Part 1 – Mise en place de la solution:

Créer une solution nommé "MyFinance" et y ajouter les projets suivants:

- "MyFinance.Domain", "MyFinance.Service", "MyFinance.Data" (3 Class Library)
- MyFinance.Console

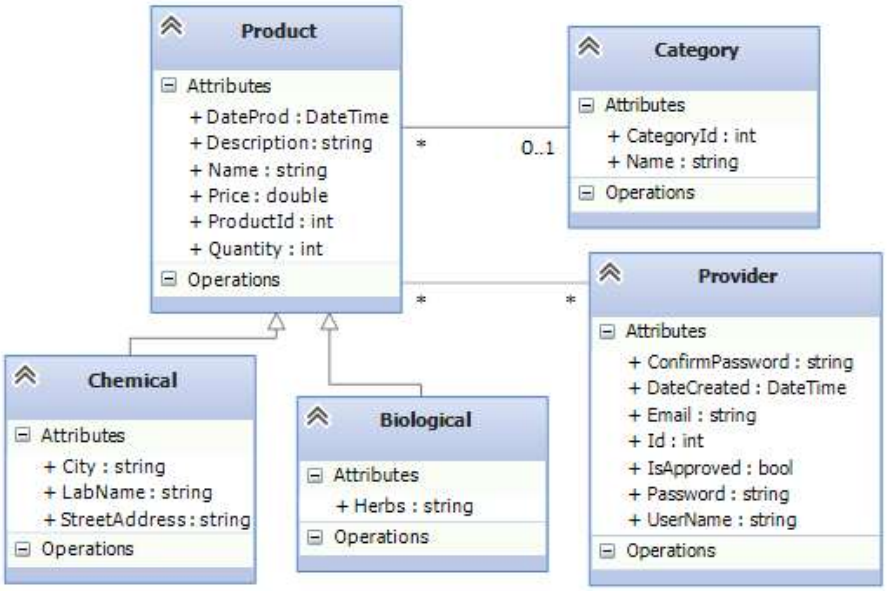
Ajouter les références entre les projets comme schématisé ci-bas

- Installer le package Entity Framework

Part 2 – Implémentation du context et des entités:

Etape 1 :

- Dans le projet "MyFinance.Domain" ajouter un nouveau dossier Entities
- Ajouter les classes schématisées dans le diagramme, ne pas oublier les propriétés de navigation et les propriétés représentant les clés étrangères.
- Décorer les propriétés de navigation par le mot clé Virtual



Etape 2 : Implémentation du Context

- Créer une classe MyFinanceContext dans le projet MyFinance.Data
- Ajouter l'heritage de DBContext
- Le constructeur de la classe context doit pointer vers une chaine de connexion présente dans le fichier App.Config
- Ajouter les DBSet nécessaires.

Etape 3 : Génération de la base de données

- Faite une opération d'ajout dans l'application console pur générer la base de donnée

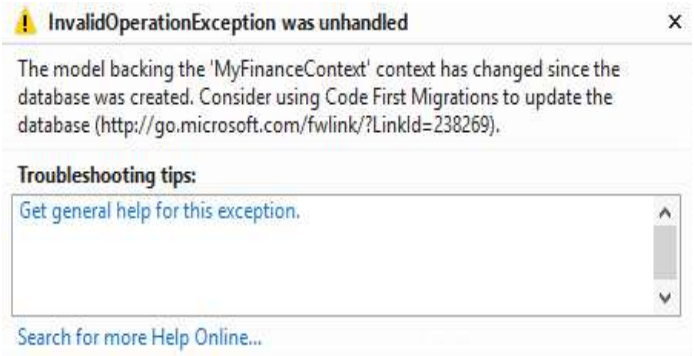
Part 3 –Mise à jour du modèle:

Etape 1 :-Let's try to add a property `public string Image { get; set; }` to the entity Product.

- Lancer l'application

L'application nous renvoie cette exception:

-That exception is thrown because the model changed and the the Entity Framework Code First doesn't know how it will behave. Therefore we have to tell the framework how it must behave.



- Ajouter la class ***MyFinanceContextInitialize*** dans le même fichier de la classe contexte : l'objectif de cette classe est d'informer EntityFramework de la stratégie à adopter lors de l'initialisation de l'application

  Nb : ***MyFinanceContextInitialize :*** `DropCreateDatabaseIfModelChanges`

- Redefinir la méthode **Seed** Hérité de la classe de base por ajouter un jeu de donnée lors de la réinitialisation de la base de donnée

- Pour mettre en marche la classe initializer ajouter ce code au constructeur de la classe **MyFinanceContext**.

```
Database.SetInitializer<MyFinanceContext>(new MyFinanceContextInitializer());
```

Etape 2 : -

Nous ne pouvons pas toujours réinitialiser la base de données au risque de perdre a chaque fois les informations qui s'y trouve.

Se pose alors une problématique. comment effectuer des modifications sur la structure de la base sans conséquence ?

- Activer la migration dans le projet DAL
- Changer le nom de la propriété `public string Image { get; set; }` à `public string ImageName { get; set; }`.
- Ajouter une migration nommé 'ModifyNameImage
- Aller à la classe "Configuration" dans le dossier "Migrations" et ajouter le code suivant dans la méthode Seed

```
context.Categories.AddOrUpdate(
          p => p.Name,  //Uniqueness property
          new Category { Name = "Medicament" },
          new Category { Name = "Vetement" },
          new Category { Name = "Meuble" }
        );
        context.SaveChanges();
```

- Exécuter la commande qui permet de mettre à jour la base de données

Part 4 – Complex Type:

Step 1 : Ajouter le **Complex Type** "Address" dans le dossier entities du projet Myfinance.Domain.

```
public class Address {
   public string StreetAddress { get; set; }
   public string City { get; set; }

}
```

Il faut suivre les règles suivantes pour qu'un type complexe soit créé par convention sinon il faut utiliser les DataAnnotaion /FluentAPI.

Conventional Complex Type Rules

1. Complex types have no key property.
2. Complex types can only contain primitive properties.
3. When used as a property in another class, the property must represent a single instance. It cannot be a collection type.

Etape 3 : Mettre à jour l'entité Chemical par le type complexe que nous venons de créer :

Partie 5 – Les annotations:

Etape 1 : Ajouter la référence "System.ComponentModel.DataAnnotions" au projet "MyFinance.Domain"

Etape 2 : Ajouter les différentes annotations qui nous permettent de configurer les entités cille suit :

- Dans la class ***Product***
  - La propriété Name doit être :
    - required
    - The user input string have the length 25 (max)
    - The property have length 50 (max)
    - An error message will be displayed if the rules are not respected.
  - La propriété Description doit être
    - Multiline
  - La propriété Price doit être
    - Currency
  - La propriété Quantity doit être
    - Positive integer
  - La propriété DateProd doit être

        - Displayed as “Production Date”
        - Valid Date
    - La propriété CategoryId doit être
        - The foreign Key property to the Category entity.

- Dans la class ***Provider*** :
    - La propriété Id doit être
        - Key (Id is already a primary key By Convention)
    - La propriété Password doit être
        - Password (hidden characters in the input)
        - Minimum length 8 characters
        - Required
    - La propriété ConfirmPassword doit être
        - Required
        - Not mapped in the database
        - Password
        - Same value as “Password” property
    - La propriété Email doit être
        - Email
        - Required

Etape 3 :

- Mettez à jour la base de données en utilisant la migration
- Lancer et tester en utilisant le projet console

Partie 6 –fluent API:

NB : les annotations et la configuration utilisant FluentApi peuvent cohabiter dans un même projet

Etape 1 : Ajouter un nouveau dossier “Configurations” dans le projet “MyFinance.Data”

Etape 2 : Ajouter la class CategoryConfiguration dans le dossier “Configurations”

- Le nom de la table correpondante à l’entité categorie dans la base de donnée doit etre ‘MyCategories’
- CategoryId est la clé primaire de la table
- La propriété name est obligatoire et a une longueur maximale de 50

Etape 3 : Ajouter la class ProductConfiguration dans le dossier “Configurations” folder :

- Configurer la relation **many-to-many** entre products et providers,
- Configurer l’héritage schématisé dans le diagramme de classe
- Configurer la relation **one-to-many** entre la class **Product** et **Category**.

Etape 4 : Configurer le type complexe Adress

Etape 5 : Mettre à jour le Context pour faire appel au class configuration que nous venons de créer

- override “OnModelCreating” }

Etape 6 : Mettre à jour la base de données en utilisant la migration

Part 7 – Une convention personnalisée :

Etape 1 : Créer une convention qui modifie le type généré par défaut Datetime à Datetime2.

Part 8 – Table porteuse de données :

Etape 1 : Dans le projet « MyFinance.Domain » ajouter la classe Client et la classe Facture schématisées dans le diagramme suivant :

0..1

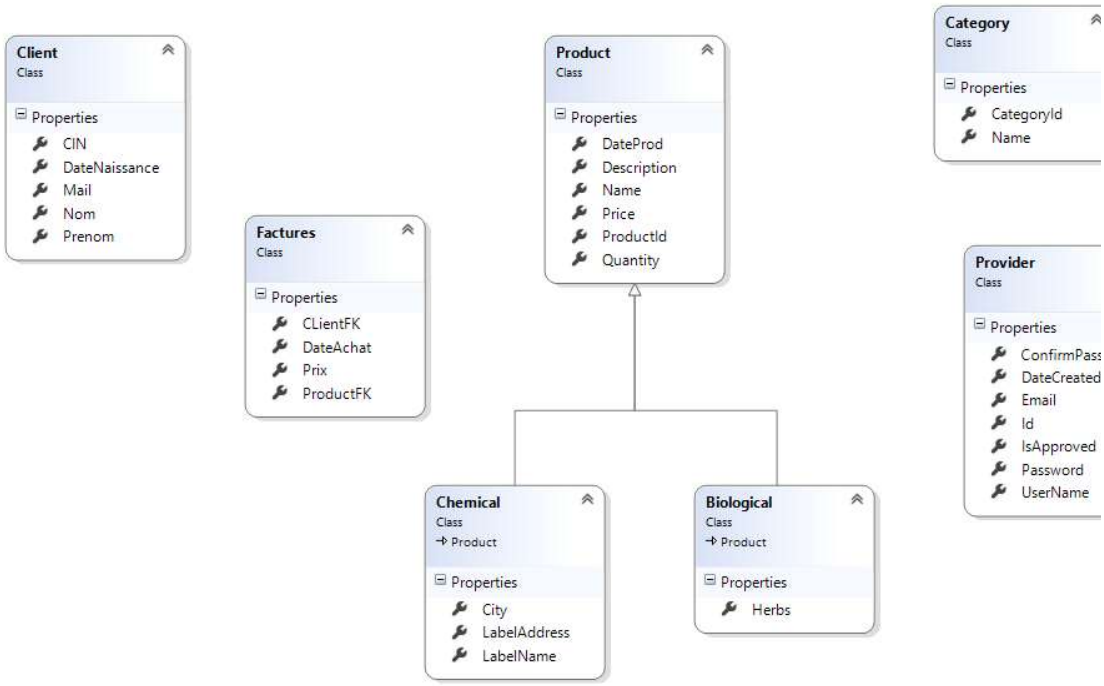
*

*

*

*

*



Etape 2 : Configurer la classe Facture.

Etape 3 : Ajouter les DbSet dans la classe MyFinanceContext.

Etape 3 : Mettez à jour la base de données.

Part 9 – Design Patterns :

Etape 1 : Créer des repositories génériques (implémenter repositoryBase)

Etape 2 : Créer la classe UnitOfWork