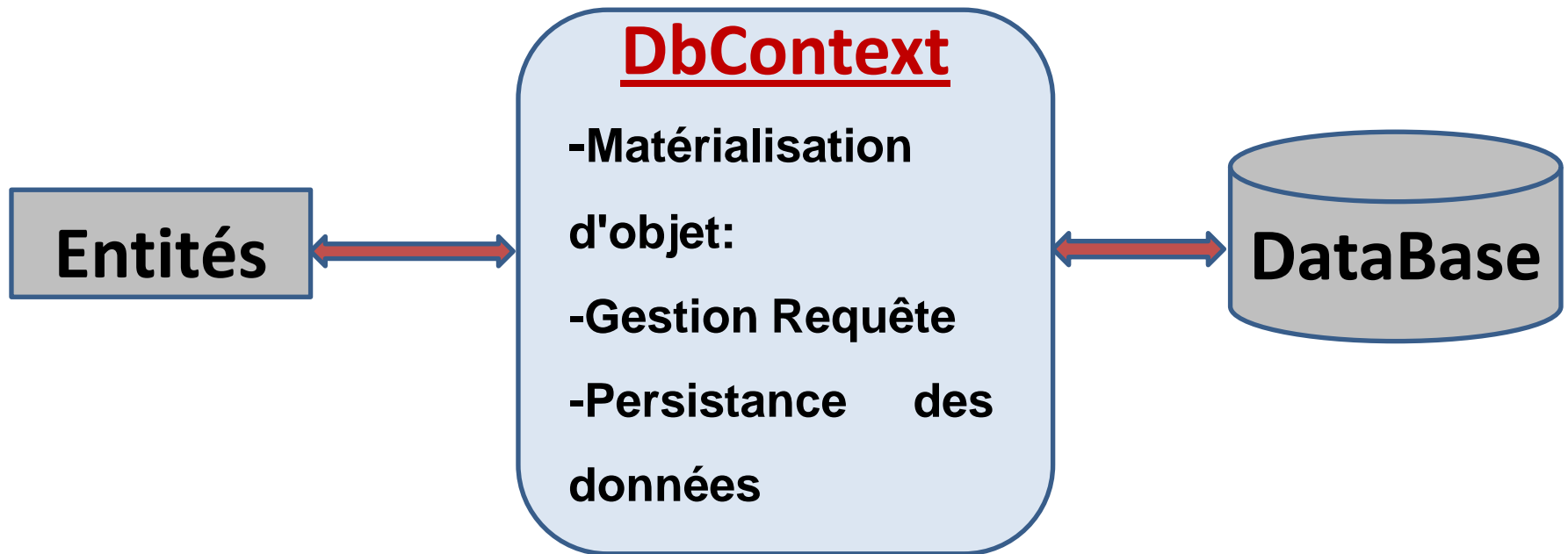
2. Création du contexte de base de données

C'est la classe principale qui coordonne les fonctionnalités d'Entity Framework pour un modèle de données spécifié

```
public partial class UserContext : DbContext
{
    public UserContext()
    {
        : base("name=UserContext")
    }
    public DbSet<User> users{ get; set; }
}
```

EF Code First -Code First Migration-

- **DbContext:** est le pont entre la base de donnée et les classes.



- **DbSet:** représente la collection de toutes les entités dans le contexte, ou qui peuvent être interrogées à partir de la base de données, d'un type donné

EF Code First -Code First Migration-

3. Spécification la chaîne de connexion:

```
<connectionStrings>  
  <add name="UserContext" connectionString="Data Source=(local); Initial Catalog=UserContext;  
    Integrated Security=True;MultipleActiveResultSets=True; AttachDbFilename=|DataDirectory|UserContext.mdf"  
    providerName="System.Data.SqlClient" />  
</connectionStrings>
```

EF Code First -Code First Migration-

3. Exécuter les migrations:

Une migration est une liste d'instruction qui permet à notre ORM d'exécuter une série de requêtes SQL

3.1.Activer la migration:

```
PM> Enable-Migrations -ContextTypeName NomDbContext
```

-L'activation de la migration crée un nouveau dossier Migrations dans notre application contenant un fichier *Configurations.cs*, permettant de définir le comportement de la migration pour le DbContext utilisé.

EF Code First -Code First Migration-

3. Exécuter les migrations:

3.1.Activer la migration:

- Configurations.cs

```
internal sealed class Configuration : DbMigrationsConfiguration<entityframework2020.Models.UserContext>
{
    public Configuration()
    {
        AutomaticMigrationsEnabled = false;
    }

    protected override void Seed(entityframework2020.Models.UserContext context)
    {
        // This method will be called after migrating to the latest version.
        // You can use the DbSet<T>.AddOrUpdate() helper extension method
        // to avoid creating duplicate seed data.
    }
}
```

EF Code First -Code First Migration-

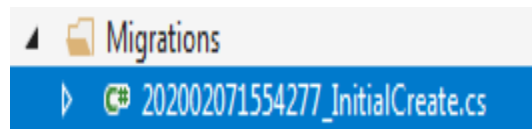
3. Exécuter les migrations

3.2.Déterminer la BD:

```
PM> Add-Migration InitialCreate
```

-InitialCreate est un nouveau fichier qui sera créé dans le dossier Migrations, avec un préfixe timestamp permettant à EF de savoir quels fichiers doivent être exécutés, et dans quel ordre. Il contient une fonction **Up()** qui contient des instructions pour créer la table avec la définition initiale de celle-ci, et une autre fonction **Down()** pour supprimer la table.

-Ce fichier pourra être utilisé pour rétablir l'état original de la base de données.



EF Code First -Code First Migration-

3. Exécuter les migrations

3.2.Déterminer la BD:

- InitialCreate.cs

```
public partial class InitialCreate : DbMigration
{
    public override void Up()
    {
        CreateTable(
            "dbo.Users",
            c => new
            {
                id = c.Int(nullable: false, identity: true),
                nom = c.String(),
                prenom = c.String(),
            })
            .PrimaryKey(t => t.id);
    }

    public override void Down()
    {
        DropTable("dbo.Users");
    }
}
```

EF Code First -Code First Migration-

3. Exécuter les migrations

3.3.Exécuter la migration:

```
PM> Update-Database -TargetMigration:"InitialCreate"
```

-Cette commande va appliquer la migration sur la base de donnée

EF Code First -Code First Migration-

❑ Cas particuliers:

○ Cas1:Plusieurs contexte dans le même projet

si on veut exécuter un autre contexte normalement on doit exécuter la commande enable-migration mais il va donner une erreur comme quoi que la migration est déjà activée c'est pourquoi on a joute le paramètre Force

```
PM> Enable-Migrations -ContextTypeName NomDbContext -Force
```

```
PM> Add-Migration InitialCreate -Force
```

```
PM> Update-Database -TargetMigration:"InitialCreate"
```

EF Code First -Code First Migration-

❑ Cas particulier:

○**Cas2: Mise à jour de BD:** Si on veut exécuter des nouveaux mises à jour dans la BD(ajouter un champ...):

1. On effectue la modification dans le Model
3. Reéxecuter la commande Add-migration → Générer un nouveau fichier InitialCreate
4. Reéxecuter la commande Update-Database sur le nouveau fichier généré

```
PM> Add-Migration InitialCreate
```

```
PM> Update-Database -TargetMigration:"InitialCreate1"
```

```
//InitialCreate1:le nouveau fichier généré
```

EF Code First -Code First Migration-

❑ Cas particulier:

○ **Cas3: Remplir la BD Lors de création:** on a la possibilité d'insérer des données dans nos tables de base de données pendant le processus d'initialisation. Pour cela on va passer à la méthode **seed**. Cette méthode est appelée lorsque la base de données est créée et chaque fois que le schéma de base de données est mis à jour.

```
protected override void Seed(entityframework2020.Models.UserContext context)
{
    var users = new List<User>();
    users.Add(new User() { id = 1, nom = "user1", prenom = "xxx" });
    users.Add(new User() { id = 2, nom = "user2", prenom = "yyy" });
    context.users.AddRange(users);
    base.Seed(context);
}
```

EF Code First -Code First Migration-

Remarque:

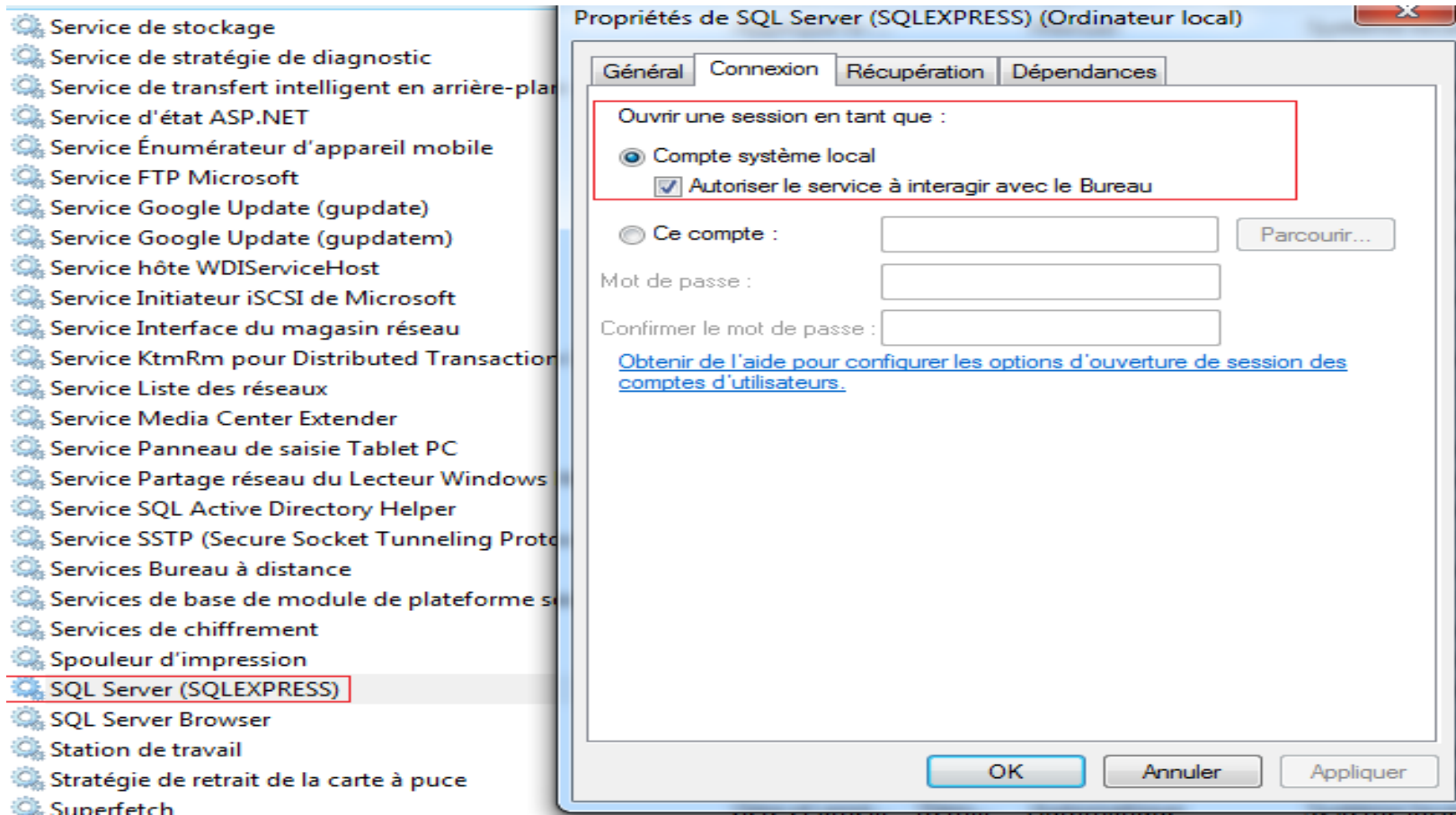
- Au cas des erreurs lors de création de la BD:
 - Vérifier qu'il n'ya pas des problèmes au niveau de chaine de connection dans le Web.config

Exemple: remplacer le DataProvider par le nom de votre server
sql server

- Vérifier les droits d'accès au fichier App_Data
- Dans l'invite de commande ,tapez « services.msc »→ trouver le service SQL Sever et avec clic droit/Propriétés→connexion et voir l'utilisateur connecté

EF Code First -Code First Migration-

- **Code First:**



EF Code First - Scaffolding-

-Le scaffolding est une technique utilisée par de nombreux frameworks MVC comme ASP.NET MVC, Ruby on Rails, Cake PHP et Node.JS etc., pour générer efficacement du code pour les opérations CRUD (créer, lire, mettre à jour et supprimer) de base de données en peu de temps. De plus, vous pouvez modifier ou personnaliser ce code généré automatiquement en fonction de vos besoins.

-Le scaffolding se compose de modèles de page, de modèles de page de champ et de modèles de filtre. Ces modèles sont appelés modèles de scaffolding et vous permettent de créer rapidement un site Web fonctionnel axé sur les données.

EF Code First - Scaffolding-

- **Fonctionnement de scaffolding en ASP MVC:**

- Les modèles de scaffolding sont utilisés pour **générer du code** pour les opérations CRUD de base dans nos applications ASP.NET MVC par rapport à notre base de données avec l'aide Entity Framework.

- En analysant votre modèle, Visual Studio est capable d'en déduire les champs dont vous allez avoir besoin pour réaliser des formulaires classiques et pour générer des actions de contrôleur

- Ces modèles utilisent le système de modèles **Visual Studio T4** pour générer des vues pour les opérations CRUD de base à l'aide d'Entity Framework.

EF Code First - Scaffolding-

- **Fonctionnement de scaffolding en ASP MVC:**

- ASP.NET Scaffolding est un générateur de code pour les applications Web ASP.NET. Visual Studio inclut des générateurs de code préinstallés pour les projets MVC et Web API.

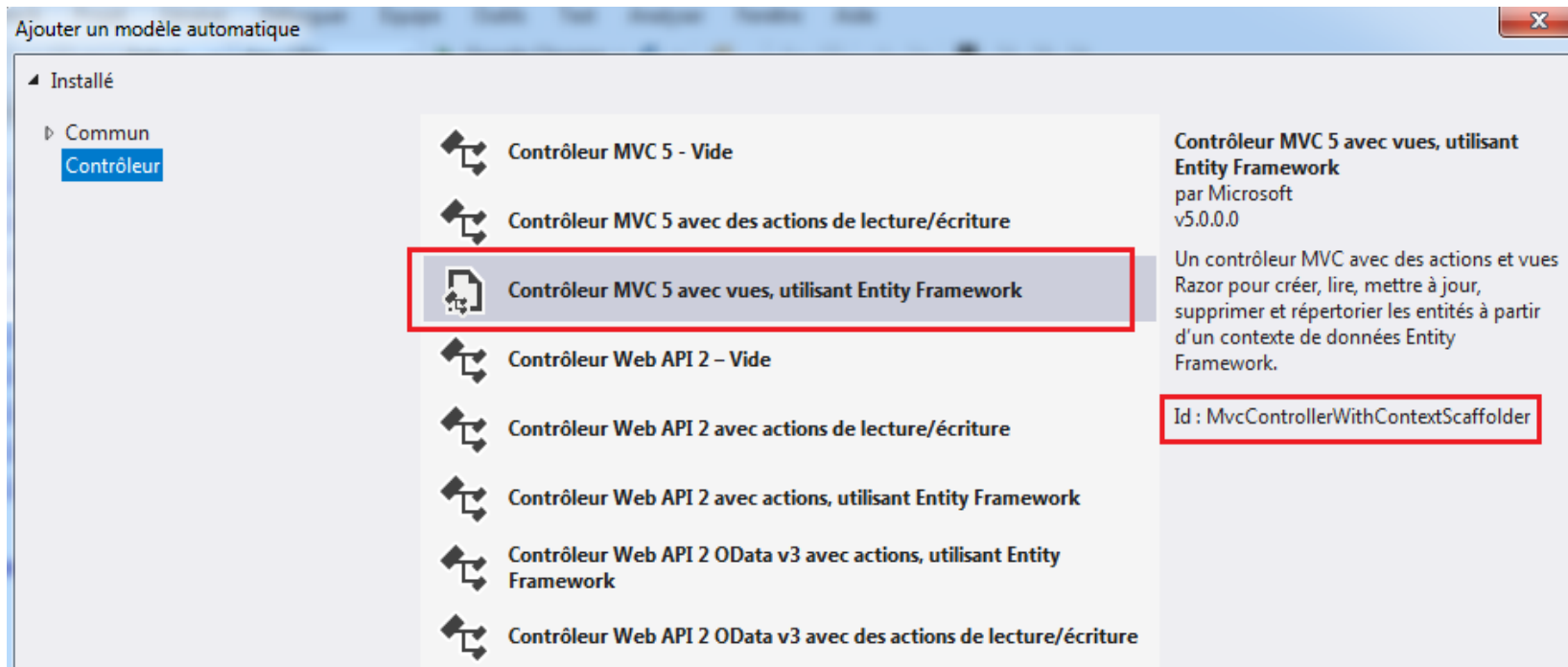
- La **génération de code source** est une opération permettant de générer automatiquement du code source. Son but est d'automatiser la production de code source .Il existe de nombreuses sources à partir desquelles générer le code source :génération à partir de programmes informatiques écrits dans un autre langage de programmation. Le logiciel réalisant cette transformation peut être appelé compilateur . Par exemple, on peut générer du code java ou C# à partir d'un programme décrit en UML

- T4(Text Template Transformation Toolkit) est un générateur de code intégré à Visual Studio

EF Code First - Scaffolding-

1. Dans le modèle créer la classe qui représente la structure de notre future table

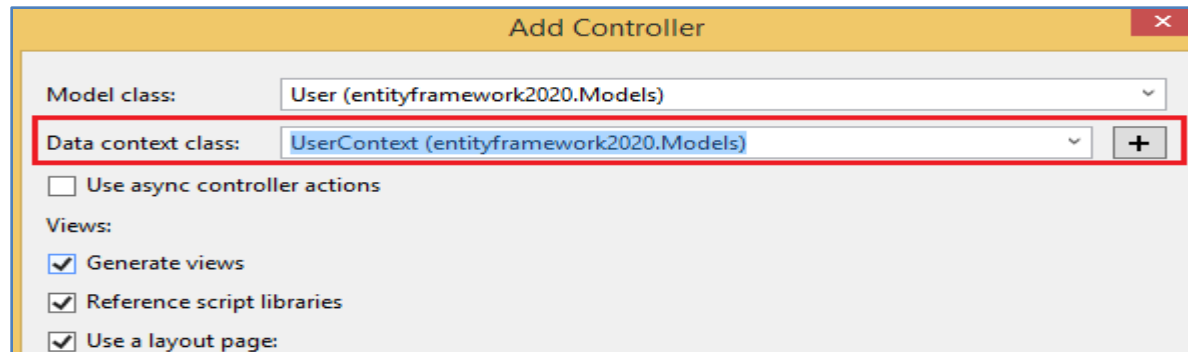
2. Ensuite on crée le controller qui va créer la BD



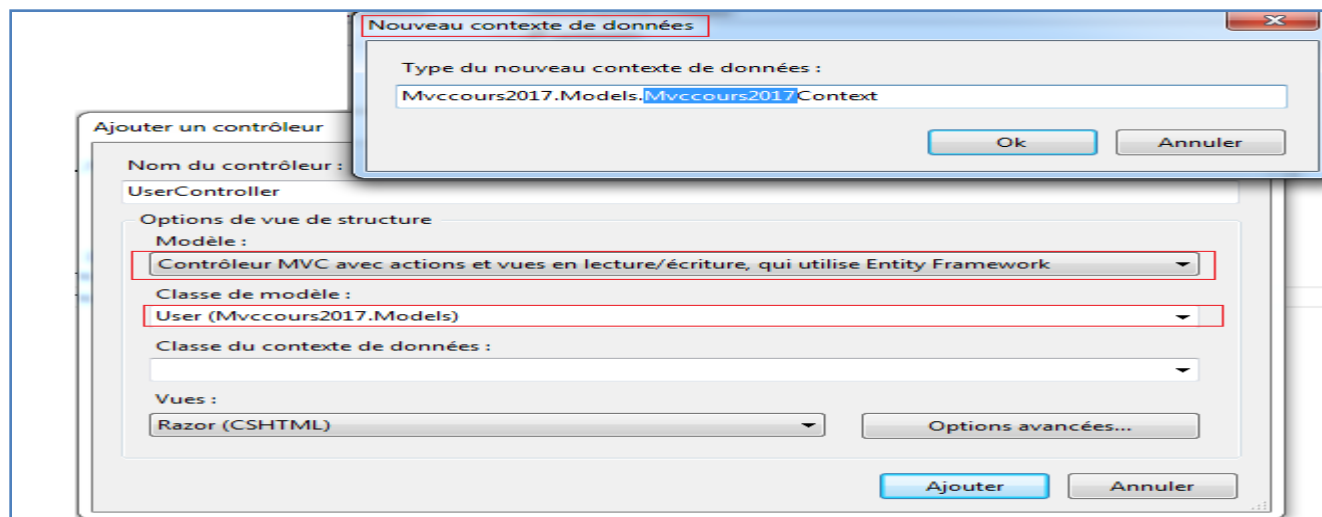
EF Code First - Scaffolding-

2. Ensuite on crée le controller qui va créer la BD

- On choisit le datacontext créé précédemment

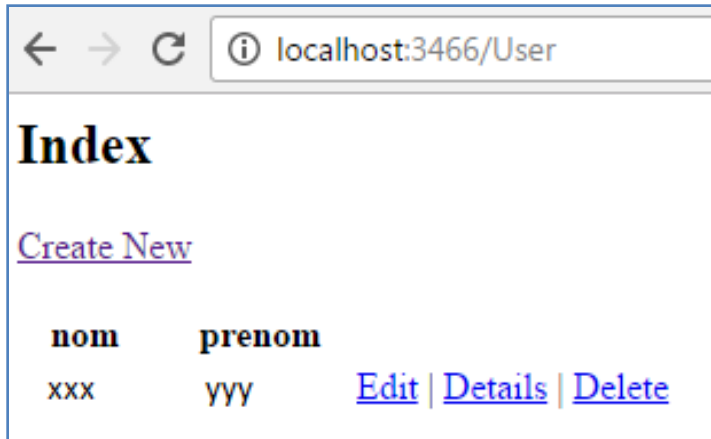


- Ou on ajoute un nouveau contexte



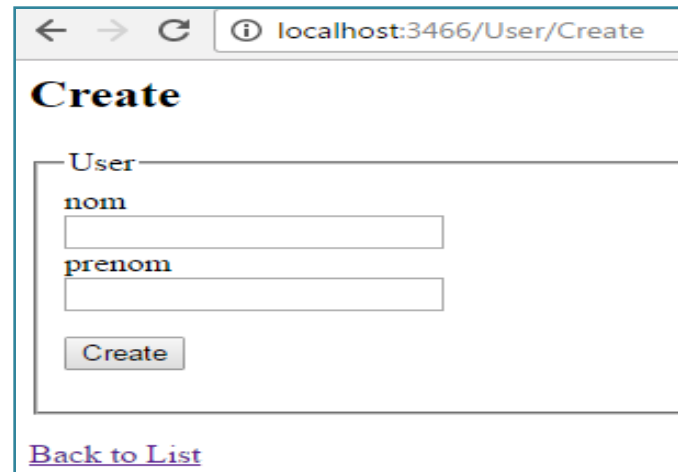
EF Code First - Scaffolding-

3. Exécuter le projet



A screenshot of a web browser window. The address bar shows "localhost:3466/User". The page title is "Index". Below the title is a link "Create New". Below that is a table with two columns: "nom" and "prenom". The first row of data shows "xxx" under "nom" and "yyy" under "prenom". To the right of the data row are three links: "Edit", "Details", and "Delete".

nom	prenom	
xxx	yyy	Edit Details Delete



A screenshot of a web browser window. The address bar shows "localhost:3466/User/Create". The page title is "Create". Below the title is a form with a label "User" and two input fields: "nom" and "prenom". Below the input fields is a "Create" button. At the bottom of the page is a link "Back to List".

User

nom

prenom

Create

[Back to List](#)

Entity Framework

Remarque: si on accède au controller généré ,on trouve que chaque action a deux versions

-**La première:** appelée lorsque on demande cette page la première fois(requête Get)

-**La deuxième:**appelée lorsque on veut valider des modifications [HttpPost])

```
public ActionResult Create()  
{return View(); }  
[HttpPost]  
public ActionResult Create(User user)  
{if (ModelState.IsValid)  
{    db.Users.Add(user);  
    db.SaveChanges();  
    return RedirectToAction("Index");  
}  
return View(user); }
```

Entity Framework

- Récupérer les données dans la vue
- 1^{ère} ligne détermine le modèle qui va être utilisé par la vue

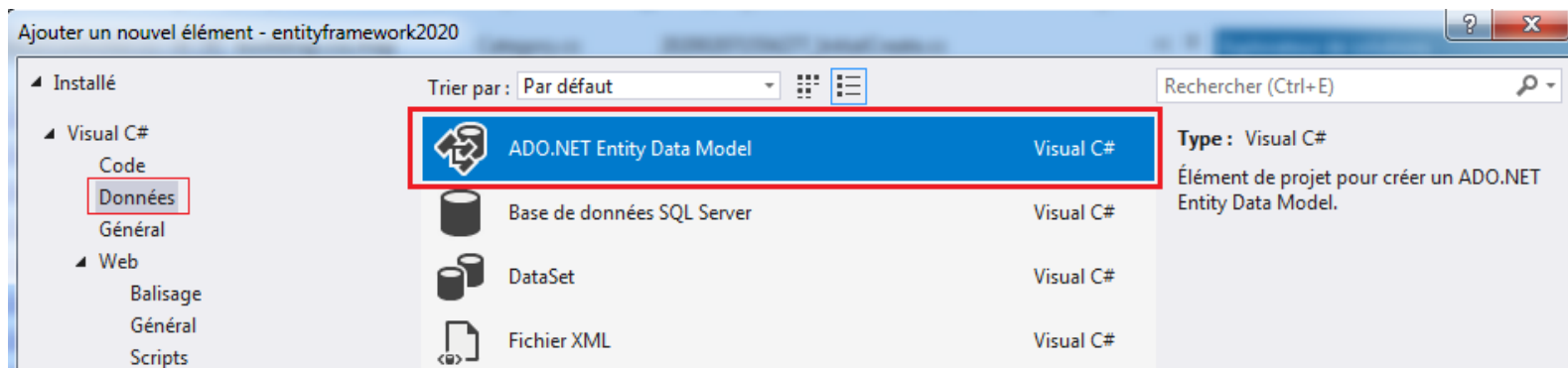
```
@model MvcCodeFirst.Models.User
```

- La vue utilise **Html helper** pour créer l'interface graphique: sont un ensemble des bibliothèques prédéfinies permettant de créer l'interface graphique **@Html.NomdeFonction**
- Lors de l'exécution, ces fonctions vont se traduire en tags html ordinaire

Entity Framework -DataBase First-

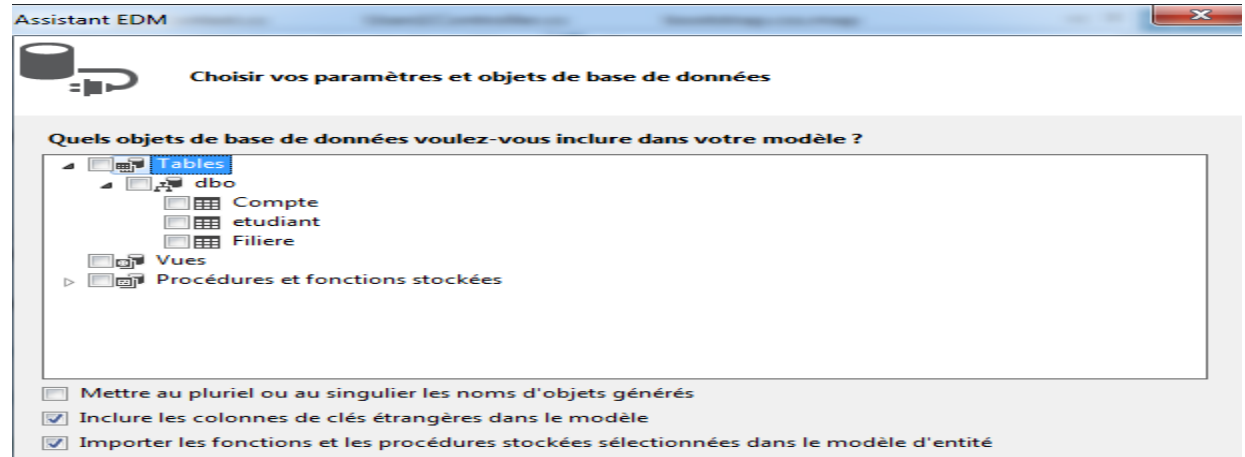
Objectif: on suppose qu'on a une base de donnée et on veut générer sa classe, ensuite on veut effectuer des opérations CRUD sur cette BD.

1. Cette fois on utilise l'approche DataBaseFirst d'Entity Framework : Click droit sur Model → ajouter nouvel élément → ADO.Net Entity Data Model → sélectionner la BD → Suivant → cocher mettre en pluriel → puis générer le projet.

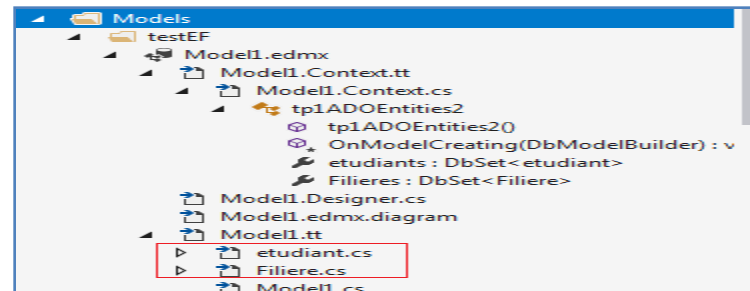


Entity Framework -DataBase First-

-Choisir les tables:



-Les fichiers générés



2.Utiliser les opérations d'Entity Framework pour réaliser des actions
CRUD sur le BD

RAZOR

Razor

-Couramment pour insérer un code c# en page web ASP on utilise les délimiteurs `<% %>`.

→ cette combinaison est assez inconfortable pour l'écriture et peut rendre le code illisible.

Solution: Razor permet de faciliter la conception des pages ASP.NET. Il introduit une syntaxe de programmation assez facilement compréhensible, qui permet d'insérer du code serveur dans une page Web qui peut également contenir du HTML, du CSS et des scripts JavaScript.

Razor

Razor est un moteur de vue qui facilite la création des vues avec ces avantages:

- il permet de limiter le nombre de caractères séparant le code serveur du code HTML ;
- il est facile à apprendre, car nécessite peu de concepts à maîtriser ;
- il améliore également l'IntelliSense avec Visual Studio.

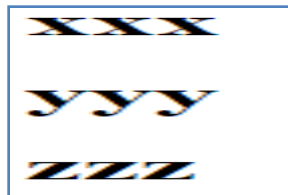
Razor

-La syntaxe : `@{ }` *// cas de plusieurs lignes*

`@` *// cas d'une seule ligne*

-Exemple:

```
@{ string[] names=new string[]{  
    "xxx",  
    "yyy",  
    "zzz",  
};  
  
}  
@foreach(var x in names)  
{<h1>@x</h1>}
```



-Résultat:

Razor

Commentaire @* *@

Razor

- Définir des fonctions:

```
@functions{  
    public string afficher(){  
        return "bonjour";  
    }  
}
```

- Appel:

```
<h1>@afficher()</h1>
```

Remarque: On peut pas intégrer un code html lors de création de fonctions

Razor

- Définir des fonctions:

```
@helper afficher2(){  
    <h1> Bonjour</h1>  
}
```

- Appel:

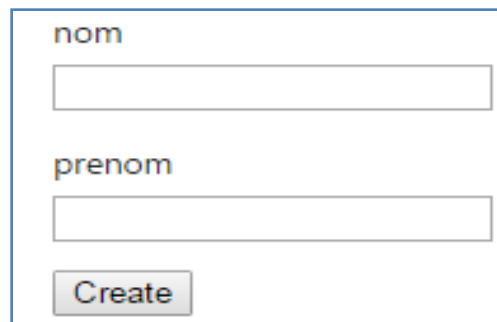
```
@afficher2()
```

LES HELPERS HTML

Les helpers HTML

- La classe html helper représente la prise en charge du rendu des contrôles HTML dans une vue(include dans la classe de base de la vue de razor `WebViewPage`)
- La propriété `@Html` est un objet de la classe **HtmlHelper** (Le symbole @ est utilisé pour accéder à l'objet côté serveur dans la syntaxe du razor).

-Exemple: Créer l'interface suivante:



The image shows a web form interface. It contains two text input fields. The first field is labeled 'nom' and the second field is labeled 'prenom'. Below these fields is a button labeled 'Create'.

Les helpers HTML

-Exemple:

-Html standard

```
@model MvcSolutionTP.Models.User

<input type="text" id="nom" value="@Model.nom" />
<input type="text" id="prenom" value="@Model.prenom" />
<input type="submit" value="Create" />
```

-Html helper

```
@model MvcSolutionTP.Models.User

@Html.TextBox("nom", Model.nom)
@Html.TextBox("prenom", Model.prenom)
<input type="submit" value="Create" />
```

→ Html helper sont un ensemble des bibliothèques prédéfinies permettant de créer l'interface graphique

Syntaxe : @Html.Méthode d'extension

Html Helper Standard

-Exemple:

-TextBox: `@Html.TextBox("txtName", null, new { @class = "textbox", placeholder = "Entrer un nom" })`

```
<input class="textbox" id="txtName" name="txtName" placeholder="Entrer un nom" type="text" value="" />
```

-RadioButton: `OUI: @Html.RadioButton("Réponse", "oui", true)`
`NON: @Html.RadioButton("Réponse", "Non", false)`

```
Oui:<input type="radio" checked="checked" name="Réponse" id="Réponse" value="oui" />  
Non:<input type="radio" name="Réponse" id="Réponse" value="Non" />
```

-CheckBox: `@Html.CheckBox("OK", false)`

```
<input type="checkbox" name="OK" id="OK" value="false" />
```

Html Helper Standard

-Exemple:

-Label:

```
@Html.Label("Nom", "Nom")
```

```
<label for="Nom">Nom</label>
```

Html Helper Standard

- **Navigation en ASP MVC**

- `@Html.ActionLink("Back to List", "Index")`: lien href qui va exécuter l'action qui se trouve dans le deuxième argument et on peut spécifier le controller comme 3ème argument

- `@Html.ActionLink("Edit", "Edit", new { id=Model.id })` : Permet d'envoyer des données dans le lien donc le programme va exécuter l'action Edit et cette dernière va prendre comme paramètre un id: `public ActionResult Edit(int id = 0)`

Html Helper Standard

- **EditorForModel:**

-Renvoie un élément d'entrée HTML pour chaque propriété du modèle.

```
public ActionResult Afficherform()
{
    User user = new User();
    user.id = 25;
    user.nom = "mmm";
    user.prenom = "ppp";
    return View(user);
}
```

```
<h2>AfficherForm</h2>
@Html.EditorForModel();
```

id	<input type="text" value="25"/>
FisrtName	<input type="text" value="mmm"/>
prenom	<input type="text" value="ppp"/>

Html Helper Standard

- Avantages:

-HTML helper standard facilite le développement rapide d'une vue mais on peut créer des applications ASPMVC sans passer par ces helpers

```
@Html.TextBox("txtName", null, new { @class = "textbox", placeholder = "Entrer un nom" })
```

-Ici l'utilisation de helper permet d'affecter le id et le name en une seule écriture

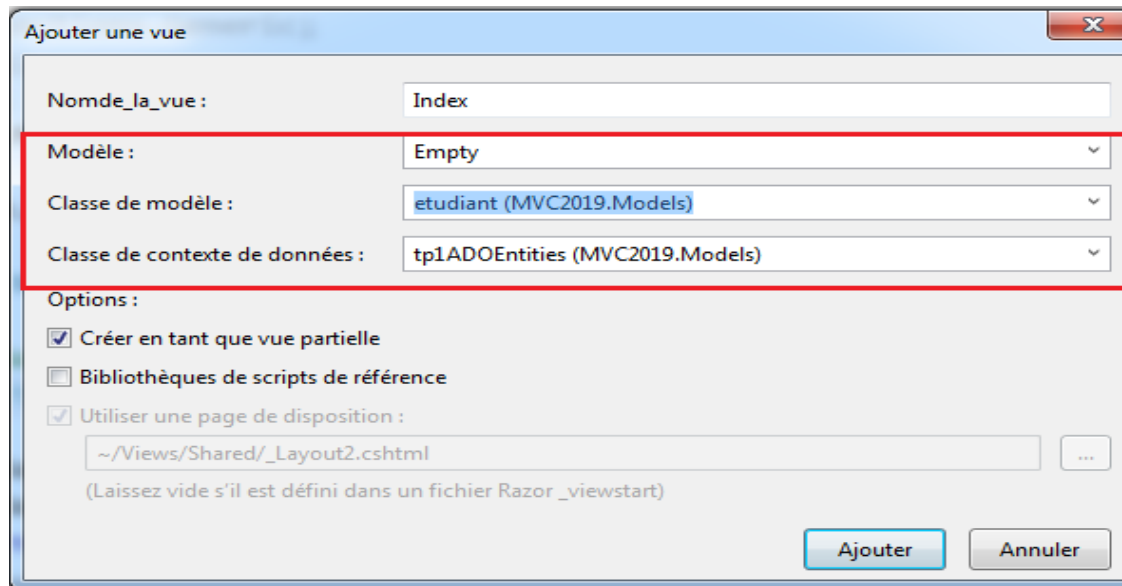
```
<input class="textbox" id="txtName" name="txtName" placeholder="Entrer un nom" type="text" value="" />
```



En utilisant les helpers on écrit moins de code

Html helper fortement typé

On a aussi les html helper fortement typé qui nécessitent de créer dès le début une vue fortement typée.



Cela ajoute automatiquement une référence de modèle sélectionné dans la vue.

Html helper fortement typé

les html helper utilisent les expressions lamda pour lier les contrôles au modèles de donnée

```
NOM: @Html.LabelFor(x => x.nom)
```

```
@Html.TextBoxFor(X => X.nom)
```

```
@Html.ValidationMessageFor(x => x.nom)
```


Html helper fortement typé

Il existe de nombreuses méthodes d'extension pour la classe Html Helper , qui crée différents contrôles html.

HtmlHelper	Html Control
Html.ActionLink	Lien d'ancrage
Html.TextBoxFor	Zone de texte
Html.TextAreaFor	TextArea
Html.CheckBoxFor	Case à cocher
Html.RadioButtonFor	Bouton radio
Html.DropDownListFor	Dropdown, combobox
Html.ListBoxFor	Boîte de liste multi-sélection
Html.HiddenFor	Champ caché
Html.PasswordFor	Boîte de texte de mot de passe
Html.DisplayFor	Texte Html
Html.LabelFor	Étiquette
Html.EditorFor	Génère des contrôles Html basés sur le type de données de la propriété du modèle spécifié, par exemple, la zone de texte pour la propriété de la chaîne, le champ numérique pour l'int, le double ou un autre type numérique.

Html helper fortement typé

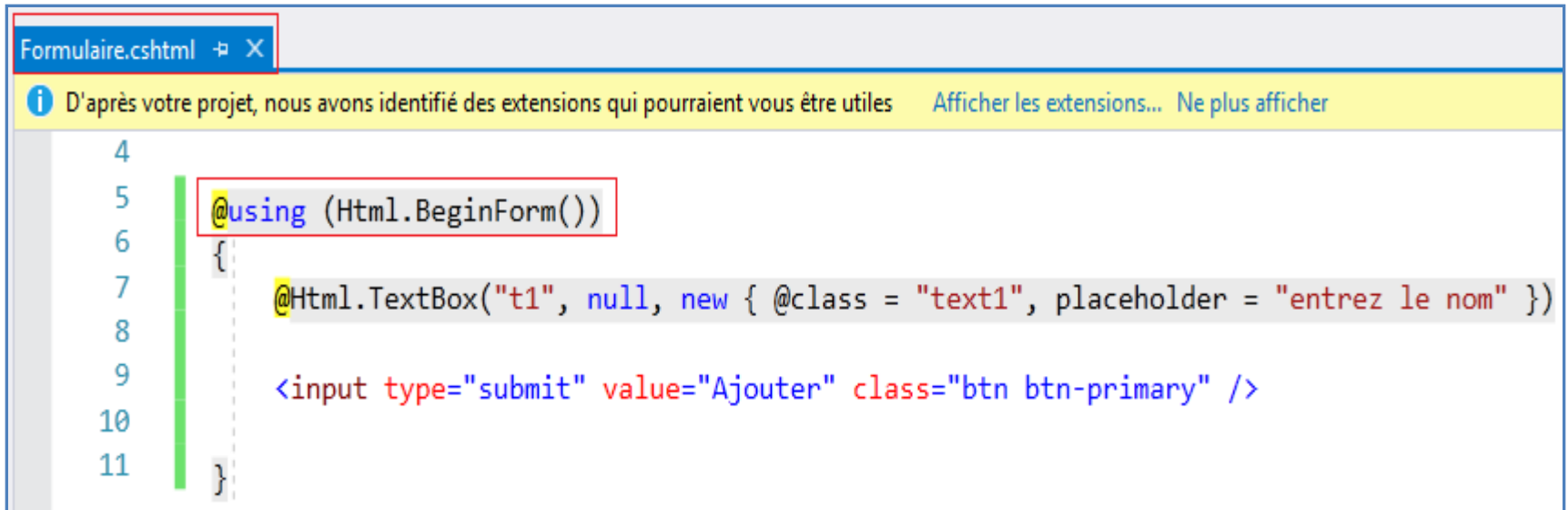
- Ses avantages:

La classe HtmlHelper génère des éléments html à l'aide de l'objet de la classe modèle en mode razor. Il lie l'objet modèle aux éléments html pour afficher la valeur des propriétés du modèle dans les éléments html et affecte également la valeur des éléments html aux propriétés du modèle tout en soumettant le formulaire Web.

```
@using (Html.BeginForm())  
{  
    @Html.LabelFor(x=>x.nom)  
    @Html.TextBoxFor(x=>x.nom)  
  
    <input type="submit" value="Ajouter" />  
}
```

Formulaire

- Sans paramètres:



```
Formulaire.cshtml
i D'après votre projet, nous avons identifié des extensions qui pourraient vous être utiles Afficher les extensions... Ne plus afficher
4
5 @using (Html.BeginForm())
6 {
7     @Html.TextBox("t1", null, new { @class = "text1", placeholder = "entrez le nom" })
8
9     <input type="submit" value="Ajouter" class="btn btn-primary" />
10
11 }
```

il recherchera une action de publication portant le même nom de la page où nous sommes. Par exemple, ici il va rechercher une action 'Formulaire' définit dans le même controller responsable de l'affichage de formulaire dès le début.

Formulaire

- Sans paramètres:

```
[HttpGet]
public ActionResult Formulaire()
{
    return View();
}
```

```
[HttpPost]
public string Formulaire(string t1)
{
    string s = "valeur tapée=" + t1;
    return s;
}
```

Formulaire

- Avec des paramètres:

Nom de l'action Nom de controller Form Method

```
@using (Html.BeginForm("Search", "Navigation", FormMethod.Post, new { @class = "my_form" }))
```

-Exemple:

```
@using (Html.BeginForm("Search", "Navigation", FormMethod.Post))
{
    @Html.TextBox("t1", null, new { @class = "text1", placeholder =
        "entrez le nom" })

    <input type="submit" value="Ajouter" class="btn btn-primary" />
}
```

```
public string Search(string t1)
{
    string s = "valeur tapée=" + t1;
    return s;
}
```

Formulaire

Remarque:

1. Toujours on peut définir plusieurs formulaires dans la même page html.
2. Si on veut enlever la validation dans un button submit → ajouter l'attribut « formnovalidate » dans le button

```
@using (Html.BeginForm("Search", "Navigation", FormMethod.Post, new { id = "my_form" }))  
{  
    @Html.TextBox("txt1", null, new { @class = "text1", form = "my_form", required = "required" })  
  
    <input type="submit" value="Action2" class="btn btn-primary" />  
  
    <input type="submit" value="Action1" formnovalidate class="btn btn-primary" />  
}
```

Formulaire

Remarque:

3. Si on veut attacher un élément à un formulaire → passer par l'attribut « form »

```
@using (Html.BeginForm("Search", "Navigation", FormMethod.Post, new { id = "my_form" }))
{
    @Html.TextBox("txt1", null, new { @class = "text1", form = "my_form", required = "required" })

    <input type="submit" value="Action2" class="btn btn-primary" />

    <input type="submit" value="Action1" formnovalidate class="btn btn-primary" />
}

@Html.TextBox("u2", null, new { @class = "text1", form = "my_form", required = "required" })
```

Formulaire

Remarque:

4. Formulaire avec plusieurs bouton submit : Afin de préciser l'action à appliquer sur chaque bouton, il y'a deux méthodes:

-Méthode 1: Mettre un seul formulaire puis dans l'action faire un test sur la valeur de bouton afin de déterminer les instructions à exécuter dans chaque cas (selon le bouton sélectionné)

```
<input name="bu" value="Ajouter" type="submit"/>
<input name="bu" value="Retour" type="submit"/>
```

```
public ActionResult Index(string bu)
{
    if (bu == "Ajouter")
    {
        //Traitement à exécuter
    }
    else
    {
        //Traitement à exécuter
    }
}
```


Formulaire

Remarque:

4. Formulaire avec plusieurs bouton submit : Afin de préciser l'action à appliquer sur chaque bouton, il y'a deux méthodes:

-Méthode2: Ajouter un deuxième formulaire

```
<input name="bu" value="Ajouter" type="submit" />  
<button name="bu" type="button" onclick="$('#RetourForm').submit()">Retour</button>  
</div>
```

```
@using (Html.BeginForm("ActionRetour", "MonController", FormMethod.Post, new { id =  
"RetourForm" })) { }
```

LES VALIDATEURS

Les validateurs

• **Etape 1**: on ajoute des annotations de validation prédéfinie dans le modèle en dessus des attributs liés aux champs qu'on veut vérifier :

-[Required] : indique que le champ ne doit pas être vide.

-[StringLength]: indique la longueur maximale de la chaîne de caractères entrés.

-[Range] : donne les bornes max et min pour une valeur numérique.

-[Compare]: compare si deux propriétés ont la même valeur.

Les validateurs

•Exemple:

```
public class Etudiant
{
    [Required]
    public String Nom { get; set; }
    // le prenom doit avoir une longueur entre 3 et 7
    [Required]
    [StringLength(7, MinimumLength = 3)]
    public String Prenom { get; set; }
    // age doit être entre 18 et 60
    [Required]
    [Range(18, 60)]
    public int age { get; set; }

    [Required]
    public string champ1 { get; set; }
    //champ2 doit être égal au champ1
    [Compare("champ1")]
    public string champ2 { get; set; }
}
```

Les validateurs

-[RegularExpression] : indique si une propriété correspond à une expression quelconque

Exemple:

```
[RegularExpression(@"\A(?:[a-z0-9!#$%&'*/+=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*/+=?^_`{|}~-]+)*  
@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.[a-z0-9](?:[a-z0-9-]*[a-z0-9])?)\Z")]  
public string Email  
{  
    get;  
    set;  
}
```

Les validateurs

- Etape2:

- Dans le controller, on vérifie la validation de formulaire :

`ModelState.IsValid=true`

- Etape3:

on fait l'appel dans la vue où on veut poser le message d'erreur par l'expression suivante :

- `@Html.ValidationMessageFor(model => model.NomPropriété)`

- `@Html.ValidationSummary()`: pour rassembler la liste des erreurs (équivalent de validation Summary en asp webforms)

Les validateurs

- Remarque:

- Si on veut personnaliser le message d'erreur

```
[Required(ErrorMessage = "Entré un nomutil.", AllowEmptyStrings = false)]  
[Display(Name = "Identifiant")]  
public string NomUtil { get; set; }
```

Les validateurs

•Remarque:

Si on veut changer le style des validateurs,soit:

-On définit le style dans le validateur

```
@Html.ValidationMessageFor(x => x.nom, "", new { @style = "color: green;border: 1px solid red;font-size:x-large" })
```

-On redéfinit les styles des classes responsables de style appliqués par défaut sur les validateur dans le fichier Site.css

```
.field-validation-error {  
    color: red; }  
  
.field-validation-valid {  
    display: none;}  
  
input.input-validation-error {  
    border: 1px solid green;}  
  
.validation-summary-errors {  
    color: blue;}
```


Fonctionnement de validation

- Coté client: Validation se fait coté client via les javascript
voir le fichier script/jquery.validate
- Coté serveur:via `ModelState.IsValid` qui teste la validation
à partir de modèle (équivalent de `Page.isvalid` en asp
webforms)