kindred

Unibet France - Test technique

septembre 2022

1 Introduction

Ce document (*ennonce.pdf*) décrit les spécifications fonctionnelles et les contraintes techniques nécessaires à la réalisation d'une petite application web minimaliste de gestion de pari en direct (listing des événements, affichage des cotes et prise de pari) mettant en oeuvre la technologie *Java* avec le framework *Spring boot*

L'application, qu'on baptisera *UniLiveTest v1.0*, consistera à exposer quelques APIs et un batch afin de répondre au besoin spécifié plus loin dans ce document.

Ce document décrit le contenu du test, la structure du livrable ainsi que le fonctionnement de l'application.

Enfin, ce même document donne des consignes sur l'envoi de la solution et le délai de réalisation. Par ailleurs, il est conseillé de le lire attentivement jusqu'àu bout avant de commencer à écrire du code.

2 Règles de lectures

— **!Important** : considéré comme obligatoire. Cela peut être une contrainte technique ou règle fonctionnelle.

3 Contenu

— ennonce.pdf : le fichier courant
— src : code source de l'application backend à compléter et renvoyer.

4 Mission

Note 1 : L'application backend est d'ores et déjà fonctionnelle : la configurtion de l'app, la connexion à la base de données ainsi que la génération de la structure de données a déjà été faite (on utilise une base H2 mémoire).

Ceci étant dit, votre mission est de :

— Implémenter 3 APIs et un batch backend en respectant la spécification technique et fonctionnelle donnée.

— Rédiger un *README.md* et un document *solution.pdf* permettant respectivement de lancer l'application et de motiver les choix techniques

Toutes les méthodes à implémenter sont référencées dans le code avec le mot clé *TODO*. Vous pouvez donc facilement les retrouver.

5 Process d'évaluation

Nous nous assurerons que l'application se compile et s'exécute en se référant au README.md et en installant les versions que vous préconisez dans solution.pdf. Une fois lancée, l'évaluation sera répartie d'une façon équitable sur les éléments suivants :

— **Général**
  — Nombre de bugs identifiés (s'il y en a)

— **Qualité de code**
  — Clareté du code
  — Architecture générale
  — Gestion des exceptions
  — Utilisation des design patterns (s'il faut)
— **Qualité rédactionnelle**
  — Pertinence des commentaires dans le code
  — Clareté du README et motivations des choix
  — Qualité de l'écrit et de français.

6 Livrable

Le projet doit pouvoir être compilé avec la version *maven* **3.x** et exécuté avec une version java de votre choix (>= 8).

Le livrable doit être un :

— Fichier zip portant le nom **unitest-solution-prenom.zip** (ou tar.gz), **OU** :
— Une *URL* vers un dépôt *git* public ou privé (avec les instructions nécessaires pour y accéder)

Une fois décompressé/accédé, l'archive/le dépôt doit avoir la strucure suivante :

— ***src*** : le projet tel que vous l'avez récupéré avec évidemment les fichiers sources que vous aurez crées.

— ***README.md*** : ce fichier doit pouvoir dresser le plan d'exécution de l'application. Il s'agit simplement d'énumérer toutes les étapes nécessaire de la préconfiguration jusqu'à l'exéuction.

— ***solution.pdf*** : ce document décrit les versions utilisées : Java et tout ce qui nécessaire à la compilation du projet. Ceci nous est utile afin de lancer le projet. Ce même fichier pourrait aussi être utilisé pour motiver les choix techniques et/ou hypothèses que vous avez dû faire pendant le développement.

7 Spécification fonctionnelle

7.1 Description détaillée

Dans ce projet, nous allons simuler un événement (*event*) en direct (*live*) qui est un match de foot entre deux équipes (Unibet IT vs Real madrid) qui va durer 5 minutes. Cet événement va avoir plusieurs marchés (markets) par exemple *"Qui va gagner le match ?"* ou encore *"qui va marquer plus que Messi ?"* ouverts ou non pour la prise de pari (*bets*). Chaque marché va avoir au moins une sélection (selection), par exemple *"Unibet IT va gagner le match"* et qui sera représentée par un nom, un état (ouverte, fermée), une cote (*odd*) et un résultat final (gagnante / perdante). Enfin, une cote a une valeur entre 1.0 et X.Y

7.2 Diagramme de classe UML

En se basant sur ce qui vient d'être dit dans la section précédente, nous pouvons déduire le schèma suivant :

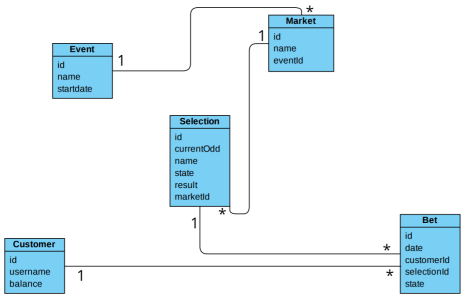

FIGURE 1 – Diagramme de classe simplifié

Dans le projet, nous utilisons *spring-data* pour initialiser le schèma ci-dessus de la base de données ainsi que l'ensemble des données de départ. Vous n'aurez donc rien à faire.

7.3 batchs et simulation de live

Nous fournissons des batchs de mise à jour de cotes et de fermeture de sélections, marchés et enfin de l'événement concerné. Durant les 5 minutes de l'événement, des selections vont changer de valeurs de cotes (à la hausse/baisse) puis se fermeront. Nous considérons qu'un marché est fermé quand toutes ses selections le sont. De la même manière, l'événement est considéré comme fini quand tous les marchés auront été fermés.

La classe *SelectionBatch* a donc deux batchs :

— **updateOddsRandomly** : mis à jour, toutes les 10 secondes, les cotes des selections non encore fermées.

— **closeSelectionRandomly** : ferme, toutes les minutes, des sélections d'une façon aléatoire dans le temps en mettant un résultat aux celles-ci (gagnante/perdante).

7.4 APIs à implementer

7.4.1 GET /api/v1/events

GET **/api/v1/events**
Récupère tous les événements de la base

Parameter

isLive Boolean | Optional | Filter uniquement sur les events live

Response application/json

200 ok

204 Pas de live

400 Request mal formée

500 Erreur serveur

7.4.2 GET /api/v1/events/{id}/selections

GET **/api/v1/events/{id}/selections ?{state}=state**
Récupère toutes les sélections d'un événément

Parameter

id Long | Obligatoire
state Enum | Optional | possible values : open, closed

Response application/json

200 ok

204 Pas de résultat

404 Evénement introuvable

400 Requête mal formée

500 Erreur serveur

7.4.3 POST /api/v1/bets/add

l'application démarre avec un utilisateur *unibest* et disposant d'une balance de 50e au démarrage. Aussi, aucune authentification n'est nécessaire pour placer le pari. Vous n'aurez rien à faire de ce côté là

POST **/api/v1/bets/add**
Enregistre un pari

Parameter

no parameter

Body application/json

```
{
    "selectionId" : "XXXXXXX",
    "cote" : XX.YY,
    "mise" : ZZ.TT
}
```

Response application/json

200	OK - Pari enregistré
400	Requête mal formée
409	Conflit, pari déjà en cours*
500	Erreur serveur
600	Balance insuffisante
601	Changement de cote
602	Selection fermée

* : Un même utilisatur ne peut prendre deux paris en même temps.

7.5 Batchs à implementer

7.5.1 payerLesParis

Après la fermeture d'une *selection*, celle-ci n'est plus disponible pour la prise de pari. Cependant, cette dernière va donner, selon son résultat, un gain ou une perte. L'enum *SelectionResult* se verra attribuer [1] deux valeurs *Win* et *Lost*

L'exercice est de récupérer tous les paris ouverts portant sur les *selections* fermées, et payer ou non le joueur selon le résultat de la sélection dont porte le pari.

!important :
— Une seule instance de ce batch doit être lancée à un instant t.
— Une pontentielle erreur ne doit pas arrêter l'ensemble des lignes à traiter.
— Le batch doit pouvoir logger à la fin l'ensemble de ligne traitées et la durée total d'exécution.

1. Le résultat d'une selection va être définie au moment de la fermeture de celle-ci et ce d'une façon aléatoire. Le batch de mise à jour des cotes s'en occupera. Vous n'aurez donc rien à faire.

— Le batch doit pouvoir être exécuté toutes les x ms secondes. Cette valeur est en fichier de propriété avec comme valeur par défaut 5000.

8 Consignes générales

8.1 Utilisation des libraires externes

Vous pouvez utiliser des libraires externes et changer le fichier *pom.xml* comme bon vous semble tant que ces choix sont motivés.

8.2 Hypothèses fonctionnelles/techniques

Dans le cas de blocage fonctionnel et/ou technique, vous pourrez faire des choix et des hypothèses pour avancer dans le test. Lors de l'évaluation, nous prendrons en compte ces remarques et on vous apportera nos réponses lors de l'entretien.

8.3 Délai de rendu

Il n'existe pas de limite de temps pour réaliser le projet. Le projet doit simplement être rendu **48 heures** avant le jour de l'entretien. Délai que nous éstimons nécessaire pour évaluer correctement le projet et de faire des retours pertinents.

En vous souhaitant un bon test.

L'équipe Unibet