Compte rendu de projet

LAVERGNE Sabrina et MAHDI MAHAMOUD Fatouma

Table des matières

Introduction - Étude du projet

Nous voulons, ici, développer une application de jeu de carte interactive basé sur le jeu connu du Jungle Speed.
Pour cela, nous implémentons une architecture web en trois couches distinctes : une couche de présentation pour l'interface utilisateur, une couche métier encapsulant la logique de jeu et une couche d'accès aux données pour la gestion de la persistance des données.
Les fonctionnalités clé de cette application sont la gestion des utilisateurs avec authentification, gestion de compte, persistance des données, la gestion des parties, en prenant en compte les réactions en temps réel des joueurs, et la gestion des scores et les calculs de statistiques.
Pour cela, des outils tels que JSP, JPA et AJAX. Nous nous intéressons, dans cette première partie, à la création des POJO et des DAO qui permettent la communication avec la base de données.

Les POJO (*Plain Old Java Object*) sont très utiles dans les projets liant une base de données car ils permettent de représenter les données de manière simple et indépendante de tout Framework.

Dans l'utilisation des POJO, on se base sur les classes Java standards pour modéliser les entités d'une application. Cette méthode permet de faciliter les opérations dans la base de données.

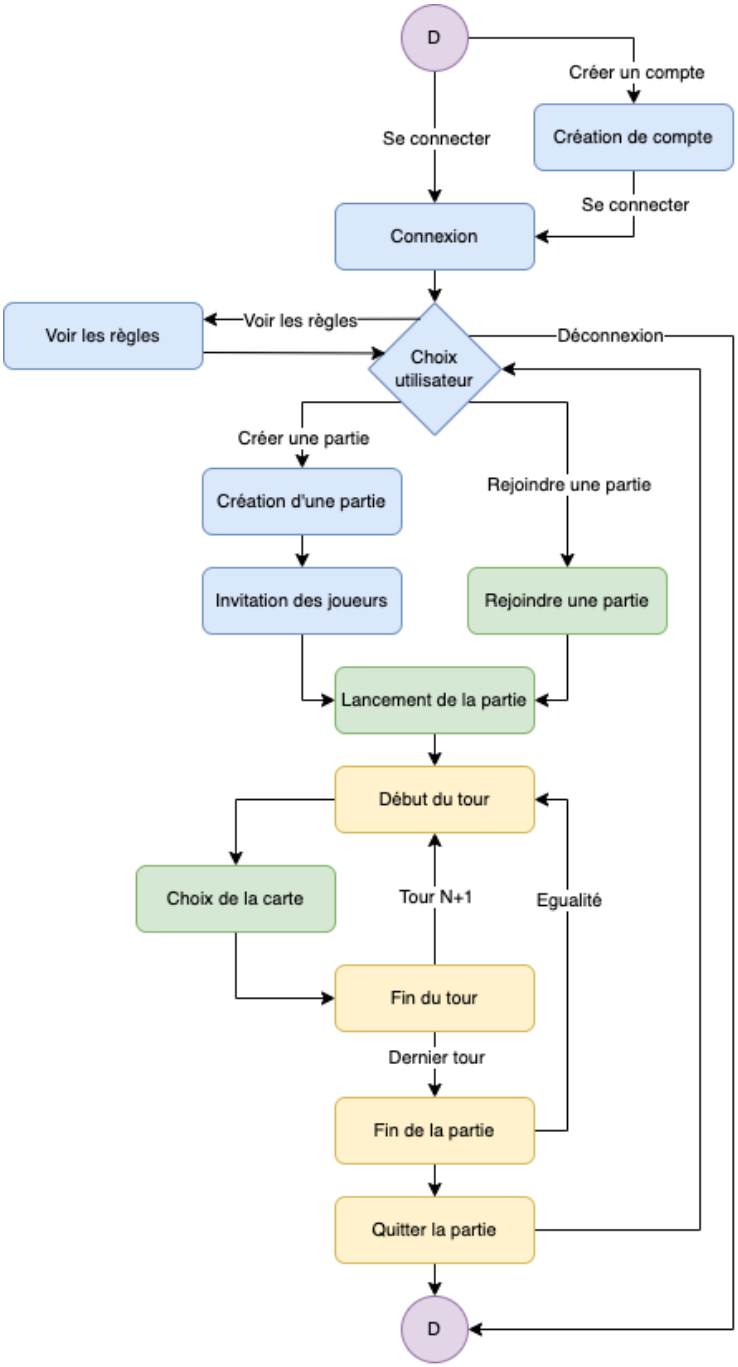
De la même manière, les DAO (*Data Access Object*) permettent de centraliser ces accès à la base. Ils permettent de fournir une interface pour les opérations CRUD (Création, Récupération, Modification et Suppression) sur les données.

Cela permet de séparer la logique d'accès aux données et la logique métier, facilitant ainsi la maintenant de l'application.

Pour la réalisation de ce projet, nous nous sommes penchées sur la réalisation d'un diagramme de cas d'utilisation qui nous permet de mieux représenter les différentes actions possibles sur notre application.

Les cas en bleu représentent les actions de l'utilisateur avant qu'il ne devienne un joueur (c'est-à-dire dans une partie), les cas en verts représentent les actions du joueur et les cas en jaunes représentent le déroulement de la partie.

L'application offre, à l'utilisateur, la possibilité de créer un compte, de se connecter, de voir les règles, de créer une partie ou d'en rejoindre une.



POJO (*Plain Old Java Object*)

Pour notre base de données, nous avons décidé d'implémenter trois tables et donc nous avons créé trois POJO :

- Utilisateur : Une personne connectée sur le site mais ne prenant pas encore part à une partie.
- Joueur : Un utilisateur une fois qu'il a créé ou rejoins une partie et qu'il est prêt à jouer.
- Partie : La gestion de la partie en cours.

Pour ce qui est des cartes, nous les gérons en dynamiques dans le fichier Java et nous n'avons, donc pas créé de lien avec la base de données.

POJO Utilisateur

Le POJO Utilisateur possède ces attributs :

- `id` : L'identifiant de l'utilisateur, généré avec une stratégie `IDENTITY`.

Cet attribut correspond à la clé primaire de la table Utilisateur dans la base de données, il est donc précédé des annotations JPA `@Id` et `@GeneratedValue(strategy = GenerationType.IDENTITY)`

- `pseudo` : Le pseudonyme choisi par l'utilisateur, une chaîne de texte non-nulle.
- `mdp` : Le mot de passe choisi par l'utilisateur, une chaîne de texte non-nulle.
- `age` : L'âge de l'utilisateur, un entier non-nul.
- `genre` : Le genre de l'utilisateur, une énumération avec les options : `MASCULIN, FEMININ, AUTRE.`
- `connecte` : Un booléen indiquant si l'utilisateur est présentement connecté ou pas.
- `enPartie` : Un booléen indiquant si l'utilisateur est présentement dans une partie ou pas.
- `nbrPartieJouee` : Un entier représentant le nombre de parties jouées par l'utilisateur.
- `nbrVictoire` : Un entier représentant le nombre de parties gagnée par l'utilisateur

- `sommeScore` : Un entier récupérant le score total de l'utilisateur pour la dernière partie jouée et l'ajoute à la somme précédente pour récupérer le score total de toutes ses parties.
- `scoreMoyen` : Un double qui calcule la moyenne du score de l'utilisateur en récupérant la valeur de `sommeScore` divisé par la valeur de `nbrPartieJouee`.
- `nbrClicsTotal` : Un entier récupérant le nombre total de clics de l'utilisateur pour la dernière partie jouée et l'ajoute à la somme précédente pour récupérer le nombre de clic total de toutes ses parties.
- `nbrClicsReussis` : Pareillement, un entier récupérant les clics qui sont annotés comme "réussis", c'est-à-dire que l'utilisateur à choisis la bonne réponse.
- `nbrClicsRapides` : Pareillement, un entier récupérant, parmi tous les clics annotés comme "réussis", ceux qui ont été les plus rapides.
- `ratioClicsReussis` : Un double qui calcul le pourcentage de clics réussis sur la totalité des clics de l'utilisateur.
- `retioClicsRapides` : Un double qui calcul le pourcentage de clics les plus rapides sur la totalité des clics réussis de l'utilisateur.

Les attributs `ratioClicsReussis` et `ratioClicsRapides` ne sont pas implémentés dans la base de données et sont donc précédés de l'annotation JPA `@Transient`.

Nous implémentons également la fonction `toString()` qui nous permet de récupérer tous les attributs sous forme d'une chaîne de caractère.

POJO Joueur

Le POJO Joueur hérite de l'entité Utilisateur et récupère donc tous ses attributs et possède, en plus, ces attributs :

- `id` : L'identifiant du joueur, un entier non-nul

Comme dans le POJO Utilisateur, cet attribut correspond à la clé primaire de la table Utilisateur dans la base de données, il est donc précédé des annotations JPA `@Id` et `@GeneratedValue(strategy = GenerationType.IDENTITY)`

- `scorePartie` : Un entier qui récupère le score de chaque tour et l'ajoute à la somme précédente.
- `utilisateurId` : L'identifiant associé à l'utilisateur, permettant de faire la connexion entre les deux entités.
- `partieId` : L'identifiant associé à la partie dans laquelle se trouve le joueur, permettant de faire la connexion entre les deux entités.

Les attributs `utilisateurId` et `partieId` sont des clés étrangères, ils sont donc précédés des annotations JPA :

- `@OneToOne` et `@JoinColumn(name = "utilisateur_id")` pour utilisateurId
- `@ManyToOne` et `@JoinColumn(name = "partie_id")` pour partieId

Nous implémentons également la fonction `toString()` qui nous permet de récupérer tous les attributs sous forme d'une chaîne de caractère.

POJO Partie

Le POJO Partie possède ces attributs :

- `id` : L'identifiant de la partie, généré avec une stratégie `IDENTITY`.

Cet attribut correspond à la clé primaire de la table Partie dans la base de données, il est donc précédé des annotations JPA `@Id` et `@GeneratedValue(strategy = GenerationType.IDENTITY)`

- `createur` : L'identifiant associé à l'utilisateur qui a créé la partie, permettant de faire la connexion entre les deux entités.

Cet attribut est une clé étrangère et est donc précédé des annotations JPA `@OneToOne` et `@JoinColumn(name = "createur")`

- `Joueurs` : La liste de tous les joueurs qui sont présents dans la partie

Cet attribut est une liste de clés étrangère, il est précédé de l'annotation `@OneToMany(mappedBy = "partie_id", cascade = CascadeType.ALL)`, qui précise que la partie peut avoir plusieurs joueurs. Chaque joueur est récupéré en faisant correspondre l'identifiant de la partie avec leur attributs "`partie_id`"

- nbrCouleur : Un entier qui correspond à la quantité de couleurs de la partie (rouge, vert, bleu et jaune), indiqué par le créateur.
- nbrNombre : Un entier qui correspond à la quantité de nombre de la partie (de 1 à 10), indiqué par le créateur.
- timer : Un entier qui correspond à la durée du timer pour chaque tour, indiqué par le créateur.
- nbrTour : Un entier qui correspond au nombre de tours de la partie, indiqué par le créateur.
- difficulte : Le niveau de difficulté de la partie, une énumération avec les options : `SIMPLE, DIFFICILE.`
- statut : Le statut de la partie, une énumération avec les options : `EN_ATTENTE, EN_COURS, TERMINEE.`
- totalClicsReussis : Un entier récupérant le total des clics qui sont annotés comme "réussis".
- totalClicsRapides : Pareillement, un entier récupérant, parmi tous les clics annotés comme "réussis", ceux qui ont été les plus rapides.

Nous implémentons également la fonction `toString()` qui nous permet de récupérer tous les attributs sous forme d'une chaîne de caractère.

DAO (*Data Access Object*)

Notre approche DAO se compose de plusieurs fichiers :

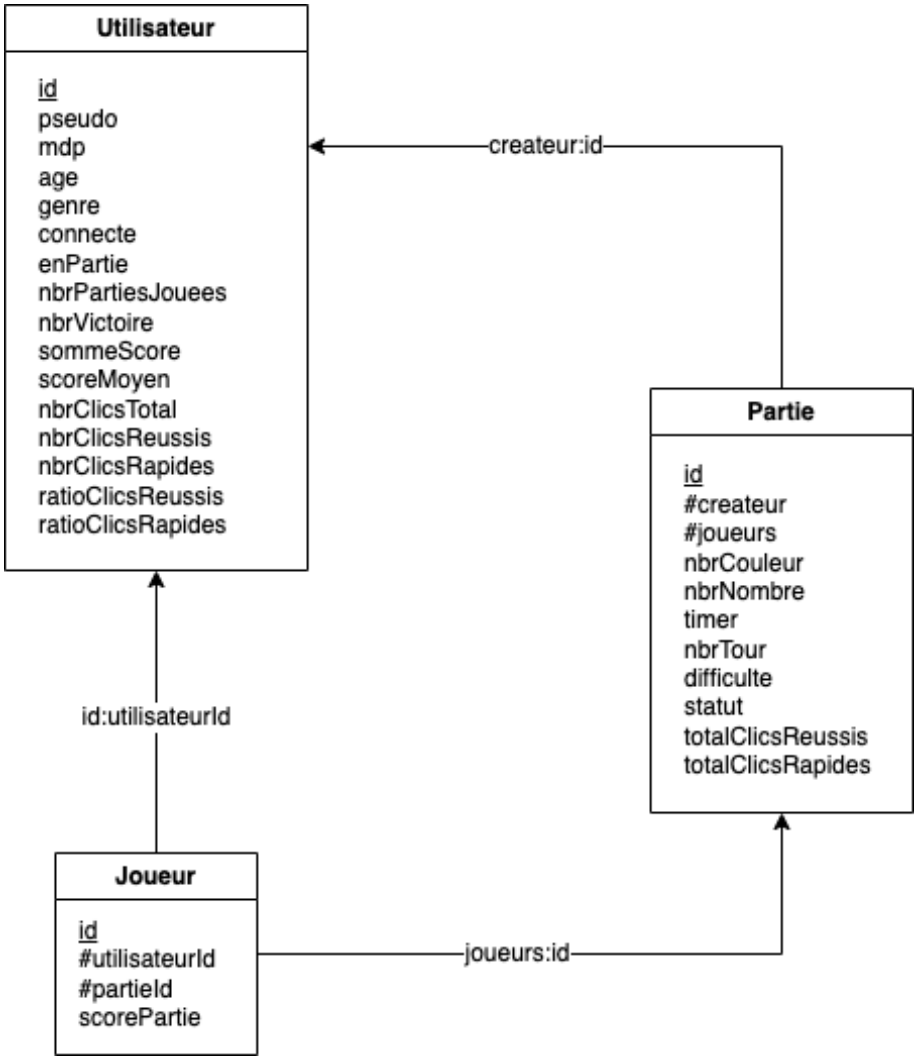
- `AbstractDAOFactory` : Ce fichier permet de définir un fichier DAO "Factory" spécifique en fonction du type de persistance énuméré dans le fichier `PersistenceKind`.
- `DAO` : Ce fichier permet de définir une classe DAO générique et abstraite qui sera ensuite appelée par les classe DAO spécifiques grâce au type d'entité de donnée `D`.
- `DAOException` : Ce fichier définit `DAOException` qui est utilisé pour signaler les problèmes spécifiques aux accès aux données via un DAO.
- `DAOFactory` : Ce fichier permet de définir la "Factory" qui est utilisée pour créer les instances de DAO spécifiques à chaque POJO.

Ainsi que les trois fichiers spécifiques aux POJO et le fichier `JPA_DAO_Factory` qui permet de faire la liaison entre les DAO spécifique et la DAO "Factory".

Pour chaque DAO spécifique, nous créons un objet de type `EntityManager` et récupérons l'instance grâce au constructeur. Puis nous créons les fonctions `findByID` et `findAll`, ainsi que les fonctions CRUD.

Base de données

Nous utilisons une base mySQL sur PhPMyAdmin sur laquelle nous créons trois tables selon nos trois POJO grâce à un fichier dump appelé `backup.sql`.
Voici notre schéma relationnel :



Conclusion

L'application des POJO et des DAO nous a permis de maîtriser des concepts clés tels que la modélisation indépendante des données et l'abstraction des accès à ces dernières. En résultat, nous avons acquis une base solide pour développer des applications Java évolutives et maintenables, tout en renforçant notre compréhension de l'importance d'une architecture logicielle bien conçue.