Rapport - Projet Application Web - Perudo

AUTEUR

Ignacio Arroyo Garza
Sami Zouari
Antonio
Mirela Majaru

2025-06-04

Contents

1 Introduction

Notre projet pour le cours d'application web consiste sur la recreation du jeu de société appelé Perudo en version en ligne et multijoueur, qui est un jeu de dés stratégique et de bluff. Le but du jeu est de deviner le nombre total de dés correspondant à une face spécifique que tous les joueurs ont sous leurs coupelles. Ce projet mettra en pratique les technologies vues en cours comme JPA et Spring Boot. Ce projet nous a aussi poussés à découvrir de nouvelles techniques comme l'utilisation des WebSockets, SPA et React.

Figure 1: Jeu Perudo

2 Règles du jeu

Voici un aperçu des règles de base :

- Préparation : Chaque joueur a cinq dés et une coupelle pour les cacher. Tous les joueurs lancent leurs dés sous leur coupelle et regardent secrètement le résultat.
- Enchères : Les joueurs prennent tour à tour pour faire une enchère sur le nombre total de dés affichant une certaine face (par exemple, "il y a cinq 4"). L'enchère doit être supérieure à la précédente, soit en augmentant le nombre de dés, soit en choisissant une face de dé plus élevée.
- Doute : Si un joueur pense qu'une enchère est trop élevée et ne peut pas être correcte, il peut crier "Dudo !" (ou "Perudo !"). Les dés de tous les joueurs sont alors révélés pour vérifier l'exactitude de l'enchère.
- Résolution : Si l'enchère était correcte ou trop basse, le joueur qui a douté perd un dé. Si l'enchère était incorrecte, le joueur qui a fait l'enchère perd un dé.

- Fin du jeu : Le jeu continue jusqu'à ce qu'un seul joueur reste avec au moins un dé, ce joueur est déclaré vainqueur.

Le jeu repose donc sur une combinaison de chance, de déduction et de psychologie pour bluffer les adversaires.

3 Analyse des Besoins

Dans cette section, nous détaillons les besoins fonctionnels et non fonctionnels de notre application web Perudo. L'objectif est de fournir une plateforme où les utilisateurs peuvent jouer à Perudo en ligne avec des fonctionnalités sociales et compétitives.

3.1 Besoins Fonctionnels

- **Gestion des Comptes Utilisateurs**
  - **Inscription et Connexion** : Les utilisateurs doivent pouvoir créer un compte en fournissant des informations de base telles que le nom d'utilisateur, l'adresse e-mail et un mot de passe. Ils doivent également pouvoir se connecter et se déconnecter de manière sécurisée.
  - **Profil Utilisateur** : Chaque utilisateur doit avoir un profil où il peut voir ses informations personnelles.

  Copiar

- **Système d'Amis**
  - **Ajout d'Amis** : Les utilisateurs doivent pouvoir ajouter des amis en utilisant un code ami unique. Cela permet de jouer avec des amis spécifiques et de voir leurs activités.
  - **Liste d'Amis** : Affichage d'une liste d'amis.
- **Boutique de Cosmétiques**
  - **Achat de Cosmétiques** : Les utilisateurs doivent pouvoir acheter des cosmétiques (comme des skins pour les dés) à partir d'une boutique en ligne intégrée.
  - **Utilisation des Cosmétiques** : Les cosmétiques achetés doivent être utilisables dans le jeu pour personnaliser l'expérience de jeu.
- **Gestion des Parties**
  - **Création et Rejoindre des Parties** : Les utilisateurs doivent pouvoir créer des parties de jeu et les rejoindre.

- **Système de Récompenses**
  - **Pièces et Trophées** : Les utilisateurs gagnent des pièces et des trophées en jouant et en remportant des parties. Les pièces peuvent être utilisées pour acheter des cosmétiques dans la boutique.
  - **Classement Global** : Un système de classement basé sur les trophées gagnés, permettant aux utilisateurs de voir leur position par rapport aux autres joueurs à l'échelle mondiale.

3.2 Besoins Non Fonctionnels

- **Performance** : L'application doit être capable de gérer plusieurs utilisateurs simultanément sans latence notable, assurant une expérience de jeu fluide.
- **Sécurité** : Les données des utilisateurs doivent être sécurisées, avec des mesures de protection contre les accès non autorisés.
- **Compatibilité** : L'application doit être compatible avec les principaux navigateurs web et dispositifs mobiles pour assurer une accessibilité maximale.
- **Expérience Utilisateur** : L'interface doit être intuitive et facile à utiliser, avec un design attrayant qui améliore l'expérience de jeu.
- **Maintenabilité** : Le code doit être bien documenté et structuré pour faciliter les mises à jour futures et l'ajout de nouvelles fonctionnalités.

En répondant à ces besoins, notre application vise à offrir une expérience de jeu enrichissante et compétitive, tout en assurant la sécurité et la satisfaction des utilisateurs.

4 Conception du Projet

L'architecture du projet Perudo repose sur une séparation claire entre le frontend (client) et le backend (serveur). Cette approche permet une meilleure maintenabilité, une évolutivité facilitée et une expérience utilisateur moderne.

4.1 Architecture Générale

- **Frontend** : Application React (JavaScript), Single Page Application (SPA), communication avec le backend via API REST (Axios) et WebSocket (STOMP) pour la gestion en temps réel des parties.
- **Backend** : Application Spring Boot (Java), expose des API REST, gère la logique métier, la persistance des données (JPA/Hibernate) et la sécurité.

- **Base de données** : PostgreSQL, utilisée pour stocker les utilisateurs, parties, historiques, cosmétiques, etc.
- **Communication** : Les WebSockets permettent la synchronisation en temps réel des états de jeu entre les joueurs.

4.2 Modélisation des Données

Les entités principales sont : `Player`, `Game`, `Bid`, `Product`, etc. Les relations sont gérées via JPA, avec des annotations pour éviter les problèmes de sérialisation (ex : `@JsonIdentityInfo`, `@JsonIgnore`).

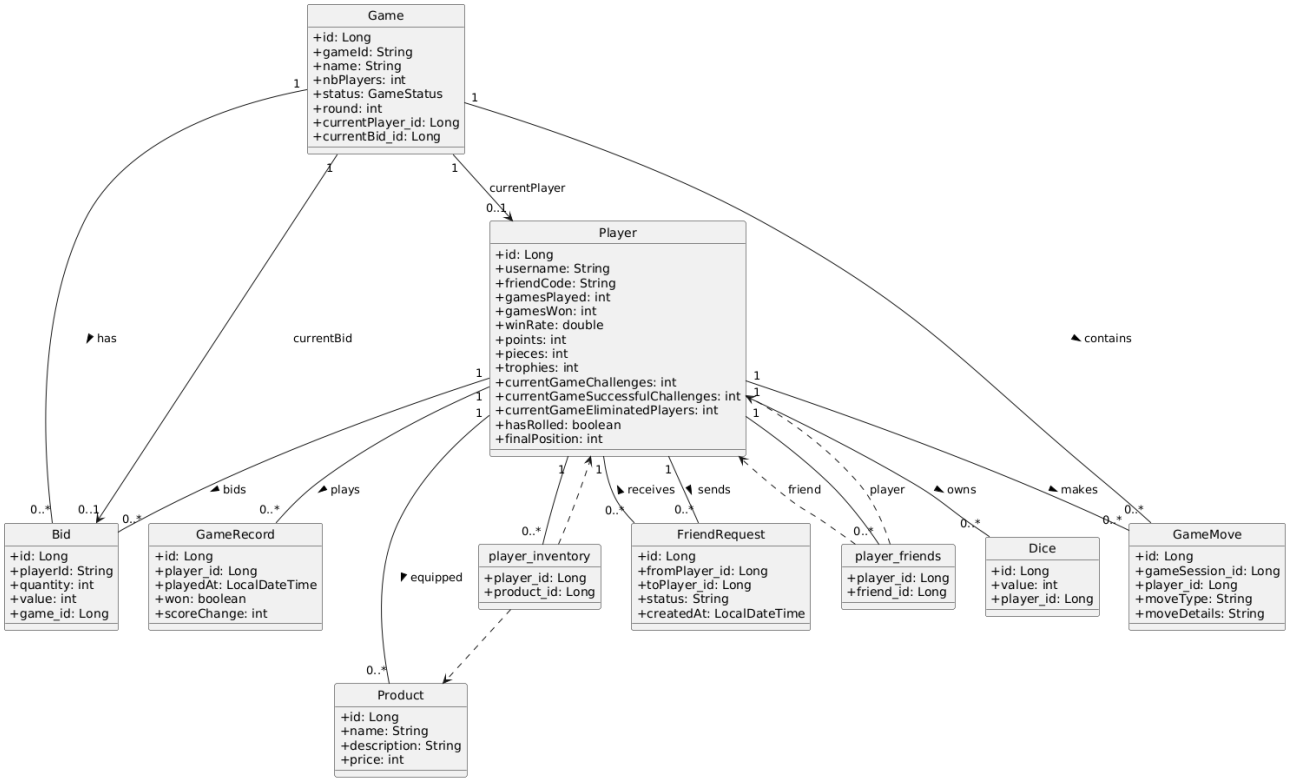

Figure 2: Diagram UML de la base de données fait avec [1]

Voici un aperçu rapide des principales entitiés utilisées et leurs relations:

1. **Player** (`@Entity @Table(name = "players")`)
   - **Attributs principaux** :
     - id: Long (PK)

  - username: String (unique)
  - friendCode: String (unique)
  - dice: List<Dice>
  - friends: Set<Player>
  - gameRecords: List<GameRecord>
  - currentGames: List<Game>
  - wonGames: List<Game>
  - pieces: int
  - trophies: int
  - points: int
  - inventory: List<Product>
  - equippedProduct: Product

2. **Game** (`@Entity @Table(name = "games")`)
   - **Attributs principaux** :
     - id: Long (PK)
     - gameId: String
     - name: String
     - nbPlayers: int
     - status: GameStatus (enum)
     - players: List<Player>
     - currentPlayer: Player
     - currentBid: Bid
     - round: int
     - turnSequence: List<Player>
3. **Dice** (`@Entity`)
   - **Attributs principaux** :
     - id: Long (PK)
     - value: int (1-6)
     - player: Player (ManyToOne)
4. **Bid** (`@Entity @Table(name = "bids")`)
   - **Attributs principaux** :
     - id: Long (PK)

  - playerId: String
  - quantity: int
  - value: int
  - game: Game (OneToOne)

5. **GameRecord** (`@Entity @Table(name = "game_records")`)
   - **Attributs principaux** :
     - id: Long (PK)
     - player: Player (ManyToOne)
     - playedAt: LocalDateTime
     - won: boolean
     - scoreChange: int
6. **GameMove** (`@Entity`)
   - **Attributs principaux** :
     - id: Long (PK)
     - gameSession: GameSession (ManyToOne)
     - player: Player (ManyToOne)
     - moveType: String
     - moveDetails: String
7. **Product** (`@Entity`)
   - **Attributs principaux** :
     - id: Long (PK)
     - name: String
     - description: String
     - price: int
     - players: List<Player>
8. **FriendRequest** (`@Entity`)
   - **Attributs principaux** :
     - id: Long (PK)
     - fromPlayer: Player (ManyToOne)
     - toPlayer: Player (ManyToOne)
     - status: String

  - createdAt: LocalDateTime

Les relations principales sont :

1. **Player** $\longleftrightarrow$ **Game** :
   - Un Player peut avoir plusieurs Games (ManyToMany)
   - Un Game a plusieurs Players
   - Un Game a un currentPlayer (ManyToOne)
2. **Player** $\longleftrightarrow$ **Dice** :
   - Un Player a plusieurs Dice (OneToMany)
   - Un Dice appartient à un Player (ManyToOne)
3. **Game** $\longleftrightarrow$ **Bid** :
   - Un Game a un currentBid (OneToOne)
4. **Player** $\longleftrightarrow$ **Player (Friends)** :
   - Un Player a plusieurs friends (ManyToMany auto-référentiel)
5. **Player** $\longleftrightarrow$ **Product** :
   - Un Player a plusieurs Products dans son inventory (ManyToMany)
   - Un Player a un equippedProduct (ManyToOne)
6. **Player** $\longleftrightarrow$ **GameRecord** :
   - Un Player a plusieurs GameRecords (OneToMany)
7. **FriendRequest** $\longleftrightarrow$ **Player** :
   - Un FriendRequest a un fromPlayer et toPlayer (ManyToOne)
   - Un Player peut avoir plusieurs FriendRequests

4.3 Choix Techniques

- **React** pour la rapidité de développement et la réactivité de l'interface.
- **Spring Boot** pour la robustesse, la sécurité et la facilité d'exposition d'API REST.
- **WebSocket** pour le temps réel (mise à jour des parties et interactions des parties).
- **JPA/Hibernate** pour la gestion de la persistance et des relations complexes.

4.4 Difficultés et Solutions

- **Boucles de sérialisation JSON** : Résolues par l'ajout d'annotations `@JsonIdentityInfo` et `@JsonIgnore` sur les entités JPA.
- **Synchronisation des données utilisateur** : Utilisation du localStorage côté frontend, synchronisé régulièrement avec le backend pour garantir l'affichage des pièces/trophées à jour.
- **Manque du player id lors du login** : Appel au backend pour récupérer les informations complètes du joueur et création d'un PlayerDTO pour passer uniquement les informations pertinentes.
- **Gestion des erreurs réseau** : Mise en place de messages d'erreur et d'états de chargement pour améliorer l'expérience utilisateur.

5 Interface Utilisateur

L'interface utilisateur a été conçue pour être intuitive et agréable, en s'inspirant des standards des jeux en ligne modernes.

- **Accueil** : Affichage du profil, des pièces, trophées, accès rapide aux parties et à la boutique.
- **Connexion/Inscription** : Formulaires simples et sécurisés.
- **Lobby et Parties** : Liste des parties en cours, possibilité de créer ou rejoindre une partie.
- **Jeu** : Affichage des dés, enchères, boutons des actions de jeu avec un pop-up à la fin de chaque manche pour dire si le joueur a bien fait de challenger ou pas. Adaptation des dés.
- **Boutique** : Achat de cosmétiques, visualisation de l'inventaire.
- **Profil et Amis** : Gestion du profil, ajout d'amis via un code unique, consultation du classement sur une page dédiée au leaderboard.

L'UI est responsive et compatible avec les principaux navigateurs.

6 Développement

Le développement s'est déroulé en plusieurs étapes :

1. **Initialisation du projet** : Création des squelettes React et Spring Boot, configuration du proxy et de la base de données (mise en place de l'environnement).

2. **Modélisation des entités** : Conception des classes JPA, gestion des relations et des contraintes.
3. **Développement des API REST** : Création des contrôleurs pour la gestion des utilisateurs, parties, amis, boutique, etc.
4. **Intégration WebSocket** : Mise en place de la communication en temps réel pour la synchronisation des parties.
5. **Développement frontend** : Création des composants React, gestion du state global (Context), intégration des appels API et WebSocket (STOMP).
6. **Gestion des erreurs et des cas limites** : Affichage d'états de chargement, gestion des erreurs réseau, feedback utilisateur.

7 Tests

- **Tests unitaires backend** : Utilisation de JUnit pour tester les services et contrôleurs Spring Boot.
- **Tests manuels frontend** : Vérification de tous les parcours utilisateur (inscription, connexion, partie, boutique, etc.).
- **Tests d'intégration** : Vérification de la communication entre le frontend et le backend (API REST et WebSocket).
- **Débogage** : Utilisation des outils de développement navigateur et des logs backend pour corriger les bugs (ex : problèmes de proxy, de sérialisation, etc.).

8 Déploiement

- **Backend** : Déployé en local sur un serveur Spring Boot, base de données PostgreSQL.
- **Frontend** : Déployé en local via le serveur de développement React (npm start), configuration du proxy pour rediriger les appels API.
- **Préparation au déploiement distant** : Le projet sera prêt, après correction d'un bug survenu lors de l'amélioration du frontend, à être déployé sur des plateformes cloud (Heroku, AWS, etc.) avec adaptation des variables d'environnement et de la reconfiguration du CORS.

9 Remarques

Le projet sera disponible depuis le GitHub : https://github.com/Ignacio-Arroyo/perudo_project. Tous les éléments mentionnés marchent correctement, le jeu avec les actions d'enchérir et de challenge, ainsi que la création et la participation à une partie, l'achat des produits depuis la boutique marchent dans l'avant-dernier push que nous avons fait (le dernier push a été une amélioration du frontend qui a provoqué un bug au niveau de la boutique). Donc pour tester le fonctionnement de la boutique, prenez la branche appelée test et vous pouvez faire le test. Pour tester le reste de l'application (jeu, ajout des amis, visualisation du leaderboard, du match history, créer un nouveau compte, se log in, etc.), ceci peut être fait depuis la branche main. Bug détecté au niveau du jeu, lors qu'une manche est finie (c'est-à-dire, un joueur a challengé), un pop-up apparaît affichant si le joueur a bien fait de challenger ou pas. Sur ce pop-up, il faut absolument appuyer sur le bouton d'OK, pour que le jeu progresse correctement. Si le bouton n'est pas appuyé pendant une durée, l'affichage du jeu ne se mettra pas à jour et ceci aura des répercussions sur la partie. Nous avons aussi gestionner la distribution des taches a l'aide d'un workspace Notion : Notion Workspace.

10 Conclusion

Ce projet nous a permis de mettre en pratique l'ensemble des compétences acquises en développement web moderne : conception d'API REST, gestion de la persistance, communication en temps réel, et développement d'interfaces réactives. Nous avons également appris à résoudre des problèmes complexes de sérialisation et de synchronisation des données. L'application Perudo offre une expérience de jeu complète, sociale et compétitive, et constitue une base solide pour de futures évolutions (tournois, IA, mobile, etc.).

References

[1] PlantUML Team. PlantUML - Visualize your UML diagrams as code. `https://www.plantuml.com`, 2024. Accessed: 2025-06-04.