Solution

Versions utilisées et décisions techniques

Versions :

- **Java**: 17
- **Apache Maven**

Décisions techniques et vue d'ensemble de l'architecture :

1. Utilisation des Data Transfer Objects (DTOs) :

J’ai crée des DTO pour chaque interaction avec le client. Cette approche optimise le flux de données et renforce la sécurité en garantissant que seules les données pertinentes sont communiquées, réduisant ainsi le risque d'exposition de données non intentionnelles.

2. Recommandation pour les identifiants :

Plutôt que d'utiliser de simples valeurs Long comme identifiants pour les événements et les sélections, il est recommandé d'utiliser des UUID. Cela rend difficile la devinette d'autres ID (J’ai pas eu le temps de changer les ID Long UUID) .

3. Hypothèses et ajouts :

- **Mise dans l'entité Bet** : j’ai ajouté le champs **Mise** dans Bet, cet information est important pour savoir le montant à payer en cas de gains .
- **Cotes dans l'entité Bet** : Il est crucial de capturer les cotes au moment de la prise du pari , j’ai donc ajoutée aussi un champ **Cote** dans Bet.
- **Direction des paris** : Le design actuel permet de parier sur le fait qu'une sélection soit vraie. Il ne supporte pas encore de parier contre une Sélection.
- **Identification du client** : L'ajout de l'ID client aux demandes post améliore le suivi.
- **Consistance de la réponse HTTP** : J'ai supposé qu'en appelant le point de terminaison post avec un mauvais selectionID, nous devrions renvoyer une réponse

HTTP 404.

4. Gestion d'exception améliorée :

Le `ExceptionHttpTranslator` et le `ExceptionType` ont été restructurés pour fournir une gestion d'exceptions plus nuancée et pour transmettre des codes de réponse HTTP personnalisés.

5. Mécanisme de verrouillage optimiste :

Pour protéger contre les paris sur des cotes obsolètes, en particulier lors de multiples opérations concurrentes, j'ai ajoutée un mechanism de verrouillage .

Les améliorations et les choix d'architecture détaillés ci-dessus visent à garantir un système de paris sûr, cohérent et efficace, même sous des charges de concurrence élevées.

Problèmes connus :

- Certains tests peuvent ne pas s'exécuter avec succès en raison d'incohérences dans les ID d'objet de la base de données à chaque lancement de l'application. Certains tests ayant des valeurs codées en dur, cette divergence peut entraîner des échecs. Les contraintes de temps ont empêché la rectification de ce problème, et il reste un domaine d'amélioration future.
- il manque des tests (unitaire , d'integration ….).

Next :

- Le code peut bien étre structurée en DDD,