CRÉATION D'UN GESTIONNAIRE DE TOURNOI

Léna ABEL
Anas BENMSATEF
Thibault LEBREUIL

2024/2025

Sommaire

1 Schéma

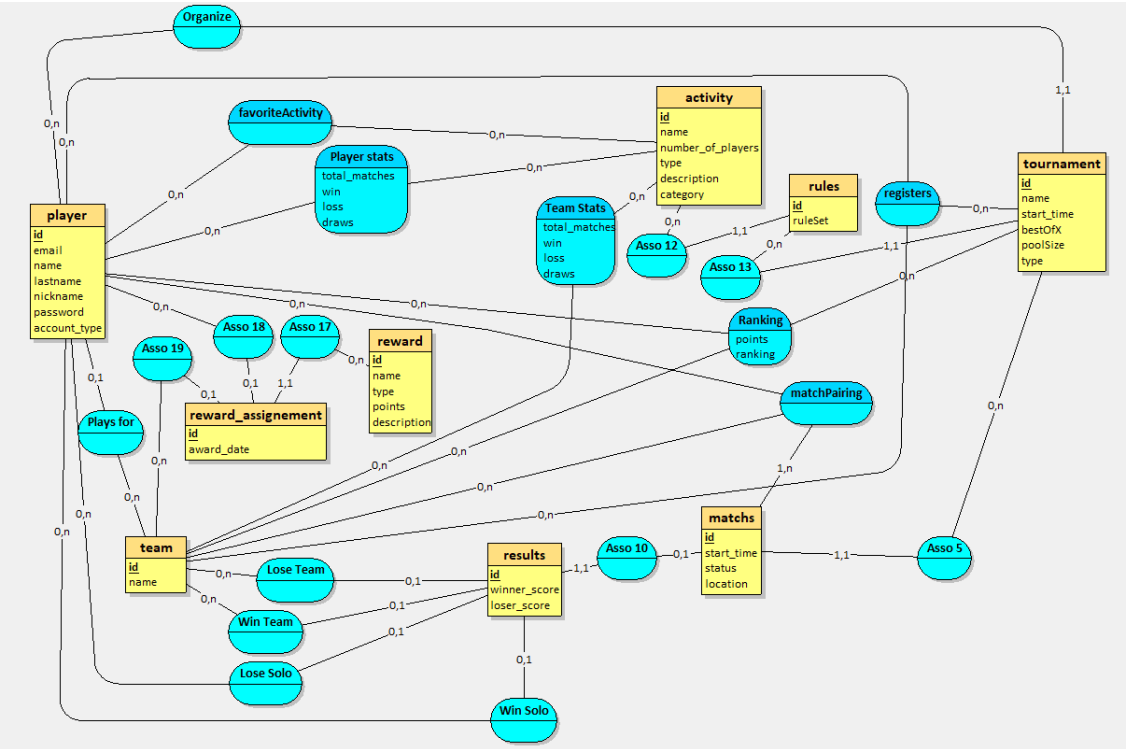

Figure 1: MCD de la base de données

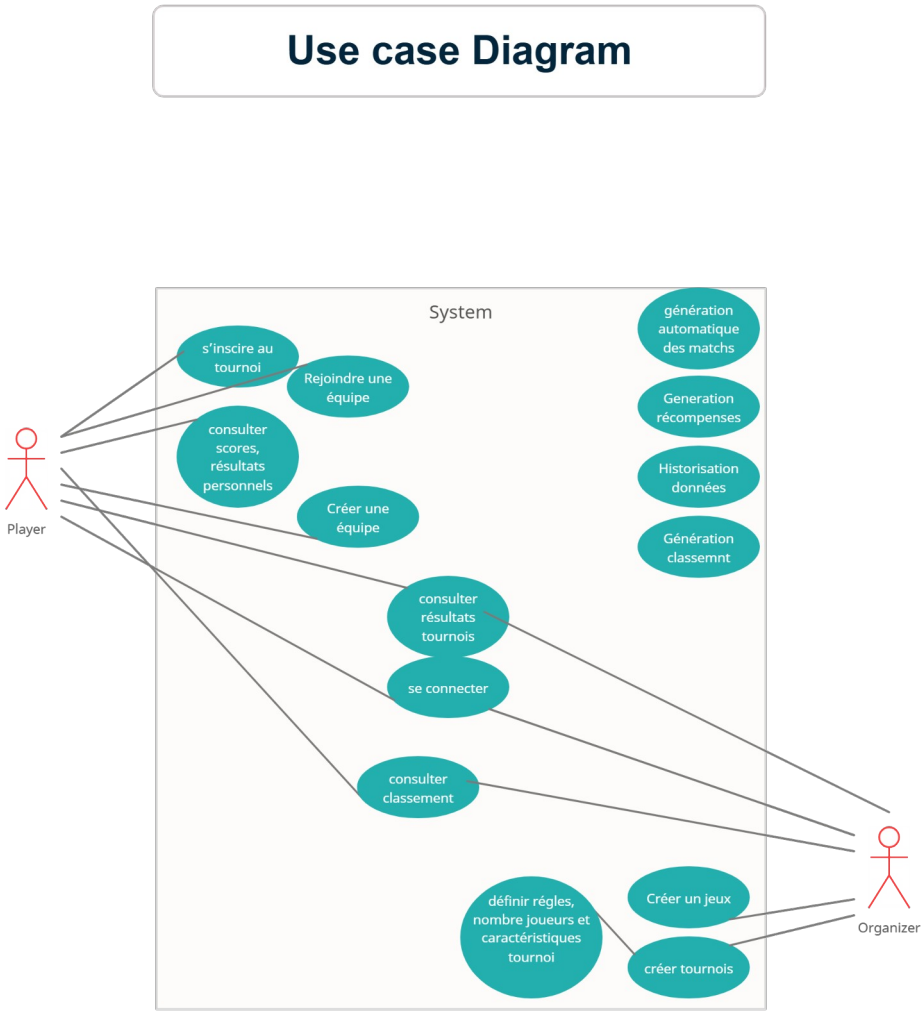

Figure 2: Diagramme de cas d'utilisation

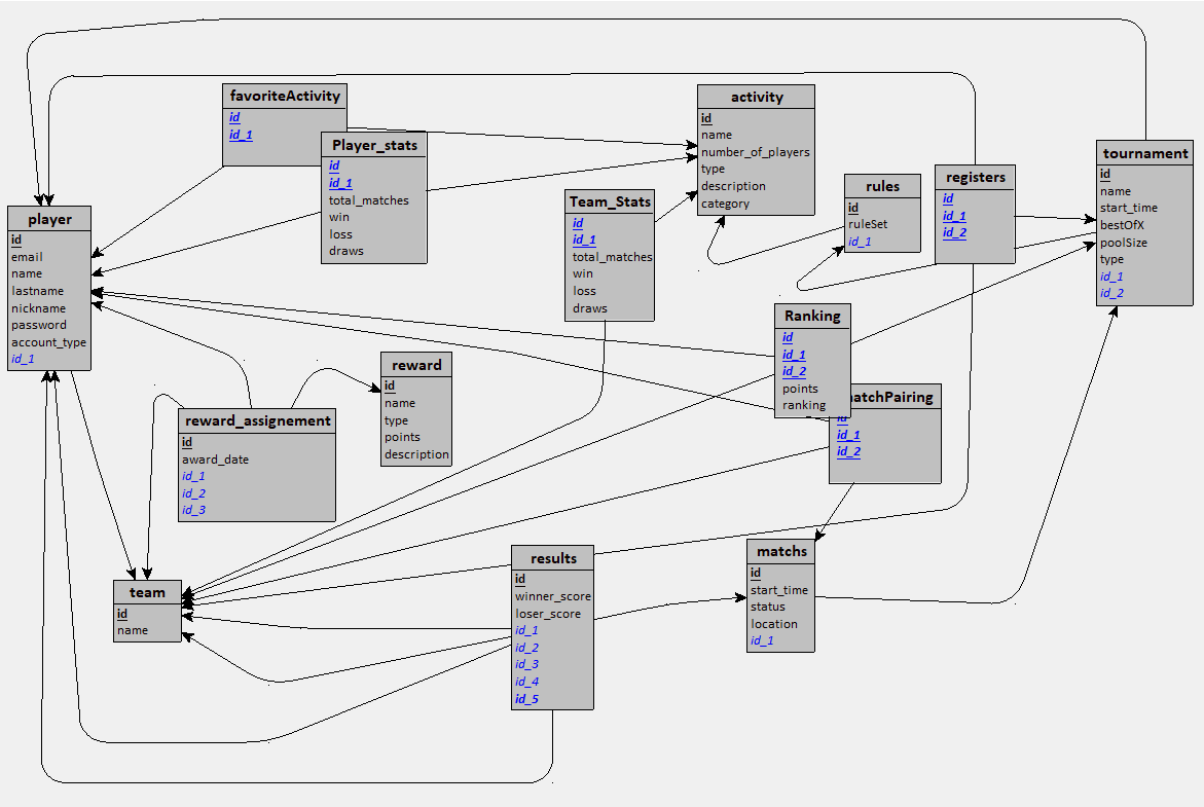

Figure 3: UML de la base de données

2 Base de données

La base de données est organisée autour de plusieurs tables principales, chacune ayant un rôle spécifique et des relations définies pour assurer la cohérence des données. Les tables clés incluent :

1. `activity` : Définit les différentes activités proposées dans les tournois.
2. `player` : Enregistre les informations sur les participants.
3. `team` : Gère les équipes composées de joueurs.
4. `tournament` : Structure les tournois et leurs caractéristiques.
5. `matchs` : Gère les rencontres organisées dans les tournois.
6. `results` : Archive les résultats des matchs.

7. `reward` : Administre les récompenses décernées aux gagnants.

2.1 Détails des Composants

- Activités (activity)
  - Cette table définit les types d'activités disponibles dans le système.
  - Chaque activité inclut :
    - Un identifiant unique (activity_id),
    - Un nom,
    - Un type (solo ou en équipe),
    - Une catégorie et une description optionnelle.
- Joueurs (player)
  - Cette table centralise les informations personnelles et d'identification des participants.
    - Les joueurs peuvent appartenir à des équipes et jouer des rôles spécifiques (player, organizer, ou admin).
    - Les mots de passe sont stockés sous forme hachée pour assurer la sécurité.
- Équipes (team)
  - Les équipes sont composées de joueurs qui participent aux tournois en groupe.
  - Une table supplémentaire, team_member, relie les joueurs à leurs équipes.
- Tournois (tournament)
  - Les tournois sont les entités principales autour desquelles le système est organisé.
  - Chaque tournoi est associé :

        - À des règles (rules),
        - À un type de tournoi (tournament_type),
        - Et à un format (format_type).
- Matchs (matchs)
    - Cette table organise les rencontres prévues dans les tournois.
    - Elle inclut des informations comme la date, l'heure et le statut du match.
- Résultats (results)
    - Les résultats des matchs sont enregistrés ici, incluant :
        - Les scores,
        - Les gagnants et perdants,
        - Les équipes ou joueurs impliqués.
- Récompenses (reward)
    - Cette table permet de gérer les prix remis aux gagnants, comme des trophées, des médailles ou des points.

2.2 Conception relationnelle

La base de données suit une approche relationnelle pour maintenir l'intégrité des données :

- Les clés primaires garantissent l'identification unique de chaque entrée.
- Les clés étrangères définissent des relations entre les tables (par exemple, un tournoi est lié à des règles spécifiques).
- Les contraintes, telles que les suppressions en cascade (`ON DELETE CASCADE`), assurent la cohérence des données lors des modifications.

2.3 Fonctionnalités Étendues

- Gestion des favoris
  - Les joueurs peuvent marquer leurs activités préférées via la table `favoriteactivity`.
- Statistiques
  - Des statistiques détaillées sur les performances des joueurs (`player_stats`) et des équipes (`team_stats`) sont suivies.
- Système de classement
  - Un classement global par tournoi est géré dans la table ranking, avec des points et des positions.
- Journaux et débogage
  - Une table `debug_log` permet de tracer les événements et les modifications, facilitant le débogage et l'audit.
- Récompenses
  - La table `reward_assignment` enregistre les récompenses attribuées, en prenant en charge les joueurs et les équipes.

3 API

L'objectif de ce projet étant d'apprendre les bases de SQL et savoir administrer une base de données, nous avons décidé d'utiliser et construire une API REST (Application Programming Interface).
Pour ce faire, nous avons décidé d'utiliser les technologies Node.js et Express.js.

Créer une API pour ce projet, nous a permis de renforcer la compréhension et la structure des bases relationnelles. Node.js dispose de bibliothèques utiles telles que mysql2, que nous avons utilisées pour développer l'API.
Aussi, l'utilisation d'une API à faciliter le débogage de certains problèmes,

notamment la conception de trigger, pour l'automatisation de création de tournois au format éliminatoire de type solo et team.

3.1 Model View Controller (MVC)

Pour structurer et organiser le projet, nous avons décidé d'adopter le modèle MVC. Ce modèle est composé de 3 couches:

1. **Modèle**:
    - Couche qui gère la logique ainsi que l'administration de données.

    Dans le projet, cette couche contient toutes les requêtes SQL qu'on a pu utiliser, ainsi que celles qu'on aurait pu utiliser par la suite.

Exemple:

```
// Créer un nouveau round
const createRound = (tournament_id, round_number) => {
    return new Promise((resolve, reject) => {
        db.query(
            "INSERT INTO tournament_round (tournament_id, round_number) VALUES (?, ?)",
            [tournament_id, round_number],
            (err, result) => {
                if (err) {
                    return reject(err);
                }
                resolve({ round_id: result.insertId, tournament_id, round_number });
            }
        );
    });
};
```

Figure 4: Requêtes pour créer un round

Dans le cadre du projet, nous avons implémenté le modèle de la table `tournament_round` (round) le CRUD(Create, Read, Update, Delete). Cela nous a permis de pratiquer et appliquer les requêtes SQL simples et complexes, renforçant

ainsi la compréhension des bases de données relationnelles et les interactions avec une API.

2. Vue:

Cette couche s'occupe de la présentation des données. Cela correspond donc à la partie visualisée par l'utilisateur. Cette partie était très importante dans le cadre du projet, car il fallait que l'interface soit à la fois conviviale mais aussi fonctionnelle, avec des fonctionnalités bien précises telles que la création de tournois, l'authentification, la consultation du classement et autres..

Exemple :

```
export default {
  name: 'TournamentsList',
  data() {
    return {
      filter: '',
      dateRange: [], // Initialize as an empty array for date range picker
      selectedGame: null,
      sortOption: null,
      fields: [
        { key: 'tournament_name', label: 'Name', sortable: true },
        { key: 'activity_name', label: 'Game', sortable: true },
        { key: 'tournament_start_time', label: 'Start Time', sortable: true },
      ],
      currentPage: 1,
      perPage: 10,
      loading: false,
      fetchError: null,
```

Ici la vue initialise la liste de tournois, pour ensuite récupérer les valeurs à travers les clés `tournament_name`, `activity_name` et `tournament_start_time`.

Ces valeurs seront donc récupérées et affichées dans l'interface, et donc accessibles aux utilisateurs.

Ces champs sont renommés `Name`, `Game` et `Start Time` pour simplifier leur lisibilité à l'utilisateur.

3. **Contrôleur**

La couche controller agit en tant qu'intermédiaire entre le modèle et la vue. Cela signifie qu'un contrôleur reçoit les informations provenant du modèle duquel il s'occupe (après les avoir appelés) et les transmet au format approprié à la vue.

Exemple :

```
// Mettre à jour un round par ID sans changer le round_id
const updateTournamentRound = (req, res) => {
    const { round_id } = req.params;
    const { tournament_id, round_number } = req.body;
    tournamentRoundModel.updateRound(round_id, tournament_id, round_number)
        .then(updatedRound => {
            if (!updatedRound) {
                return res.status(404).json({ error: 'Round not found' });
            }
            res.json(updatedRound);
        })
        .catch(err => {
            console.error('Error updating round:', err);
            res.status(500).json({ error: 'Error updating round' });
        });
};
```

Figure 5: Controller pour mettre à jour un tournoi

Ici, le contrôleur responsable du modèle vu précédemment, reçoit les informations `d'UPDATE` du modèle, puis les transmet à la vue, en lui spécifiant le nom de chaque élément envoyé.

4 Ajout : Sécurité et Gestion des Accès

Pour assurer la sécurité et le bon fonctionnement du système, plusieurs mesures et fonctionnalités ont été intégrées à la base de données et à l'API :

4.1 Sécurité des Données

- **Hachage des mots de passe :**
  - Tous les mots de passe des utilisateurs (joueurs, administrateurs, organisateurs) sont hachés à l'aide d'algorithmes sécurisés comme `bcrypt` avant d'être stockés dans la base de données. Cela garantit que les informations sensibles ne soient jamais stockées en clair.
- **Gestion des permissions :**
  - Les utilisateurs se voient attribuer des rôles spécifiques (exemple : admin, organizer, player) via une colonne dédiée dans la table player. Ces rôles définissent les actions autorisées pour chaque utilisateur, et des vérifications sont effectuées au niveau des contrôleurs pour éviter tout accès non autorisé.
- **Audit et historique**
  - Une table `audit_log` enregistre les actions critiques effectuées dans le système (création, modification ou suppression de tournois, de matchs, etc.) pour permettre un suivi précis et prévenir les abus.

4.2 Authentification et Autorisation

- **JWT (JSON Web Token) :**
  - L'authentification des utilisateurs se fait via des tokens JWT. Lorsqu'un utilisateur se connecte, un token est généré et utilisé pour valider ses requêtes ultérieures.
- **Expiration des sessions :**

- Les tokens JWT ont une durée de validité limitée. Les utilisateurs doivent se reconnecter pour obtenir un nouveau token une fois que leur session a expiré.

- **Protection contre les attaques courantes :**
  - Protection contre le Cross-Site Scripting (XSS) : Les données affichées dans la vue sont validées et encodées avant d'être rendues publiques.

Table des figures