

Documentation SlayerGates & MotherGates

Préambule

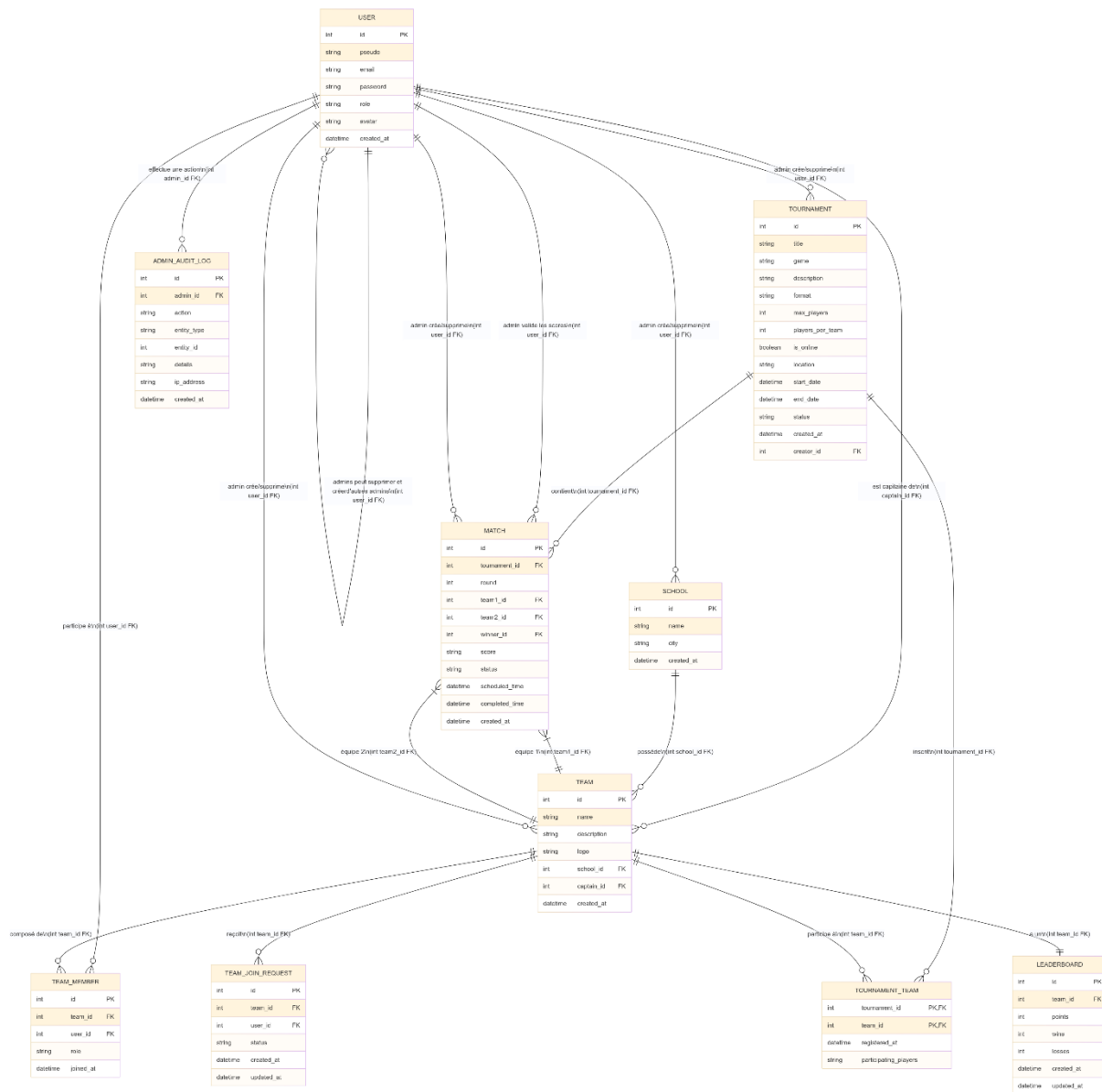
La présente documentation décrit la structure et l'organisation de la base de données partagée entre les projets SlayerGate (site internet) et MotherGates (interface desktop pour les administrateurs). Cette base de données centralise les informations essentielles liées à la gestion des utilisateurs, des équipes, des tournois et des matchs, assurant ainsi une synchronisation et une cohérence des données entre les deux plateformes.

Contenu de la Base de Données

La structure de la base de données repose sur plusieurs entités fondamentales :

- **USER** : Stocke les informations des utilisateurs inscrits.
- **TOURNAMENT** : Gère les tournois, leurs formats et participants.
- **SCHOOL** : Référence les établissements scolaires affiliés.
- **TEAM** : Regroupe les équipes participantes et leurs caractéristiques.
- **MATCH** : Enregistre les matchs et leurs résultats.
- **TEAM_MEMBER** : Suivi des membres des équipes et de leur statut.
- **TOURNAMENT_TEAM** : Gestion des équipes inscrites aux tournois.
- **TEAM_JOIN_REQUEST** : Historique des demandes d'adhésion aux équipes.
- **LEADERBOARD** : Classement des équipes selon leurs performances.
- **ADMINAUDITLOG** : Historique des modifications administrateurs

MCD :



MLD :

USER (id INT PRIMARY KEY AUTO_INCREMENT, pseudo VARCHAR(255) NOT NULL, email VARCHAR(255) UNIQUE NOT NULL, password VARCHAR(255) NOT NULL, role VARCHAR(50) NOT NULL, avatar VARCHAR(255), created_at DATETIME DEFAULT CURRENT_TIMESTAMP)

SCHOOL (id INT PRIMARY KEY AUTO_INCREMENT, name VARCHAR(255) NOT NULL, city VARCHAR(255) NOT NULL, created_at DATETIME DEFAULT CURRENT_TIMESTAMP)

TEAM (id INT PRIMARY KEY AUTO_INCREMENT, name VARCHAR(255) NOT NULL, description TEXT, logo VARCHAR(255), school_id INT, captain_id INT, created_at DATETIME DEFAULT CURRENT_TIMESTAMP, FOREIGN KEY (school_id) REFERENCES SCHOOL(id), FOREIGN KEY (captain_id) REFERENCES USER(id))

TOURNAMENT (id INT PRIMARY KEY AUTO_INCREMENT, title VARCHAR(255) NOT NULL, game VARCHAR(255) NOT NULL, description TEXT, format VARCHAR(255), max_players INT, players_per_team INT, is_online BOOLEAN, location VARCHAR(255), start_date DATETIME, end_date DATETIME, status VARCHAR(50), created_at DATETIME DEFAULT CURRENT_TIMESTAMP, creator_id INT, FOREIGN KEY (creator_id) REFERENCES USER(id))

MATCH (id INT PRIMARY KEY AUTO_INCREMENT, tournament_id INT, round INT, team1_id INT, team2_id INT, winner_id INT, score VARCHAR(50), status VARCHAR(50), scheduled_time DATETIME, completed_time DATETIME, created_at DATETIME DEFAULT CURRENT_TIMESTAMP, FOREIGN KEY (tournament_id) REFERENCES TOURNAMENT(id), FOREIGN KEY (team1_id) REFERENCES TEAM(id), FOREIGN KEY (team2_id) REFERENCES TEAM(id), FOREIGN KEY (winner_id) REFERENCES TEAM(id))

TEAM_MEMBER (id INT PRIMARY KEY AUTO_INCREMENT, team_id INT, user_id INT, role VARCHAR(50), joined_at DATETIME DEFAULT CURRENT_TIMESTAMP, FOREIGN KEY (team_id) REFERENCES TEAM(id), FOREIGN KEY (user_id) REFERENCES USER(id))

TOURNAMENT_TEAM (tournament_id INT, team_id INT, registered_at DATETIME DEFAULT CURRENT_TIMESTAMP, participating_players TEXT, PRIMARY KEY (tournament_id, team_id), FOREIGN KEY (tournament_id) REFERENCES TOURNAMENT(id), FOREIGN KEY (team_id) REFERENCES TEAM(id))

TEAM_JOIN_REQUEST (id INT PRIMARY KEY AUTO_INCREMENT, team_id INT, user_id INT, status VARCHAR(50) NOT NULL, created_at DATETIME DEFAULT CURRENT_TIMESTAMP, updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP, FOREIGN KEY (team_id) REFERENCES TEAM(id), FOREIGN KEY (user_id) REFERENCES USER(id))

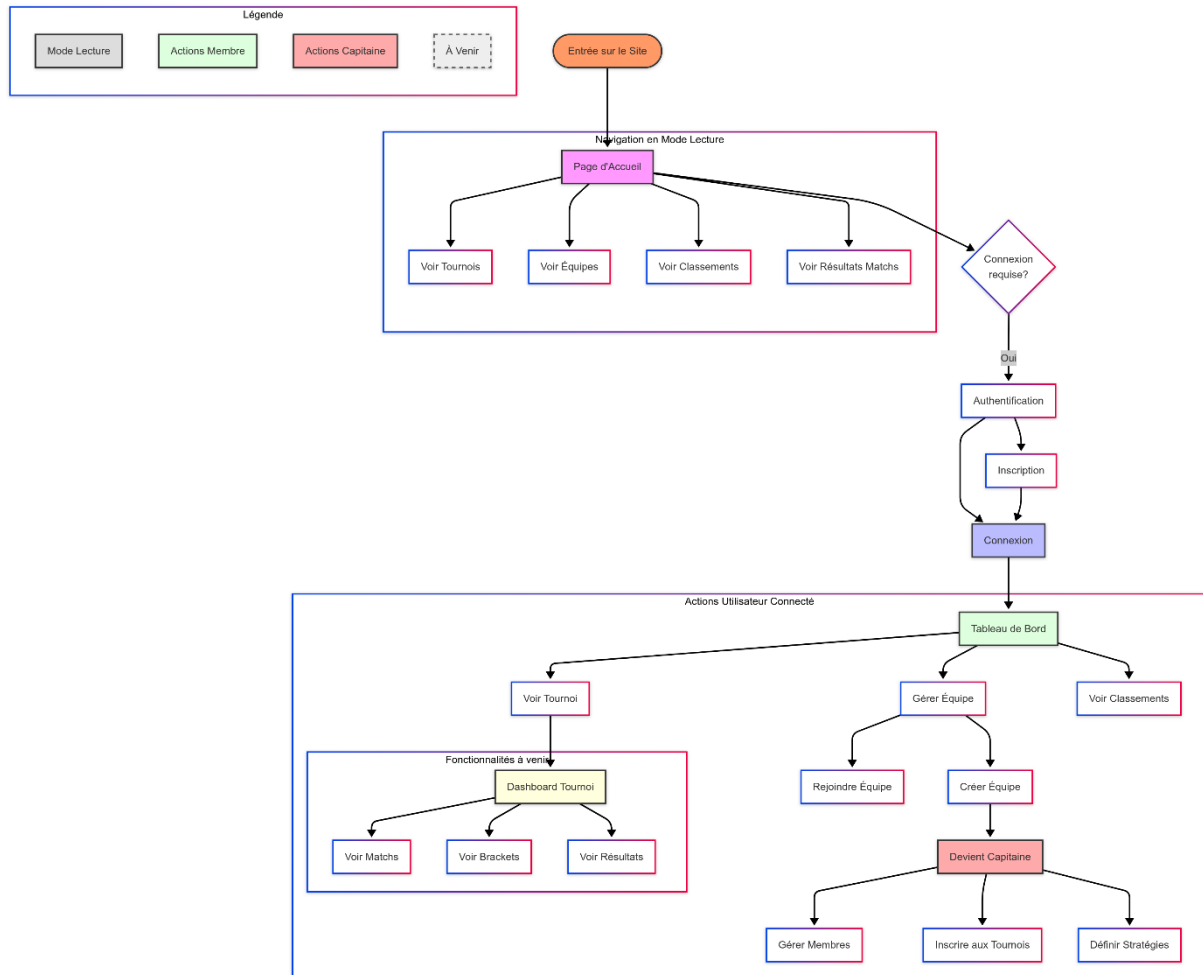
LEADERBOARD (id INT PRIMARY KEY AUTO_INCREMENT, team_id INT UNIQUE, points INT DEFAULT 0, wins INT DEFAULT 0, losses INT DEFAULT 0, created_at DATETIME DEFAULT CURRENT_TIMESTAMP, updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP, FOREIGN KEY (team_id) REFERENCES TEAM(id))

ADMIN_AUDIT_LOG (id INT PRIMARY KEY AUTO_INCREMENT, admin_id INT, action VARCHAR(255) NOT NULL, entity_type VARCHAR(255) NOT NULL, entity_id INT, details TEXT, ip_address VARCHAR(50), created_at DATETIME DEFAULT CURRENT_TIMESTAMP, FOREIGN KEY (admin_id) REFERENCES USER(id))

SlayerGates

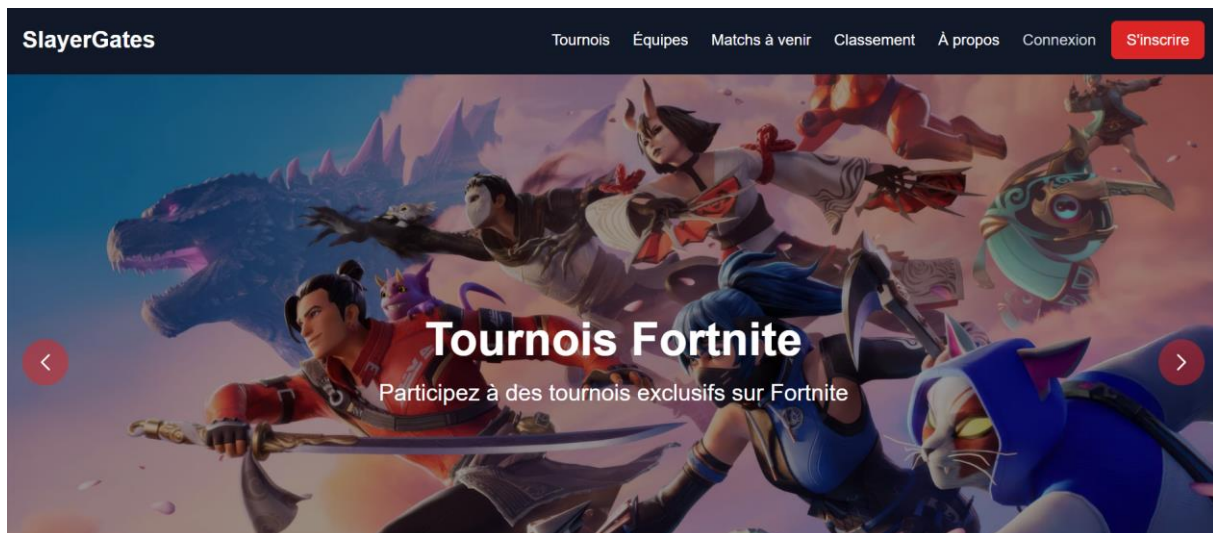
DOCUMENTATION UTILISATEURS-SLAYERGATES

DCU :



1. Présentation et Contexte

SlayerGates est une plateforme web dédiée à l'organisation et à la gestion des tournois esports inter-académiques. La plateforme est conçue pour permettre aux utilisateurs de s'inscrire et de participer activement aux compétitions, tout en assurant une gestion centralisée par des administrateurs via l'interface MotherGates.



2. Fonctionnalités Principales pour les Utilisateurs

- **Création de Compte :**

Tout utilisateur peut se créer un compte sur la plateforme. Une fois inscrit, il peut accéder aux fonctionnalités réservées aux utilisateurs connectés.

Créer un compte

Pseudo

Email

Mot de passe

Le mot de passe doit contenir :

- Au moins 8 caractères
- Au moins une lettre majuscule
- Au moins un chiffre
- Au moins un caractère spécial (!@#\$%^&*()_.,?";'<>)

Confirmer le mot de passe

SlayerGates

TournoisÉquipesMatches à venirClassementÀ proposConnexionS'inscrire

Connexion

Email

Mot de passe

Se connecter

Pas encore inscrit ?

Créer un compte

- **Gestion d'Équipe :**
 - **Création d'Équipe :** Si l'utilisateur n'appartient à aucune équipe, il peut créer une équipe. En créant une équipe, il devient automatiquement le capitaine.

SlayerGates

TournoisÉquipesMatches à venirClassementÀ proposG GabsDéconnexion

Équipes

Créer une équipe

T

Team Saclay
Universite Paris-Saclay

1 membres • 0 tournois

Voir l'équipe

T

Team Telecom
Telecom Paris

1 membres • 0 tournois

Voir l'équipe

4

4Esport
Efrei Paris

2 membres • 0 tournois

Voir l'équipe

The screenshot shows the 'Créer une équipe' form on the SlayerGates website. The form is centered on a light gray background. It includes a text input for 'Nom de l'équipe', a dropdown menu for 'École' with the placeholder 'Sélectionnez une école', and a file upload section for 'Logo de l'équipe' with a 'Choisir un fichier' button and the text 'Aucun fichier choisi'. Below the logo section, it states 'Formats acceptés : JPG, JPEG, PNG, WEBP. Taille maximale : 2 MB'. A red 'Créer l'équipe' button is at the bottom right of the form.

SlayerGates

Tournois Équipes Matches à venir Classement À propos **Gabs** Déconnexion

Créer une équipe

Nom de l'équipe

École

Sélectionnez une école

Logo de l'équipe

Choisir un fichier Aucun fichier choisi

Formats acceptés : JPG, JPEG, PNG, WEBP. Taille maximale : 2 MB

Créer l'équipe

- **Rejoindre une Équipe :**
 - L'utilisateur peut également envoyer une demande pour rejoindre une équipe existante, sous réserve de validation par le capitaine.

The screenshot shows the '4Esport' team page. It displays the team name '4Esport', the school 'École Efrei Paris', the captain 'Capitaine admin', and a list of members 'Membres de l'équipe' including 'admin' and 'baldur'. A red 'Rejoindre l'équipe' button is at the bottom.

4Esport

École
Efrei Paris

Capitaine
admin

Membres de l'équipe

admin
baldur

Rejoindre l'équipe

- **Consultation :**

Tous les utilisateurs (connectés ou non) peuvent consulter la liste des tournois, le classement et des matchs publiés sur la plateforme.

3. Rôles et Interactions

- **Utilisateurs Sans Compte :**

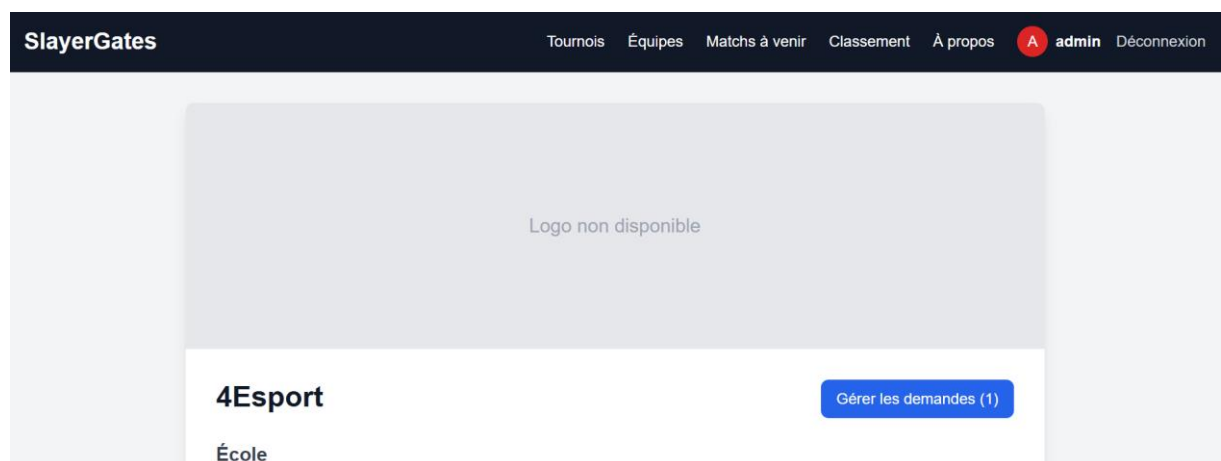
- Peuvent uniquement consulter la liste des tournois, du classement et des matchs publiés sur la plateforme.

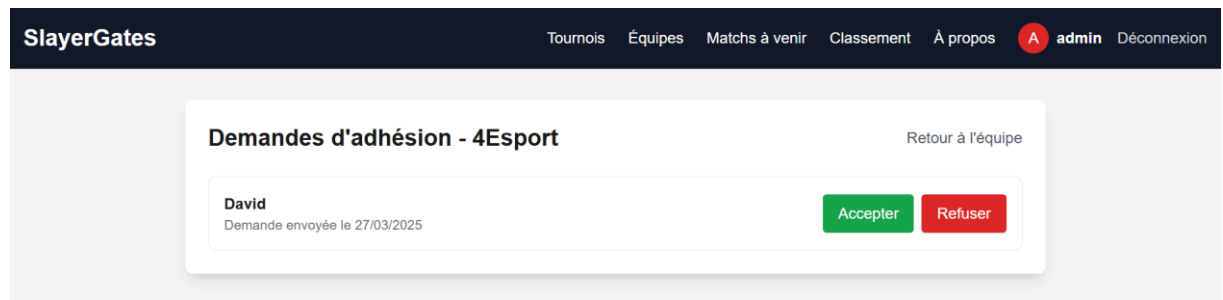
- **Utilisateurs avec un Compte (Utilisateurs Connectés) :**

- Possèdent un compte personnel.
- Peuvent consulter l'ensemble des fonctionnalités accessibles aux utilisateurs connectés, telles que la consultation détaillée des tournois et des matchs.
- Ont la possibilité d'envoyer une demande pour rejoindre une équipe existante.
- Ont également la possibilité de créer une nouvelle équipe s'ils n'appartiennent pas encore à une équipe. Ils deviendront par défaut capitaine d'équipe.

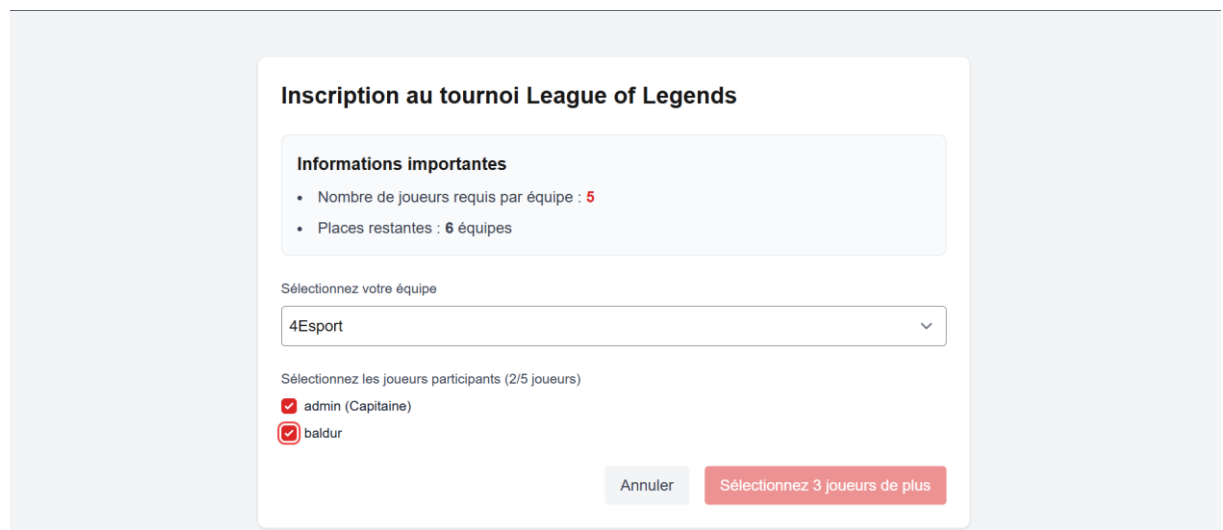
- **Utilisateurs Capitaines (Créateurs d'Équipe) :**

- Lorsqu'un utilisateur crée une équipe, il devient automatiquement le capitaine et dispose de droits spécifiques, tels que la modification du logo de l'équipe et la validation des demandes d'adhésion.





Le capitaine et le seul habilité à inscrire son équipe à un tournoi.



DOCUMENTATION TECHNIQUE – SLAYERGATES

1. ARCHITECTURE SYSTÈME

1.1 Vue d'Ensemble

- **Type d'Application** : Application web
- **Architecture** : Next.js avec API Routes
- **Language Principal** : JavaScript/TypeScript

- **Interface** : React avec Tailwind CSS

2. COMPOSANTS PRINCIPAUX

2.1 Frontend (Web)

- Framework : Next.js
- Styling: Tailwind CSS
- Components React
- Gestion d'état côté client

2.2 Backend

- API Routes Next.js
- ORM : Prisma
- Authentication JWT
- Services métier

3. DÉPENDANCES PRINCIPALES

3.1 Frontend

- **Next.js**
 - Routing système
 - Server-side rendering
 - API integration
- **Tailwind CSS**
 - Styling responsive
 - Composants UI

Exemple de composant React avec authentication :

```
// components/NavBar.js
const NavBar = () => {
  const { user, loading, logout, checkAuth } = useAuth();
  const router = useRouter();

  useEffect(() => {
    checkAuth();
  }, []);

  return (
    <nav className="bg-gray-900 text-white p-4">
      <div className="container mx-auto flex justify-between items-center">
        <Link href="/" className="text-2xl font-bold hover:text-gray-300">
          SlayerGates
        </Link>

        <div className="flex items-center space-x-6">
          <Link href="/tournaments" className="hover:text-red-500">
            Tournois
          </Link>
          <Link href="/teams" className="hover:text-red-500">
            Équipes
          </Link>
          { /* ... autres liens ... */ }
        </div>
      </div>
    </nav>
  );
};
```

Hook d'authentification personnalisé :

```
// hooks/useAuth.js
export function useAuth() {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);
  const [isCaptain, setIsCaptain] = useState(false);

  const checkAuth = async () => {
    try {
      const response = await fetch('/api/auth/check');
      const data = await response.json();

      if (data.user) {
        setUser(data.user);
        // Vérification du statut de capitaine
        const captainResponse = await fetch('/api/teams/captain');
        const captainData = await captainResponse.json();
        setIsCaptain(Array.isArray(captainData) && captainData.length > 0);
      } else {
        setUser(null);
        setIsCaptain(false);
      }
    } catch (error) {
      console.error('Erreur de vérification:', error);
      setUser(null);
    }
  };

  // ... autres fonctions d'authentification ...

  return { user, loading, checkAuth, isCaptain };
}
```

3.2 Backend

- **Prisma ORM**
 - Mapping base de données
 - Migrations
 - Requêtes type-safe
- **JWT/bcrypt**
 - Authentification

- Hachage sécurisé

4. SÉCURITÉ

4.1 Authentification

- JWT pour les sessions
- Hachage bcrypt des mots de passe
- Validation des tokens
- Gestion des rôles utilisateurs

Exemple de création de token JWT :

```
// lib/auth.js
export async function createToken(user) {
  try {
    const token = jwt.sign(
      {
        userId: user.id,
        email: user.email,
        pseudo: user.pseudo,
        role: user.role
      },
      JWT_SECRET,
      { expiresIn: '7d' }
    );
    return token;
  } catch (error) {
    console.error('Erreur lors de la création du token:', error);
    throw error;
  }
}
```

4.2 Protection des Données

- Validation des entrées
- Protection CSRF

- Rate limiting
- Prévention des injections SQL via Prisma

Exemple de validation des données (inscription) :

```
// api/auth/register/route.js
export async function POST(request) {
  try {
    const body = await request.json();
    const { pseudo, email, password, role } = body;

    // Validation des données
    if (!pseudo || !email || !password) {
      return Response.json({ error: 'Tous les champs sont requis' }, { status: 400 });
    }

    // Vérification des contraintes de longueur
    if (pseudo.length > 50) {
      return Response.json({ error: 'Le pseudo ne doit pas dépasser 50 caractères' }, { status: 400 });
    }
    if (email.length > 255) {
      return Response.json({ error: 'L\'email ne doit pas dépasser 255 caractères' }, { status: 400 });
    }

    // ... suite du code d'inscription
  } catch (error) {
    return Response.json({ error: 'Erreur serveur' }, { status: 500 });
  }
}
```

4.3 Authentification Utilisateur

Exemple de login

```
// api/auth/login/route.js
export async function POST(request) {
  try {
    const { email, password } = await request.json();

    if (!email || !password) {
      return Response.json(
        { error: 'Email et mot de passe requis' },
        { status: 400 }
      );
    }

    const user = await prisma.user.findUnique({
      where: { email },
      select: {
        id: true,
        email: true,
        password: true,
        pseudo: true,
        role: true
      }
    });

    // ... suite de la validation et création du token
  } catch (error) {
    return Response.json({ error: 'Erreur serveur' }, { status: 500 });
  }
}
```

5. EXEMPLES D'IMPLÉMENTATION

5.1 API Routes

Exemple de route pour les tournois :

```
// api/tournaments/route.js
export async function GET() {
  try {
    await prisma.$connect();

    // Récupération des tournois avec leurs relations
    const tournaments = await prisma.tournament.findMany({
      include: {
        teams: {
          include: {
            team: {
              include: {
                captain: true,
                members: true,
                school: true
              }
            }
          }
        },
        participants: {
          include: {
            user: true
          }
        }
      }
    });

    return Response.json(tournaments);
  } catch (error) {
    console.error('Erreur lors de la récupération des tournois:', error);
    return Response.json(
      { error: 'Erreur serveur' },
      { status: 500 }
    );
  }
}
```

6. DÉPLOIEMENT

6.1 Prérequis

- Node.js 18+

- Python 3.8+
- PostgreSQL

6.2 Installation

1. Installation des dépendances Node.js

npm install

2. Configuration de la base de données

npx prisma generate

npx prisma migrate dev

DOCUMENTATION DES INCIDENTS - SLAYERGATES

1 Problèmes d'Authentification

- **Gestion des Sessions JWT**
 - **Problème** : Expiration des tokens non gérée côté client
 - **Impact** : Déconnexions inattendues des utilisateurs
 - **Solution** : Implémentation d'un système de refresh token et mise à jour automatique
 - **Prévention** : Ajout de vérifications régulières de la validité du token
- **Conflits de Rôles**
 - **Problème** : Confusion dans la gestion des permissions utilisateur/capitaine
 - **Impact** : Accès incorrects aux fonctionnalités d'administration d'équipe
 - **Solution** : Refonte du système de rôles avec hiérarchie claire
 - **Prévention** : Tests unitaires pour chaque niveau de permission

2 Problèmes de Base de Données

- **Relations Complexes**

- **Problème** : Cycles de relations dans le schéma Prisma
- **Impact** : Erreurs lors des migrations et requêtes complexes
- **Solution** : Restructuration des relations avec références unidirectionnelles
- **Exemple** :

```
// Avant
model Team {
  members    User[]
  captain    User    @relation("TeamCaptain")
}
```

```
// Après
model Team {
  members    TeamMember[]
  captainId  Int?
  captain    User?    @relation("TeamCaptain", fields: [captainId], references: [id])
}
```

- **Performance des Requêtes**

- **Problème** : Temps de réponse lents sur les pages de tournois
- **Impact** : Expérience utilisateur dégradée
- **Solution** :
 - Optimisation des requêtes Prisma
 - Mise en place de pagination
 - Création d'index appropriés

3. Problèmes d'Interface Utilisateur

- **Responsive Design**

- **Problème** : Mise en page cassée sur mobiles
- **Impact** : Interface inutilisable sur certains appareils
- **Solution** : Refonte avec approche mobile-first
- **Exemple** :

```
/* Avant */  
.tournament-grid {  
  display: grid;  
  grid-template-columns: repeat(4, 1fr);  
}
```

```
/* Après */  
.tournament-grid {  
  display: grid;  
  grid-template-columns: repeat(1, 1fr);  
}  
  
@media (min-width: 640px) {  
  .tournament-grid {  
    grid-template-columns: repeat(2, 1fr);  
  }  
}  
  
@media (min-width: 1024px) {  
  .tournament-grid {  
    grid-template-columns: repeat(4, 1fr);  
  }  
}
```

- **État de Chargement**

- **Problème** : Absence d'indicateurs de chargement
- **Impact** : Confusion des utilisateurs lors des actions
- **Solution** : Implémentation de composants de loading et feedback

4. Problèmes de Performance

- **Chargement Initial**

- **Problème** : Temps de chargement excessif de la page d'accueil
- **Impact** : Taux de rebond élevé
- **Solution** :
 - Optimisation des images
 - Lazy loading des composants
 - Mise en cache côté serveur

- **Gestion des Ressources**

- **Problème** : Fuites de mémoire dans les composants React
- **Impact** : Ralentissements progressifs de l'application
- **Solution** : Nettoyage correct des effets et listeners
- **Exemple** :

```
useEffect(() => {  
  const subscription = someAPI.subscribe();  
  
  return () => {  
    subscription.unsubscribe();  
  };  
}, []);
```

5. Problèmes de Déploiement

- **Migrations Base de Données**

- **Problème** : Échecs de migration en production
- **Impact** : Indisponibilité du service
- **Solution** :
 - Tests de migration en pré-production
 - Procédure de rollback automatisée
 - Sauvegarde avant migration

6. Problèmes de Sécurité

- **Injection SQL**

- **Problème** : Vulnérabilités potentielles dans les requêtes
- **Impact** : Risque de compromission des données
- **Solution** :
 - Utilisation systématique de Prisma
 - Validation stricte des entrées
 - Tests de sécurité réguliers

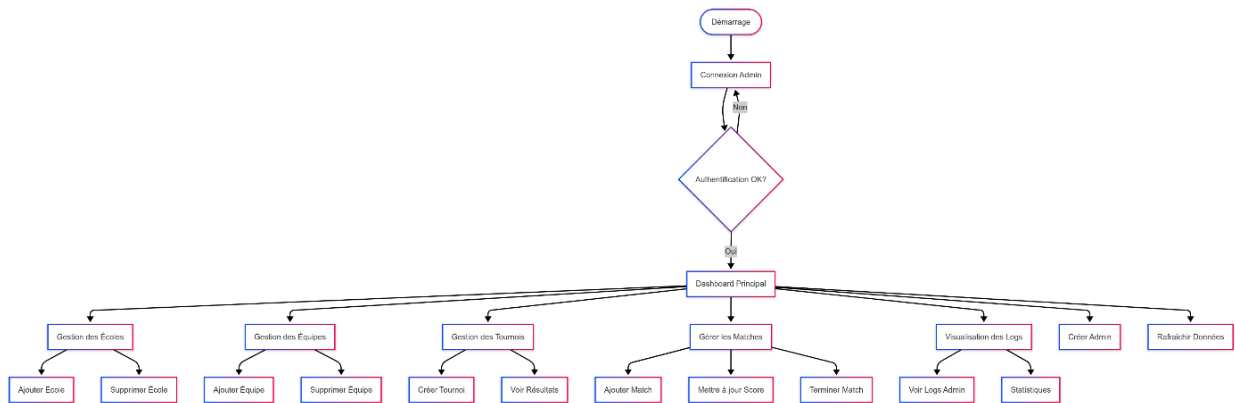
- **Protection CSRF**

- **Problème** : Absence de protection sur certaines routes
- **Impact** : Vulnérabilité aux attaques CSRF
- **Solution** :
 - Middleware de validation des tokens
 - Headers de sécurité appropriés

MotherGates

GUIDE UTILISATEUR – MOTHERGATES

DCU :



1. PRÉSENTATION

MotherGates est l'interface administrateur de la plateforme SlayerGates, conçue pour gérer les tournois e-sport interacadémiques. Cette application desktop vous permet de contrôler l'ensemble de la plateforme de manière sécurisée et efficace.

2. FONCTIONNALITÉS PRINCIPALES

2.1 Gestion des Tournois

- **Créer un tournoi**
 - Remplissez les informations de base (titre, jeu, description)
 - Définissez le format et les limites de participants
 - Choisissez entre tournoi en ligne ou présentiel
 - Fixez les dates de début

Admin: DoomAdmin

Créer un administrateur Rafraîchir tout

Matchs Tournois Équipes et joueurs Écoles Logs

Inscriptions aux tournois Places disponibles Gérer les tournois Créer un tournoi

Nom*:

Jeu*:

Description*:

Format:

Nombre max d'équipes*:

Joueurs par équipe*:

Date de début*:

Créer le tournoi

• Gérer les tournois existants

- Consultez la liste des inscriptions
- Suivez les places disponibles
- Modifiez les paramètres si nécessaire
- Supprimez un tournoi si besoin

Admin: DoomAdmin

Créer un administrateur Rafraîchir tout

Matchs Tournois Équipes et joueurs Écoles Logs

Inscriptions aux tournois Places disponibles Gérer les tournois Créer un tournoi

| ID | Nom | Jeu | Date de début | Statut |
|-----|-------------------|----------------|------------------|---------|
| 1.2 | Valduron Cup | Valduron | 2025-04-14 15:10 | opening |
| 2.1 | League of Legends | League of L... | 2025-04-07 15:10 | opening |

Créer le tournoi

Admin: DoomAdmin

Créer un administrateur Rafraîchir tout

Matchs Tournois Équipes et joueurs Écoles Logs

Inscriptions aux tournois Places disponibles Gérer les tournois Créer un tournoi

| Tournoi | Places max | Places prises | Places restantes |
|----------------------|------------|---------------|------------------|
| 1. Valduron Cup | 8 | 2 | 6 |
| 2. League of Legends | 8 | 2 | 6 |

Créer le tournoi

2.2 Gestion des Écoles

• Ajouter une école

- Enregistrez le nom de l'école
- Indiquez la ville
- Validez les informations

MotherGates Admin v1.0.0

Admin: DoomAdmin

Créer un administrateur

Rafraîchir tout

Matches

Tournois

Équipes et joueurs

Écoles

Logs

Nom de l'école:

Ville:

Créer une école

Supprimer l'école

| ID | Nom | Ville |
|-----|--------------------------|---------------|
| 1 6 | EPITA | Paris |
| 2 3 | ESSEC Business School | Cergy |
| 3 7 | Efrei Paris | Villejuif |
| 4 4 | HEC Paris | Jouy-en-Josas |
| 5 5 | INSA Centre Val de Loire | Blois |
| 6 2 | Telecom Paris | Paris |
| 7 1 | Universite Paris-Saclay | Saclay |

- **Gérer les écoles**

- Consultez la liste des écoles
- Supprimez une école (possible uniquement si aucune équipe n'y est rattachée)

2.3 Gestion des Équipes

- **Supervision des équipes**

- Visualisez les équipes par école
- Gérez les membres
- Suivez les inscriptions aux tournois
- Validez les demandes d'adhésion

Équipes

Joueurs

Supprimer l'équipe

| | Date de création | Nom | Capitaine | École | Statut |
|---|------------------|--------------|-----------|-------------------------|--------|
| 1 | 2025-03-24 18:10 | Team Telecom | Player2 | Telecom Paris | Active |
| 2 | 2025-03-24 18:10 | Team Saclay | Player1 | Universite Paris-Saclay | Active |

2.4 Gestion des Matches

Création des matchs

- Notifications lorsqu'un tournoi est prêt
- Choisir les équipes qui doivent s'affronter
- Possibilité de supprimer les matchs

Admin: DoomAdmin

Créer un administrateurRafraîchir tout

MatchesTournoisÉquipes et joueursÉcolesLogs

MatchesHistorique des matchs

Créer un matchMettre à jour le score

Supprimer le match

| | ID | Tournoi | Round | Équipe 1 | Équipe 2 | Score | Statut | Date prévue |
|---|----|---------------|-------|--------------|--------------|-------|---------|------------------|
| 1 | 5 | League of ... | 2 | Team Telecom | Team Saclay | | pending | 29/03/2025 20:55 |
| 2 | 2 | Valorant Cup | 1 | Team Saclay | Team Telecom | | pending | 15/04/2025 18:10 |

2.5 Gestion des Utilisateurs

Modération des users

- Voir la liste des joueurs (pseudo, email, équipe, école, date)
- Possibilité de supprimer un utilisateur

ÉquipesJoueurs

Rechercher: Pseudo ou email...École: Toutes les écolesÉquipe: Toutes les équipesDate depuis: Tout

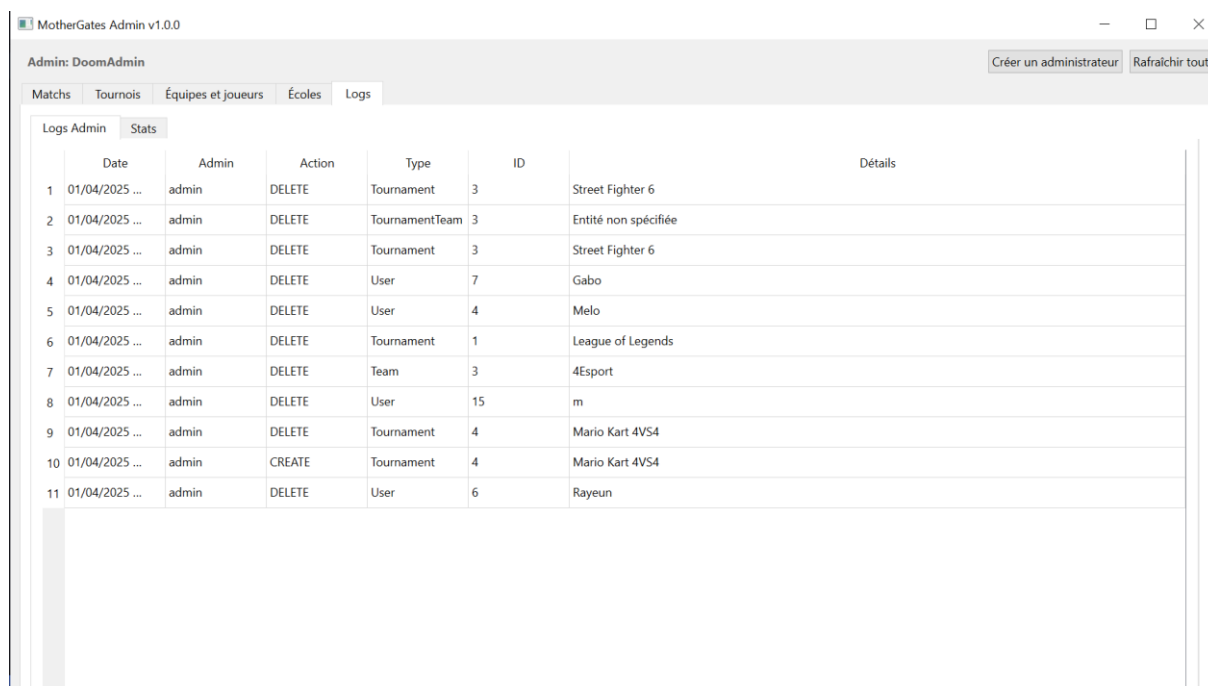
Supprimer le joueur

| | Date de création | Pseudo | Email | École(s) | Équipe(s) |
|---|------------------|-----------|-----------------------------|-------------------------|--------------|
| 1 | 31/03/2025 19:14 | DoomAdmin | melchior.arrouche@yahoo.com | Sans école | Sans équipe |
| 2 | 29/03/2025 20:20 | Macan | eleonorecabou@gmail.Com | Sans école | Sans équipe |
| 3 | 27/03/2025 15:31 | Gabs | icomebackbaby@gmail.com | Sans école | Sans équipe |
| 4 | 27/03/2025 13:47 | baldur | blackgalaxy@voila.fr | Sans école | Sans équipe |
| 5 | 24/03/2025 18:10 | Player2 | player2@example.com | Telecom Paris | Team Telecom |
| 6 | 24/03/2025 18:10 | Player1 | player1@example.com | Universite Paris-Saclay | Team Saclay |
| 7 | 24/03/2025 18:10 | admin | melchior.a2r@gmail.com | Sans école | Sans équipe |

2.4 Analyse et Logs

Disposition d'outil d'analyse

- Liste des actions/modifications des admins sur la plateforme
- Graphiques de performance de la plateforme SlayerGates



The screenshot shows the MotherGates Admin v1.0.0 interface. At the top, there's a header bar with the title 'Admin: DoomAdmin' and two buttons: 'Créer un administrateur' and 'Rafraîchir tout'. Below the header, there's a navigation bar with tabs: 'Matches', 'Tournois', 'Équipes et joueurs', 'Écoles', and 'Logs'. The 'Logs' tab is selected. Under the 'Logs' tab, there are two sub-tabs: 'Logs Admin' and 'Stats'. The 'Logs Admin' sub-tab is active, displaying a table of logs. The table has columns: 'Date', 'Admin', 'Action', 'Type', 'ID', and 'Détails'. There are 11 rows of logs, numbered 1 to 11. The logs show various actions like 'DELETE' and 'CREATE' performed by 'admin' on different entities like 'Tournament', 'User', and 'TournamentTeam'.

| | Date | Admin | Action | Type | ID | Détails |
|----|----------------|-------|--------|----------------|----|----------------------|
| 1 | 01/04/2025 ... | admin | DELETE | Tournament | 3 | Street Fighter 6 |
| 2 | 01/04/2025 ... | admin | DELETE | TournamentTeam | 3 | Entité non spécifiée |
| 3 | 01/04/2025 ... | admin | DELETE | Tournament | 3 | Street Fighter 6 |
| 4 | 01/04/2025 ... | admin | DELETE | User | 7 | Gabo |
| 5 | 01/04/2025 ... | admin | DELETE | User | 4 | Melo |
| 6 | 01/04/2025 ... | admin | DELETE | Tournament | 1 | League of Legends |
| 7 | 01/04/2025 ... | admin | DELETE | Team | 3 | 4Esport |
| 8 | 01/04/2025 ... | admin | DELETE | User | 15 | m |
| 9 | 01/04/2025 ... | admin | DELETE | Tournament | 4 | Mario Kart 4VS4 |
| 10 | 01/04/2025 ... | admin | CREATE | Tournament | 4 | Mario Kart 4VS4 |
| 11 | 01/04/2025 ... | admin | DELETE | User | 6 | Rayeun |

3. UTILISATION QUOTIDIENNE

3.1 Connexion

1. Lancez l'application MotherGates
2. Entrez vos identifiants administrateur
3. Vérifiez que vous êtes bien connecté (indicateur en haut de l'interface)

3.2 Navigation

- Les onglets permettent de basculer entre les fonctionnalités

- Un système de logs trace toutes les actions importantes

3.3 Bonnes Pratiques

- Vérifiez toujours les informations avant validation
- Sauvegardez régulièrement vos modifications
- Consultez les logs en cas de doute
- Déconnectez-vous après utilisation

4. RÉOLUTION DES PROBLÈMES COURANTS

4.1 Problèmes de Connexion

- Vérifiez votre connexion Internet
- Assurez-vous d'utiliser les bons identifiants
- Contactez le support technique si le problème persiste

4.2 Erreurs Courantes

- **Message "Accès refusé"**: Vérifiez vos droits d'administrateur
- **Échec de création** : Vérifiez que tous les champs obligatoires sont remplis
- **Erreur de suppression** : Assurez-vous qu'il n'y a pas de dépendances

5. RECOMMANDATIONS SYSTÈME

Configuration Requise

- Python 3.10 ou supérieur
- Système d'exploitation : Windows/Linux/macOS

- Connexion Internet stable
- Écran haute résolution recommandé

DOCUMENTATION TECHNIQUE – MOTHERGATES

1. ARCHITECTURE SYSTÈME

1.1 Vue d'Ensemble

- **Type d'Application** : Application desktop
- **Architecture** : Modèle-Vue-Contrôleur (MVC)
- **Langage Principal** : Python 3.10+
- **Interface Graphique** : PyQt6/PySide6
- **Base de Données** : PostgreSQL (hébergée sur Neon.tech)

1.2 Composants Principaux

1. Frontend (GUI)

- Framework : PySide6 v6.8.2
- Widgets personnalisés
- Interface responsive
- Gestion des événements Qt

2. Backend

- ORM : SQLAlchemy v2.0.23

- Connecteur BD : psycopg2-binary v2.9.9
- Gestion des sessions
- Services métier

3. Base de Données

- PostgreSQL 15+
- Schéma relationnel
- Indexation optimisée
- Triggers et contraintes

2. DÉPENDANCES PRINCIPALES

2.1 Interface Graphique

- **PySide6** (v6.8.2)
 - Widgets Qt
 - Gestionnaire de mise en page
 - Signaux et slots

3. SÉCURITÉ

3.1 Authentification

- JWT pour les sessions
- Hachage des mots de passe
- Validation des tokens
- Gestion des rôles admins

4. LOGGING ET MONITORING

4.1 Système de Logs

- Niveaux de log hiérarchiques

5. FONCTION CRITIQUE

5.1 Création de Match

Fonction : MatchService.create_match

```
def create_match(self, tournament_id: int, team1_id: int, team2_id: int,
                 round: int, scheduled_time: Optional[datetime] = None) -> Match:
    """
    Crée un nouveau match dans un tournoi.

    Vérifie l'existence du tournoi et des équipes avant d'ajouter le match à la base de données.
    """
    # Vérifier l'existence du tournoi
    tournament = self.db.query(Tournament).filter(Tournament.id == tournament_id).first()
    if not tournament:
        raise ValueError("Tournoi introuvable")

    # Vérifier l'existence des équipes et leur inscription au tournoi
    team1 = self.db.query(Team).filter(Team.id == team1_id).first()
    team2 = self.db.query(Team).filter(Team.id == team2_id).first()
    if not team1 or not team2:
        raise ValueError("Une ou plusieurs équipes sont introuvables")

    # Création et enregistrement du match
    match = Match(
        tournament_id=tournament_id,
        team1_id=team1_id,
        team2_id=team2_id,
        round=round,
        scheduled_time=scheduled_time
    )

    self.db.add(match)
    self.db.commit()
    self.db.refresh(match)
    return match
```

Utilité: Gère la création des matchs dans le système en vérifiant l'existence des équipes et du tournoi. Utilisée lors de l'ajout de nouveaux matchs dans l'interface utilisateur.

5.2. Validation des Administrateurs

Fonction : UserService.validate_admin_password

```
def validate_admin_password(self, password: str) -> tuple[bool, str]:
    """
    Valide un mot de passe administrateur selon des critères stricts.
    Retourne un tuple (is_valid, message).
    """
    import string

    common_passwords = [
        "admin123", "password", "123456", "qwerty", "azerty",
        "motdepasse", "administrator", "adminadmin"
    ]

    if len(password) < 12:
        return False, "Le mot de passe doit contenir au moins 12 caractères"
    if not any(char in string.punctuation for char in password):
        return False, "Le mot de passe doit contenir au moins un caractère spécial"
    if not any(char.isdigit() for char in password):
        return False, "Le mot de passe doit contenir au moins un chiffre"
    if not any(char.isupper() for char in password):
        return False, "Le mot de passe doit contenir au moins une lettre majuscule"
    if password.lower() in common_passwords:
        return False, "Ce mot de passe est trop courant et non sécurisé"

    return True, "Mot de passe valide"
```

Utilité : Garantit des mots de passe administrateurs robustes en appliquant plusieurs règles de sécurité. Utilisée lors de la création et de la modification des comptes administrateurs.

5.3 Mise à Jour des Scores

Fonction : MatchService.update_score

```
def update_score(self, match_id: int, score: str, winner_id: Optional[int] = None):
    """
    Met à jour le score d'un match et, optionnellement, définit le gagnant.
    """
    import re

    match = self.db.query(Match).filter(Match.id == match_id).first()
    if not match:
        raise ValueError("Match introuvable")

    if not re.match(r'^\d+[-:]\d+$', score):
        raise ValueError("Format de score invalide")

    match.score = score

    if winner_id:
        match.winner_id = winner_id

    self.db.commit()
```

Utilité : Permet la mise à jour des scores et l'enregistrement des gagnants après les matchs. Utilisée lors de la gestion des résultats.

5.4 Gestion des Utilisateurs

Fonction: UserService.create_admin

```
def create_admin(self, pseudo: str, email: str, password: str) -> User:
    """
    Crée un nouvel administrateur après validation des informations.
    """
    from werkzeug.security import generate_password_hash

    # Validation du mot de passe
    is_valid, message = self.validate_admin_password(password)
    if not is_valid:
        raise ValueError(f"Mot de passe invalide : {message}")

    # Vérifier si l'utilisateur existe déjà
    existing_user = self.db.query(User)
        .filter((User.pseudo == pseudo) | (User.email == email))
        .first()

    if existing_user:
        raise ValueError("Un utilisateur avec ce pseudo ou cet email existe déjà")

    # Création de l'administrateur
    user = User(
        pseudo=pseudo,
        email=email,
        password=generate_password_hash(password),
        role='admin'
    )

    self.db.add(user)
    self.db.commit()
    self.db.refresh(user)
    return user
```

Utilité : Permet la création de comptes administrateurs tout en vérifiant l'unicité des identifiants et la robustesse des mots de passe. Utilisée dans l'interface d'administration.

DOCUMENTATION DES INCIDENTS – MOTHERGATES

1. Gestion des Erreurs et Exceptions

1.1 Erreurs de Base de Données

- **Problème** : Connexions multiples à la base de données
 - **Solution** : Utilisation d'une seule instance de SessionLocal() par widget et fermeture propre des connexions.
- **Problème** : Erreurs de transaction non gérées
 - **Solution** : Ajout de try/except autour des opérations de base de données.

1.2 Gestion des Utilisateurs

- **Problème** : Validation des mots de passe des administrateurs
 - **Solution** : Implémentation de règles strictes de validation dans UserService.validate_admin_password().
 - **Date** : 2025-04-01
- **Problème** : Création d'administrateurs multiples
 - **Solution** : Ajout de vérifications d'existence avant création.

2. Interface Utilisateur

2.1 Widgets et Dialogues

- **Problème** : Fermeture incorrecte des dialogues
 - **Solution** : Ajout de gestionnaires d'événements closeEvent() pour fermer les connexions.
- **Problème** : Sélection multiple dans les tables
 - **Solution** : Ajout de vérifications de sélection dans les méthodes de suppression.

2.2 Gestion des Données

- **Problème** : Suppression d'écoles avec équipes associées
 - **Solution** : Ajout de vérifications et messages d'erreur explicites.
- **Problème** : Mise à jour des scores des matchs
 - **Solution** : Ajout de confirmation avant modification.

3. Audit et Journalisation

3.1 Suivi des Actions

- **Problème** : Actions non enregistrées
 - **Solution** : Implémentation de AdminAuditService pour toutes les opérations critiques.

3.2 Gestion des Logs

- **Problème** : Logs incohérents
 - **Solution** : Configuration centralisée du logging avec setup_logging().

4. Performance et Optimisation

4.1 Chargement des Données

- **Problème** : Chargement lent des données
 - **Solution** : Optimisation des requêtes avec SQLAlchemy et chargement différé.

5. Sécurité

5.1 Accès aux Données

- **Problème** : Accès non sécurisé aux données
 - **Solution** : Implémentation de vérifications des droits d'accès.

5.2 Validation des Entrées

- **Problème** : Données non validées
 - **Solution** : Ajout de validations côté serveur et client.

Notes

- Tous les incidents ont été résolus avec des tests unitaires et d'intégration.
- La documentation sera mise à jour régulièrement.
- Les dates indiquées correspondent à la dernière mise à jour du document.