

# PlateForme C#.Net

**Ing. Meryem OUARRACHI**

# Plan

- ❑ **L'environnement .Net**
- ❑ **Initiation à la programmation C#**
- ❑ **Programmation Orienté Objet C#**
- ❑ **Programmation avancée en C#**

## CHAPITRE 6:

**ASP .Net core**

# 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**

**11. Sécurité du projet ASP MVC**

# Spécificités de ASP

# Objectif

- .Net possède un ensemble de fonctionnalités dédiées à la création et à la gestion des applications Web.
- ASP.NET permet de créer des sites Web dynamiques.

# Privilèges d'ASP

- Plus sécurisée :Selon le *cabinet de sécurité WhiteHat Security*, la sécurité est légèrement meilleure avec *ASP .NET* qu'avec *JSP*, à cause du fait qu'il y a une meilleure orientation de la sécurité pour les développeurs. Mais, les chiffres sont très proches l'un de l'autre, la densité de vulnérabilité est de 27,2 pour le .NET et de 30,0 pour le Java.
- Solution utilisée par des nombreux gouvernements et institutions financières.
- Exemple des sites en ASP:Massar,Stackoverflow

# Modèles de programmation ASP.Net

ASP.NET prend en charge un certain nombre de modèles de programmation pour la création d'applications Web

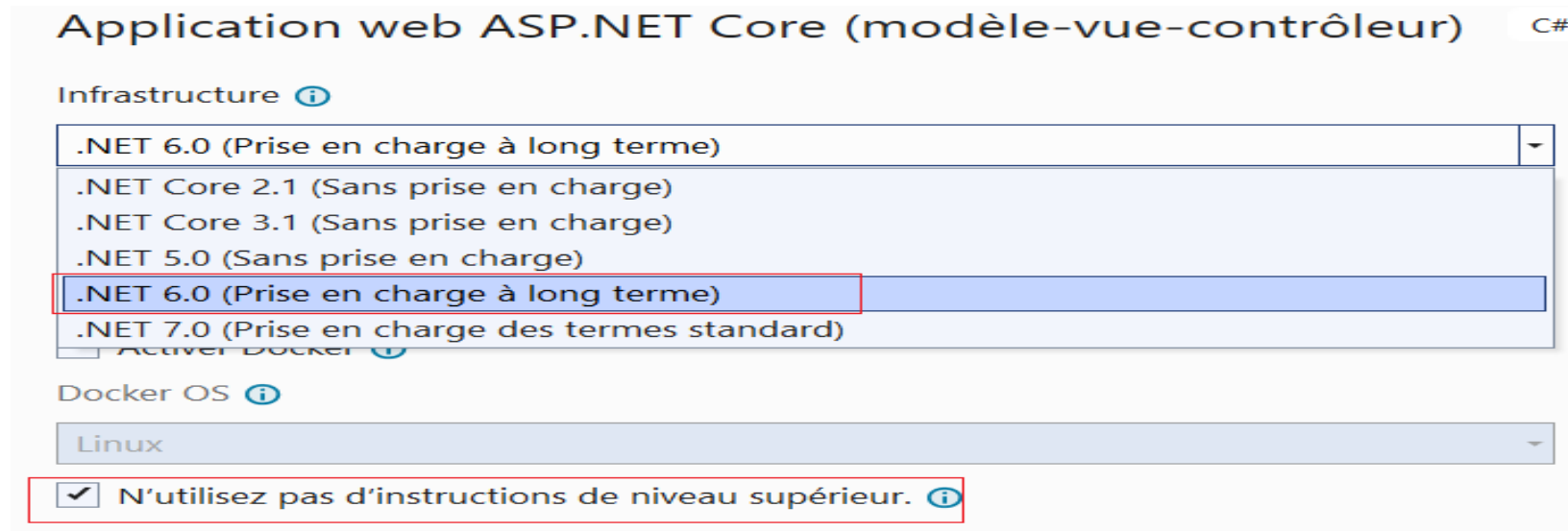
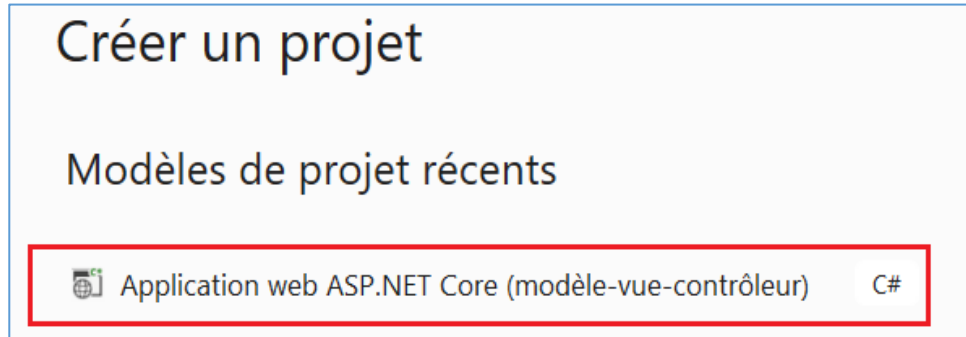
- ❑ **ASP WEB Forms:** Framework pour la construction de pages modulaires à partir de composants, avec des événements d'interface utilisateur en cours de traitement côté serveur.
- ❑ **ASP MVC:** permet de créer des pages Web à l'aide du modèle de conception Modèle – Vue – Contrôleur
- ❑ **ASP .Net Core:** framework open source multiplateforme et hautes performances pour la création d'applications web modernes.



# Structure du projet MVC

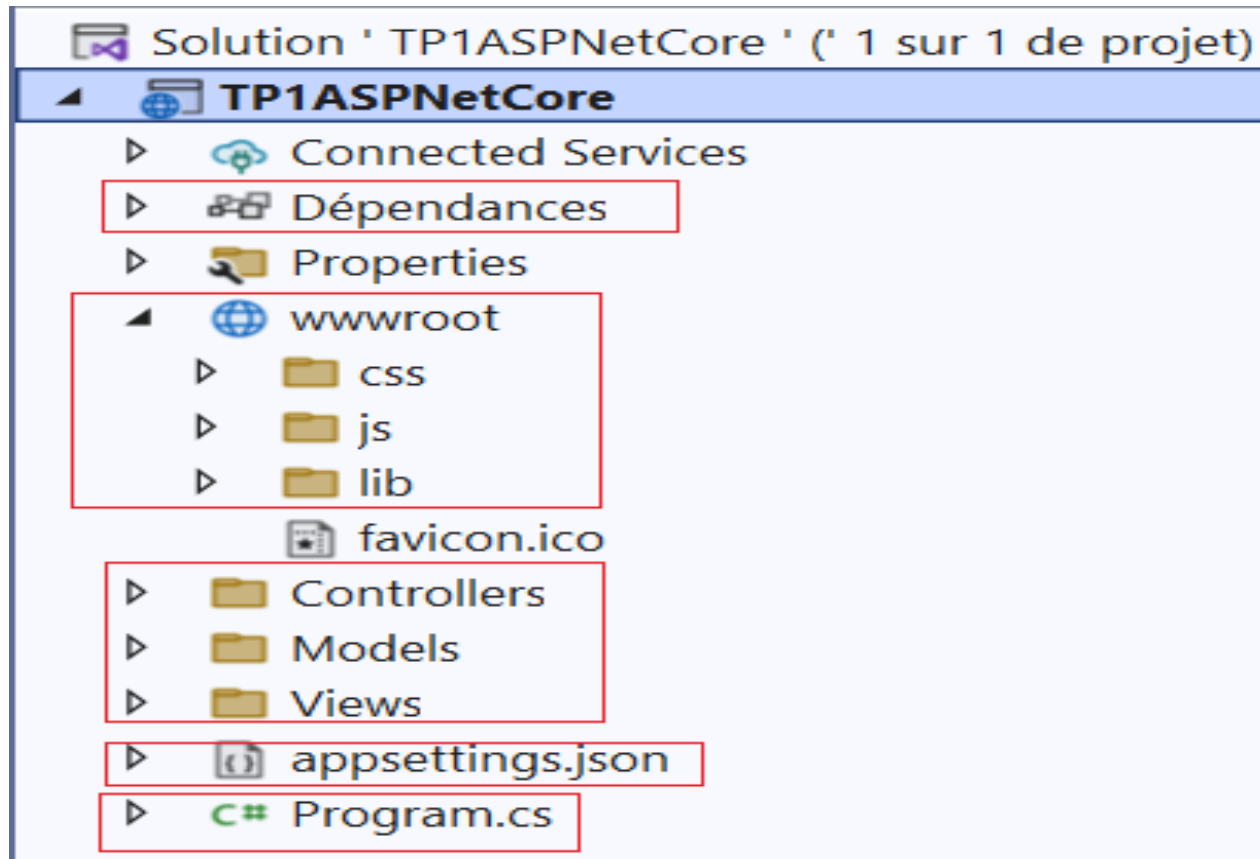
# Architecture d'une application ASP.Net Core

Ajouter nouveau projet → Application web ASP.Net Core (Modèle MVC)



# Architecture d'une application ASP.Net Core

Ajouter nouveau projet → ASP.Net Core



# Architecture d'une application ASP.Net Core

- Dépendances:** contient toutes les librairies utilisées dans le projet. lorsque nous ajouterons de nouvelles dépendances via NuGet, elles seront ajoutées ici.
- Appsetting.json:** fichier de configuration de l'application :paramètres de chaine de connection
- wwwroot:** contient les fichiers statiques de l'application:html,css,images,javascripts...
- Program.cs:**repérse la classe du démarrage càd le main. Il dispose toutes les configuration nécessaires de démarrage du projet

# Architecture d'une application ASP.Net Core

## •Program.cs:

```
public class Program
{
    0 références
    public static void Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args);

        // Add services to the container.
        builder.Services.AddControllersWithViews();

        var app = builder.Build();

        // Configure the HTTP request pipeline.
        if (!app.Environment.IsDevelopment())
        {
            app.UseExceptionHandler("/Home/Error");
            // The default HSTS value is 30 days. You may want to change
            app.UseHsts();
        }
        app.UseHttpsRedirection();
        app.UseStaticFiles();
        app.UseRouting();
        app.UseAuthorization();
        app.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");

        app.Run();
    }
}
```

# Exécution d'un projet ASP.Net core

- Dans un projet ASP.NET Core, le serveur web est généralement configuré via Kestrel. Kestrel est un serveur web open source développé par Microsoft et destiné à être utilisé avec les applications ASP.NET Core. Il est conçu pour être léger, performant et extensible. Kestrel peut être utilisé seul ou en combinaison avec un serveur web inversé tel qu'IIS (Internet Information Services) ou Nginx pour traiter certaines tâches spécifiques.
- **Serveur par Défaut** : Kestrel est le serveur web par défaut pour les applications ASP.NET Core. Lorsque vous créez une nouvelle application ASP.NET Core, Kestrel est généralement configuré comme le serveur qui va écouter les requêtes HTTP.

# Exécution d'un projet ASP.Net core

- Les modèles de projet ASP.NET Core utilisent Kestrel par défaut lorsqu'ils ne sont pas hébergés avec IIS. Dans le fichier suivant Program.cs généré par modèle, la méthode `WebApplication.CreateBuilder` appelle `UseKestrel` en interne :

```
var builder = WebApplication.CreateBuilder(args);  
var app = builder.Build();
```

# Exécution d'un projet ASP.Net core

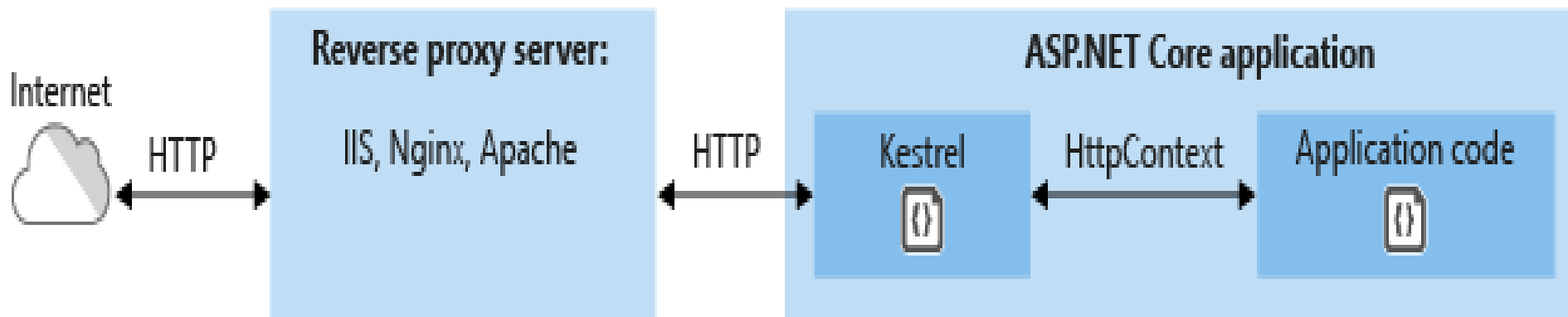
- Changer le port
- Si on veut spécifier le port à utiliser passer au fichier appsetting.cs

```
"Kestrel": {  
  "Endpoints": {  
    "Http": {  
      "Url": "http://localhost:5400"  
    },  
    "Https": {  
      "Url": "https://localhost:5401"  
    }  
  }  
}
```



# Exécution d'un projet ASP.Net core

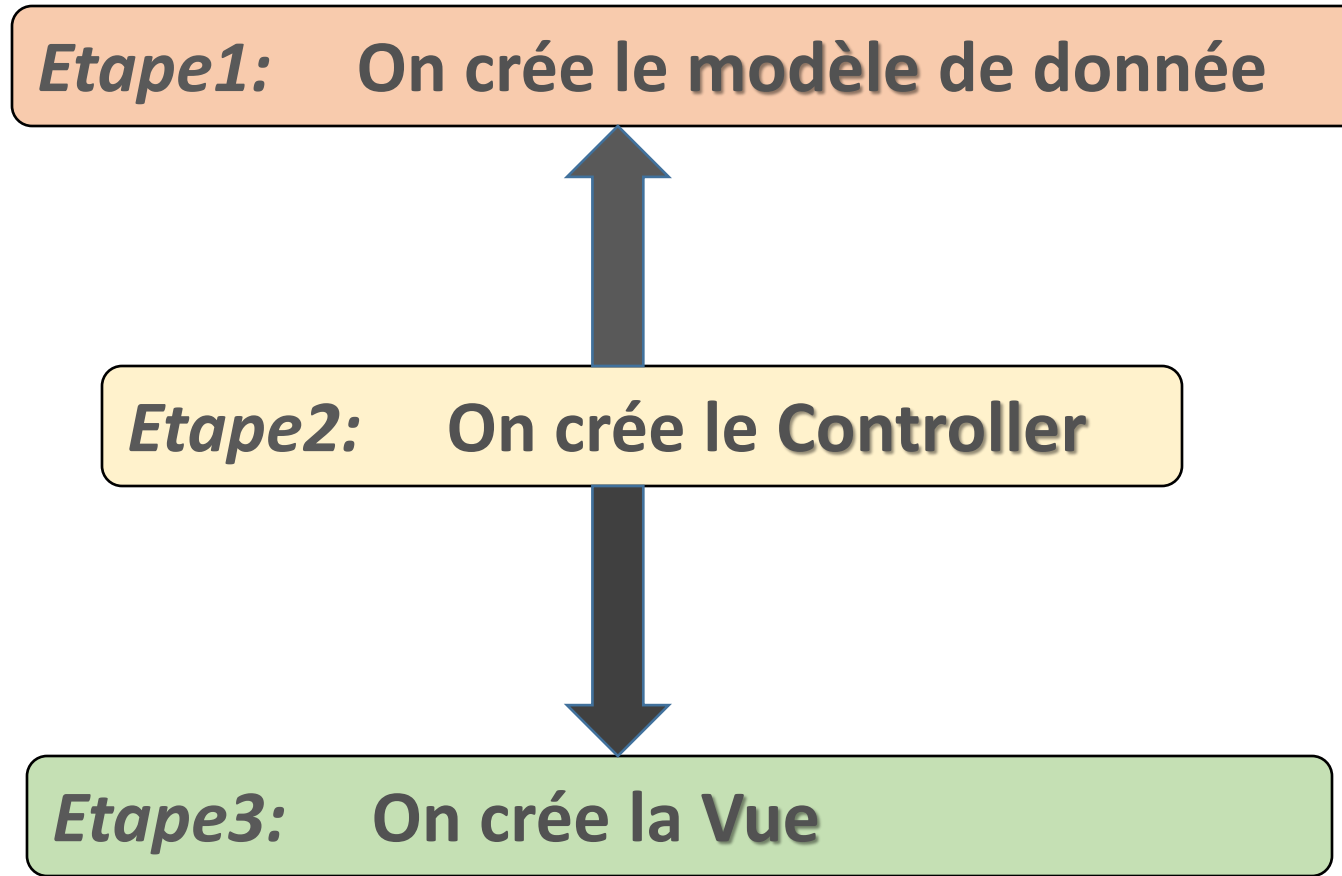
- **Reverse Proxy** : Bien que Kestrel soit performant, il est souvent recommandé de l'utiliser avec un serveur web inversé tel qu'IIS ou Nginx dans un déploiement de production. Le serveur web inversé agit comme un intermédiaire entre Kestrel et Internet, assurant des fonctionnalités telles que la gestion des connexions SSL, la mise en cache, la compression, etc.



# Le patron de conception MVC

- Le MVC est un pattern architectural qui sépare les données (le modèle), l'interface homme-machine (la vue) et la logique de contrôle (le contrôleur).
- Ce modèle de conception impose donc une séparation en 3 couches :
- Le modèle** : Il représente les données de l'application. Il définit aussi l'interaction avec la base de données et le traitement de ces données.
- La vue** : Elle représente l'interface utilisateur, ce avec quoi il interagit. Elle n'effectue aucun traitement, elle se contente simplement d'afficher les données que lui fournit le modèle.
- Le contrôleur** : Il gère l'interface entre le modèle et le client. Il va interpréter la requête de ce dernier pour lui envoyer la vue correspondante. Il effectue la synchronisation entre le modèle et les vues.

# Etape de création des pages ASP MVC



# Modèle

-C'est une classe qui devra se créer dans le dossier Model, porte les données de projets

```
namespace MvcCours2017.Models
{
    public class Etudiant
    {
        public int idEt;
        public string nom;
        public string prenom;
        public int age;
    }
}
```

# Controller

- Il représente le point de démarrage de projet ASP MVC
- Il a pour rôle d'interpréter les requêtes de client pour lui envoyer les réponses(vues).
- C'est une classe qui va contenir un ensemble des actions.
- Création du controller: click droit sur le dossier controller→Ajouter un controller →il existe plusieurs modèles de contrôleur. Pour l'instant, nous choisirons le contrôleur MVC - Vide.

# Controller

-La classe Controller fournit des méthodes qui répondent aux requêtes HTTP envoyées à un site Web ASP.NET MVC.

```
public class EtudiantController : Controller
{
    //
    // GET: /Etudiant/

    public ActionResult Index()
    {
        return View();
    }
}
```

# Controller

- Dans une classe de contrôleur, chaque méthode publique représente une action.
- ActionResult signifie que la méthode va réaliser l'action correspondant à son nom.
- Dans cet exemple l'application donnera l'URL suivante <http://localhost/Etudiant/Index>.
- L'exécution de cet exemple va déclencher une erreur: car on a pas encore créer la vue.

# View

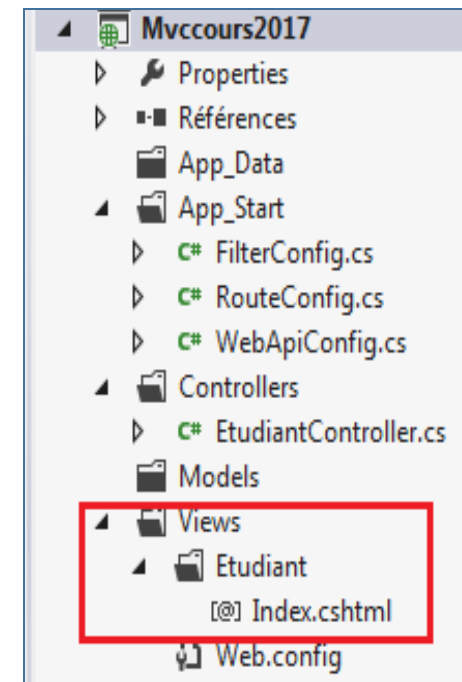
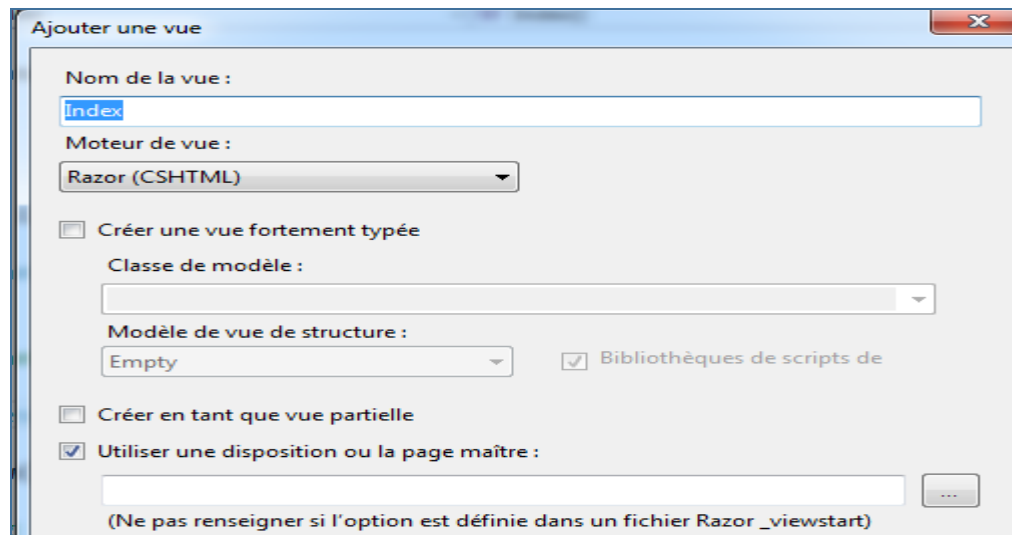
- `return View()` dans le controller: amène le programme à chercher dans le dossier View, un sous-répertoire portant le même nom que le controller dans le quel on trouve la vue(a le même nom que l'action)
- Càd dans le dossier View on doit avoir un sous-répertoire Etudiant portant une page Index(nom de l'action exécuté).
- A savoir que par défaut, ASP.NET MVC va chercher la vue dans le sous-répertoire du même nom que le contrôleur, mais si elle n'existe pas, il ira voir ensuite dans le sous-répertoire Shared.



# View

## -Création de vue:

On clique au dessus de l'action du controller qui va être responsable de l'affichage de vue et on choisit ajouter une vue



# View

-Si on veut appeler une vue qui porte un nom différent que l'action on met `return View("Index2")` mais toujours la recherche est dans le même sous répertoire.

```
public ActionResult Index()  
{  
    return View("Index2");  
}
```

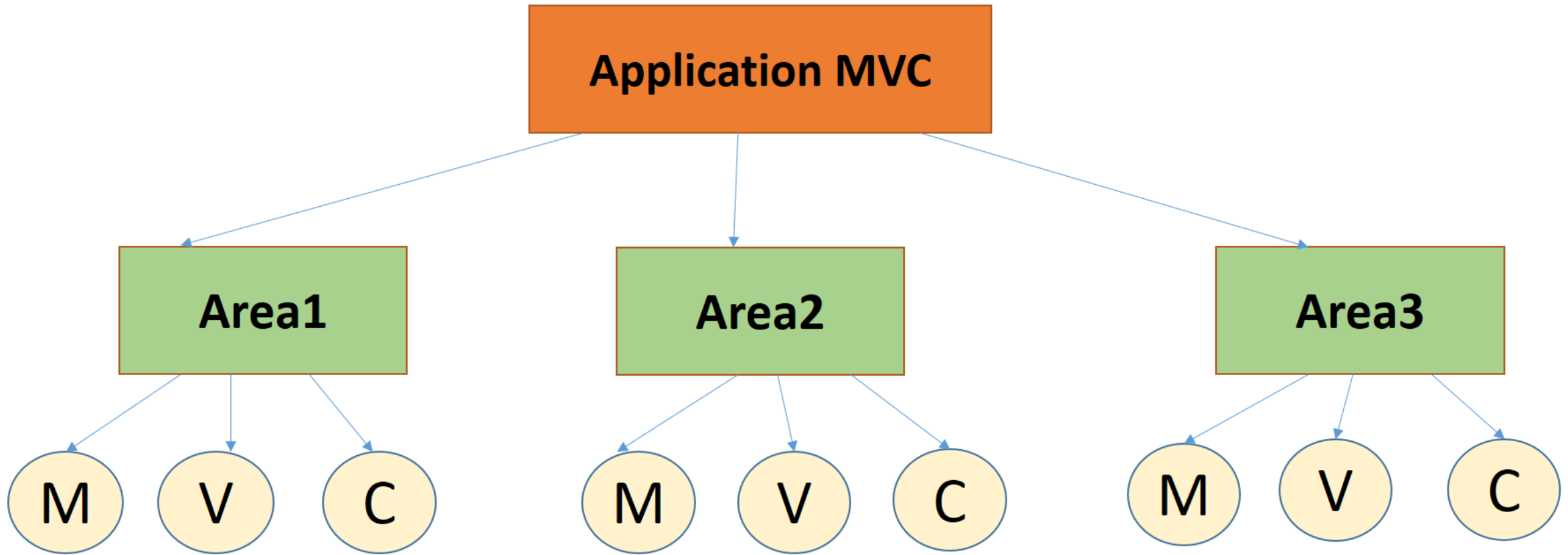
# Area en ASP.Net core

- L'architecture MVC sur laquelle reposent les projets ASP.NET MVC offre une disposition par défaut aux fichiers du projet. Ainsi, tous les contrôleurs de l'application sont créés par défaut dans un dossier Contrôleur, les vues dans un dossier Views et le modèle dans le dossier correspondant

# Area en ASP.Net core

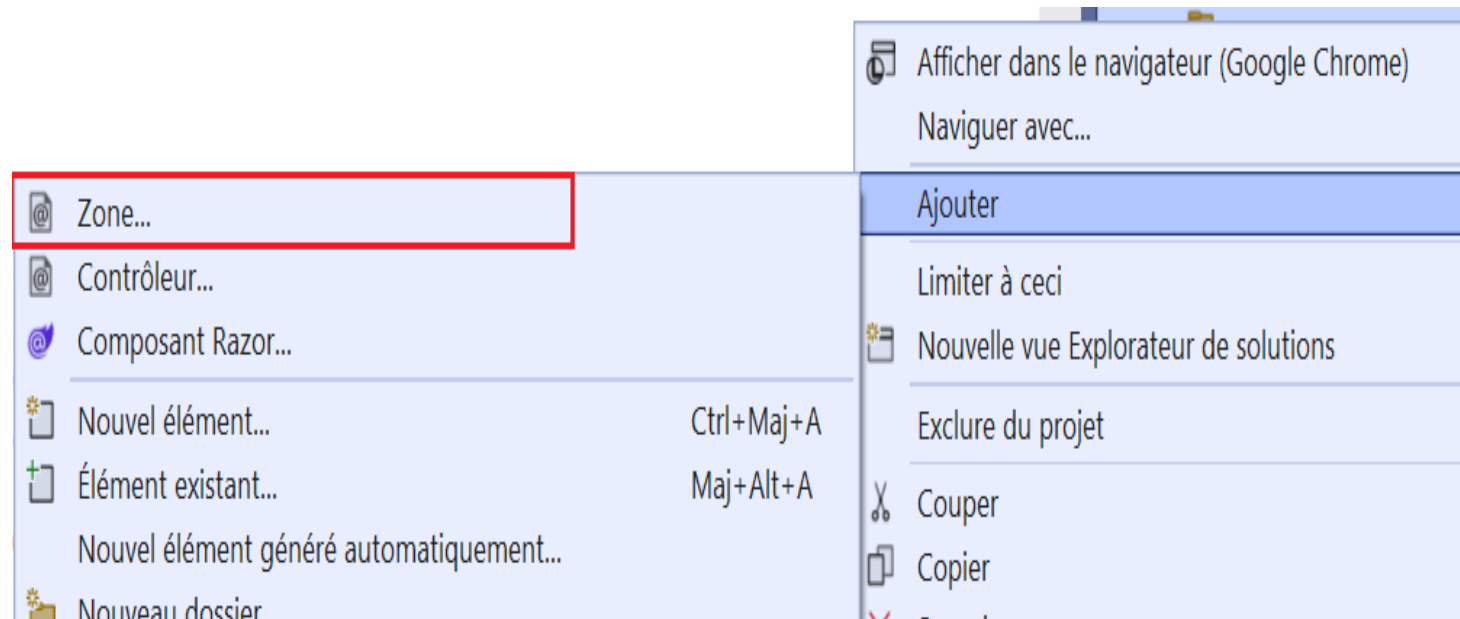
-Une alternative à l'architecture MVC standard d'ASP.NET MVC, qui organise les fichiers du projet par défaut dans des dossiers spécifiques tels que Contrôleurs, Vues et Modèles, serait d'opter pour une approche basée sur la modularité. Cette approche permettrait de diviser le projet en modules distincts, offrant ainsi un emplacement dédié à chaque module pour ses contrôleurs et ses vues. Ce modèle modulaire peut rendre la maintenance et l'identification des composants beaucoup plus efficaces, surtout dans le cas de projets volumineux comprenant un grand nombre de fichiers.

# Area en ASP.Net core



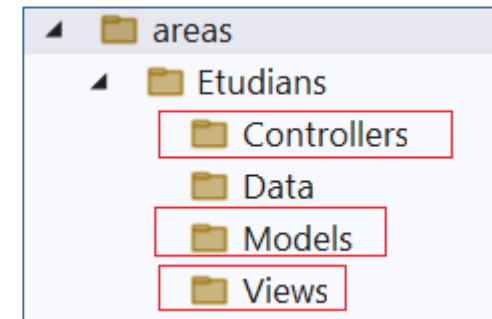
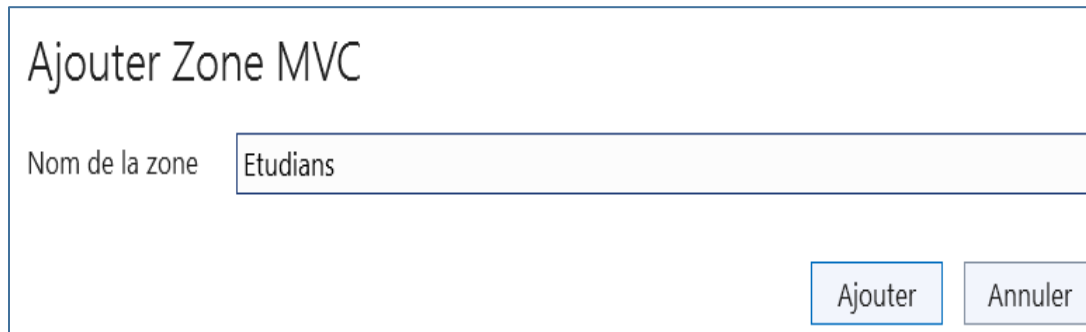
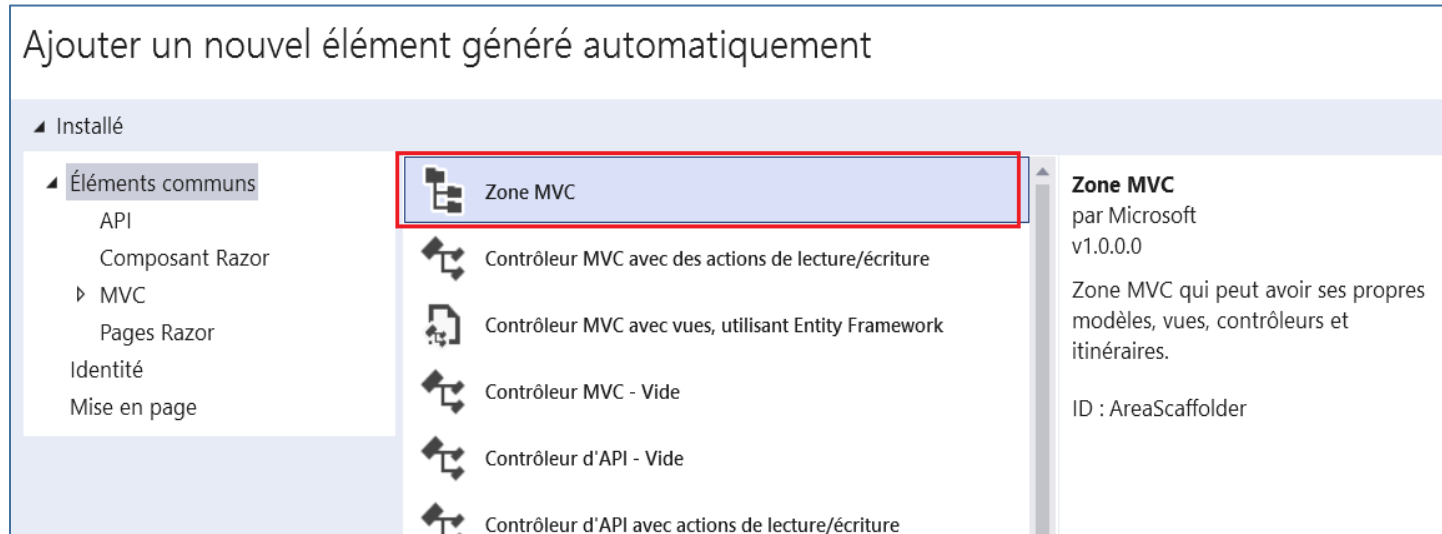
# Area en ASP.Net core

-Pour l'ajouter,il suffit de faire un click droit sur le projet et ajouter un nouveau dossier nommé « areas » si vous effectuée cette opération, vous allez trouver automatiquement un nouveau élément qui s'ajoute appelé area à chaque fois on souhaite ajouter un nouvel élément au dossier areas



# Area en ASP.Net core

-Cliquer sur area→Area MVC(Zone MVC)



# Area en ASP.Net core

Définir areas dans le routage de Program.cs

```
app.MapControllerRoute(  
    name: "Etudians",  
    pattern: "{area:exists}/{controller=Home}/{action=Index}/{id?}");  
  
app.MapControllerRoute(  
    name: "default",  
    pattern: "{controller=Home}/{action=Index}/{id?}" );
```

Ajouter le routage dans le controller

```
namespace Tp1NetCore.areas.Etudians.Controllers  
{  
    [Area("Etudians")]  
    0 références  
    public class HomeController : Controller  
    {  
        public IActionResult Index()  
        {  
            return View();  
        }  
    }  
}
```



# Area en ASP.Net core

**Remarque:** Lors de la définition des areas dans le program.cs, laisser la partie default au dernier « app.MapControllerRoute » sinon on est obligé de définir le routage pour chaque action

```
[Area("Etudians")]  
0 références  
public class HomeController : Controller  
{  
    // [Route("/Home/Index")]  
    0 références  
    public IActionResult Index()  
    {  
        return View();  
    }  
}
```

# Gestion d'Etat

# Synchronisation entre le modèle et la vue

- **Envoi des données via le controller:**

-Pour envoyer des données du modèle vers la vue,il faut les passer dans les paramètres de View.

```
public ActionResult Index2()
{
    Etudiant e = new Etudiant();
    e.idEt = 1;
    e.prenom = "sara";
    e.nom = "hafidi";
    e.age = 20;
    return View(e);
}
```

# Synchronisation entre le modèle et la vue

- **Récupération des données dans la vue:**

- Chaque vue a une propriété *Model* qui sauvegarde l'objet envoyé par le controller

**@Model.nom**

# Synchronisation entre le modèle et la vue

- **Récupération des données dans la vue**

- Si on veut afficher une liste des valeurs,on doit passer par l'instruction **foreach**

- Toujours lors d'une instruction foreach dans la vue,il faut ajouter

**@model IEnumerable<MvcTP2.Models.etudiant>**

Au lieu de **@model MvcTp2.Models.etudiant**

# ViewData

- **Passer des valeurs du contrôleur vers la vue:**

- On utilise le dictionnaire de clé/valeur ViewData

Dans le controller: **ViewData["Nom"] = "yyy";**

- **Récupération de valeur:**

- Dans la vue **@ViewData["Nom"]**

# ViewBag

-Autre façon pour passer les données de controller vers la vue est la propriété ViewBag

**-Création:** `ViewBag.Nom="tttt"`

**-Récupération:** `@ViewBag.Nom;`

-On peut passer des données Via le ViewData et les récupérer via ViewBag et vice versa

# RedirectToAction

- Si le Controller ne veut pas afficher une vue mais il veut se rediriger vers une autre action:

**RedirectToAction("Index3");**

```
public ActionResult Index3()
{
    ViewData["nom"] = "xxxx";
    return View();
}

public ActionResult Index4()
{
    int valeur = 10;
    return RedirectToAction("Index3");
}
```

- <http://localhost:4979/Etudiant/Index4> → affiche xxxx



# Redirect(Url.Action(...))

- Si le controller veut se rediriger vers une action qui se trouve dans un autre controller:

**Redirect(Url.Action("Index", "accueil"));**

# Envoi des données via le View

-Supposons que l'utilisateur cette fois qui veut envoyer des données dans un lien

<http://localhost:4979/Etudiant/Index5?x=eee&y=fff>

Récupération: utiliser ces données comme des paramètres des actions.

```
public ActionResult Index5(string x ,string y)
{
    ViewData["nom"] = x;
    ViewData["prenom"] = y;
    return View();
}
```

# Cookies

## -Création:

```
CookieOptions option = new CookieOptions();
```

```
option.HttpOnly = true;
```

```
Response.Cookies.Append("Cookie1", value, option);
```

```
//ou bien Response.Cookies.Append("Cookie1", value);
```

## -Récupération:

```
String s =
```

```
this.ControllerContext.HttpContext.Request.Cookies["Cookie1"];
```

## -Suppression

```
Response.Cookies.Delete("Cookie1");
```

# Cookies

## **-Vérification**

```
this.ControllerContext.HttpContext.Request.Cookies["Cookie1"] != null
```

# Sessions

1. Dans le fichier `program.cs`: *ajouter le service de session*

```
builder.Services.AddSession();
```

2. Dans le fichier `program.cs`: *activer le service de session dans l'application*

```
app.UseSession();
```

# Sessions

## **-Création:**

```
HttpContext.Session.SetString("key1", value);
```

```
HttpContext.Session.SetInt32("key2",15);
```

## **-Récupération:**

```
var name = HttpContext.Session.GetString ("key1");
```

```
var age = HttpContext.Session.GetInt32 ("key1");
```

## **Vérification**

```
HttpContext.Session.GetString ("key1")!= null
```

## **Supprimer:**

```
HttpContext.Session.Remove ("key1");
```

# Mise en forme

# Page Layout

Dans un site web, on aura souvent une partie qui se répète dans toutes les pages comme l'entête/pied de page, menu





# Page Layout

- On peut réaliser cela via les Pages Layout.
- Elles permettent de définir une mise en page et les éléments à inclure dans toutes les pages.
- Il est logique de commencer le projet par la construction d'une Page Layout.

# Page Layout

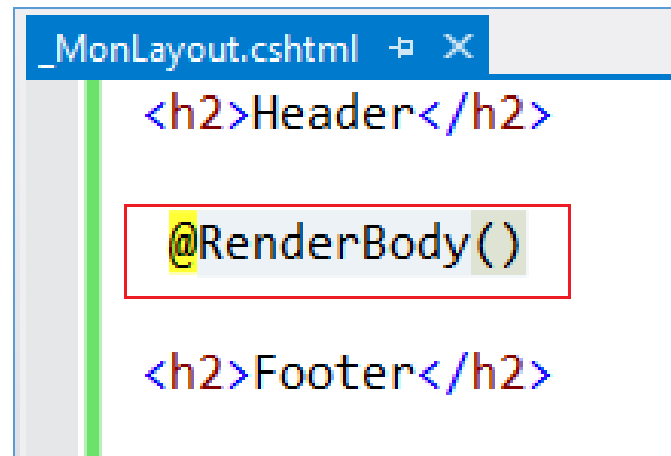
- C'est l'équivalent de Master page en ASP WebForms
- Voir \_Layout.cshtml

```
        <ul id="menu">
            <li>@Html.ActionLink("Accueil", "Index", "Home")</li>
            <li>@Html.ActionLink("À propos de", "About", "Home")</li>
            <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
        </ul>
    </nav>
</div>
</div>
</header>
<div id="body">
    @RenderSection("featured", required: false)
    <section class="content-wrapper main-content clear-fix">
        @RenderBody()
    </section>
</div>
<footer>
    <div class="content-wrapper">
```

- RenderBody(): permet d'affecter le contenu de page qui exploite layout

# Page Layout

- Chaque vue a une variable nommée Layout qui détermine la Template à appliquer sur une page.
- Cette propriété est affectée dans la première ligne de la vue
- Création de Layout:** on le crée dans le répertoire **Shared** de projet→ajouter une vue



```
_MonLayout.cshtml
<h2>Header</h2>
@RenderBody()
<h2>Footer</h2>
```

# Page Layout

**-Appel de Layout:** on le crée dans le répertoire Shared de projet→ajouter une vue



```
Index2.cshtml  X  _MonLayout.cshtml
@{
    Layout = "~/Views/Shared/_MonLayout.cshtml";
}

<h2>Index2</h2>

<p>@Model.prenom</p>
```

*Remarque:* par recommandation les noms des pages partagés doivent commencer par « \_ »

# Page Layout

**Remarque:-** Si on n'a pas une ligne au début d'une vue qui appelle le layout à utiliser, le programme cherche le layout défini dans la page `_ViewStart.chnml` du projet.

```
@{  
    Layout = "_Layout";  
}
```

# Partial View

- C'est l'équivalent de User contrôle en ASP WebForms.
- Il s'agit d'une partie de view qu'on peut l'intégrer dans d'autres view.
- **Création:** ajouter une nouvelle vue

Ajouter une vue

Nom de la vue :  
Partial1

Moteur de vue :  
Razor (C#HTML)

☐ Créer une vue fortement typée  
Classe de modèle :  
Modèle de vue de structure :  
Empty

☒ Créer en tant que vue partielle

☒ Utiliser une disposition ou la page maître :  
(Ne pas renseigner si l'option est définie dans un fichier Razor \_viewstart)

ID de ContentPlaceHolder :  
MainContent

Ajouter Annuler

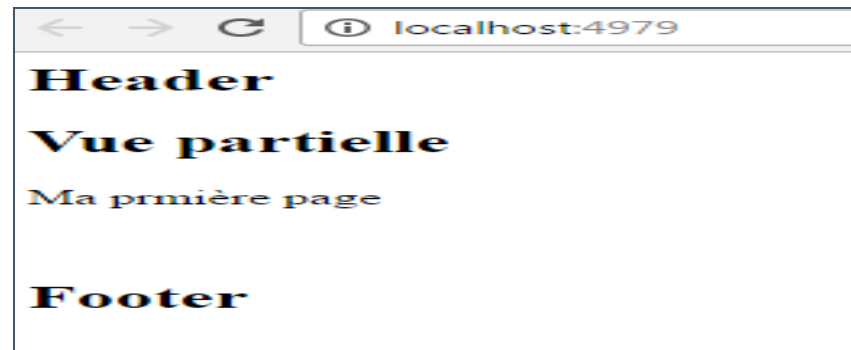
# Partial View

- Appel :

- Intégrer cette ligne dans l'emplacement où on veut appeler ce partial view

```
<div>  
    <partial name="Partial1" />  
</div>
```

- Exécution:



# Section

❑ **Problématique:** Supposons qu'on a un ensemble des pages qui utilisent le même Layout page mais on veut laisser une partie dans Layout se change selon la vue à afficher

-Autrement dit rendre une section dans layout modifiable selon la vue à utiliser .

❑ **Solution:** Utiliser la notion de section



# Section

- **Création:** On définit la section dans Layout page via l'écriture suivante càd dans cet endroit la section va être appeler

```
@await RenderSectionAsync("Section1", required: false)
```

- **Appel:** Ensuite on définit le contenu de section dans la vue qui utilise le précédent layout page

```
@section Section1
{
    <p>*****Contenu spécifique de cette vue*****</p>
}
```

# 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 .
- L'objectif principal d'Entity Framework est de permettre aux développeurs de travailler avec des données de manière orientée objet plutôt qu'en utilisant des requêtes SQL directes. Il facilite la gestion et la manipulation des données en fournissant un modèle de données basé sur des objets qui correspond aux structures de la base de données.

# 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.

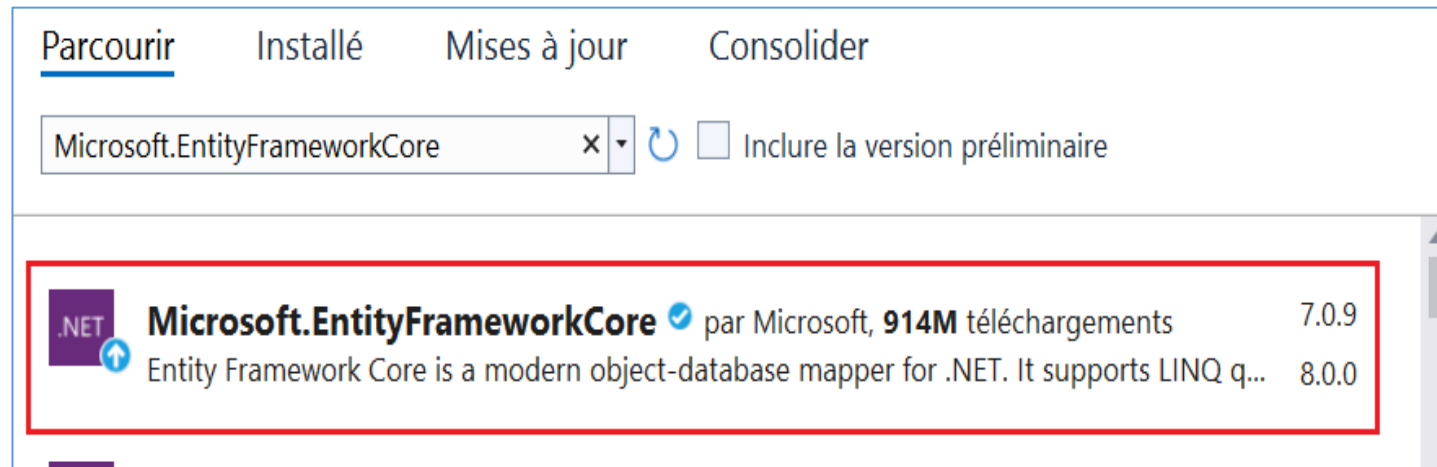
# Les approches Entity Framework

**-Database First : Création à partir de la base de données :** Avec l'approche "Database First", le modèle de données est généré à partir d'une base de données existante. Entity Framework examine la structure de la base de données et crée automatiquement les classes d'entités et le contexte de base de données correspondants.

**-Code First : Création à partir du code :** Avec l'approche "Code First", le modèle de données est défini en utilisant des classes et des propriétés dans le code source. Les classes d'entités sont écrites en premier, et la base de données est générée ensuite à partir de ces classes.

# Entity Framework

Pour utiliser entity framework dans les projets ASP.Net, on doit l'installer dans le projet concerné via Nuget à partir du gestionnaire de package pour la solution:



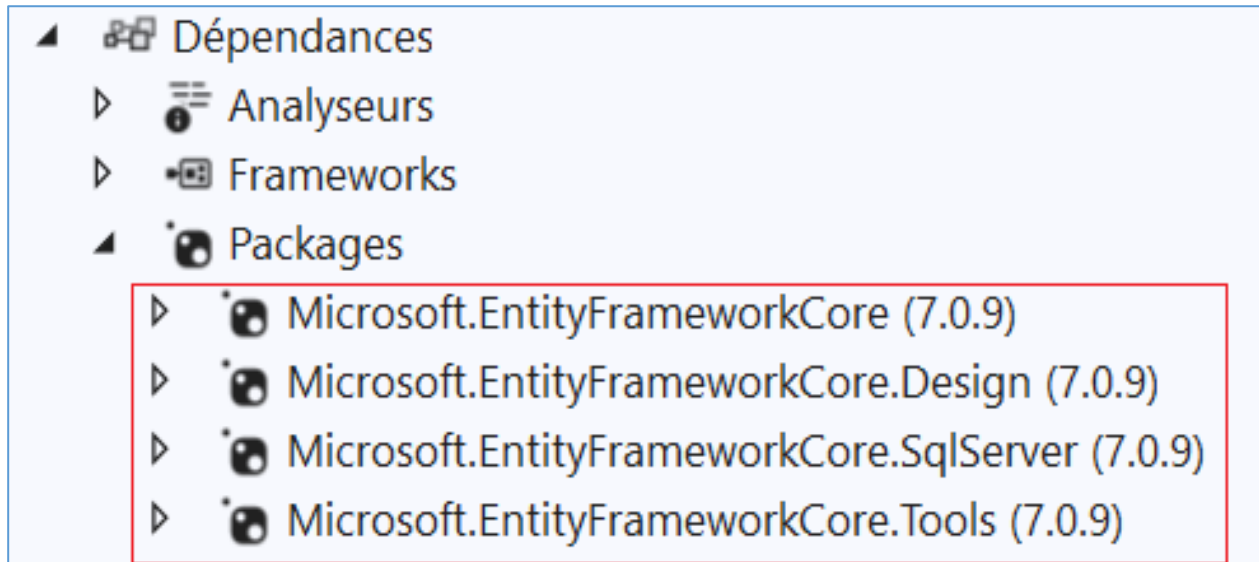
# Entity Framework

Installer ces package:

- ✓ Microsoft.EntityFrameworkCore →7.0.9
- ✓ Microsoft.EntityFrameworkCore.sqlServer →7.0.9
- ✓ Microsoft.EntityFrameworkCore.Design →7.0.9
- ✓ Microsoft.EntityFrameworkCore.Tools →7.0.9
- ✓ Microsoft.AspNetCore.Identity.EntityFrameworkCore →6.0.20

# Entity Framework

Une fois l'installation se termine, on trouve ces éléments ajoutés dans le package de projet:





# Opérations Entity Framework

**ClientDataContext cl = new ClientDataContext();**

Sélectionner plusieurs lignes	<b>var</b> x =cl.clients.ToList();
Sélectionner une ligne en fonction de sa clé primaire	<b>Client</b> x=cl.clients.Find(id=2);
Sélectionner selon une condition	<b>var</b> x = cl.clients.Where(p => p.id > 5 && p.id < 10)

# Opérations Entity Framework

**ClientDataContext cl = new ClientDataContext();**

Récupérer la première ligne

```
Client x=cl.clients. .FirstOrDefault(m => m.Nom ==  
"client1");
```

```
Client y=cl.clients.FirstOrDefault();
```

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

# 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

# Entity Framework -Code First-

## 1.Création du modèle

-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-

## 1.Création du modèle

### -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
- Ajouter l'annotation Key pour la variable qui sera la clé de la table

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

# Entity Framework -Code First-

## 1.Création du modèle

-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 :

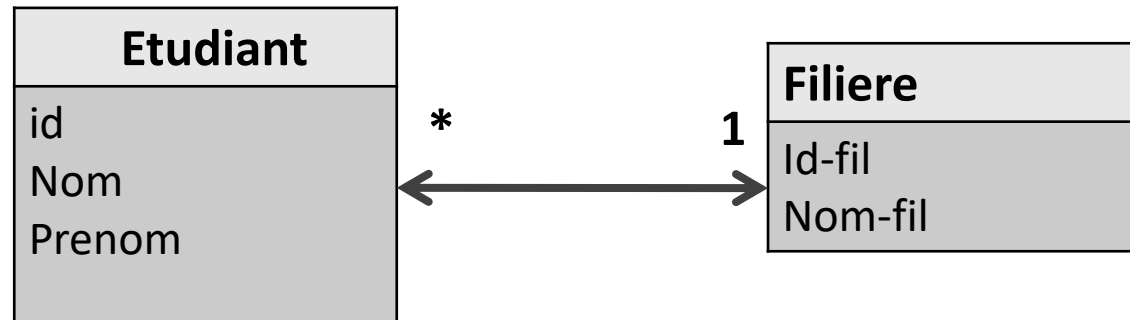
```
using System.ComponentModel.DataAnnotations;
```



# Entity Framework -Code First-

## 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

# Entity Framework -Code First-

## 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; }
}
```

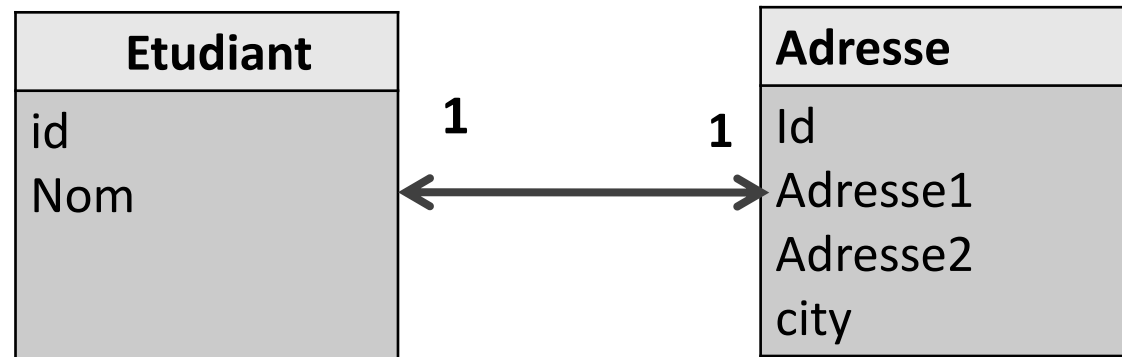
```
public partial class Filiere
{
    public Filiere()
    {
        this.etudiants = new HashSet<etudiant>();
    }
    [Key]
    public int Id_filiere { get; set; }
    public string Nom_filiere { get; set; }

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

# Entity Framework -Code First-

## 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.

# Entity Framework -Code First-

## 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; }

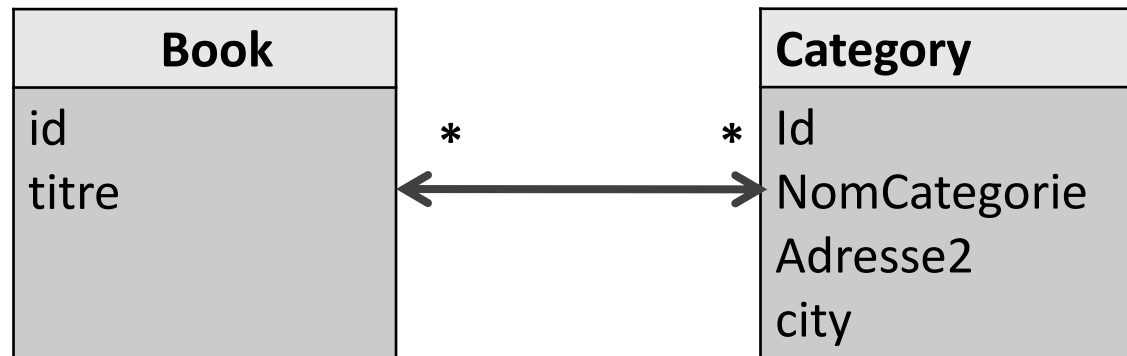
    public string Address1 { get; set; }
    public string Address2 { get; set; }
    public string City { get; set; }
    public string Country { get; set; }

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

# Entity Framework -Code First-

## 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.

# Entity Framework -Code First-

## 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; }
}
```

# Entity Framework -Code First-

## 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; }
}
```

# Entity Framework -Code First-

- **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



# Entity Framework -Code First-

## 3. Spécification la chaîne de connexion:

```
<connectionStrings>  
  <add name="UserContext" connectionString="Data Source=user-PC\SQLEXPRESS; Initial Catalog=UserContext;  
    MultipleActiveResultSets=True;Integrated Security=True;" providerName="System.Data.SqlClient" />  
</connectionStrings>
```

# Entity Framework -Code First-

## ○ Résumé:

- Etape1:Création de modèle
- Etape2:Création du contexte de base de données et spécification la chaîne de connexion:
- Etape3:Exécuter des commandes pour générer la base de données

# Entity Framework -Code First- *Exemple*

## ❑ Etape1:Création de modèle

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

```
public class User
{
    public int Id { get; set; }

    [StringLength(50)]
    public string Nom { get; set; }

    [Required]
    [StringLength(50)]
    public string Prenom { get; set; }

    [Required]
    [DataType(DataType.EmailAddress)]
    [StringLength(50)]
    public string Email { get; set; }
}
```

# Entity Framework -Code First- *Exemple*

## ❑ Etape2: Création de datacontext

1. En Model → Nouveau dossier Context → On crée la classe de contexte, qui sera un composant middleware pour la communication avec la base de données. Elle possède des propriétés DbSet qui contiennent les données de table de la base de données.

```
public class ApplicationDbContext : DbContext
{
    public ApplicationDbContext(DbContextOptions options)
        : base(options)
    {
    }

    public DbSet<User> Users { get; set; }
}
```

# Entity Framework -Code First- *Exemple*

## ❑ Etape2: Création de datacontext

2. Ajouter les paramètres de connexion DB dans le fichier appsettings.json

```
"ConnectionStrings": {  
  "ConnectionDb": "Server=DESKTOP-6PAFM8E\\SQLEXPRESS;Database=tp1Netcore;  
    TrustServerCertificate=True; Trusted_Connection=True;  
    MultipleActiveResultSets=true;"  
}
```

# Entity Framework -Code First- *Exemple*

## ❑ Etape2: Création de datacontext

### 3. Définir le contexte dans le fichier program.cs

```
public static void Main(string[] args)
{
    var builder = WebApplication.CreateBuilder(args);

    // Add services to the container.
    builder.Services.AddControllersWithViews();

    builder.Services.AddDbContext<tp1Context>(option =>
        option.UseSqlServer(builder.Configuration.GetConnectionString("ConnectionDb")));

    var app = builder.Build();|
```

# Entity Framework -Code First- *Exemple*

## ❑ Etape3: génération de la base de donnée

### 1.Déterminer la BD et définir la migration

- Add-Migration NomDeLaMigration

ou

- Add-Migration NomDeLaMigration -Context NomDuContexte

→La commande Add-Migration va créer dans notre projet un dossier “**Migrations**”, avec du code représentant le schéma de notre base de données.

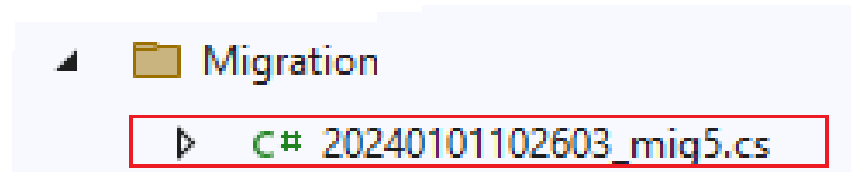
# Entity Framework -Code First- *Exemple*

## ❑ Etape3: génération de la base de donnée

### 1.Déterminer la BD et définir la migration

-Une fois on exécute cette commande 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.





# Entity Framework -Code First- *Exemple*

## □ Etape3: génération de la base de donnée

### 1.Déterminer la BD et définir la migration

#### ○ mig5.cs

```
public partial class mig5 : Migration
{
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.CreateTable(
            name: "User",
            columns: table => new
            {
                id = table.Column<int>(type: "int", nullable: false)
                    .Annotation("SqlServer:Identity", "1, 1"),
                nom = table.Column<string>(type: "nvarchar(max)", nullable: false),
                prenom = table.Column<int>(type: "int", nullable: false)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_User", x => x.id);
            });
    }

    protected override void Down(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.DropTable(
            name: "User");
    }
}
```

# Entity Framework -Code First- *Exemple*

❑ **Etape3: génération de la base de donnée**

## 2.Exécuter la migration

- **Update-Database –Context** → La commande Update-Database va utiliser le code précédent pour mettre à jour la base de données.

# Entity Framework 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.

# Entity Framework 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.

# Entity Framework 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

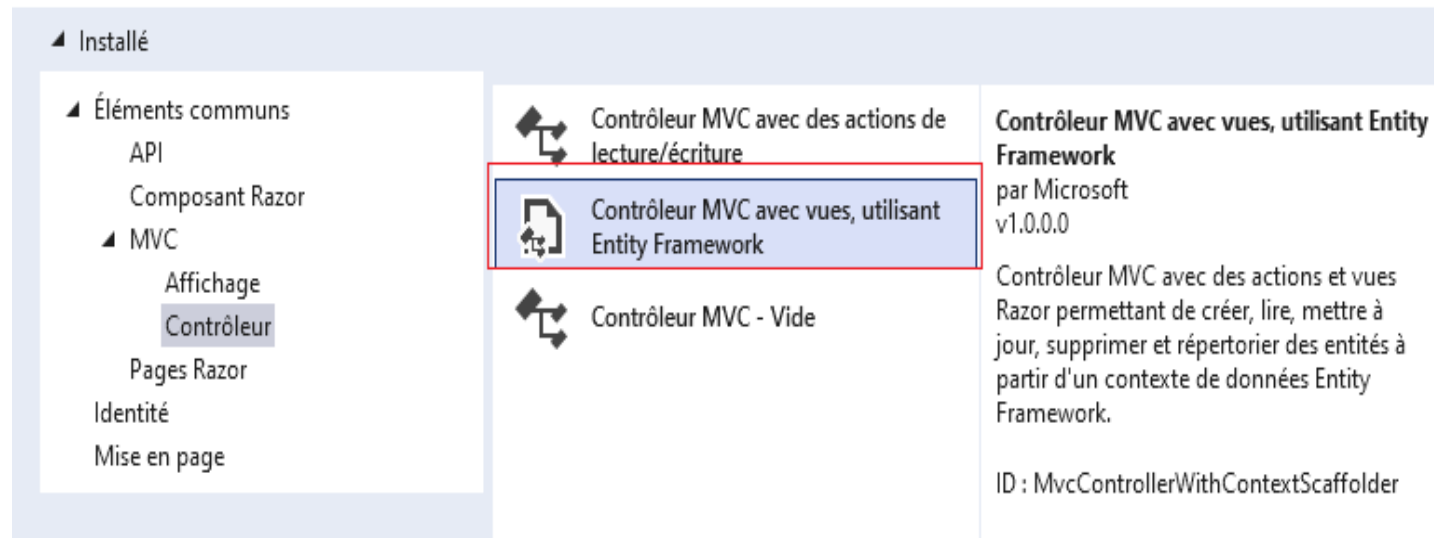
- T4(Text Template Transformation Toolkit) est un générateur de code intégré à Visual Studio

# Entity Framework Code First-Scaffolding-

## ❑ Au façon pour effectuer Effectuer Etape2 et Etape3-

-**Etape2:** Controller → ajouter « Controleur MVC avec vues,utilisant Entity Framework »

Ajouter un nouvel élément généré automatiquement



# Entity Framework Code First-Scaffolding-

❑ Au façon pour effectuer Effectuer Etape2 et Etape3-

-**Etape2:**Controller → ajouter « Controleur MVC avec vues,utilisant Entity Framework »

Ajouter Contrôleur MVC avec vues, utilisant Entity Framework

Classe de modèle: User (Tp1NetCore.Models)

Classe DbContext class: <Obligatoire> +

Ajouter un contexte de données

Nouveau type de contexte de données: Tp1NetCore.Data.Tp1NetCoreContext

Ajouter Annuler

Ajouter Annuler

-Le dataContext va être générer dans un Dossier appelé « Data »

-La chaine de connexion s'ajoute dans le fichier « appsettings.json »

# Entity Framework Code First-Scaffolding-

## □Au façon pour effectuer Etape2 et Etape3-

**-Etape3:** génération de la base de donnée

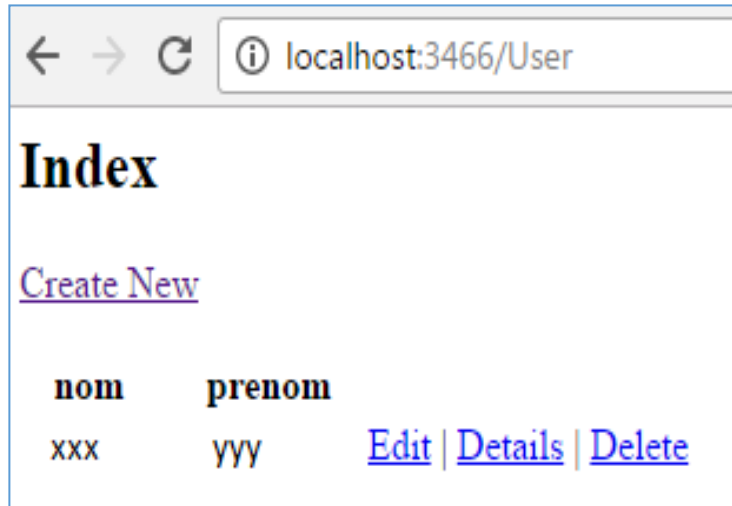
**Add-Migration tp1**→La commande Add-Migration va créer dans notre projet un dossier “Migrations”, avec du code représentant le schéma de notre base de données.

**Update-Database**→La commande Update-Database va utiliser le code précédent pour mettre à jour la base de données.



# EF Code First -Code First - *Exemple*

## 3. Exécuter le projet

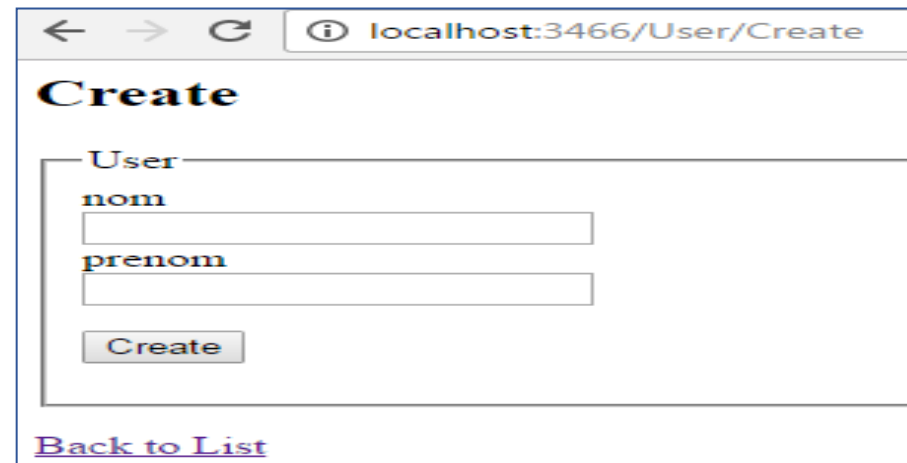


← → ↻ ⓘ localhost:3466/User

## Index

[Create New](#)

nom	prenom	
xxx	yyy	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>



← → ↻ ⓘ localhost:3466/User/Create

## Create

User

nom

prenom

[Back to List](#)

# EF Code First -Code First - *Exemple*

**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 IActionResult Create()
{
    return View();
}
```

[HttpPost]

```
public async Task<IActionResult> Create(User user )
{
    if (ModelState.IsValid)
    {
        _context.Add(user );
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(user );
}
```

# EF Code First -Code First - *Exemple*

- **Récupérer les données dans la vue**

-1<sup>ère</sup> ligne détermine le modèle qui va être utilisé par la vue

```
@model Tp1NetCore.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

# EF Code First -Code First - *Exemple*

## ❑ Cas particulier:

○**Cas1: Mise à jour de BD:** Si on veut exécuter des nouvelles mises à jour dans la BD( ajouter un champ...):

1.On effectue la modification dans le Model

2.Ajouter une référence pour les nouvelles classes dans le contexte

0 références

```
public virtual DbSet<Etudiant> Etudiants{ get; set; }
```

0 références

```
public virtual DbSet<Filiere> Filieres { get; set; }
```

# EF Code First -Code First Migration-

## ❑ Cas particulier:

○ **Cas1: Mise à jour de BD:** Si on veut exécuter des nouvelles mises à jour dans la BD( ajouter un champ...):

3. Supprimer l'ancienne migration

**Remove-Migration -Context TP1NetCore.Models.Tp1NetCoreDataContext**

3. Reéxecuter la commande Add-migration → Générer un nouveau fichier de migration

4. Reéxecuter la commande Update-Database sur le nouveau fichier généré

# EF Code First -Code First Migration-

## ❑ Cas particulier:

- **Cas1: Mise à jour de BD:** Si on veut exécuter des nouvelles mises à jour dans la BD( ajouter un champ...):

**Add-Migration newmigration -Context **NomDuContexte****

**Update-Database -Context **NomDuContexte****

# EF Code First -Code First Migration-

## ❑ Cas particulier:

○**Cas2: 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 de la base de données. 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.

# EF Code First -Code First Migration-

## ❑ Cas particulier:

### ○Cas2: Remplir la BD Lors de création

```
public class VotreDbContext : DbContext
{
    // ... autres membres DbContext

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        // ... configurations du modèle

        // Appel à la méthode Seed
        Seed(modelBuilder);
    }

    private void Seed(ModelBuilder modelBuilder)
    {
        // Ajout de données initiales pour votre table
        modelBuilder.Entity<VotreEntite>().HasData(
            new VotreEntite { Id = 1, Propriete1 = "Valeur1", Propriete2 = "Valeur2" },
            // ... autres données
        );
    }
}
```

**Exemple:**

```
new User { id=1,nom="user1",prenom="xxx"},
```



# EF Code First -Code First Migration-

## Remarque:

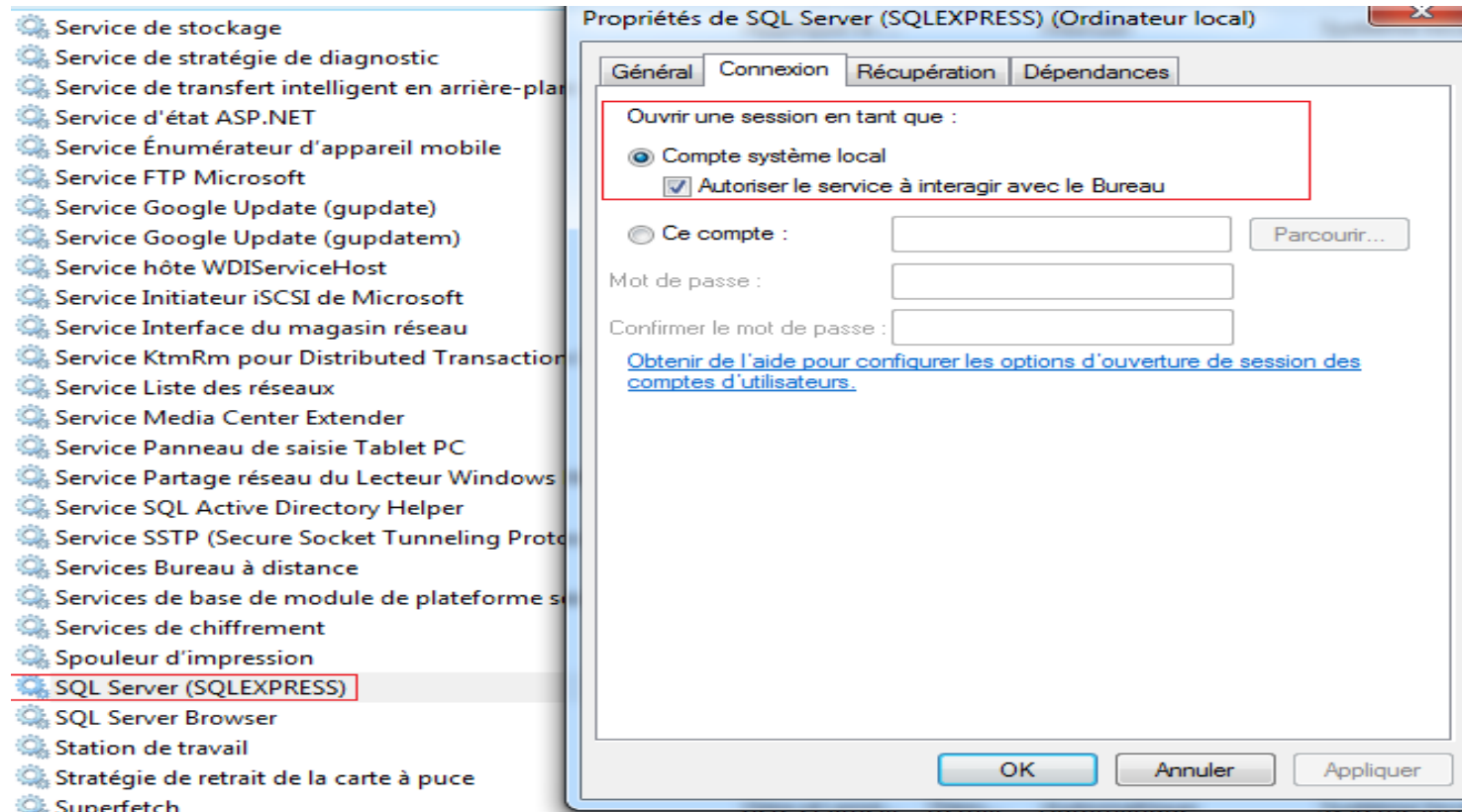
- Au cas des erreurs lors de création de la BD:
  - Vérifier qu'il n'ya pas des problèmes au niveau de chaine de connection dans le Web.config

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:**



# 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.

# Entity Framework -DataBase First-

□ **Etape1:** Création le modèle EF basé sur notre base de données

Pour cela exécuter la commande suivante dans la console Nuget

```
Scaffold-DbContext "Server=DESKTOP-
```

```
ITM0GTO\SQLEXPRESS;Database=Tp1NetCore;Trusted_Connection=true;TrustS  
erverCertificate=True;MultipleActiveResultSets=true;"
```

```
Microsoft.EntityFrameworkCore.SqlServer -o Models -Force
```

# Entity Framework -DataBase First-

❑ **Etape2:** Enregistrez votre contexte avec l'injection de dépendance

1. Supprimer la configuration de contexte en ligne:

Dans ASP.NET Core, la configuration de chaîne de connexion est généralement effectuée dans appsetting.json. Pour se conformer à ce modèle, nous allons déplacer la configuration du fournisseur de base de données vers appsetting.json

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
=> optionsBuilder.UseSqlServer("Server=DESKTOP-ITN0GTO\\SQLEXPRESS;Database=Tp1NetCore.Data;
```

# Entity Framework -DataBase First-

□ **Etape2:** Enregistrez votre contexte avec l'injection de dépendance

## 1. Supprimer la configuration de contexte en ligne:

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
// => optionsBuilder.UseSqlServer("Server=DESKTOP-ITM0GT0\\SQLEXPRESS;Database=Tp1NetCore.Data;");
{ }
```

```
"ConnectionStrings": {
|
|
| "Tp1NetCoreContext": "Server=DESKTOP-ITM0GT0\\SQLEXPRESS;Database=Tp1NetCore.Data;";
|
}
```

# Entity Framework -DataBase First-

❑ **Etape2:**Enregistrez votre contexte avec l'injection de dépendance

## 2.Enregistrez et configurez le contexte dans Program.cs

Pour que nos contrôleurs MVC puissent utiliser Tp1NetCoreContext, nous allons l'enregistrer en tant que service.

```
builder.Services.AddDbContext<Tp1NetCoreDataContext>  
    (option => option.UseSqlServer(builder.Configuration.GetConnectionString("Tp1NetCoreContext")));
```