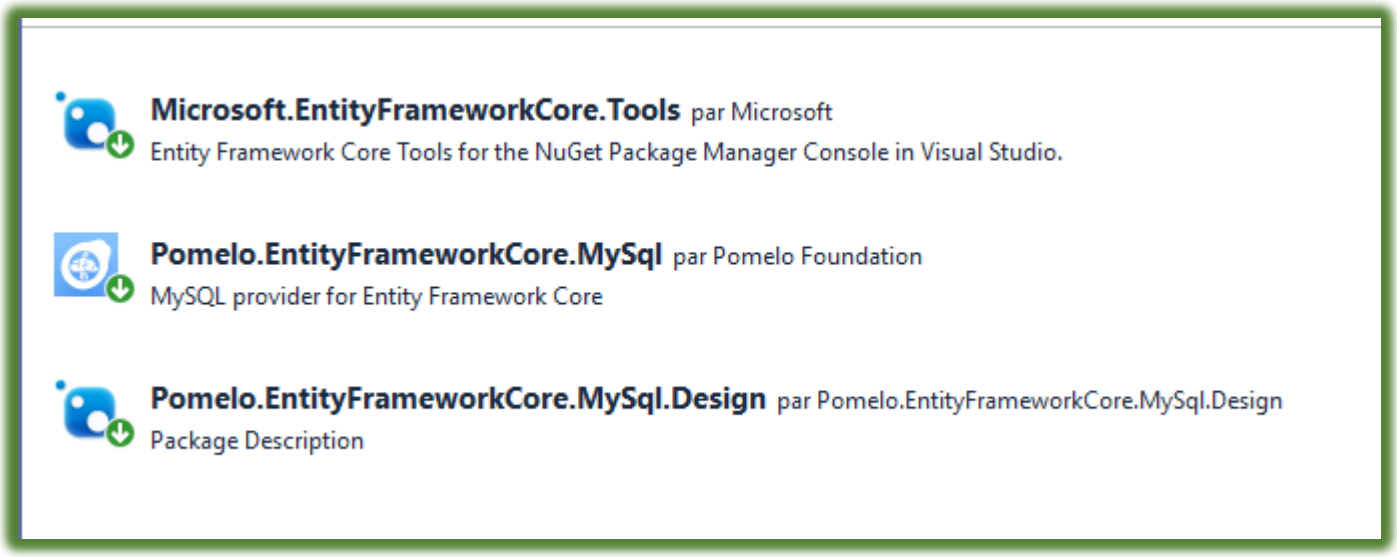
TP4 – ACCES AUX DONNEES

Manga WEB ASP CORE

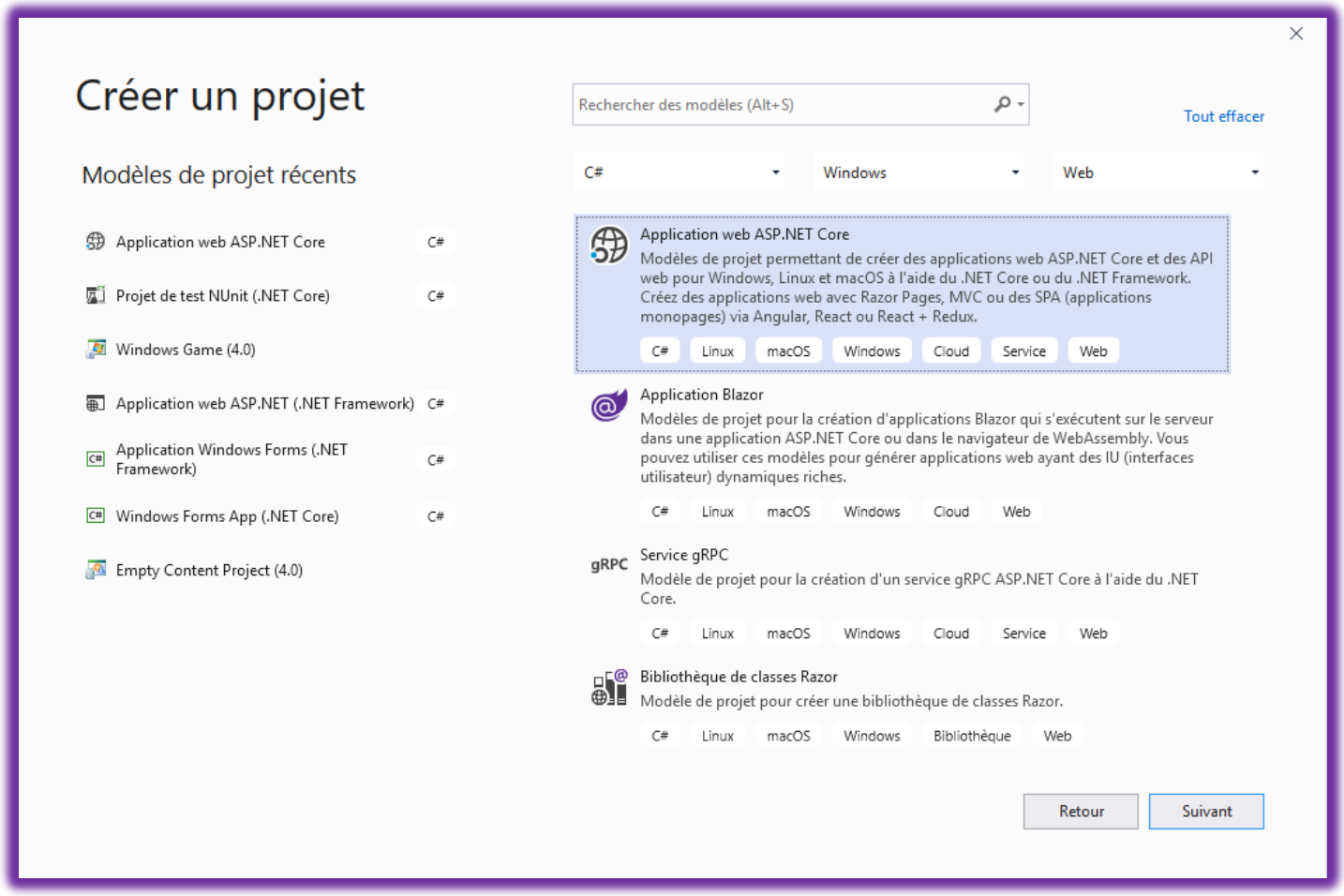
La bibliothèque de classes Entity Framework propose un mécanisme de mapping entre un modèle relationnel et un modèle objet. Le framework utilise le langage Linq pour interroger les données présentes dans les classes.

Middleware : .net et Mysql

Pour se connecter à Mysql avec Entity, nous devons télécharger les paquets Nuget suivants :

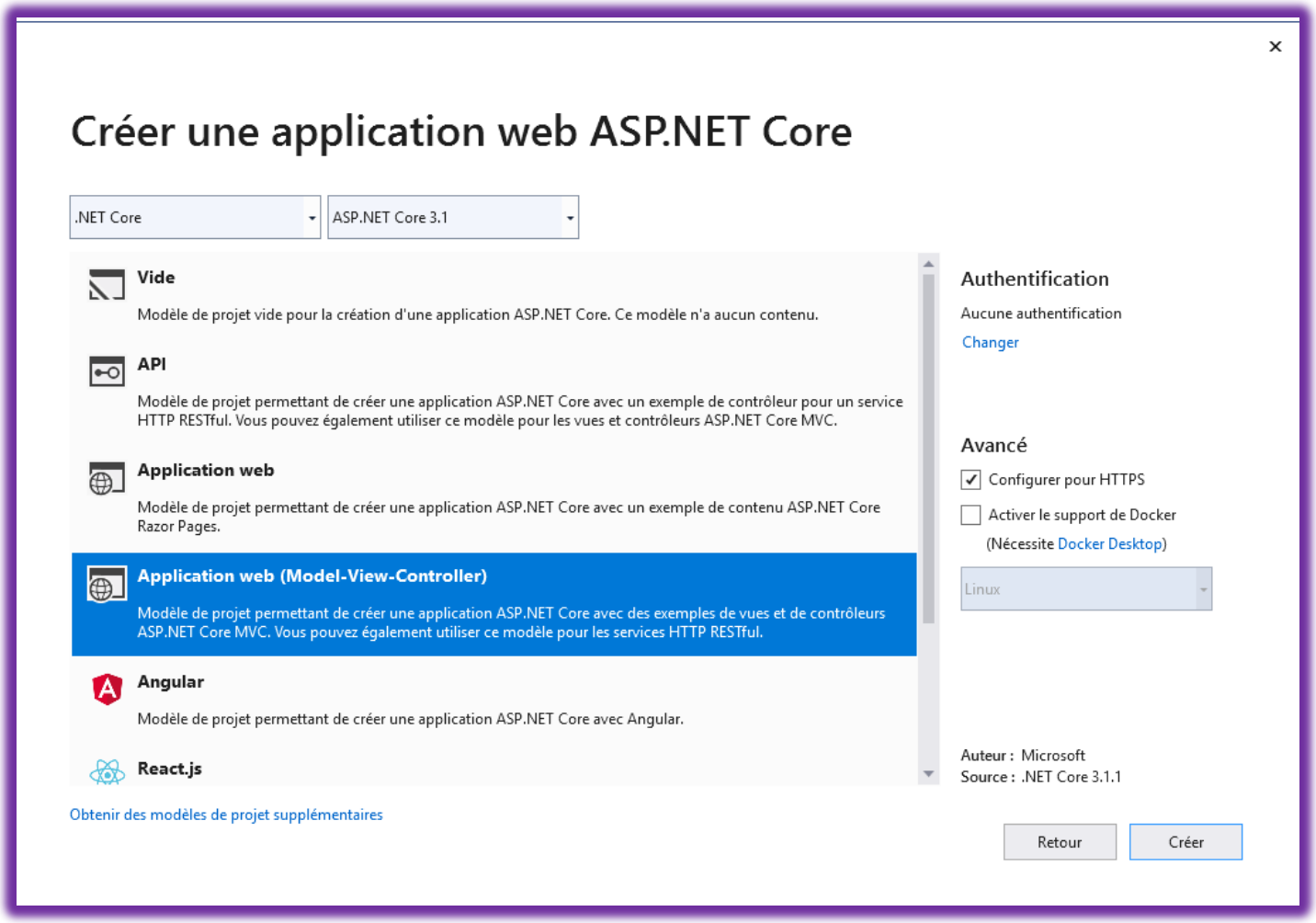


Etude d'un exemple : Les Mangas

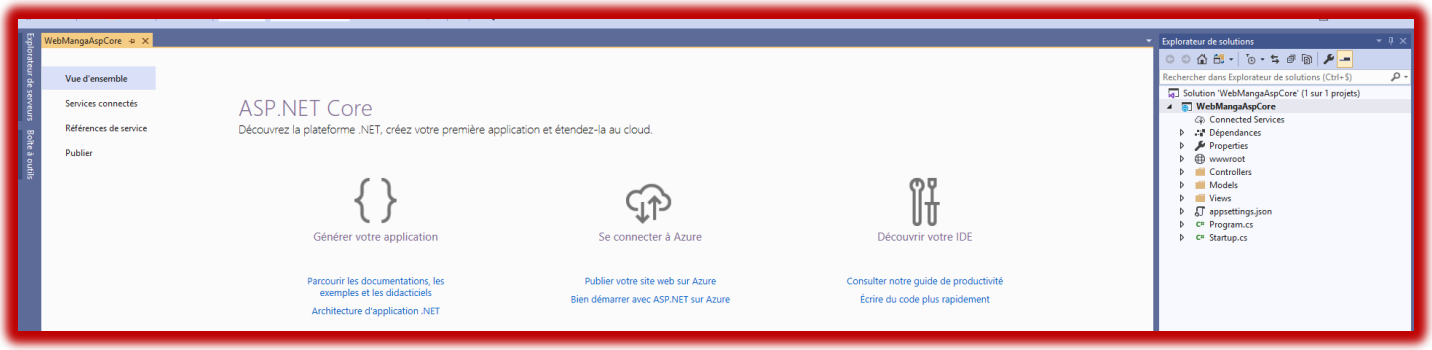


Création d'une application Web

Créons un nouveau projet, de type Application Web MVC



Le projet peut être créé.

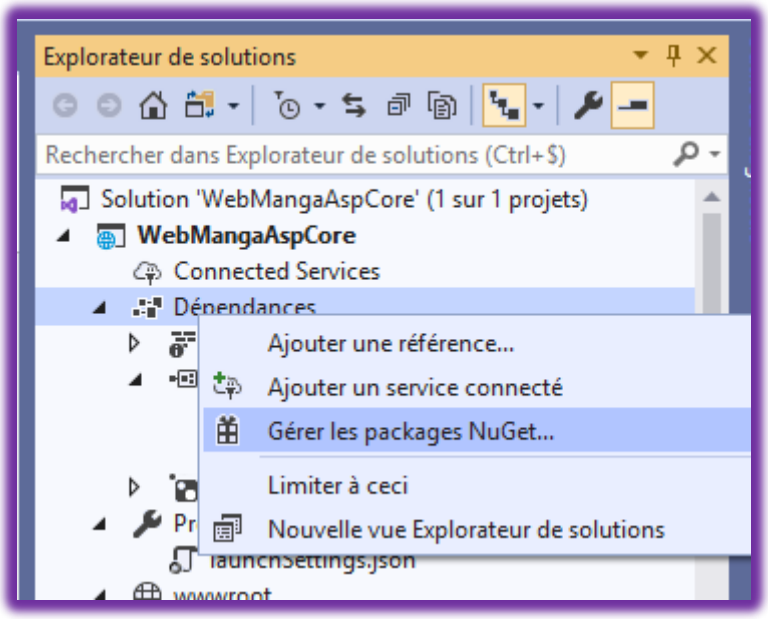


Génération des classes du domaine : ORM Entity

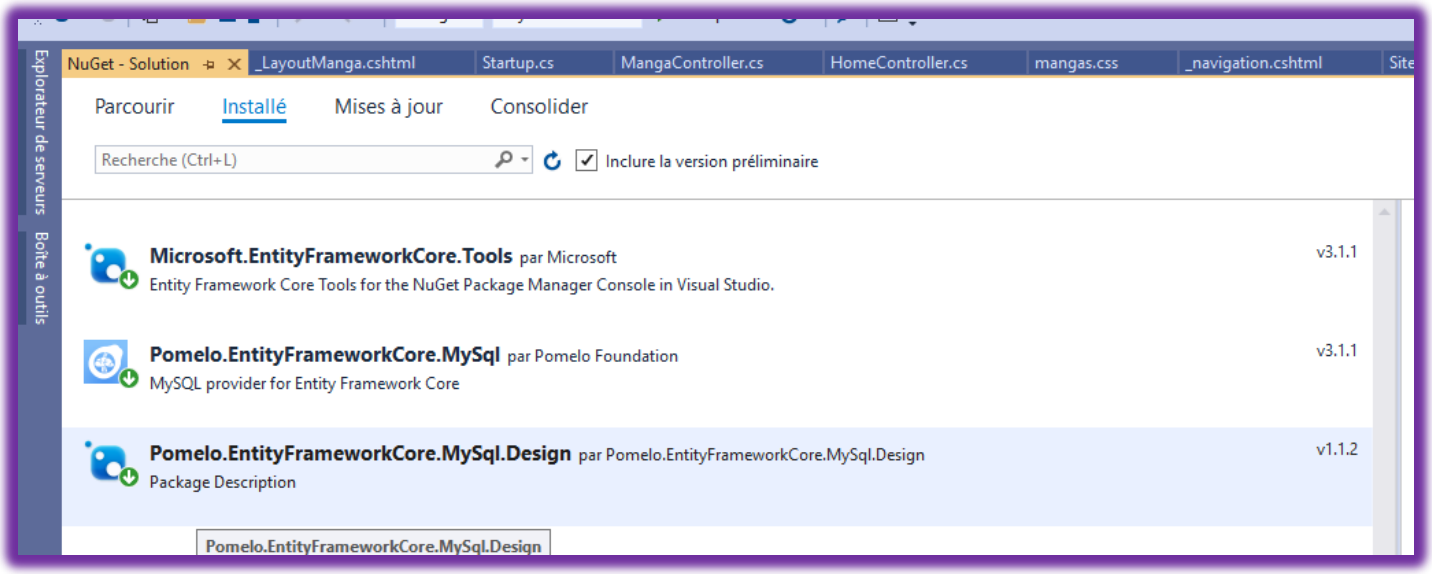
L'application va utiliser l'ORM Entity et pour cela, il faut commencer par installer les paquets Nuguet.

Gestion des paquets Nuget

On utilise le gestionnaire de paquets :



et on installe :



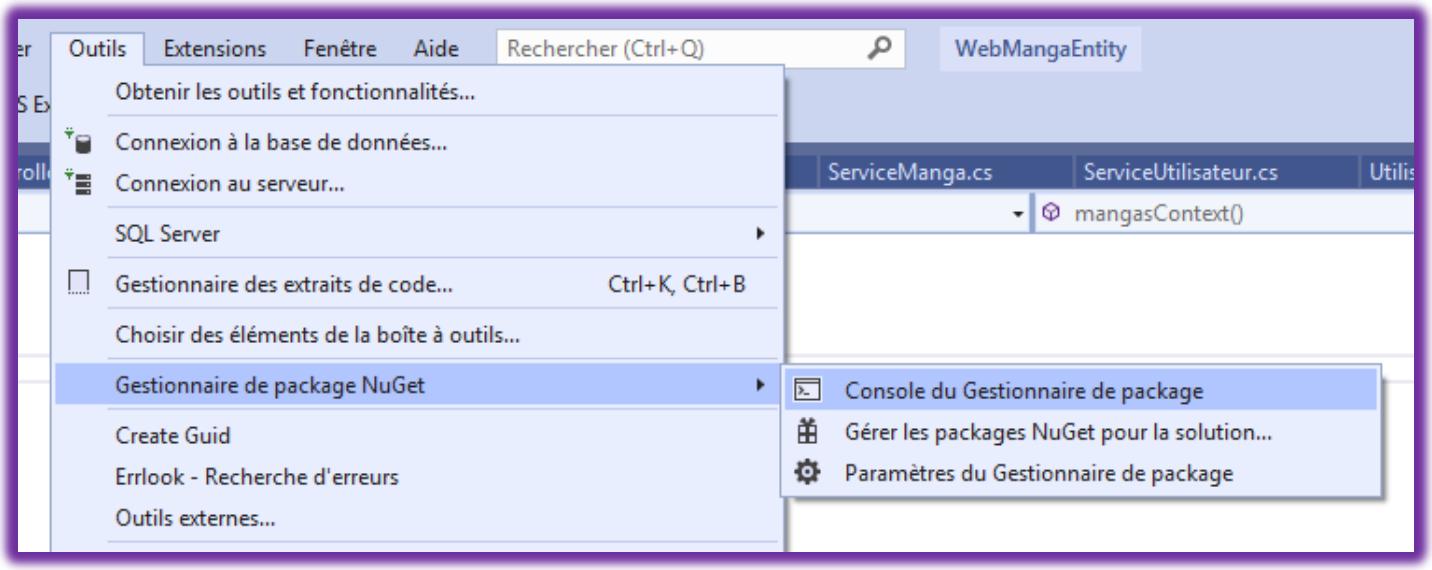
Les paquets Pomelo sont une alternative aux paquets de Microsoft qui ne fonctionnent pas.

Génération des tables

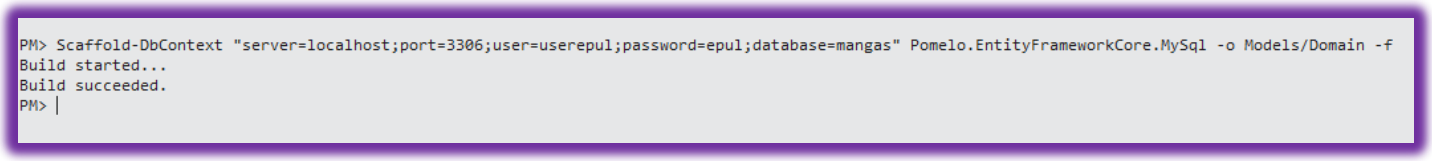
La génération des tables va se faire à l'alide la console Nuget. On va utiliser la commande :

```
Scaffold-DbContext "server=localhost;port=3306;user=userepul;password=epul;database=mangas"
Pomelo.EntityFrameworkCore.MySql -o Models/Domain -f
```

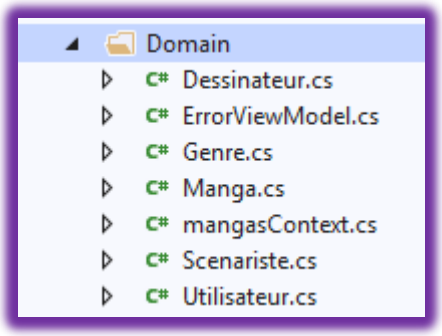
Vous lancez la console NuGet :



Sous la console Nuget vous entrez la commande suivante



Le système vous génére dans le dossier Models/Domain :

- Les tables issues du mappingt avec la base de données
  o Manga
  o Dessinateur
  o …
- Une classe mangasContext qui sera le point d'entrée de l'utilisation de l'ORM
	

mangasContext

Cette classe établit la liaison avec la base de données, elle contient :

- L'accès aux tables

```
        }

        1 référence
        public virtual DbSet<Dessinateur> Dessinateur { get; set; }
        1 référence
        public virtual DbSet<Genre> Genre { get; set; }
        3 références
        public virtual DbSet<Manga> Manga { get; set; }
        1 référence
        public virtual DbSet<Scenariste> Scenariste { get; set; }
        1 référence
        public virtual DbSet<Utilisateur> Utilisateur { get; set; }

        0 références
```

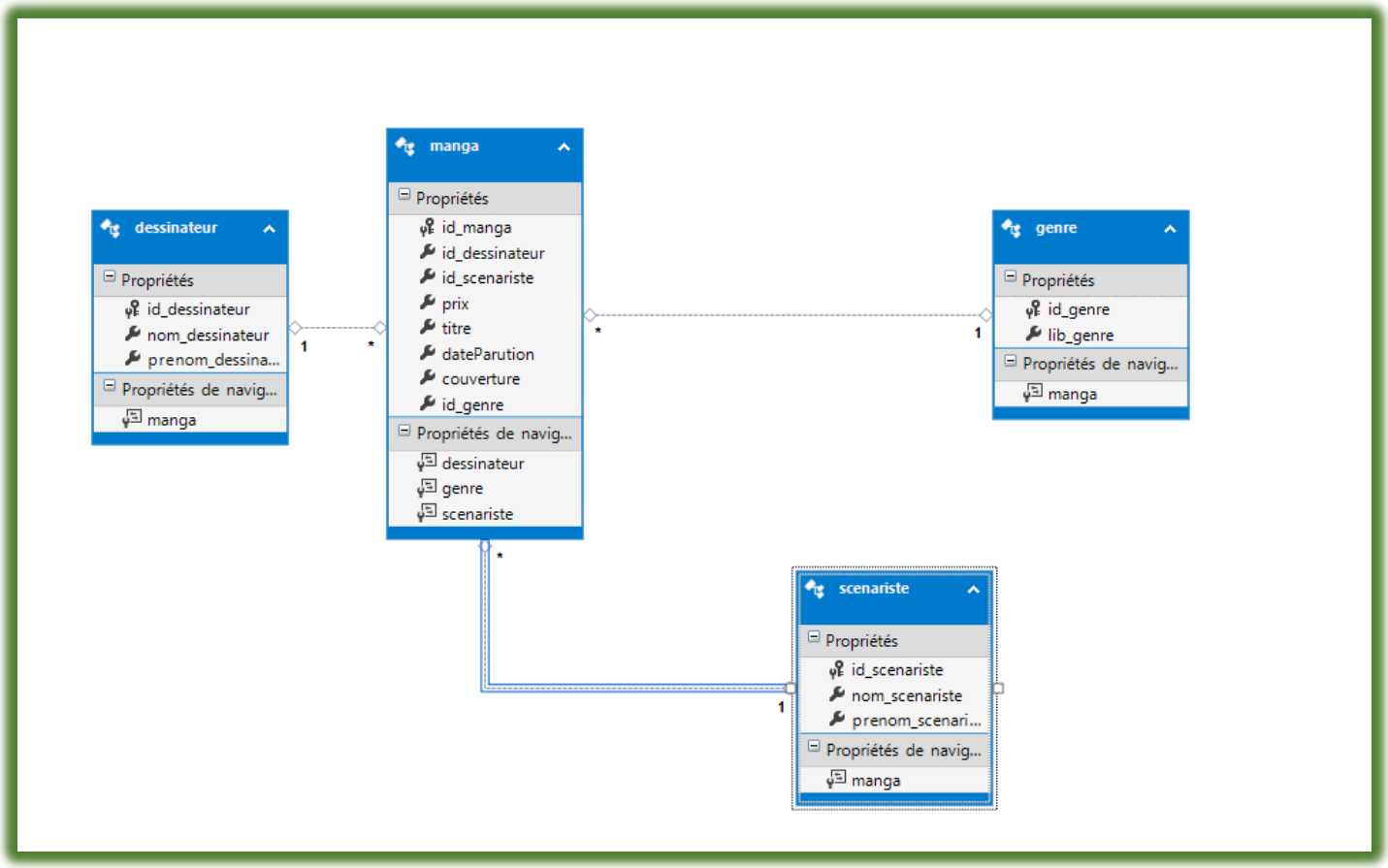
- La chaîne de connexion

```
0 références
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        optionsBuilder.UseMySql("server=localhost;port=3306;user=userepul;password=epul;database=mangas", x => x.ServerVersion("10.1.32-mariadb"));
    }
}
```

- La description des tables

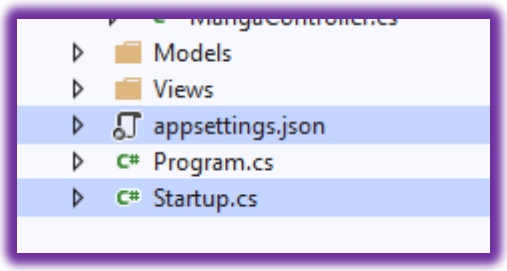
```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Dessinateur>(entity =>
    {
        entity.HasKey(e => e.IdDessinateur)
            .HasName("PRIMARY");

        entity.ToTable("dessinateur");

        entity.Property(e => e.IdDessinateur)
            .HasColumnName("id_dessinateur")
            .HasColumnType("int(11)");

        entity.Property(e => e.NomDessinateur)
            .IsRequired()
            .HasColumnName("nom_dessinateur")
            .HasColumnType("varchar(50)")
            .HasCharSet("utf8")
            .HasCollation("utf8_bin");

        entity.Property(e => e.PrenomDessinateur)
            .IsRequired()
```

Dans l'ancienne version ASP.NET, on obtenait le diagramme de classes suivant



Configuration de l'application

Il faut à présent configurer l'application en intervenant sur deux fichiers :

- appsettings.json - Startup.cs	

appsttings.json

Ce fichier contient des paramètres de configuration, on va donc ajouter la chaîne de connexion pour accéder à la base de données :

Schéma : http://json.schemastore.org/appsettings

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",

  "ConnectionString": {
    "MysqlString": "host=localhost;database=mangas;user id=userepul;password=epul"
  }
}
```

Startup.cs

Ce fichier contient les services qui seront utilisées dans l'application. On va ajouter le service d'accès à la base de données via le framework Entity dans la méthode ConfigureServices. Il faut ajouter les paquetages suivants :

- `using Microsoft.EntityFrameworkCore;`
- `using WebMangaEntity.Models.Domain;`

```
using Microsoft.EntityFrameworkCore;
using WebMangaEntity.Models.Domain;

namespace WebMangaEntity
{
    2 références
    public class Startup
    {
        0 références
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        2 références
        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        0 références
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddControllersWithViews();
            services.AddDbContext<mangasContext>(options =>
                    options.UseMySql(Configuration.GetConnectionString("ConnectionString")));
        }
```

Votre application est configurée.

Développement de l'application

Développement en couches

Une application client serveur se doit de suivre une structure qui met en œuvre les couches suivantes :

- **La couche présentation** : elle représente la partie front office de l'application, c'est elle qui est en relation directe avec l'utilisateur. Sa responsabilité s'arrête au contrôle quand cela est possible de la saisie des données.
-
- **La couche persistance** n'est plus à la charge du développeur. Elle est représentée par la couche de mapping du framework entity.

- **La couche domain** posséde les classes qui sont en relation avec la base de données. Ces dernières sont générées automatiquement dans la couche de mapping. On ajoute à cette couche une classe nommée **xxxxContext** qui regroupe tous les accès à la couche de mapping. Cette couche est en relation avec la couche présentation et la couche de mapping.

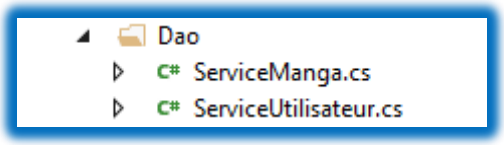
- **La couche DAO** posséde les traitements regroupés dans des classes de service. Elle s'appuie sur la couche Domain.

- **La couche mesExceptions** comprend les classes et les structures de données nécessaires à la gestion des erreurs. Dans notre exemple, nous trouverons la classe MonException qui génère une exception personnalisée. Cette couche est accessible depuis toutes les autres.

Couche DAO

Dans cette couche, nous trouverons les classes de service qui contiendront les différents traitements de l'application en s'appuyant sur la couche de mapping.

La classe ServiceUtilisteur va utiliser un singleton pour être sûr **de n'avoir qu'une seule instance** de la couche de mapping



Couche MesExceptions

On commence par ajouter à notre projet une bibliothèque de classes nommée Utilitaires

Elle offre les classes suivantes :

- MonException
- sErreurs

Classe MonException :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WebManga.Models.MesExceptions
{
    public class MonException : Exception
    {
        private string utilisateur;
        private string application;
        private string systeme;
        private string support = "Si l'erreur persiste, relevez les messages ci-dessus et prenez
contact avec le support technique.";
        public MonException()
        {
            utilisateur = application = systeme = "";
        }
        public string Utilisateur
        {
            get { return utilisateur; }
            set { utilisateur = value + "\r\n"; }
        }

        public string Application
        {
            get { return application; }
            set { application = "Origine de l'erreur : " + value + "\r\n"; }
        }
        public string Systeme
        {
            get { return  systeme; }
            set {  systeme = "Erreur système : " + value + "\r\n"; }
        }

        public MonException(string u, string a, string s)
        {
             utilisateur =  application =  systeme = "";
            if (u != "")
                 utilisateur = u + "\r\n";
            if (a != "")
                 application = "Origine de l'erreur : " + a + "\r\n";
            if (s != "")
                 systeme = "Erreur système : " + s + "\r\n";
        }
            public string MessageUtilisateur()
            {
                return ( utilisateur);
            }
            public string MessageApplication()
            {
                return ( application);
            }
            public string MessageSysteme()
            {
                return ( systeme);
            }
            public string MessageSupport()
            {
                return ( support);
```

```
            }
            public string Messages()
            {
                if ( systeme == "")
                     support = "";
                return ( utilisateur +  systeme +  support);
            }
        }
    }
```

Serreurs

C'est une classe pour conserver les informations dues aux erreurs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace WebManga.Models.MesExceptions
{
    public class Serreurs
    {
        private String _MessageUtilisateur, _MessageApplication;
        public Serreurs(String mu, String ma)
        {
            _MessageUtilisateur = mu;
            _MessageApplication = ma;
        }
        public String MessageUtilisateur()
        {
            return (_MessageUtilisateur);
        }
        public String MessageApplication()
        {
            return (_MessageApplication);
        }
    }
}
```

Couche présentation

Elle comprend les fenêtres de l'application dont :

- Fiche d'accueil avec un menu. Les items des menus sont désactivés tant que l'utilisateur ne se logue pas à l'application.

Menu

Cette couche est en liaison directe avec la couche metier mais ne voit pas la couche Persistance.

Affichage des mangas de notre contexte

Vous allez créer :

- Une page principale layout avec un menu
- Une page qui présente les mangas dans un tableau

Page principale : _LayoutManga

Cette page va contenir le menu de notre application. Elle appartient au répertoire shared du projet. Voici son code :

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - WebManga</title>

    <environment include="Development">

        <link rel="stylesheet" href="~/lib/bootstrap/dist/css/Site.css" />
        <link rel="stylesheet" href="~/lib/bootstrap/dist/css/mangas.css" />
        <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    </environment>

</head>
<body>

    <partial name="~/Views/Shared/_navigation.cshtml" />
    <div class="container body-content">
        @RenderBody()

    </div>


    <footer class="border-top footer text-muted">
        <div class="container">
            <div class="row">
                <div class="col-md-12">
                    <div class="footer-copyright text-center py-3">
                        © 2020 Copyright:CV
                        <p> @DateTime.Now.Year - Web Manga Application</p>
                    </div>
                </div>
            </div>
        </div>
    </footer>
    <script src="~/lib/jquery/dist/jquery.min.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
    <script src="~/js/site.js" asp-append-version="true"></script>
    @RenderSection("Scripts", required: false)
</body>
</html>
```

Page _Navigation.cshtml

Elle contient la NavBar

```
<div class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
```

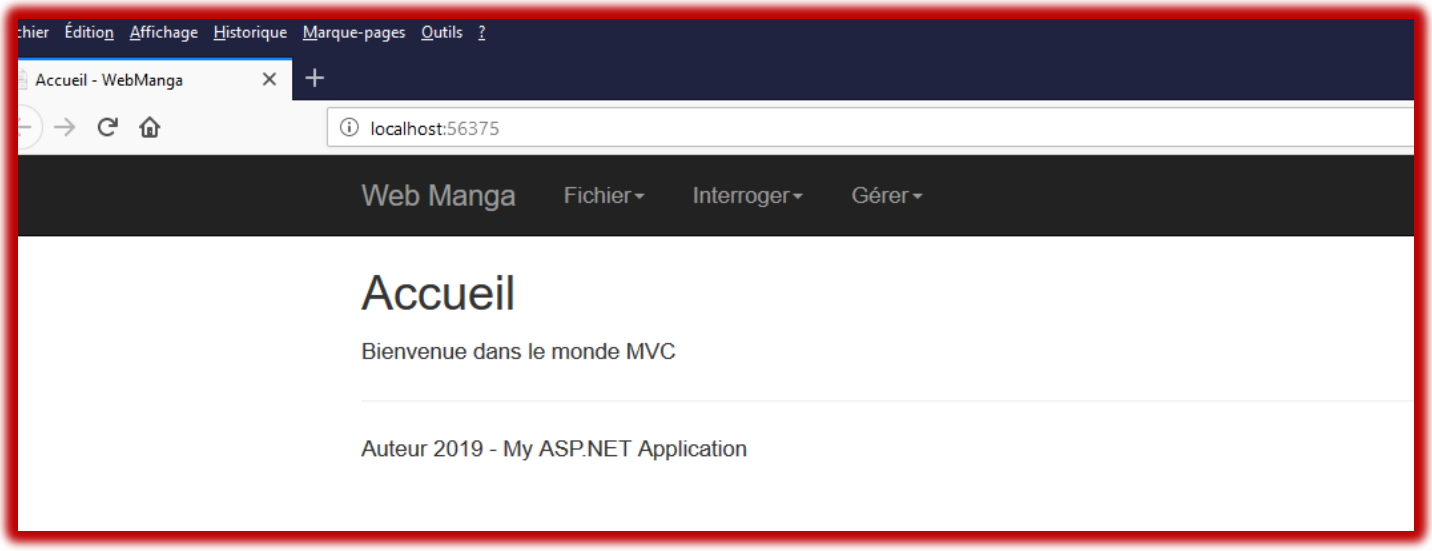
```
        <div class="navbar-header">
            <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-
collapse">
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            @Html.ActionLink("Web Manga", "About", "Home", new { area = "" }, new { @class =
"navbar-brand" })
        </div>
        <div class="navbar-collapse collapse">
            <ul class="nav navbar-nav">
                <li class="dropdown">
                    <a href="#" class="dropdown-toggle" data-toggle="dropdown">Fichier<b
class="caret"></b></a>
                    <ul class="dropdown-menu">
                        <li class="dropdown-header">Application</li>
                        <li>@Html.ActionLink("Connexion", "Index", "Connexion")</li>
                        <li class="dropdown-header">----------</li>
                        <li role="separator" class="divider"></li>
                        <li>@Html.ActionLink("Quitter", "", "")</li>

                    </ul>
                </li>
                <li class="dropdown">
                    <a href="#" class="dropdown-toggle" data-toggle="dropdown">Interroger<b
class="caret"></b></a>
                    <ul class="dropdown-menu">
                        <li class="dropdown-header">Fiches</li>
                        <li role="separator" class="divider"></li>
                        <li>@Html.ActionLink("Liste des Mangas", "Index", "Manga")</li>
                        <li>@Html.ActionLink("Manga par genre", "Genre", "Manga")</li>
                    </ul>
                </li>
                <li class="dropdown">
                    <a href="#" class="dropdown-toggle" data-toggle="dropdown">Gérer<b
class="caret"></b></a>
                    <ul class="dropdown-menu">
                        <li class="dropdown-header">Gestion</li>
                        <li role="separator" class="divider"></li>
                        <li>@Html.ActionLink("Ajouter Manjoa", "Ajouter", "Manga")</li>
                        <li>@Html.ActionLink("Modifier Manjoa", "Modifier", "Manga")</li>

                        <li role="separator" class="divider"></li>
                        <li class="dropdown-header">Manga</li>
                        <li role="separator" class="divider"></li>
                        <li>@Html.ActionLink(">Augmenter Prix", "Augmenter", "Manga")</li>

                    </ul>
                </li>
            </ul>
        </div>
    </div>
</div>
```

Page index

On va utiliser notre layout en créant une page Index.cshtml de départ.
Voici le code
On peut noter que cette page va venir s'insérer dans layout.

```
@{  ViewBag.Title = "Accueil";
    Layout = Layout = "~/Views/Shared/_LayoutManga.cshtml";
}
<h2>Accueil</h2>
<div>
    <p>
       Bienvenue dans le monde MVC  <span style="color:red">@ViewData["Nom"]</span>
    </p>
</div>
```

Exécution

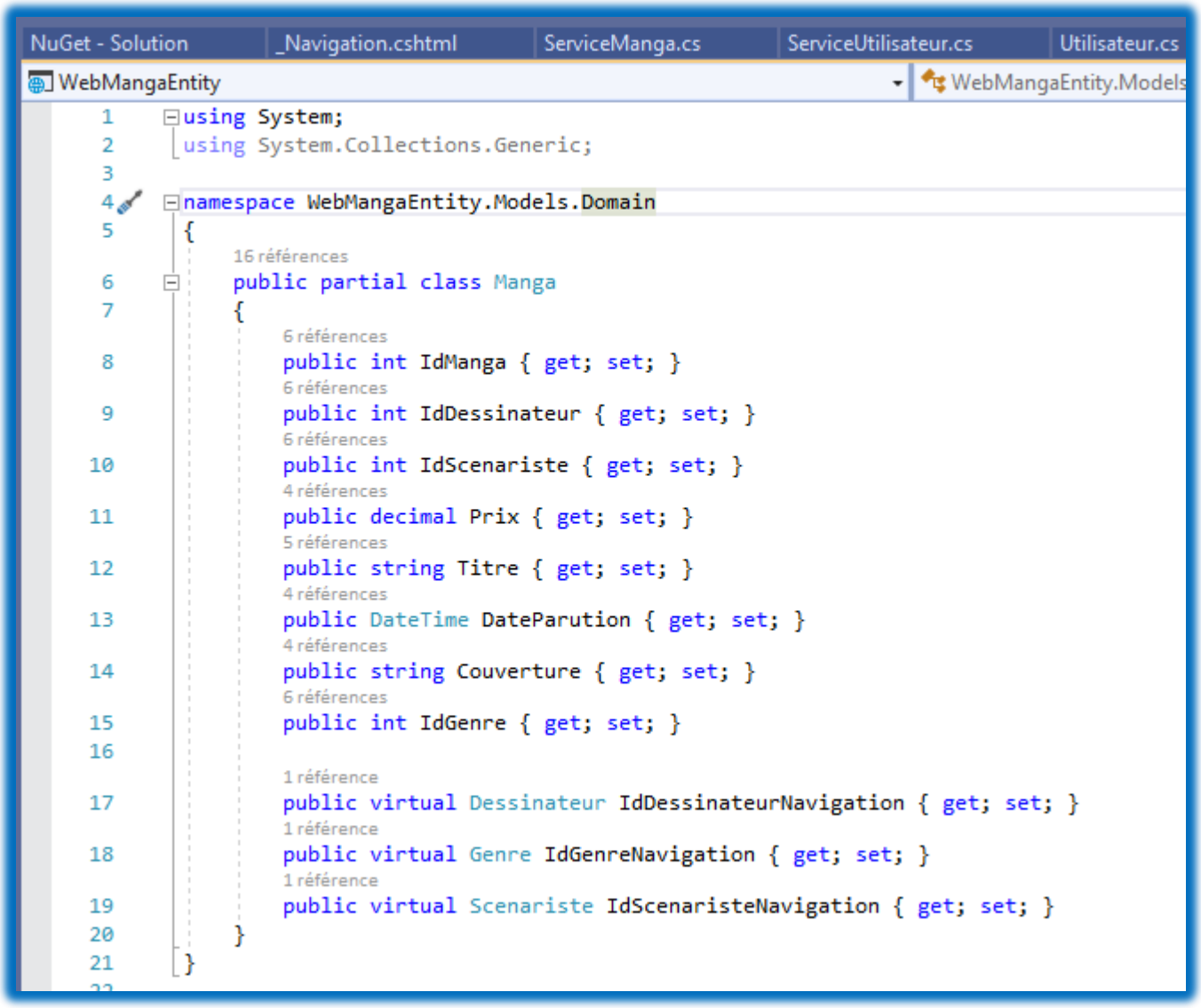


Mise en place des items du menu

Nous allons commencer à développer l'application. La première page que nous allons créer sera l'affichage des mangas.

Impact sur le modèle

Aucun impact, on va utiliser les classes du Domain comme elles sont générées.

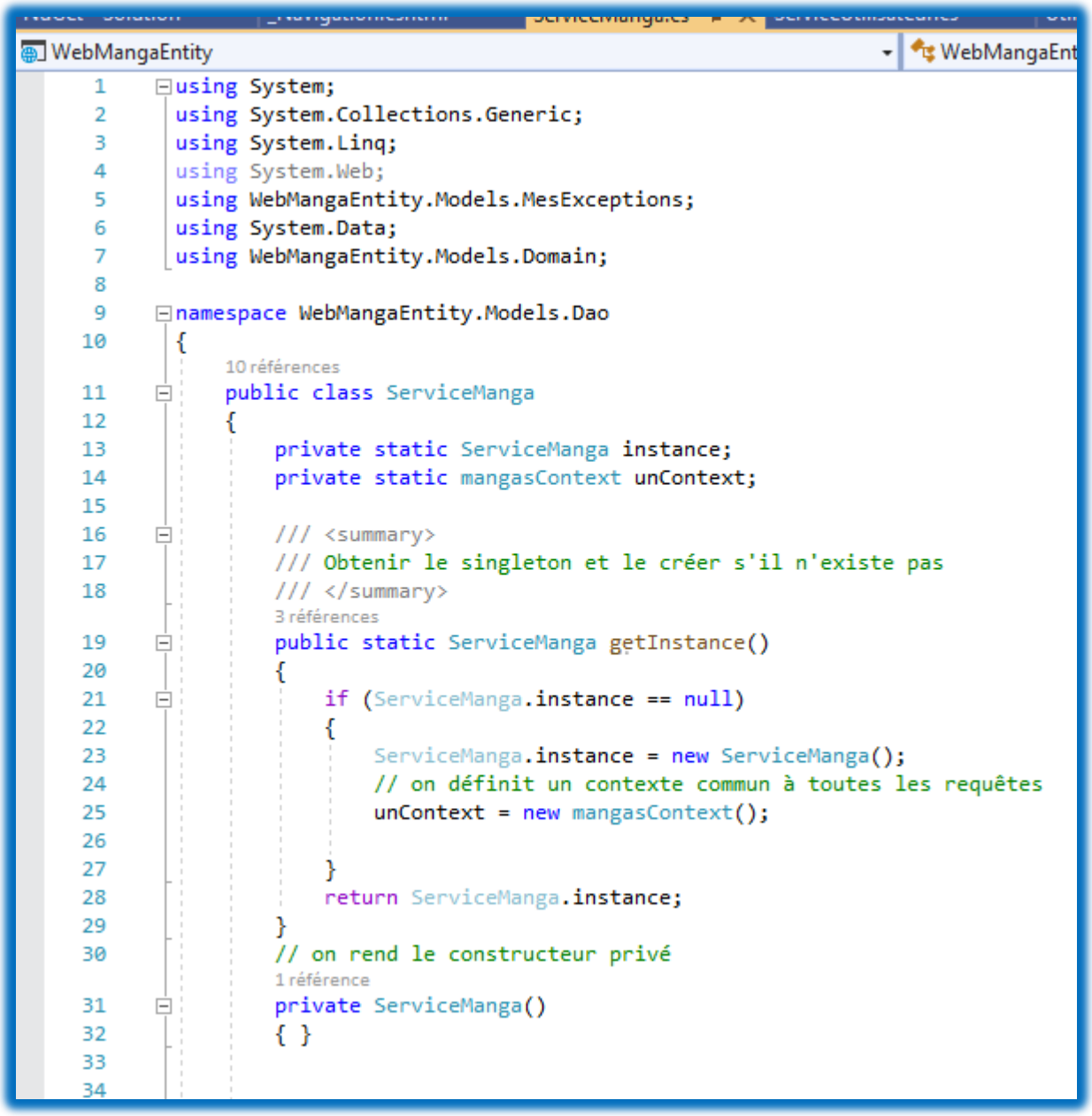


Impact sur la classe ServiceManga

On va ajouter les traitements concernant les mangas qui interrogent la base de données en utilisant le langage Linq.

Constructeur de cette classe

Le constructeur de cette classe va créer une instance sur le modèle mangaContext si celui-ci est nul. Pour cela, on va utiliser un singleton



```
WebMangaEntity                                              WebMangaEnt
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Web;
5   using WebMangaEntity.Models.MesExceptions;
6   using System.Data;
7   using WebMangaEntity.Models.Domain;
8
9   namespace WebMangaEntity.Models.Dao
10  {
        10 références
11      public class ServiceManga
12      {
13          private static ServiceManga instance;
14          private static mangasContext unContext;
15
16          /// <summary>
17          /// Obtenir le singleton et le créer s'il n'existe pas
18          /// </summary>
            3 références
19          public static ServiceManga getInstance()
20          {
21              if (ServiceManga.instance == null)
22              {
23                  ServiceManga.instance = new ServiceManga();
24                  // on définit un contexte commun à toutes les requêtes
25                  unContext = new mangasContext();
26
27              }
28              return ServiceManga.instance;
29          }
30          // on rend le constructeur privé
            1 référence
31          private ServiceManga()
32          { }
33
34
```

Manga sur un numéro

On développe une classe statique qui va utiliser une référence sur le modèle mangasContext. La connexion à la base de données se fait par le modèle mangasContext.

```
2 références | 0 exceptions
public  manga GetunManga(int id)
{
    manga unManga = null;

    Serreurs er = new Serreurs("Erreur sur lecture des Mangas", "ServiceManga.getunManag()");
    try
    {
        unManga = (from m in unMangaEntity.manga
                   where m.id_manga == id
                   select m)
                   .FirstOrDefault();

        return unManga;
    }
    catch (Exception e)
    {
        throw new MonException(er.MessageUtilisateur(), er.MessageApplication(), e.Message);
    }
}
```

Liste de tous les mangas

On va utiliser un DataTable car les données sont issues de plusieurs tables, on ne peut pas renvoyer une collection typée.

```
/// <summary>
/// Lister les magas de la base
/// </summary>
/// <returns>Liste des mangas</returns>
1 référence | 0 exceptions
public  DataTable ListMangas()
{
    Serreurs er = new Serreurs("Erreur sur lecture des mangas.", "Servivemanga.ListeMangas()");
    DataTable dt = new DataTable();
    /// On construit les colonnes
    /// du datatable
    try
    {
        dt.Columns.Add("id_manga", typeof(int));
        dt.Columns.Add("lib_genre", typeof(String));
        dt.Columns.Add("nom_dessinateur", typeof(String));
        dt.Columns.Add("nom_scenariste", typeof(String));
        dt.Columns.Add("dateParution", typeof(DateTime));
        dt.Columns.Add("titre", typeof(String));
        dt.Columns.Add("prix", typeof(Double));
        dt.Columns.Add("couverture", typeof(String));

        var mesMangas = (from m in unMangaEntity.manga
                         join g in unMangaEntity.genre on m.id_genre equals g.id_genre
                         join d in unMangaEntity.dessinateur on m.id_dessinateur equals d.id_dessinateur
                         join s in unMangaEntity.scenariste on m.id_scenariste equals s.id_scenariste
                         orderby m.id_manga
                         select new { m.id_manga, g.lib_genre, d.nom_dessinateur, s.nom_scenariste, m.dateParution,m.titre, m.prix, m.couverture }
                         );
        foreach (var res in mesMangas)
        {
            dt.Rows.Add(res.id_manga, res.lib_genre, res.nom_dessinateur, res.nom_scenariste, res.dateParution, res.titre,res.prix,res.couverture);
        }
        return dt;
    }
    catch (Exception e)
    {
        throw new MonException(er.MessageUtilisateur(),
            er.MessageApplication(), e.Message);
    }
}
```

Les méthodes sont appelées à travers des exceptions. Il faut donc mettre en place une couche Exception dans le modèle.

Fonctionnement de l'application : affichage des mangas

Sur l'item du menu Liste des Mangas de la page _layoutCommercial, on appelle le contrôleur de la classe Manga : MangaController.

En fait, au lancement de l'application, le système construit une table des routes grâce au traitement RegisterRoutes de la classe Startup.cs

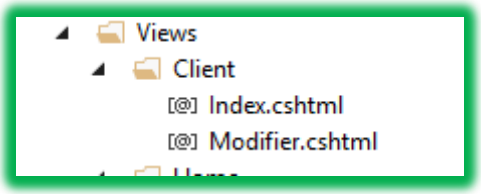
```
0 références
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
        // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
        app.UseHsts();
    }
    app.UseHttpsRedirection();
    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
    });
}
}
}
```

MangaController (c'est surprenant mais c'est la norme de Microsoft, Controller devient implicite), on recherche alors la méthode Index qui va :

- Instancier un objet de type Service
- Appeler la méthode ListMangas () qui sera appelée à travers l'instance du constructeur.

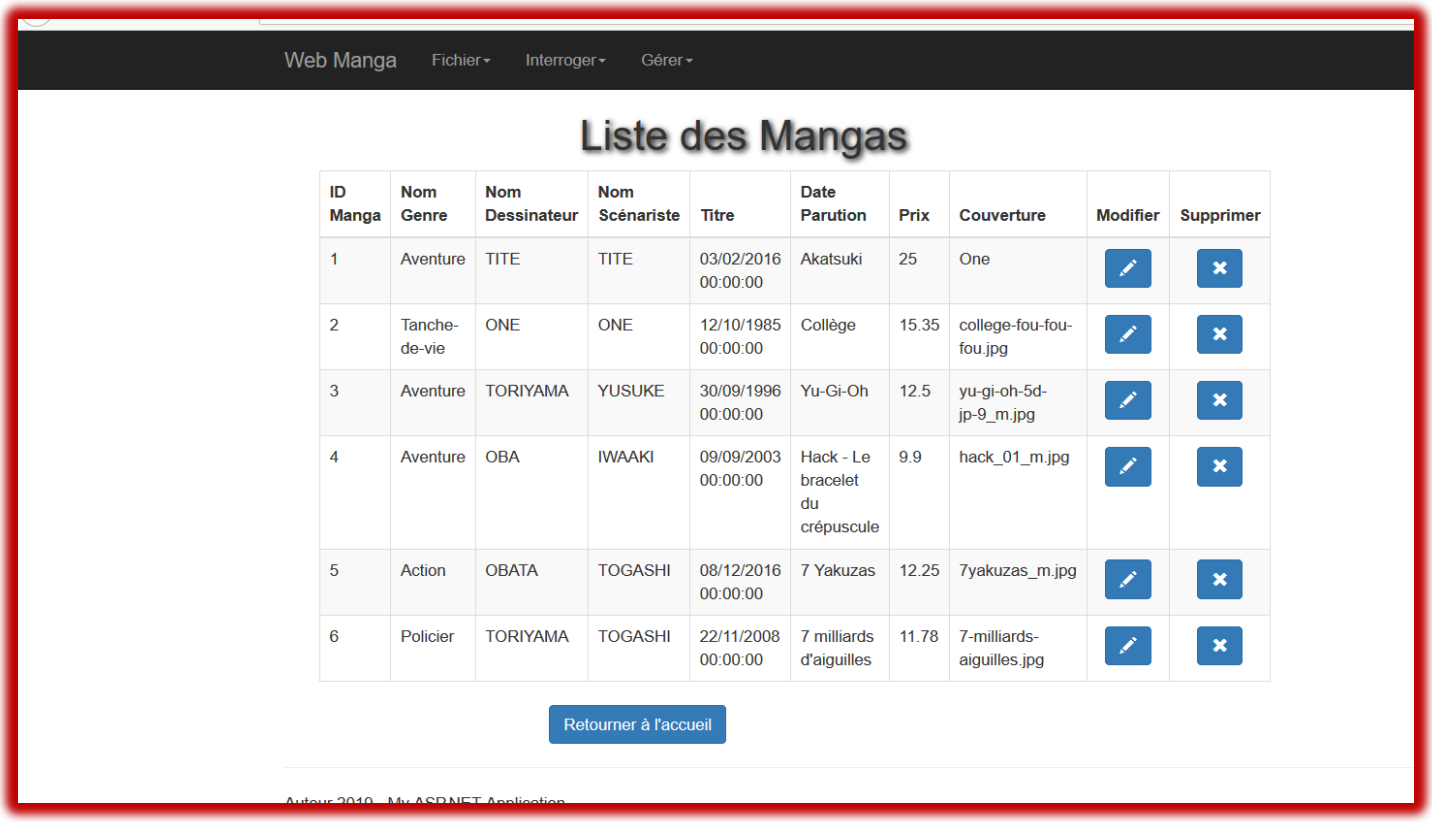
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using WebMangaEntity.Models.MesExceptions;
using WebMangaEntity.Models.Dao;
using WebMangaEntity.Models.Domain;
using System.Data;
using Microsoft.AspNetCore.Mvc;

namespace WebMangaEntity.Controllers
{
    0 références
    public class MangaController : Controller
    {
        // GET: Manga
        0 références
        public ActionResult Index()
        {
            DataTable mesMangas = null;
            try
            {
                mesMangas = ServiceManga.getInstance().ListMangas();
            }
            catch (MonException e)
            {
                ModelState.AddModelError("Erreur", "Erreur lors de la récupération des mangas : " + e.Message);
            }
            return View(mesMangas);
        }
```

Le contrôleur va alors rechercher une vue nommée Manga et appeler sa page Index



```
@{
    ViewBag.Title = "Liste des Mangas";
    Layout = "~/Views/Shared/_LayoutManga.cshtml";
}
@if (Model != null)
{ <div class="container">
        <div class="col-md-8 col-sm-8">
            <div class="blanc">
                <h1>Liste des Mangas </h1>
            </div>

            <table class="table table-bordered table-striped table-responsive">
                <thead>
                    <tr>
                        <th> ID Manga</th>
                        <th> Nom Genre</th>
                        <th> Nom Dessinateur</th>
                        <th>Nom Scénariste</th>
                        <th>Titre</th>
                        <th>Date Parution </th>
                        <th>Prix</th>
                        <th>Couverture</th>
                        <th style="width:80px">Modifier</th>
                        <th style="width:80px">Supprimer</th>
                    </tr>
                </thead>
                @foreach (DataRow dataRow in Model.Rows)
                {
                <tr>
                    @foreach (var item in dataRow.ItemArray)
                    {
                        <td>
                            @item.ToString()

                        </td>
                    }
                    <td style="text-align:center;">
                        <span>
                            @Html.ActionLink("  ", "Modifier", "Manga", new { id = dataRow[0] }, new { @class = "btn btn-default btn-primary glyphicon glyphicon glyphicon-pencil" })
                        </span>
                    </td>
                    <td style="text-align:center;">
                        <span>
                            @Html.ActionLink("  ", "Supprimer", "Manga", new { id = dataRow[0] }, new { @class = "btn btn-default btn-primary glyphicon glyphicon glyphicon-remove" })
                        </span>
                    </td>
                </tr>
                }
            </table>
            <div class="form-group">
                <div class="col-md-6 col-md-offset-3 col-sm-6 col-sm-offset-3">
                    @Html.ActionLink("Retourner à l'accueil", "/", "", new { @class = "btn btn-default btn-primary" })
                </div>
```

L'exécution donne :



Modification d'un manga

Pour réaliser ce traitement, on part du bouton de modification. Le code est :

```
<span>
    @Html.ActionLink("  ", "Modifier", "Manga", new { id = dataRow[0] }, new { @class = "btn btn-default btn-primary glyphicon glyphicon glyphicon-pencil" })
</span>
```

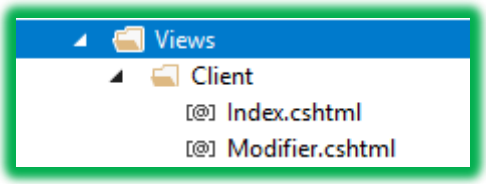
- Le premier paramètre indique un texte de lien (commentaire)
- Le second, la méthode à rechercher dans le contrôleur
- Le troisième indique le contrôleur : MangaController
- Le quatrième indique un paramètre pour la méthode Modifier
- Le cinquième indique des attributs HTML

Contrôleur Manga

Dans ce contrôleur, il faut écrire la méthode modifier avec comme paramètre l'id du manga. On va rechercher le manga sur son ID et appeler un formulaire de modification

```
// GET: Commande/Edit/5
0 références | 0 requête | 0 exceptions
public ActionResult Modifier(int id)
{
    manga unManga = null;
    try
    {
        unManga = ServiceManga.getInstance().GetunManga(id);
        return View(unManga);
    }
    catch (MonException e)
    {
        return HttpNotFound();
    }
}
```

Le formulaire recherché doit appartenir à la vue Manga et doit s'appeler Modifier



Formulaire Modifier

Il récupère en paramètre un objet Manga à modifier

```
}
<h2>Modifier un Manga</h2>
@if (Model != null)
{
    using (Html.BeginForm())
    {
        @Html.AntiForgeryToken()
            <div class="form-horizontal">
                <hr />
                @Html.ValidationSummary(true, "", new { @class = "text-danger" })
                <div class="form-group">
                    @Html.LabelFor(model => model.IdManga, htmlAttributes: new { @class = "control-label col-md-2" })
                    <div class="col-md-10">
                        @Html.EditorFor(model => model.IdManga, new { htmlAttributes = new { @class = "form-control", @disabled = "" } })
                        @Html.ValidationMessageFor(model => model.IdManga, "", new { @class = "text-danger" })
                        <input name="IdManga" type="HIDDEN" value=@(@Model.IdManga) id="IdManga" />
                    </div>
                </div>

                <div class="form-group">
                    <label class="control-label col-md-2">Id dessinateur</label>
                    <div class="col-md-10">
                        <input name="IdDessinateur" type="text" class="form-control" value=@(@Model.IdDessinateur) id="Id_dessinateur" />
                    </div>
                </div>

                <div class="form-group">
                    <label class="control-label col-md-2">Id Scénariste</label>
                    <div class="col-md-10">
                        <input name="IdScenariste" type="text" class="form-control" value=@(@Model.IdScenariste) id="Id_scenariste" />
                    </div>
                </div>

                <div class="form-group">
                    <label class="control-label col-md-2">Prix du manga</label>
                    <div class="col-md-10">
                        <input name="Prix" type="text" class="form-control" value=@(@Model.Prix) id="Prix" />
                    </div>
                </div>
                <div class="form-group">
                    <label class="control-label col-md-2">titre du manga</label>
                    <div class="col-md-10">
                        <input name="Titre" type="text" class="form-control" value=@(@Model.Titre) id="Titre" />
                    </div>
                </div>

                <div class="form-group">
                    <label class="control-label col-md-2">Couverture</label>
                    <div class="col-md-10">
                        <input name="Couverture" type="text" class="form-control" value=@(@Model.Couverture) id="Couverture" />
                    </div>
                </div>
```

```
                <div class="form-group">
                    <label class="control-label col-md-2">Date parution</label>
                    <div class="col-md-10">
                        <input name="DateParution" type="text" class="form-control" value=@(@Model.DateParution) id="DateParution" />
                    </div>
                </div>
                <div class="form-group">
                    <label class="control-label col-md-2">Genre du manga</label>
                    <div class="col-md-10">
                        <input name="IdGenre" type="text" class="form-control" value=@(@Model.IdGenre) id="ID Genre" />
                    </div>
                </div>
                <div class="form-group">
                    <div class="col-md-offset-2 col-md-10">
                        <input type="submit" value="Enregistrer" class="btn btn-primary" />

                    </div>
                </div>
            </div>

    }
}
```

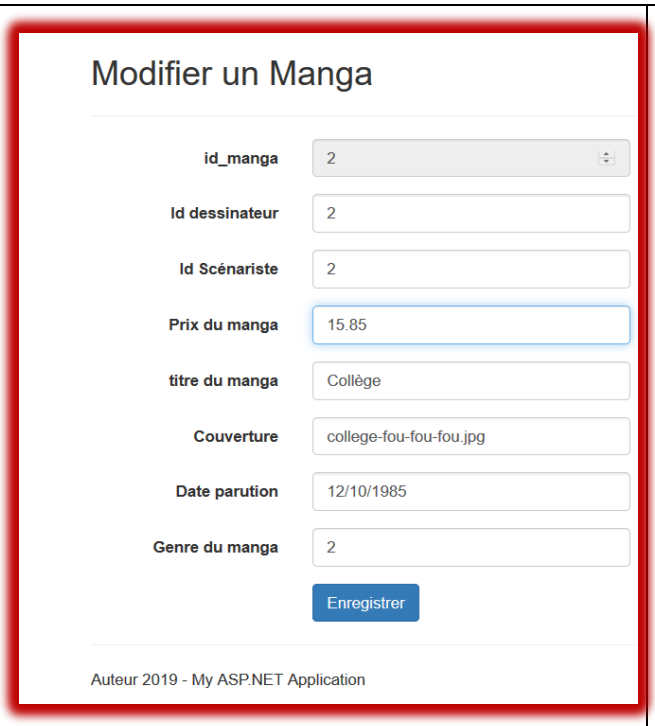
C'est un formulaire, using (Html.BeginForm()), à la sortie du formulaire, on revient à une méthode Modifier dans le contrôleur du manga. L'appel se fera en mode POST. Il recevra en paramètre un objet Manga formé avec les noms des champs qui sont identiques avec les propriétés de la classe Manga (Binding)

```
[HttpPost]
0 références | 0 requête | 0 exceptions
public ActionResult Modifier(manga unM)
{
    try
    {
        ServiceManga.getInstance().UpdateManga(unM);
        return View();
    }
    catch (MonException e)
    {
        return HttpNotFound();
    }
}
```

Le contrôleur va appeler une méthode de modification de la classe ServiceManga

```
/// <summary>
/// Fonction qui met à jour un manga
/// </summary>
/// <param name="unM"></param>
1 référence
public void UpdateManga(Manga unM)
{
    Manga unManga = null;
    Serreurs er = new Serreurs("Erreur sur l'écriture d'un manga.", "ServiceManga.update()");

    // requête de sélection pour une mise à jour .

    unManga = GetunManga(unM.IdManga);

    // On modifie les données

    unManga.IdDessinateur = unM.IdDessinateur;
    unManga.IdGenre = unM.IdGenre;
    unManga.IdScenariste= unM.IdScenariste;
    unManga.DateParution = unM.DateParution;
    unManga.Prix = unM.Prix;
    unManga.Titre = unM.Titre;
    unManga.Couverture = unM.Couverture;


    // On enregistre les modifications dans la base de données .
    try
    {
        unContext.SaveChanges();
    }
    catch (Exception e)
    {
        throw new MonException(er.MessageUtilisateur(),
            er.MessageApplication(), e.Message);
    }
}
```

Exécution



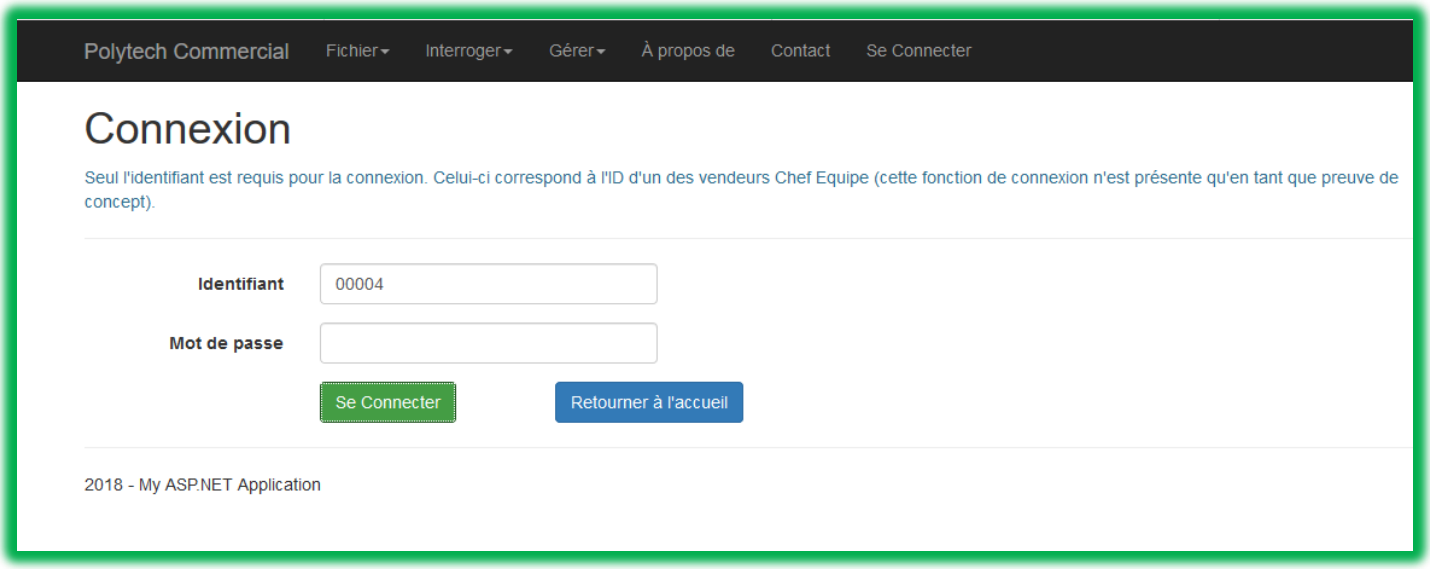
Application à compléter

Connexion

Ajouter une fonctionnalité d'authentification en créant une table utilisateurs avec :

- Idlogin
- Mot de passe
- Fonction (Admin ou Lecture)
- Nom
- Prénom

Le mot de passe doit être crypté et vous devez éviter l'injonction SQL pour le contrôle. Donc le mot de passe est vérifié dans le code. Seuls, les administrateurs pourront gérer la base.



	Travail à faire
1.2	Réalisez ce traitement
1.3	Affichez la liste des mangas avec leur couverture Implémentez les déplacements du fichier couverture lors d'une modification Ce traitement n'est accessible qu'à un utilisateur Administrateur

Annexe

Variables de session

Bibliothèque
using System.Web.HttpContext.Current;

//Stocker une valeur
Session["login"]=”cvial”;

Accéder à une variable de session
unlogin = Session["login"] ;

//effacer une variable de session
Session["login"]=null;