TP 3 Seed Data & ViewModels & Upload Images

//Seed DATA au niveau du OnModelCreating

1. Ajouter un fichier JSOn contenant les enregistrements

```
[
{"Id":"e1081557-8301-405c-887a-c44e6be1954e","Name":"Informatique"},
{"Id":"55c00488-dc7e-4909-bb85-708cba3a48e7","Name":"Electromenager"}]
```

2. Faire les modifications du OnModelCreating pour ajouter les enregistrements à la base de données

```
string CatJson = System.IO.File.ReadAllText("Categories.Json");
            List<Categorie> categories=System.Text.Json.
                JsonSerializer.Deserialize<List<Categorie>>(CatJson);
            //Seed to categorie
            foreach (Categorie c in categories)
                modelBuilder.Entity<Categorie>()
                        .HasData(c);
```

→ **JsonSerializer.Deserialize** Fournit les fonctionnalités permettant de sérialiser des objets ou des types valeur en JSON et de désérialiser JSON en objets ou types valeur.

Deserialize(JsonDocument, Type, JsonSerializerContext)	Convertit la JsonDocument valeur JSON représentant une seule valeur JSON en un `returnType`.

3. Ajouter un contrôleur et une vue permettant d'ajouter une "sousCategorie" ayant une dépendance avec categorie (one-to-many).

Penser à définir des dépendances nullables dans sousCategorie.

```
public IActionResult Create()
        {
            var c =_context.categories.ToList();
            ViewBag.Categories = c.Select(c
                => new SelectListItem()
                {
                    Text = c.Name
                ,
                    Value = c.Id.ToString()
                });
            return View();
           }
[HttpPost]
        public IActionResult Create(SousCategorie sscat)
        {
            if (!ModelState.IsValid)
            {
                List<Categorie> cat = _context.categories.ToList();
                 //ViewBag est utilisé pour transférer des données temporaires (non inclus dans le modèle) du contrôleur à la vue.
                ViewBag.Categories = cat.Select(c
                => new SelectListItem()
                {
                    Text = c.Name              //texte à afficher de la selectList
                ,
                    Value = c.Id.ToString()    //Valeur de la selectList
                });
                return View();
            }
            sscat.Id = Guid.NewGuid();
            _context.sscategories.Add(sscat);
            _context.SaveChanges();
            return RedirectToAction(nameof(Index));
        }
```

4. Génération de la vue Fortement typée

```
 <div class="form-group">
                <label asp-for="categorieId" class="control-label">Categorie Name</label>
                <select asp-for="categorieId" class ="form-control" asp-
items="@ViewBag.categories"></select>
   <span asp-validation-for="categorieId" class="text-red"></span>
</div>.....
```

5. Ajout du contrôle des erreurs du ModelState via ViewBag

ModelState encapsule les erreurs qui peuvent venir à partir de la liaison du modèle et de la validation du Modèle.

ModelState. IsValid renvoie true si le modèle reçu en paramètre de l'action est valide à savoir que les règles de validations appliquées en utilisant les attributs d'annotations de données.

Ajouter ce code au niveau du contrôleur et vue. Expliquez.

```
ViewBag.Errors = ModelState.Values
                    .SelectMany(v => v.Errors).Select(e => e.ErrorMessage)
                       .ToList();


   //View, s'il y a des erreurs, on va les afficher
@if (ViewBag.Errors != null)
            {
                <div class="text-red ml">
                    <ul>
                        @foreach (string error in ViewBag.Errors)
                        {
                            <li class="ml">@error</li>
                        }
                    </ul>
                </div>
               }
```

- Ajoutons à présent les deux propriétés suivantes à la classe « Produit »

```
public string? ImageFile { get; set; }
public DateTime? DateAjoutProduit  { get; set; }
```

- Définir les migrations et mettre à jour la base de données
- Ajouter un ViewModel « ProduitVM » permettant de créer un nouveau produit incluant une image

```
    public class ProduitVM
 {
     public Produit produit { get; set; }


     public IFormFile photo { get; set; }
       }
```

Pourquoi on utilise les ViewModels dans le cas suivant ?

IFormFile représente un fichier envoyé avec httpRequest. Cette interface nous permet de lire un contenu d'un fichier via un stream.

Etant donnée qu'on ne peut pas définir un champ IFormFile mappé à une entité, nous avons utilisé ViewModels pour assurer ce binding.

- Ajouter ProduitController qui contient ActionResult Create pour la création d'un nouveau produit.

```
[HttpPost]
[ValidateAntiForgeryToken]

public IActionResult Create(ProduitVM model, IFormFile photo)
```

```
        {
            if (photo == null)
                return Content("File not uploaded");

            try
            {
//Combine trois chaînes dans un seul path

                var path = Path.Combine(webHostEnvironment.WebRootPath, "images",
photo.FileName);
//fournit un stream pour la lecture et ecriture dans un fichier
                using (FileStream stream = new FileStream(path, FileMode.Create))
                {
                    photo.CopyTo(stream);
                    stream.Close();
                }

                model.produit.ImageFile = photo.FileName;
//Mapping entre Model et ViewModel

                var produit = new Produit
                {
                    Id = new Guid(),
                    Name = model.produit.Name,
                    DateAjoutProduit = model.produit.DateAjoutProduit,
                    ImageFile = photo.FileName,

                };
                _context.Add(produit);
                _context.SaveChanges();

                return RedirectToAction(nameof(Index));

            }
            catch (Exception)
            {
                throw;
            }
        }
```

- Ajouter une vue fortement typée (Faites les modifications nécessaires)

```
@model WebApplication2023MVCCore.Models.ViewModels.ProduitVM

@{
    ViewData["Title"] = "Create";
}

<h1>Create</h1>

<h4>Produit</h4>
<hr />
<div>
    <a asp-action="Index">Back to List</a>
</div>

<div>
    <form asp-action="Create" method="post" enctype="multipart/form-data">
        <div asp-validation-summary="ModelOnly"></div><input type="hidden" asp-
for="produit.Id"/>
        <div>


            <table class="table table-sm table-bordered table-striped">

                <tr>
                    <th> <label asp-for="produit.Name"></label></th>
```

```
                    <td> <input asp-for="produit.Name" class="form-control" placeholder="Enter
produit name" /><span asp-validation-for="produit.Name"></span></td>
                </tr>
                <tr>
                    <th> <label asp-for="produit.DateAjoutProduit"></label></th>
                    <td> <input asp-for="produit.DateAjoutProduit" class="form-control"
placeholder="Enter ajout date" /><span asp-validation-
for="produit.DateAjoutProduit"></span></td>
                </tr>
                <tr>
                <th><label asp-for="photo"></label></th>
                <td><input asp-for="photo" accept="image/*" type="file" class="form-control"
/>
                <span asp-validation-for="photo" class="text-danger" />
                </td>
                </tr>

                <tr>
                    <th>  <button type="submit" class="btn btn-primary"
style="width:107px">Save Produit</button></th>
                    <td> </td>
                </tr>
                <tr>
                    <th>@Html.ActionLink("Back To List", "Index", new { /* id=item.PrimaryKey
*/ }, new { @class = "btn btn-success" })</th>
                    <td> </td>
                </tr>
            </table>
        </div>
    </form>

</div>

@section Scripts {
    @{
        await Html.RenderPartialAsync("_ValidationScriptsPartial");
    }
    }
```