TP Docker – Films

Objectifs

Apprendre à :

- Utiliser **Dockerfile** et **docker-compose** pour créer et conteneuriser des projets Python personnels.
- Utiliser **Git** pour sauvegarder son projet et gérer les branches.
- Utiliser **Jupyter Notebook** pour documenter le projet.

Tâches

1. Création du repository

- Créer un **repository Git** contenant un fichier `README.md` décrivant le futur projet.
- Cloner ce repository en local.

2. Structure final du projet

```
tp/
├── docker-compose.yml
├── read/
│   ├── Dockerfile
│   ├── data/
│   │   └── movies.csv
│   ├── models/
│   │   └── Movie.py
│   └── read_data.py
└── write/
    ├── Dockerfile
    ├── data/
    │   └── movies.csv
    ├── exceptions/
    │   ├── InvalidAgeLimitException.py
    │   ├── InvalidGenreException.py
    │   ├── InvalidTitleException.py
    │   └── ...
    ├── models/
    │   └── Movie.py
    └── manip_data.py
```

3. Données

- Copier le fichier `movies.csv` dans les dossiers `read/data` et `write/data`.

4. Modèle

Dans le dossier `models`, créer une classe `Movie` contenant :

- L'attribut de classe :

- `id` (int) (il sera incrémenté dans le constructeur mais attention il commencera à 31 -> movie.csv)

- Les attributs :
  - `titre` (str)
  - `annee_production` (int)
  - `genre` (str)
  - `age_limite` (int)
- Une méthode spéciale `__str__` qui affichera les informations du film de manière lisible.
- Commiter votre travail.

5. Exceptions

Dans le dossier `exceptions`, créer une exception personnalisée pour chaque type d'erreur utilisateur, exemple :

- `InvalidTitleException`
- `InvalidYearException`
- `InvalidGenreException`
- `InvalidAgeLimitException`

Ces exceptions seront utilisées pour valider les entrées lors de la création ou la modification d'un film.

- Commiter votre travail.

6. Script d'écriture – `manip_data.py`

Dans le dossier `write`, créer le script `manip_data.py` qui aura pour rôle de manipuler les données du fichier CSV.

Fonctionnalités attendues (avec une méthode par action) :

1. **Ajouter un film**
   - Demander les informations nécessaires à l'utilisateur.
   - Créer un objet `Movie`.
   - Ajouter ce film dans le fichier `movies.csv`.
   - Commiter votre travail & merge.
2. **Modifier un film**
   - Demander l'id du film à modifier.
   - Redemander toutes les informations du film (remplacement complet).
   - Mettre à jour le fichier CSV.
   - Commiter votre travail & merge.
3. **Supprimer un film**
   - Supprimer un film sur la base de son id.
   -

Commiter votre travail & merge .

Si possible, chaque action doit être développée sur une **branche Git dédiée**, puis mergée dans la branche principale.

7. Script de lecture – `read_data.py`

Dans le dossier `read`, créer le script `read_data.py` qui permettra de lire et filtrer les films depuis le fichier CSV.

Fonctionnalités attendues :

1. Récupérer un film par son titre.
   - Commiter votre travail & merge.
2. Récupérer la liste des films ayant une limite d'âge inférieure ou égale à une valeur donnée.
   - Commiter votre travail & merge.
3. Récupérer la liste des films d'un certain genre.
   - Commiter votre travail & merge.
4. Récupérer la liste des films produits entre deux années données (année de début et année de fin).
   - Commiter votre travail & merge.

8. Docker

Dockerfile

Créer un `Dockerfile` pour chaque projet (`read` et `write`) :

- Utiliser une image de base **Python**.
- Copier le contenu du projet dans le répertoire `/home/votre-prenom`.
- Exécuter le script principal avec la commande :

```
CMD ["python", "script.py"]
```

Docker Compose

Créer un fichier `docker-compose.yml` permettant de :

- **Builder** les images `read` et `write`.
- Créer un **volume partagé** `data_volume`.
- Monter ce volume sur le dossier `data` des deux conteneurs (`read` et `write`) afin que les modifications effectuées dans `manip_data.py` soient visibles dans `read_data.py`.

9. Tests

Vérifier depuis le terminal Docker que toutes les commandes et fonctionnalités sont opérationnelles :

- Ajout, modification et suppression de films dans `write`.
- Lecture et filtrage corrects dans `read`.

10. Documentation Jupyter

Créer un **notebook Jupyter** qui documente le projet. Il devra contenir les sections suivantes :

1. **Récupération du projet**
    - Commandes Git pour cloner le repository.
2. **Lancement du projet sous Docker**
    - Commandes pour construire et exécuter les conteneurs.
3. **Utilisation des applications**
    - Description des fonctionnalités disponibles dans `read` et `write`.
    - Explication des entrées valides et invalides.
    - Exemples d'exécution.

Bonus

Pour que votre travail reste fonctionnel, créer une branche git pour chaque bonus.

Partie 1 - Transformation de `genre` en Enum

- Créer une classe Enum avec des genres valides (Action, Drame, etc.).
- Modifier la classe `Movie` pour n'accepter que des valeurs de cet Enum.
- Adapter les exceptions pour détecter les genres invalides.
- Commiter votre travail & merge.

Partie 2 - Les genres comme liste (multi-genres)

- Stocker les genres comme une liste dans la class Movie (`["Action", "Drame"]`). Dans le CSV, les genres seront séparés par `;`.
- Adapter les méthodes de lecture/écriture pour gérer ces listes.
- Commiter votre travail & merge.

Partie 3 - Classe `Acteur` + Fichier CSV dédié

- Créer un fichier `actors.csv` contenant les acteurs (`id`, `nom`, `prenom`, `age`).
- Créer une classe `Acteur` dans `models/Acteur.py`.
- Créer un fichier `movies_actors.csv` pour gérer la jointure film <-> acteur (`id_film`, `id_acteur`).
- Dans `manip_data.py`, effectuer les changement pour :
    - Ajouter des acteurs.
    - Associer un acteur à un film.
- Dans `read_data.py`, effectuer les changement pour :
    - Rechercher un film selon un acteur.
- Commiter votre travail & merge.