Mission Principale #01

GÉNÉRATEUR DE CODE

Votre mission consiste à créer un outil en ligne de commande permettant de prendre un fichier JSON et de générer le code représentant ce modèle dans un langage de programmation spécifique.

Cet outil sera exécuté à la ligne de commande et aura des paramètres précis qui pourront être passés. Ces paramètres auront des comportements et des validations spécifiques.

Cet outil devra supporter la génération de code pour plusieurs langages possibles donc le C#. L'information sur les autres langages à supporter sera seulement annoncée plus tard.

Dans le but d'assurer l'efficacité du nouvel outil, il devra contenir de bons tests unitaires sur les parties importantes des fonctionnalités demandées.

L'utilisation de l'outil se fera de cette manière :

```
codegen -n Namespace.Models.MyClass -f test.json
```

Pour simplifier le développement de cet outil, vous devriez regarder les **Run/Debug Configurations** de votre IDE pour voir comment y passer les différents arguments :

- https://www.jetbrains.com/help/rider/Get_Started_with_Run_Debug_Configurations.html#step-1-understand-run-debug-configurations

Veuillez consulter les informations suivantes pour obtenir le détails des fonctionnalités requises.

Paramètres de la ligne de commande

Obligatoires

-n, --name <fully_qualified_class_name>
-f, --file <json_file_name>

Optionnels

-c, --code <lang_code>
Si non présent, par défaut, le code sera généré en C#.
-v, --verbose :
Si présent, affichera le code généré dans la console

```
codegen -n Namespace.Models.MyClass -f test.json

codegen --name Namespace.Models.MyClass --file test.json
```

L'exécution du programme ci-dessus va générer le fichier **MyClass.cs** avec l'espace de nom **NameSpace.Models** contenant le code C# généré.

```
codegen -n Namespace.Album -f test.json -c csharp -v
```

L'exécution du programme ci-dessus va générer le fichier **Album.cs** avec l'espace de nom **NameSpace** contenant le code C# généré et va également afficher le code généré dans la console.

```
codegen
```

L'exécution du programme ci-dessus affichera un message d'erreur indiquant comment utiliser le programme avec les différents paramètres.

Vous pouvez utiliser une librairie pour vous aider:

- https://github.com/dotnet/command-line-api

Validations requises

Il est nécessaire de valider tous les différents cas d'erreurs possibles. Par exemple, il faut s'assurer de valider que les paramètres obligatoires soient présents (voir **Paramètres de la ligne de commande**) et que le JSON contenu dans le fichier soit valide.

Il faudra un ou des messages d'erreur claires en conséquence de ces validations.

Génération de code

Le code généré doit également être formaté correctement et doit être utilisable dans un programme (e.g. il doit compiler). Vous devez utiliser des **List<T>** ou **Dictionary<TKey, TValue>** si une collection est nécessaire au lieu d'utiliser un tableau.

Étant donné que vous devez supporter plusieurs langages, il est recommandé de structurer son code selon les responsabilités des classes suivantes:

> Un **Parser** lira le fichier JSON pour récupérer tous les *tokens* représentant les différentes classes et variables dans une structure de données de votre choix. Ensuite, un **CodeGenerator** utilisera cette structure de données pour générer le code.
>
> *JSON Input ——> Class/Field Tokens ——> Generated Code*

Des informations additionnelles vous seront fournies sur des cas limites reliés à la génération du code. Toutefois, vous devez vous assurer qu'il n'y ait pas de duplication de champs ou de classes dans la génération du code.

Tests Unitaires

Les tests unitaires devront être bien faits et couvrir les fonctionnalités critiques de votre outil.

Pour vous aider à rendre les tests meilleurs et plus efficaces, vous avez le droit d'utiliser certaines librairies telles que:

- **Verify** : https://github.com/VerifyTests/Verify
- **Moq** : https://github.com/moq/moq

Informations additionnelles

Certains détails ne sont pas dévoilés pour le moment de manière volontaire et des informations additionnelles seront fournis dans les jours qui suivront. Toutefois, il est toujours possible de me poser des questions pour obtenir ces informations.

Évaluation

Cette mission compte pour 10% de la note finale et sera évalué sur le **Fonctionnement** et la **Qualité du code.**

Le fonctionnement et la qualité du code sont aussi importants l'un que l'autre. Un mauvais fonctionnement pourrait pénaliser la note pour la qualité du code et une mauvaise qualité du code pourra pénaliser la note pour le fonctionnement.

La remise du travail sera le **25 février avant 8h00** et la remise se fera par **GitHub classroom**. Il suffit simplement de pousser tous vos changements dans le dépôt Git qui sera créé lors de l'utilisation du lien GitHub classroom pour cette mission.

Attention : Si vous faites des *commits* après la remise du travail, ceci sera considéré comme un retard!