TP Individuel

Le but de ce TP individuel est de faire une bibliothèque de classes (.NET Standard) qui sera réutilisable dans le projet final du cours (TP collectif).

Le projet final permettra de faire une application Web ASP.NET Core afin de gérer de façon très simple un inventaire de magasin. La réalisation demandée dans ce TP est le module de gestion des données.

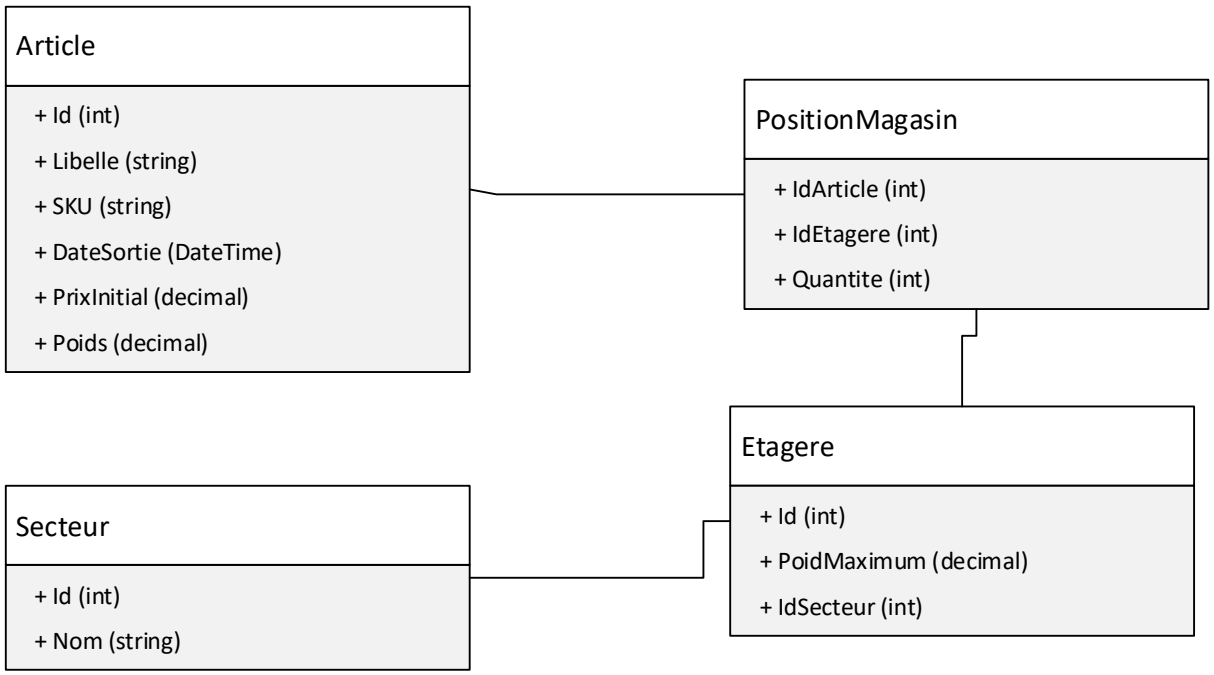
Votre production sera donc réutilisée dans le TP collectif.

Ce qui est attendu comme livrable pour correction :

- Le code source de la libraire
- Un moyen de tester le code source de la librairie (application console, web, projet de tests, etc.)

Vous êtes libre de choisir la ou les technologie(s) de persistance qui vous intéresse (XML, JSON, Entity Framework, autre…). *(note : EF fortement recommandé)*

Le modèle de cette librairie est le suivant (cardinalités volontairement omises du fait de la présence des clés étrangères) :



Le poids est exprimé en gramme

Information : lorsque vous allez coder le modèle de données, n'oubliez pas les collections d'objets qui ne sont pas sur ce schéma.

Le pattern repository (équivalent aux exercices) doit être implémenté au niveau « Article »

La librairie doit exposer les possibilités suivantes :

- Récupérer un article par son Id

- Récupérer la totalité des articles (de trois façons : global + par étagère + par secteur)
- Ajouter/modifier/supprimer un article
- Récupérer le prix moyen par secteur

Elle est le dernier rempart avant la base de données. A cet effet, il est nécessaire que la librairie empêche que des articles soient ajoutés sur une étagère et dépasse son poids maximum.

Exemple de données de test :

Secteur

Id	Nom
1	Secteur A
2	Secteur B

Etagère

Id	PoidsMaximum	IdSecteur
1	15000	1
2	17000	1
3	15500	2
4	12000	2

Article

Id	Libelle	SKU	DateSortie	PrixInitial	Poids
1	Tablette	123456	10/02/19	499.99	499
2	Telephone	789101	02/03/19	299.59	258
3	PC	147852	05/05/18	1566.23	1890
4	Bureau	258963	02/06/10	350	9500

Position

IdArticle	IdEtagere	Quantite
1	1	10
2	1	2
1	3	15

Informations :

- Pour pouvoir utiliser les décimaux, il faut formater les chiffres avec un point (« . ») comme séparateur de décimaux (ex : 1559.56) et ajouter la lettre « m » après le chiffre (ex : 1559.56m) pour que le compilateur C# initialise correctement la variable.
- Si vous faites des tests unitaires/intégration, il faut installer le package NuGet (« FluentAssertions ») sur votre projet de tests pour pouvoir utiliser les méthodes Should() et autres.

Chaque initiative supplémentaire sera reconnue à sa juste valeur dans la note.
Le fait que la solution ne compile n'est pas éliminatoire (mais c'est mieux quand même !).

Ce qui sera évalué :

- Le degré de remplissage des objectifs demandés
- La qualité et la propreté du code produit (bonne séparation, bonne organisation, code facilement lisible (nom de variables, méthodes, etc.)
- La facilité de tester le code pour l'évaluation

Idées de BONUS :

- Faire la séparation entre les abstractions (interfaces) et le implémentations (classes)
- Faire des tests unitaires/intégrations
- Faire plusieurs implémentations substituables

- Implémenter la logique CQS (Command Query Separation) : fournir une API de lecture distincte de celle d’écriture de façon explicite