Projet Informatique UE 5-2 Programmation JAVA V : Gestion de Joueurs et Parties pour un Jeu Vidéo en Ligne

Votre mission est de développer une application de gestion pour un jeu vidéo en ligne multi-joueurs. Cette application sera divisée en deux microservices Spring Boot :

1. **Service de Gestion des Joueurs** : Gérer les informations des joueurs, leur profil, leurs statistiques, etc.
2. **Service de Gestion des Parties** : Gérer les parties de jeu, le score, les performances, et la participation des joueurs.

Ces deux services devront communiquer pour mettre à jour les statistiques des joueurs en fonction des résultats de chaque partie.

Objectifs

1. Créer deux applications Spring Boot indépendantes (Gestion des Joueurs et Gestion des Parties).
2. Concevoir une base de données relationnelle pour chaque application avec gestion des relations.
3. Implémenter toutes les couches (Controller, Service, Repository,...) dans chaque application.
4. Mettre en place une communication entre les deux applications via une API REST.
5. Rédiger des tests unitaires et d'intégration pour garantir le bon fonctionnement.

Spécifications du Projet

1. **Application de Gestion des Joueurs**

But : Gérer les informations des joueurs, leurs amis et leur historique de statistiques.

- **Base de Données** :
  - **Joueur** : id, nom, pseudonyme, email, niveau, points_totaux
  - **Ami** : id, id_joueur, id_ami
- **Relations** :
  - Un **joueur** peut avoir plusieurs **amis** (One-to-Many).
- **Endpoints** :
  - Ajouter, modifier, supprimer un joueur.
  - Récupérer les informations et les statistiques d'un joueur.
  - Ajouter et gérer les amis d'un joueur.

2. **Application de Gestion des Parties**

But : Gérer les informations des parties et l'enregistrement des performances des joueurs.

- **Base de Données** :
    - **Partie** : id, date, type_partie, score_maximum, id_hote
    - **Participation** : id, id_partie, id_joueur, score, victoire
- **Relations** :
    - Une **partie** est jouée par plusieurs **joueurs** (Many-to-Many) via la table **Participation**.
    - Un **joueur** participe à plusieurs parties.
- **Endpoints** :
    - Créer, modifier et supprimer une partie.
    - Récupérer les informations d'une partie spécifique.
    - Enregistrer la participation d'un joueur et son score.

3. **Communication entre les Applications**

- Lorsqu'une partie est terminée dans le **Service de Gestion des Parties**, un appel REST est effectué vers le **Service de Gestion des Joueurs** pour mettre à jour les statistiques et le score total du joueur en fonction de ses performances.
- Le **Service de Gestion des Joueurs** maintient un historique des scores totaux, du nombre de victoires, et du niveau du joueur, et peut incrémenter son niveau selon le score accumulé.

4. **Structure des Couches**

Pour chaque application, créez les couches suivantes :

- **Controller** : Gère les requêtes HTTP.
- **DTO** : Représentation du modéle à envoyer
- **Service** : Contient la logique métier.
- **DAO** : Data Access Object, facilite l'accés au
- **Repository** : Gère l'accès aux données.
- **Entity**: Représentation du modéle des données

Scénarios d'Utilisation

1. **Service de Gestion des Joueurs**

- **Créer un joueur** : Ajouter un nouveau joueur avec son pseudonyme, son email et son niveau de départ.
- **Ajouter un ami** : Associer un joueur à un autre pour construire son réseau d'amis.
- **Récupérer le profil d'un joueur** : Accéder aux informations du joueur, à ses statistiques et à son réseau d'amis.

2. **Service de Gestion des Parties**

- **Créer une partie** : Ajouter une nouvelle partie en précisant la date, le type, et l'hôte.
- **Enregistrer la participation d'un joueur** : Ajouter un joueur à une partie avec son score.

- **Mettre à jour les statistiques des joueurs après une partie** : Une fois la partie terminée, les scores sont enregistrés et le service de gestion des joueurs est mis à jour via un appel REST.

Endpoints REST - Exemples

1. **Service de Gestion des Joueurs**
    - `POST /joueurs` : Ajouter un nouveau joueur.
    - `GET /joueurs/{id}` : Obtenir le profil et les statistiques d'un joueur.
    - `POST /joueurs/{id}/amis` : Ajouter un ami pour un joueur.
2. **Service de Gestion des Parties**
    - `POST /parties` : Créer une nouvelle partie.
    - `POST /parties/{id}/participations` : Enregistrer la participation d'un joueur avec son score.
    - `GET /parties/{id}` : Obtenir les informations et les participants d'une partie.

Livrables

1. **Fonctionnalités** : implémentées et fonctionnelles.
2. **Code Source** : Organisé, testé et documenté.
3. **Documentation** : Explications techniques, choix de conception, guide d'utilisation et guide d'installation **EN ANGLAIS**
4. **Tests** : Tests unitaires et d'intégration fonctionnel.
5. **Schéma de la Base de Données** : Modéle schématique de la base de données utilisé et Script de création de la Base de données.
6. **Base de données** : Le choix de la base de données est laissé libre, mais elle doit être relationnelle (MySQL, PostgreSQL, SQL Server, Oracle, etc.). L'utilisation de H2 n'est pas autorisée.

Avec ce projet, vous apprendrez à gérer des microservices pour un jeu en ligne, à organiser des relations complexes entre les entités de joueurs et parties, et à tester l'interaction entre deux services distincts.