Spécifications Backend du Projet

Ce document détaille les composants clés du backend pour le système de gestion des inscriptions et réservations sportives, en se basant sur les exigences du cahier des charges et en intégrant l'authentification Firebase.

1. Modèles de Données (Django Models)

Les modèles de données représentent la structure des informations stockées dans la base de données PostgreSQL. Chaque modèle correspondra à une table dans la base de données.

1.1 Modèle `Utilisateur`

Représente un utilisateur du système (employé, conjoint, enfant). Ce modèle sera lié à l'UID Firebase pour la gestion de l'authentification.

- `firebase_uid` (CharField, unique) : L'identifiant unique de l'utilisateur dans Firebase Authentication.
- `matricule` (CharField, unique, nullable) : Matricule de l'employé (si applicable).
- `nom` (CharField)
- `prenom` (CharField)
- `categorie` (CharField, choix: 'Employé', 'Conjoint', 'Enfant')
- `service_affectation` (CharField, nullable) : Service d'affectation de l'employé.
- `cnie` (CharField, unique, nullable) : Numéro de CNIE pour les adultes.
- `date_naissance` (DateField) : Date de naissance pour tous les bénéficiaires.
- `est_employe` (BooleanField) : Indique si l'utilisateur est un employé (pour distinction femme employée/non-employée piscine).
- `est_admin` (BooleanField, défaut: False) : Indique si l'utilisateur a des privilèges d'administration.

- `parent` (ForeignKey vers Utilisateur, nullable) : Pour lier les conjoints/enfants à un employé.

1.2 Modèle `Salle`

Représente une salle sportive ou une infrastructure réservable.

- `nom` (CharField, unique)
- `description` (TextField)
- `capacite` (IntegerField)
- `type_salle` (CharField, choix: 'Basket', 'Minifoot', 'Tennis', 'Ping-pong', 'Salle de sport', 'Piscine')
- `code_salle` (CharField, unique) : Ex: C001-1.
- `categorie_genre` (CharField, choix: 'Homme', 'Femme', 'Mixte') : Pour les salles spécifiques.

1.3 Modèle `Creneau`

Représente un créneau horaire disponible pour une salle ou une piscine.

- `salle` (ForeignKey vers Salle)
- `jour_semaine` (IntegerField, choix: 0=Lundi, ..., 6=Dimanche)
- `heure_debut` (TimeField)
- `heure_fin` (TimeField)
- `quota_max` (IntegerField) : Nombre maximum de réservations pour ce créneau.
- `est_reservable` (BooleanField, défaut: True) : Permet de désactiver un créneau temporairement.

1.4 Modèle `Reservation`

Représente une réservation effectuée par un utilisateur.

- `utilisateur` (ForeignKey vers Utilisateur)
- `creneau` (ForeignKey vers Creneau)

- `date_reservation` (DateField) : Date spécifique de la réservation.
- `statut` (CharField, choix: 'En attente', 'En attente documents', 'Validée', 'Refusée')
- `type_paiement` (CharField, choix: 'RIB', 'TPE', 'N/A') : Pour les salles.
- `montant_paye` (DecimalField, nullable) : Montant payé pour la réservation de salle.
- `reçu_paiement` (FileField, nullable) : Fichier du reçu de paiement (pour les salles).
- `code_validation` (CharField, unique, nullable) : Code généré par le système pour la validation.
- `qr_code_image` (ImageField, nullable) : Image du QR code généré.
- `date_demande` (DateTimeField, auto_now_add=True)
- `date_validation` (DateTimeField, nullable)

1.5 Modèle `GroupePiscine`

Gère les groupes spécifiques pour la piscine.

- `nom_groupe` (CharField, unique) : Ex: 'Femmes Employées - Groupe A'.
- `categorie_utilisateur` (CharField, choix: 'Homme', 'Femme Employée', 'Femme Non-Employée', 'Enfant', 'Employé Continu')
- `jours_piscine` (CharField) : Jours attribués (ex: 'Lundi, Jeudi').
- `heure_debut` (TimeField)
- `heure_fin` (TimeField)
- `frequence_actuelle` (IntegerField, défaut: 0) : Pour le suivi de la fréquence.

1.6 Modèle `Activite`

Représente toutes les activités proposées, réservables ou non.

- `nom` (CharField, unique)
- `description` (TextField)

- `est_reservable` (BooleanField, défaut: False) : True pour Salles et Piscine.
- `droits_acces` (TextField) : Qui peut y accéder (employés, conjoints, enfants, grand public).
- `horaires_recommandes` (TextField) : Horaires généraux ou recommandations.

1.7 Modèle `Competition`

Informations sur les compétitions associées aux activités.

- `activite` (ForeignKey vers Activite)
- `nom_competition` (CharField)
- `description` (TextField)
- `date_debut` (DateField)
- `date_fin` (DateField)
- `modalites_inscription` (TextField) : Comment s'inscrire (hors système, formulaire interne).
- `resultats_passes` (TextField, nullable) : Liens ou descriptions des résultats.

2. Spécifications des API REST (Django REST Framework)

Les API seront construites en utilisant Django REST Framework, avec des ViewSets pour les opérations CRUD standard et des vues personnalisées pour les logiques spécifiques. L'authentification sera basée sur la validation des tokens Firebase.

2.1 Authentification et Profil Utilisateur

- **Endpoint :** `/api/auth/firebase-login/`
  - **Méthode :** `POST`
  - **Description :** Valide le token Firebase ID et crée/met à jour le profil `Utilisateur` dans la base de données.
  - **Requête :** `{'id_token': '...'}`

    - **Réponse :** `{'access_token': '...', 'refresh_token': '...', 'user_data': {...}}` (token interne si nécessaire, et données utilisateur).
- **Endpoint :** `/api/users/me/`
    - **Méthode :** `GET`
    - **Description :** Récupère le profil de l'utilisateur authentifié.
    - **Authentification :** Requiert un token valide.
- **Endpoint :** `/api/users/me/`
    - **Méthode :** `PUT` / `PATCH`
    - **Description :** Met à jour le profil de l'utilisateur authentifié.
    - **Authentification :** Requiert un token valide.
- **Endpoint :** `/api/users/`
    - **Méthode :** `POST` (pour création manuelle par admin)
    - **Description :** Crée un nouvel utilisateur (admin seulement).
    - **Authentification :** Requiert un token admin valide.

2.2 Gestion des Activités

- **Endpoint :** `/api/activites/`
    - **Méthode :** `GET`
    - **Description :** Liste toutes les activités (réservables ou non).
    - **Filtres :** `?est_reservable=true`
- **Endpoint :** `/api/activites/<id>/`
    - **Méthode :** `GET`
    - **Description :** Détails d'une activité spécifique.

2.3 Gestion des Salles et Créneaux

- **Endpoint :** `/api/salles/`
    - **Méthode :** `GET`
    - **Description :** Liste toutes les salles.
    - **Filtres :** `?type_salle=Piscine`, `?categorie_genre=Homme`

- **Endpoint :** `/api/salles/<id>/`
  - **Méthode :** `GET`
  - **Description :** Détails d'une salle.
- **Endpoint :** `/api/salles/<id>/creneaux/`
  - **Méthode :** `GET`
  - **Description :** Liste les créneaux disponibles pour une salle spécifique.
  - **Filtres :** `?date=YYYY-MM-DD`, `?est_reservable=true`
- **Endpoint :** `/api/salles/`
  - **Méthode :** `POST` (admin seulement)
  - **Description :** Crée une nouvelle salle.
- **Endpoint :** `/api/salles/<id>/`
  - **Méthode :** `PUT` / `PATCH` (admin seulement)
  - **Description :** Met à jour une salle.
- **Endpoint :** `/api/creneaux/`
  - **Méthode :** `POST` (admin seulement)
  - **Description :** Crée un nouveau créneau.
- **Endpoint :** `/api/creneaux/<id>/`
  - **Méthode :** `PUT` / `PATCH` (admin seulement)
  - **Description :** Met à jour un créneau.

2.4 Gestion des Réservations (Client)

- **Endpoint :** `/api/reservations/`
  - **Méthode :** `POST`
  - **Description :** Crée une nouvelle demande de réservation (salle ou piscine).
  - **Requête :** `{'creneau_id': ..., 'date_reservation': 'YYYY-MM-DD', 'ayants_droit': [...], 'type_paiement': 'RIB', 'montant_paye': ...}`
  - **Authentification :** Requiert un token valide.
- **Endpoint :** `/api/reservations/me/`

    - **Méthode :** `GET`
    - **Description :** Liste les réservations de l'utilisateur authentifié.
    - **Filtres :** `?statut=En attente`, `?statut=Validée`
    - **Authentification :** Requiert un token valide.
- **Endpoint :** `/api/reservations/<id>/upload-justificatif/`
    - **Méthode :** `POST`
    - **Description :** Télécharge le reçu de paiement et/ou le QR code pour une réservation de salle.
    - **Requête :** `{'reçu_paiement': <fichier>, 'code_validation': '...'}`
    - **Authentification :** Requiert un token valide.

2.5 Gestion des Réservations (Administration)

- **Endpoint :** `/api/admin/reservations/`
    - **Méthode :** `GET`
    - **Description :** Liste toutes les réservations (admin seulement).
    - **Filtres :** `?statut=En attente`, `?salle_id=...`, `?utilisateur_id=...`
    - **Authentification :** Requiert un token admin valide.
- **Endpoint :** `/api/admin/reservations/<id>/validate/`
    - **Méthode :** `POST`
    - **Description :** Valide une réservation et génère le QR code/code interne.
    - **Authentification :** Requiert un token admin valide.
- **Endpoint :** `/api/admin/reservations/<id>/reject/`
    - **Méthode :** `POST`
    - **Description :** Refuse une réservation.
    - **Authentification :** Requiert un token admin valide.

2.6 Gestion des Groupes Piscine

- **Endpoint :** `/api/groupes-piscine/`
    - **Méthode :** `GET`

    - **Description :** Liste tous les groupes de piscine.
- **Endpoint :** `/api/groupes-piscine/`
    - **Méthode :** `POST` (admin seulement)
    - **Description :** Crée un nouveau groupe de piscine.
- **Endpoint :** `/api/groupes-piscine/<id>/`
    - **Méthode :** `PUT` / `PATCH` (admin seulement)
    - **Description :** Met à jour un groupe de piscine (y compris la fréquence).

2.7 Tableau de Bord et Rapports

- **Endpoint :** `/api/dashboard/stats/`
    - **Méthode :** `GET`
    - **Description :** Fournit les statistiques clés pour le tableau de bord (nombre de réservations validées/en cours, taux d'occupation, etc.).
    - **Authentification :** Requiert un token admin valide.
- **Endpoint :** `/api/dashboard/export-excel/`
    - **Méthode :** `GET`
    - **Description :** Exporte les données de réservation filtrées au format Excel.
    - **Authentification :** Requiert un token admin valide.

3. Vues et ViewSets (Django REST Framework)

Pour chaque modèle et fonctionnalité, des vues (ou ViewSets) seront implémentées pour gérer la logique métier et l'interaction avec la base de données.

3.1 Vues d'Authentification

- `FirebaseLoginView` (APIView) :
    - Gère la réception du `id_token` Firebase.
    - Vérifie le token avec le SDK Admin Firebase.
    - Récupère ou crée l'utilisateur dans la base de données locale (`Utilisateur` modèle).

    - Génère et retourne un token d'authentification interne (JWT) pour les requêtes subséquentes au backend.

3.2 ViewSets Standard (ModelViewSet)

- `UtilisateurViewSet` :
    - Opérations CRUD pour les utilisateurs.
    - Permissions : `IsAuthenticated` pour `me/`, `IsAdminUser` pour `list` / `create` / `update` / `delete`.
- `SalleViewSet` :
    - Opérations CRUD pour les salles.
    - Permissions : `AllowAny` pour `list` / `retrieve`, `IsAdminUser` pour `create` / `update` / `delete`.
- `CreneauViewSet` :
    - Opérations CRUD pour les créneaux.
    - Permissions : `AllowAny` pour `list` / `retrieve`, `IsAdminUser` pour `create` / `update` / `delete`.
- `ReservationViewSet` :
    - Opérations CRUD pour les réservations.
    - Permissions : `IsAuthenticated` pour `list` / `create` / `retrieve` (pour l'utilisateur courant), `IsAdminUser` pour toutes les opérations sur toutes les réservations.
    - Logique de validation du statut (changement de 'En attente' à 'En attente documents' après création).
- `GroupePiscineViewSet` :
    - Opérations CRUD pour les groupes de piscine.
    - Permissions : `AllowAny` pour `list` / `retrieve`, `IsAdminUser` pour `create` / `update` / `delete`.
- `ActiviteViewSet` :
    - Opérations CRUD pour les activités.

    - Permissions : `AllowAny` pour `list` / `retrieve`, `IsAdminUser` pour `create` / `update` / `delete`.
- `CompetitionViewSet` :
    - Opérations CRUD pour les compétitions.
    - Permissions : `AllowAny` pour `list` / `retrieve`, `IsAdminUser` pour `create` / `update` / `delete`.

3.3 Vues Personnalisées et Logique Métier

- `ReservationUploadJustificatifView` (APIView) :
    - Gère le téléchargement du reçu de paiement et du code de validation.
    - Met à jour le statut de la réservation à 'En attente de validation admin'.
- `AdminReservationValidateView` (APIView) :
    - Valide une réservation après vérification (reçu, quotas).
    - Génère le QR code et le code interne de réservation.
    - Envoie des notifications.
- `AdminReservationRejectView` (APIView) :
    - Refuse une réservation.
    - Envoie des notifications.
- `DashboardStatsView` (APIView) :
    - Récupère les données agrégées pour le tableau de bord.
    - Calcule les indicateurs de fréquence pour la piscine.
- `DashboardExportExcelView` (APIView) :
    - Génère un fichier Excel des réservations filtrées.

4. Intégration Firebase Authentication

Le backend ne gérera pas directement les identifiants et mots de passe des utilisateurs. Il s'appuiera sur Firebase Authentication pour cela. Le processus sera le suivant :

1. **Frontend s'authentifie avec Firebase :** L'application mobile/web utilise le SDK Firebase pour authentifier l'utilisateur (email/mot de passe, Google, etc.).
2. **Frontend obtient un `id_token` Firebase :** Après une authentification réussie, Firebase fournit un `id_token` (JWT).
3. **Frontend envoie le `id_token` au Backend :** Pour accéder aux ressources protégées du backend, le frontend inclura ce `id_token` dans l'en-tête `Authorization` (Bearer Token).
4. **Backend vérifie le `id_token` :** Le backend utilisera le SDK Admin Firebase pour vérifier la validité et l'intégrité du `id_token` reçu. Cela inclut la vérification de la signature, de l'expiration, et de l'émetteur.
5. **Backend mappe l'utilisateur Firebase à l'utilisateur interne :** Une fois le token validé, le backend extrait l'UID Firebase de l'utilisateur. Il cherchera un utilisateur correspondant dans sa base de données locale (`Utilisateur` modèle). Si l'utilisateur n'existe pas, il sera créé (lors de la première connexion). Si l'utilisateur existe, son profil sera mis à jour si nécessaire.
6. **Backend autorise la requête :** Après avoir identifié l'utilisateur, le backend appliquera les permissions et autorisations basées sur le rôle de l'utilisateur (admin ou utilisateur standard) pour traiter la requête.

Cette approche garantit que la gestion de l'authentification est déléguée à Firebase, simplifiant le backend tout en maintenant la sécurité.