TP MVC : Application de Revue de Films avec Django en utilisant le Pattern MVC

Contexte

Dans ce TP, vous allez améliorer une application web simple permettant aux utilisateurs de consulter et de soumettre des critiques de films. Cette application utilisera le framework Django, qui met en œuvre le pattern MVC (Modèle-Vue-Contrôleur) sous la forme MVT (Modèle-Template-Vue) pour organiser le code de manière efficace.

Étapes du TP :

1. Installation de Django et Configuration de l'Environnement
2. Repérer les modèles
3. Repérer les vues
4. Repérer les templates
5. Réponse aux questions

Questions :

Il a été demandé à un développeur d'ajouter une vue pour proposer des recommendantion à un utilisateur, pour cela il a développé 'RecommendedMoviesView'.

- Qu'en pensez-vous en termes d'architecture ?
- Auriez-vous fait autrement ? Si oui, expliquer comment, essayer de le faire.

Correction

Le problème principal avec cette implémentation du pattern MVC dans Django devient évident dans la classe RecommendedMoviesView. Voici les problèmes architecturaux :

1. **Violation du Principe de Responsabilité Unique** :

- La vue contient une logique métier complexe pour calculer les recommandations
- Elle mélange la logique de présentation avec la logique de recommandation
- Le code de recommandation devrait être dans le modèle ou dans un service dédié

2. **Couplage Fort** :

- La logique de recommandation est étroitement couplée à la vue
- Toute modification de l'algorithme de recommandation nécessite de modifier la vue
- Le code n'est pas réutilisable dans d'autres contextes (API, commandes, etc.)

3. **Testabilité Réduite** :

- Tester la logique de recommandation nécessite de passer par la vue
- Impossible de tester l'algorithme de recommandation indépendamment
- Difficile de mocker les dépendances pour les tests unitaires

Une meilleure approche serait de :

1. Créer un service dédié pour la logique de recommandation :

```python
# services.py
class MovieRecommendationService:
    def get_recommendations(self, user, limit=10):
        # Logique de recommandation ici
        pass
```

2. Modifier la vue pour utiliser ce service :

```python
# views.py
class RecommendedMoviesView(LoginRequiredMixin, ListView):
    template_name = 'movies/recommended_movies.html'
    context_object_name = 'recommended_movies'

    def get_queryset(self):
        service = MovieRecommendationService()
        return service.get_recommendations(self.request.user)
```

Cette solution respecte mieux les principes SOLID et le pattern MVC en :

- Séparant les responsabilités

- Améliorant la testabilité
- Rendant le code plus modulaire et réutilisable
- Facilitant la maintenance et l'évolution du code