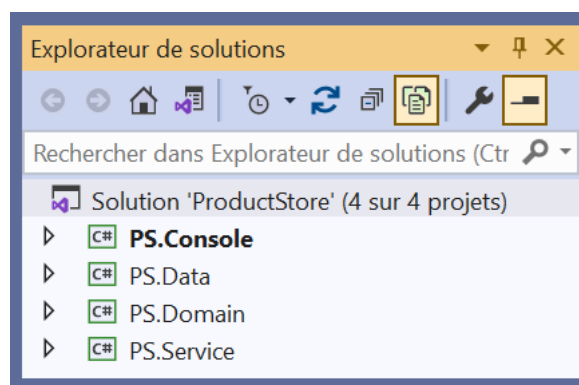


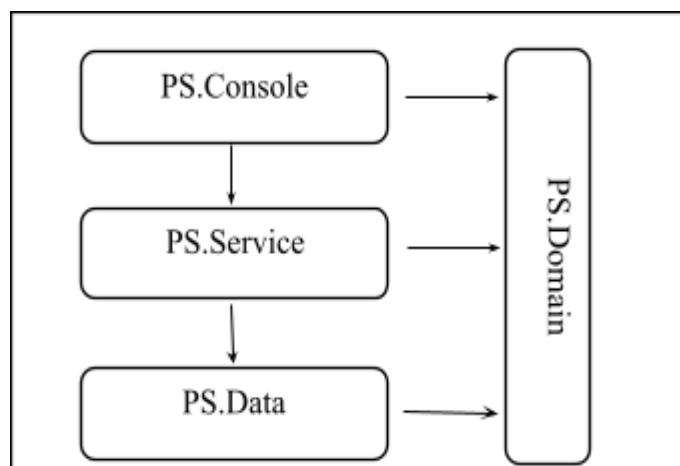
(Correction)

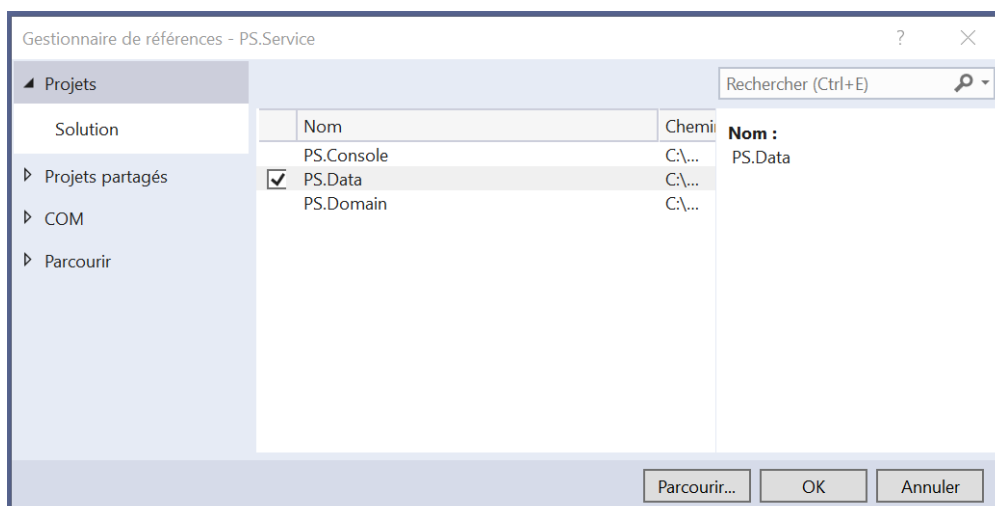
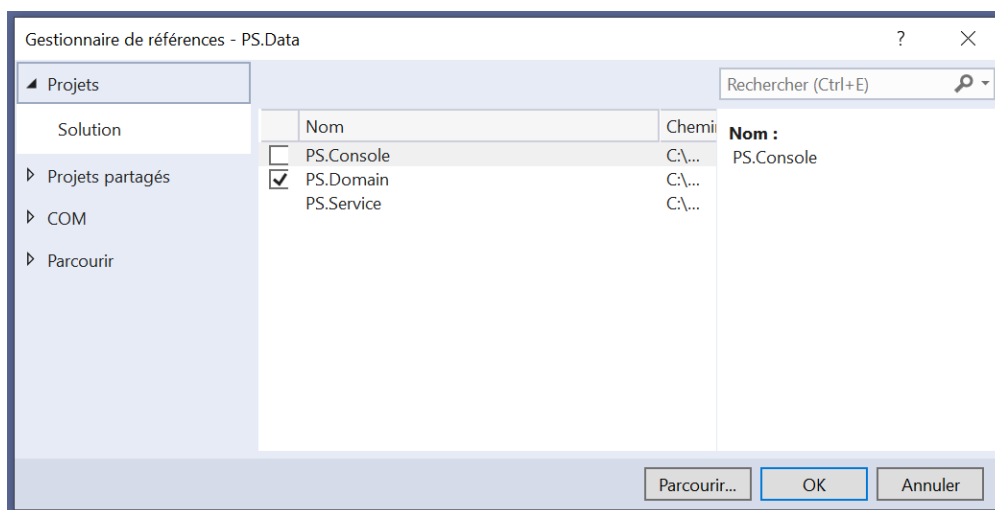
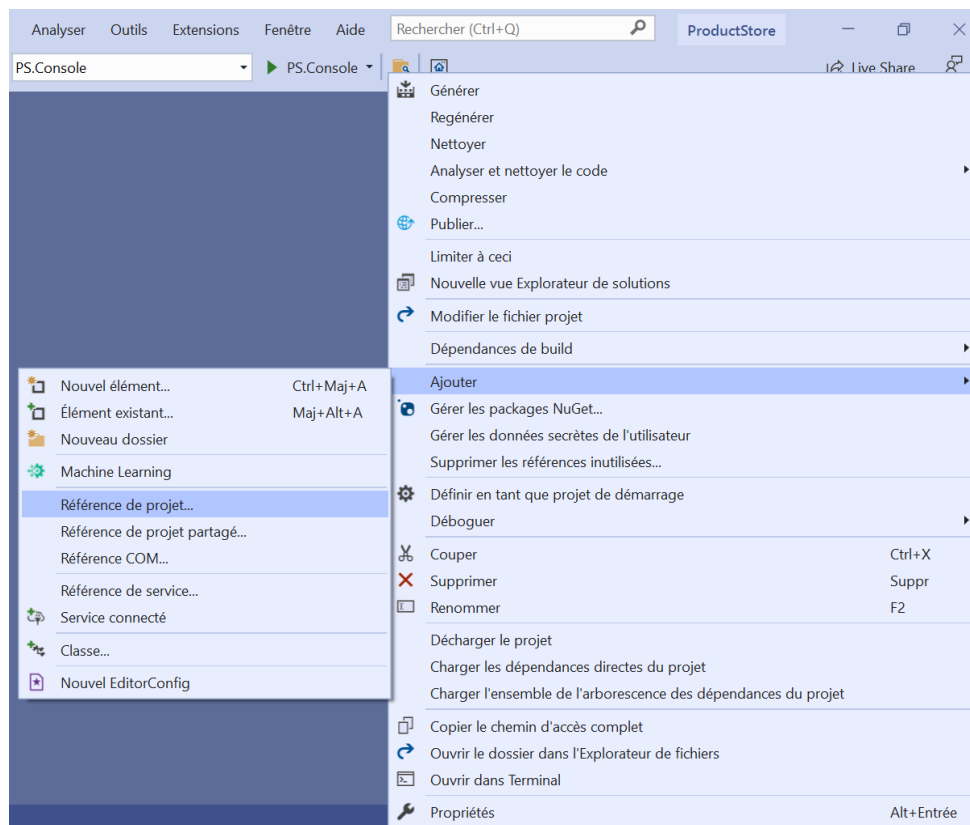
**Partie 3.1 : Mise en place**

1. Dans la solution ProductStore, créer un autre projet nommé « PS.Data » de type « Bibliothèque de classes (.Net Core) ».

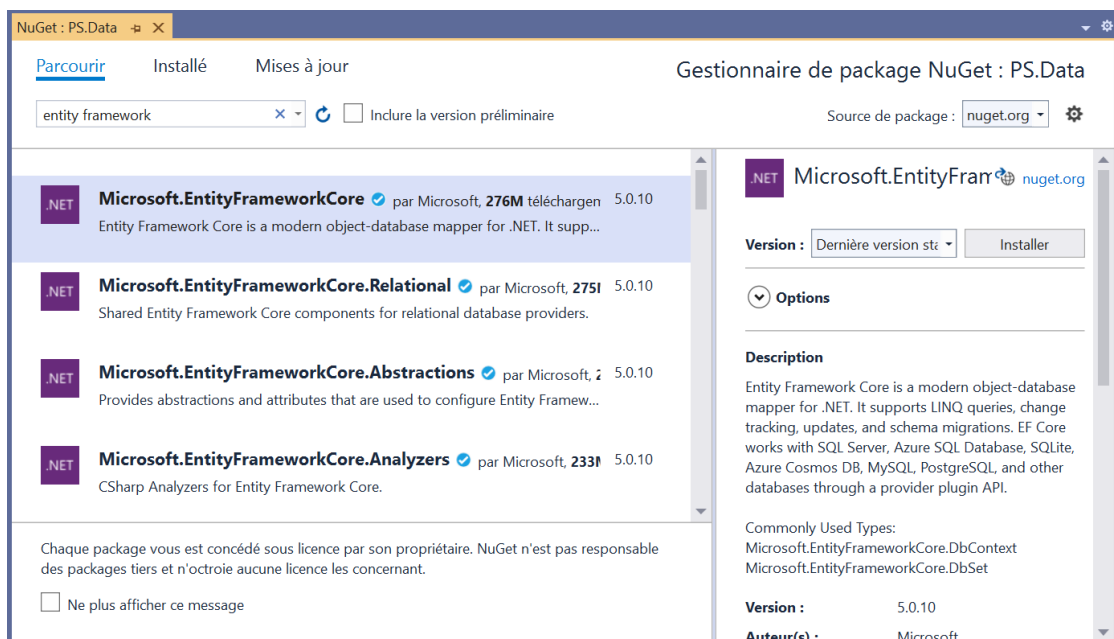
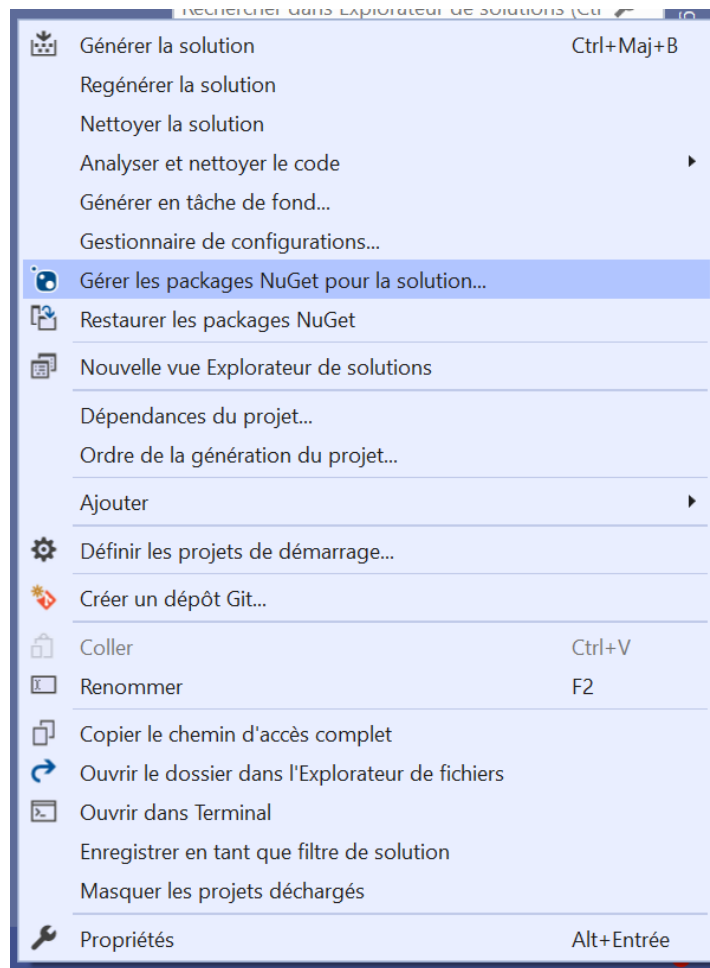


2. Ajouter les références entre les projets: Le projet PS.Data doit référencer le projet PS.Domain. Le projet PS.Service doit référencer le projet PS.Data.





### 3. Installer Entity Framework Core



### **Partie 3.2 : Implémentation du Context**

Une fois les entités sont créées. L'approche Code-First exige la classe contexte dérivée de DbContext et contient les DbSet qui représentent les entités qui seront par la suite transformées en tables. DbSet est une collection d'entités, donc généralement va prendre le nom de l'entité au pluriel. Un DbSet représente une image d'une table. DbContext est une partie importante de l'EF Core. C'est la classe qui est responsable de l'interaction avec la base de données.

4. Créer une classe PSContext dans le projet PS.Data
5. Ajouter l'héritage de DbContext
6. Créer le constructeur par défaut de la classe PSContext
7. Redéfinir la méthode OnConfiguring

Le point de départ de toutes les DbContext configurations est DbContextOptionsBuilder . Il existe trois façons d'utiliser ce générateur :

- Dans la méthode OnConfiguring
- Dans AddDbContext et méthodes associées
- Construit explicitement avec new

8. Ajouter la chaîne de connexion

Chaque DbContext instance doit être configurée pour utiliser un et un seul fournisseur de base de données. (Les différentes instances d'un DbContext sous-type peuvent être utilisées avec différents fournisseurs de bases de données, mais une seule instance ne doit en utiliser qu'une seule.) Un fournisseur de base de données est configuré à l'aide d'un Use\* appel spécifique. Dans notre cas, on va utiliser le fournisseur de base de données SQL Server. Ces Use\* méthodes sont des méthodes d'extension implémentées par le fournisseur de base de données.

9. Installer EntityFrameworkCore.SqlServer

Afin que la méthode d'extension Use\* puisse être utilisée, le package NuGet du fournisseur de base de données SQL Server doit être installé.

10. Ajouter les DBSets de toutes les entités.

```

namespace PS.Data
{
    public class PSContext : DbContext 5
    {
        public PSContext() 6
        {
            Database.EnsureCreated(); 11
        }
        public DbSet<Category> Categories { get; set; } 10
        public DbSet<Provider> Providers { get; set; }
        public DbSet<Product> Products { get; set; }
        public DbSet<Biological> Biologicals { get; set; }
        public DbSet<Chemical> Chemicals { get; set; }
        protected override void OnConfiguring(DbContextOptionsBuilder 7
optionsBuilder)
        {
            optionsBuilder.UseSqlServer(@"Data
Source=(localdb)\mssqllocaldb;Initial Catalog=ProductStoreDB;Integrated 8
Security=true");
            base.OnConfiguring(optionsBuilder);
        }
    }
}

```

### **Partie 3.3 : Génération de la BD**

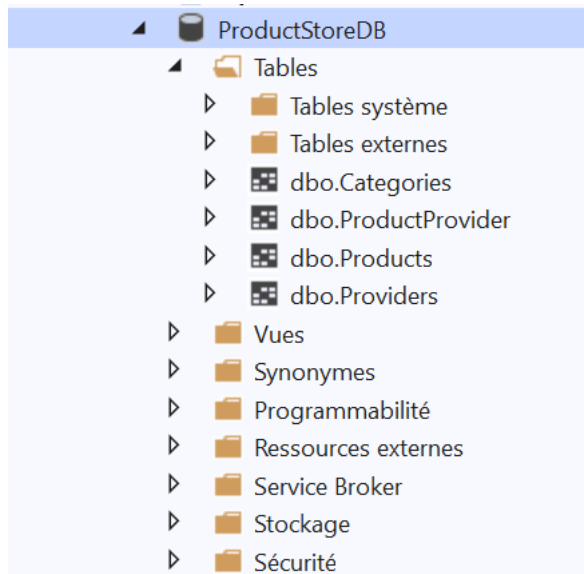
11. Appeler la méthode Database.EnsureCreated() dans le constructeur par défaut de la classe PSContext.
12. Faire une opération d'ajout dans l'application console et exécuter afin de générer la base de données.

```

namespace PS.Console
{
    class Program
    {
        static void Main(string[] args)
        {
            Scenario1();
        }
        static void Scenario1()
        {
            using (var context = new PSContext())
            {
                //context.Database.EnsureCreated();
                //Create
                System.Console.WriteLine("Create");
                //Instancier un objet Product
                Product P = new Product { Name = "Prod1", DateProd = DateTime.Now };
                //Ajouter l'objet au DBSET
                context.Products.Add(P);
                //Persister les données
                context.SaveChanges();
            }
        }
    }
}

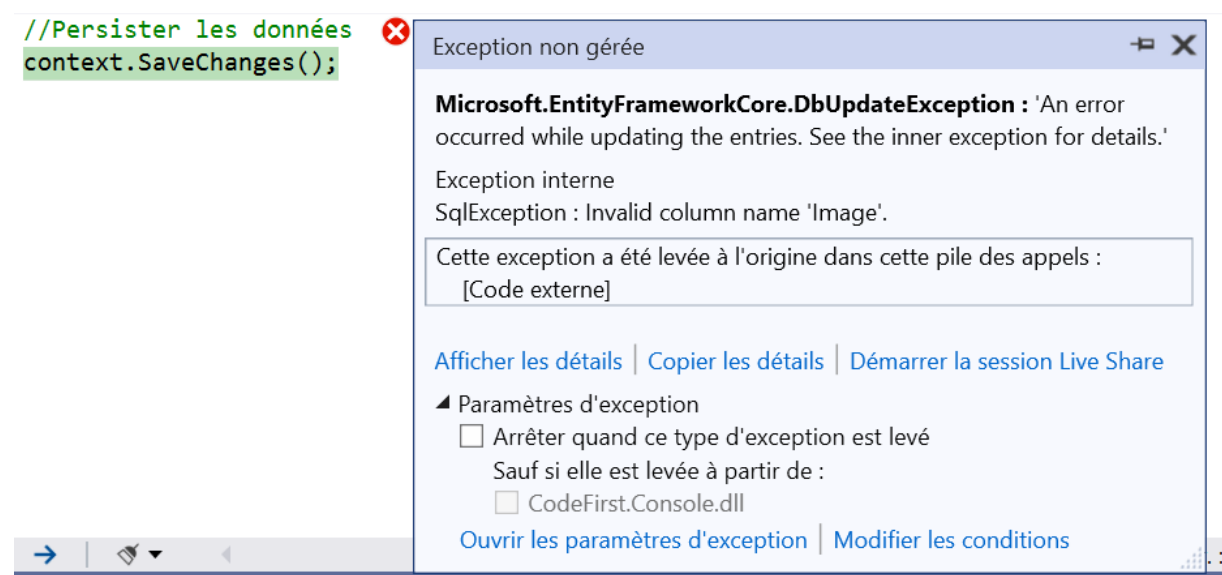
```

Le nom de la base de données de la chaîne de connexion est ProductStoreDB. EF Core CodeFirst va créer une nouvelle base de données nommée ProductStoreDB.



### Partie 3.4 : Migrations

13. Ajouter la propriété `public string Image { get; set; }` à l'entité Product puis lancer l'application. L'application nous renvoie cette exception:

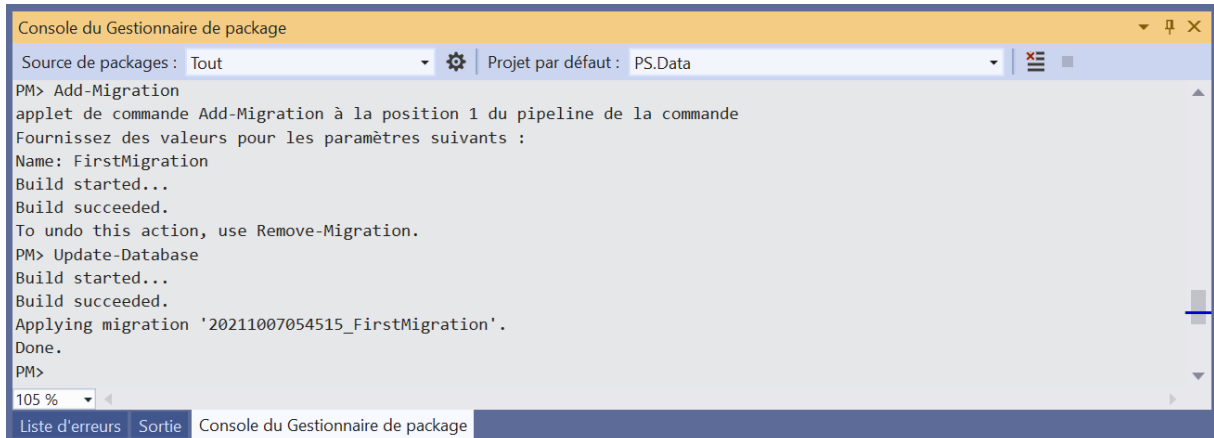


EF Core utilise un outil de migration qui met à jour automatiquement le schéma de la base de données lorsque votre modèle change sans perdre toutes les données existantes ou d'autres objets de base de données.

14. Installer EntityFrameworkCore.Design dans le projet PS.Console

15. Installer EntityFrameworkCore.Tools dans le projet PS.Data

16. Exécuter les commandes qui permettent d'ajouter une migration et de mettre à jour la base de données en ciblant le projet PS.Data



```
Console du Gestionnaire de package
Source de packages : Tout
Projet par défaut : PS.Data
PM> Add-Migration
applet de commande Add-Migration à la position 1 du pipeline de la commande
Fournissez des valeurs pour les paramètres suivants :
Name: FirstMigration
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
PM> Update-Database
Build started...
Build succeeded.
Applying migration '20211007054515_FirstMigration'.
Done.
PM>
```

EFCore Code-First dispose de deux commandes pour faire la migration:

**Add-Migration:** pour enregistrer les modifications que vous avez apportées à vos entités.

**Update-Database:** pour appliquer les modifications en attente à la base de données basée sur la dernière version créée avec la commande "Add-Migration"

La première exécution de la commande Add-Migration va ajouter un dossier Migrations à notre projet PS.Data. Ce nouveau dossier contient deux fichiers :

La classe `PSContextModelSnapshot`: Représente un instantané du modèle actuel. Cette classe est utilisée pour déterminer ce qui a changé lors de la création d'une nouvelle migration afin de pouvoir mettre la base de données à jour avec le modèle. Cette classe est ajoutée au dossier Migrations lors de la création de la première migration et mise à jour à chaque migration ultérieure.

Une classe de migration: Le code dans cette migration représente les objets qui sont créés dans la base de données. Cette classe contient une méthode `Up()` qui contient des instructions pour la création et une autre méthode `Down()` pour la suppression.

17. Changer le nom de la propriété public string `Image {get; set;}` à `public string ImageName {get; set;}`. Ajouter une migration nommée `ModifyNameImage` et mettre à jour la base de données.

Après la modification du nom de la propriété dans la classe `Product`, on doit exécuter ces deux commandes pour appliquer cette modification dans la table `Products` de notre base de données:

**Add-Migration ModifyNameImage:** L'exécution de la commande Add-Migration enregistre les modifications que vous avez apportées à la table `Product`. Une classe nommée `ModifyNameImage` a été générée automatiquement dans le dossier Migrations qui contient le code suivant:

## Update-Database

```
namespace PS.Data.Migrations
{
    1 référence
    public partial class MigModifyNameImage : Migration
    {
        0 références
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.RenameColumn(
                name: "Image",
                table: "Products",
                newName: "ImageName");
        }

        0 références
        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.RenameColumn(
                name: "ImageName",
                table: "Products",
                newName: "Image");
        }
    }
}
```

18. Restaurer le schéma de la base de données à l'état précédent.

Il suffit d'exécuter ces deux commandes sachant que FirstMigration est le nom que nous avons choisi pour notre première migration:

```
Add-Migration -Migration FirstMigration
```

```
Update-Database
```

### **Partie 3.5 : Types d'entité détenus**

19. Ajouter le type détenu "Address" dans le projet PS.Domain.

EF Core permet de modéliser des types d'entités qui peuvent uniquement apparaître dans les propriétés de navigation d'autres types d'entités. Il s'agit de types d'entités détenues. L'entité contenant un type détenu est son propriétaire.

Les entités détenues sont essentiellement une partie de l'entité propriétaire et ne peuvent pas exister sans elle.



Les types détenus ne sont pas inclus dans le modèle selon les conventions par défaut de EF Core. On doit utiliser la `OwnsOne` méthode dans `OnModelCreating` ou annoter le type avec l'attribut `Owned` pour configurer le type en tant que type détenu.

Dans notre exemple, `Address` est un type d'entité détenu. Il est utilisé comme propriété de l'entité `Chemical`.

Nous pouvons utiliser l'attribut `Owned`:

```
namespace PS.Domain
{
    [Owned]
    public class Address
    {
        public string StreetAddress { get; set; }
        public string City { get; set; }
    }
}
```

Il est également possible d'utiliser la méthode `OwnsOne` dans `OnModelCreating`. `OnModelCreating` est une méthode de la classe contexte qu'on peut substituer afin de configurer davantage le modèle qui a été découvert par convention à partir des types d'entités exposés dans les `DbSet`:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Chemical>().OwnsOne(p => p.MyAddress);
}
```

Si la propriété `Address` est privée dans l'entité `Chemical`, on peut utiliser cette version de la méthode `OwnsOne` :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Chemical>().OwnsOne(typeof(Address), "MyAddress");
}
```

20. Mettre à jour l'entité `Chemical` par le type détenu que nous venons de créer

```
namespace PS.Domain
{
    public class Chemical: Product
    {
        public string LabName { get; set; }
        public Address MyAddress { get; set; }
    }
}
```

### **Partie 3.6 : Annotations**

21. Ajouter la référence “System.ComponentModel.DataAnnotations” au projet PS.Domain
22. Ajouter les différentes annotations qui nous permettent de configurer les entités comme suit :

Dans la class Product:

- La propriété Name doit être :
  - obligatoire
  - de longueur maximale entrée par l'utilisateur 25
  - de longueur maximale dans la base de données 50
  - Un message d'erreur si les règles ne sont pas respectées
- La propriété Description doit être:
  - multiligne
- La propriété Price doit être:
  - de type devise
- La propriété Quantity doit être:
  - un entier positif
- La propriété DateProd doit être:
  - affichée sous le nom “Production Date”
  - de type date valide
- La propriété CategoryId doit être:
  - la clé étrangère de l'entité Category

```

public class Product:Concept
{
    [DataType(DataType.ImageUrl), Display(Name = "Image")]
    public string Image { get; set; }
    [Display(Name = "Production Date")]
    [DataType(DataType.DateTime)]
    public DateTime DateProd { get; set; }
    [DataType(DataType.MultilineText)]
    public string Description { get; set; }
    [Required(ErrorMessage = "Name Required")]
    [StringLength(25, ErrorMessage = "Must be less than 25 characters")]
    [MaxLength(50)]
    public string Name { get; set; }
    [DataType(DataType.Currency)]
    public double Price { get; set; }
    public int ProductId { get; set; }
    [Range(0, int.MaxValue)]
    public int Quantity { get; set; }
    //foreign Key properties
    public int? CategoryId { get; set; }
    //navigation properties
    [ForeignKey("CategoryId")] //useless in this case
    public Category MyCategory { get; set; }
    public IList<Provider> Providers { get; set; }
...}

```

Dans la class Provider :

- La propriété Id doit être:
  - clé primaire (Id est déjà une clé primaire par convention par défaut)
- La propriété Password doit être:
  - de type Password (masquée)
  - de longueur maximale entrée par l'utilisateur 8 caractères
  - obligatoire
- La propriété ConfirmPassword doit être:
  - obligatoire
  - ne doit pas être mappée dans la base de données
  - de type Password (masquée)
  - a la même valeur que la propriété Password
- La propriété Email doit être:
  - une adresse mail valide
  - obligatoire

```

public class Provider {
    [NotMapped]
    [Required(ErrorMessage = "Confirm Password is required")]
    [DataType(DataType.Password)]
    [Compare("Password")]
    public string ConfirmPassword { get; set; }
    public DateTime? DateCreated { get; set; }
    [Required, EmailAddress]
    public string Email { get; set; }
    [Key]    // optional !
    public int Id { get; set; }
    public bool IsApproved { get; set; }
    [Required(ErrorMessage = "Password is required")]
    [DataType(DataType.Password)]
    [MinLength(8)]
    public string Password { get; set; }
    public string UserName { get; set; }
    public virtual IList<Product> Products { get; set; }
...}

```

23. Mettre à jour la base de données en utilisant la migration
24. Lancer et tester en utilisant le projet console