Unibet - Test technique

Versions utilisées

- Java 17 (dernière LTS)
- Maven 3.8.1 (intégré dans intelliJ)

Dépendances ajoutées

- Open api 1.6.2 afin d'avoir une interface pour tester les endpoints

Lancement du projet

- Configuration Spring boot classique sur intelliJ
- Sélectionner le JDK 17
- Sélectionner la classe main du projet : UnibetLiveTestApplication
- Adresse du swagger si besoin : localhost:8887/swagger-ui/index.html
- *Optionnel* : le délais d'exécution du batch se trouve dans le fichier application.properties : 'batch.rate.pay'

Architecture

Après avoir lu l'intégralité de l'énoncé et réfléchi au projet, j'ai choisi de partir sur une architecture classique avec trois couches : contrôleurs, services et repository. Les classes de display n'étaient pas forcément nécessaire mais je ne préfère pas retourner directement les entités. Si il est nécessaire de faire du traitement (modification ou ajout de champs) nous sommes plus libres de manipuler une classe seulement pour le display. J'ai hésité à faire une architecture hexagonale avec du DDD mais ici la couche métier n'était pas très complexe et j'ai supposé qu'il

est préférable de privilégier la performance dans le cas de mise à jour de côte, d'enregistrement de paris etc.

Pistes d'améliorations

Gérer l'état des paris perdus. Je n'ai traité dans le batch des paiements que la mise à jour des bets gagnants. Il faudrait traiter les bets perdus en même temps ou dans un autre batch éventuellement afin de ne pas perdre en performance sur celui-ci.

Faire un test d'intégration pour test le batch de paiement.

Ajouter une date de fin sur les événements pour savoir quand est ce qu'ils ne sont plus live et intégrer le filtre dans la requête en base de donnée directement pour ne pas remonter toute la base puis filtrer côté java.