ChatGPT

Spécifications techniques pour une plateforme de padel gamifiée

Introduction et objectifs

Le padel connaît une croissance rapide en Europe et de nombreux joueurs souhaitent organiser des matchs, suivre leur progression et participer à des tournois. L'objectif de ce projet est de concevoir un **site web** et une **application mobile Android** permettant de gérer des parties de padel de façon ludique, inspirée de jeux compétitifs comme *League of Legends* ou *Rocket League*. La plateforme doit offrir un système de classement Elo, des fonctionnalités de matchmaking et de réservation de courts, un portail pour les tournois, des systèmes de recommandations/notation des adversaires, ainsi qu'une gamification complète (paliers, missions, XP, avatars, etc.).

Compte tenu d'un budget limité, les solutions proposées s'appuient principalement sur des logiciels libres ou gratuits et des offres de type « free tier » pour l'hébergement. Les choix techniques visent également la **scalabilité** et une maintenance aisée.

Vue d'ensemble de l'architecture

- **Front-end web :** site responsive (desktop + mobile) construit avec **React** et **Next.js** pour le routage et le rendu côté serveur. React permet de créer des interfaces réactives, et la bibliothèque React Native (utilisée pour l'application mobile) partage les concepts de composants et d'état.
- **Application mobile :** développée avec **React Native**, un framework qui permet de **créer des applications natives pour Android et iOS** en JavaScript tout en conservant des performances natives [1]. Les composants React Native s'appuient sur les API natives des plateformes, ce qui évite de compromettre l'expérience utilisateur [2].
- **Backend/API :** service RESTful écrit en **Node.js** avec **Express**. Express est un framework web léger et non prescriptif qui fournit une base robuste pour des applications Web et des API tout en restant minimaliste [3]. Les API doivent être documentées (Swagger/OpenAPI) afin d'être consommées par le site, l'application et d'éventuels services externes.
- **Base de données : PostgreSQL**, système de gestion de bases de données relationnelles open-source réputé pour sa fiabilité et ses performances [4]. Il permet de combiner des données relationnelles (joueurs, matchs, réservations) et de stocker des structures JSON lorsqu'un schéma flexible est nécessaire. Des index géospatiaux (extension PostGIS) seront utilisés pour optimiser les requêtes basées sur la localisation.
- **Carte et géolocalisation :** utilisation de **Leaflet**, une bibliothèque JavaScript open-source légère (≈42 Ko) conçue pour créer des cartes interactives mobiles [5]. Elle s'intègre aisément à React et propose des marqueurs, pop-ups et géoJSON [6]. Les tuiles proviendront d'**OpenStreetMap** et le géocodage (conversion d'adresses en coordonnées) sera assuré par **Nominatim**, un moteur de recherche d'adresses open-source qui « permet de rechercher des données OSM par nom ou adresse (géocodage) et de générer des adresses à partir de coordonnées » [7].
- **Calculs de distance :** pour proposer des adversaires ou des courts proches, le backend utilisera la **formule de Haversine**. Cette formule calcule la distance entre deux points sur une sphère à partir de leur latitude et de leur longitude [8], ce qui convient à des distances géographiques.

- **Notifications et temps réel :** utilisation de **WebSockets** (Socket.io) pour recevoir en temps réel les invitations de match, les confirmations de score et les notifications de mission. Pour les notifications push sur mobile, on peut s'appuyer sur **Firebase Cloud Messaging (FCM)**, inclus dans le plan gratuit Spark de Firebase [9].
- **Hébergement et déploiement :** déploiement via un **VPS** ou une plateforme PaaS (Railway, Render) utilisant des conteneurs Docker pour isoler les services (API, base de données, client). La base PostgreSQL peut être hébergée sur un service gratuit (ElephantSQL free tier) ou autopilotée. Les assets (images d'avatars, sons) seront stockés dans un stockage objet compatible S3 (Backblaze B2, MinIO).

1 Système de classement Elo

1.1 Algorithme et principes

Le **système Elo** attribue à chaque joueur une note numérique reflétant son niveau. La différence de note entre deux joueurs sert à prédire l'issue d'un match : un joueur ayant 100 points de plus qu'un autre a une probabilité d'environ 64 % de gagner, et une différence de 200 points correspond à 76 % de chances [10]. Après chaque rencontre, des points sont transférés du perdant au gagnant. Si le joueur favori gagne, peu de points sont échangés ; en cas de victoire surprise du joueur moins bien classé, beaucoup de points sont transférés [11]. Ce mécanisme corrige les notes qui sont trop élevées ou trop basses [12].

Dans cette plateforme :

- **Classement individuel :** chaque joueur possède un score Elo initial (par exemple 1000). Le score est mis à jour après chaque match en fonction du résultat (victoire, défaite) et du niveau des adversaires. Un coefficient « K » variable (plus élevé pour les débutants) accélère la convergence.
- **Classement par paire :** pour les matchs en double, une paire est considérée comme une entité distincte avec sa propre note Elo, sans impacter le score individuel. Le calcul peut appliquer la même formule Elo en considérant la moyenne des notes individuelles de chaque équipe.
- **Paliers visuels :** des seuils définis (Bronze : <1200 pts, Argent : 1200–1399 pts, Or : 1400–1599, Platine : ≥1600) déterminent un **palier** affiché dans l'interface. Les joueurs voient leur progression et obtiennent des **badges** lors du franchissement d'un palier.
- **Historique** : chaque mise à jour de score est enregistrée dans une table `rating_history` (joueur, note avant/après, variation, date, match). Cela permet d'afficher des graphiques d'évolution (Sparkline) et de justifier les changements de niveau.

1.2 Flux utilisateur

1. **Inscription et profil :** le joueur crée un compte, renseigne un pseudo, une localisation (géocodée via Nominatim) et choisit un avatar.
2. **Consultation du classement :**
3. **Top global/régional :** pages affichant le classement général et par région (via champ `region` sur l'utilisateur). Les filtres permettent de voir les classements par tranche d'âge ou sexe si ces données sont collectées.
4. **Mon profil :** affichage de la note actuelle, du palier, des badges obtenus et de la progression (courbe de variation). Un bouton « Historique des matchs » ouvre la liste des matchs joués avec résultat et variation de score.
5. **Visionnage d'un autre joueur :** on peut consulter la fiche d'un joueur, voir son palier et ses statistiques (taux de victoire, points marqués, dernière activité), envoyer une demande d'ami ou proposer un match amical.

1.3 Modèle de données principal (Elo)

Entité	Attributs clés	Description
Player	`id`, `username`, `email`, `password_hash`, `avatar_url`, `home_lat`, `home_lon`, `elo_score`, `xp`, `level`, `tier`, `region`	Représente un utilisateur. Le `tier` est calculé d'après `elo_score`.
Pair	`id`, `player1_id`, `player2_id`, `elo_score`, `created_at`	Unique pour chaque combinaison de deux joueurs. Utilisé pour le classement en double.
Match	`id`, `pair_a_id`, `pair_b_id`, `score_a`, `score_b`, `court_id`, `played_at`, `status` (proposé, confirmé, terminé), `validated` (booléen)	Enregistre une rencontre et le score.
RatingHistory	`id`, `player_id`, `match_id`, `rating_before`, `rating_after`, `rating_change`, `created_at`	Permet d'historiser les variations de la note Elo.

1.4 API / Backend

- `GET /players/{id}` : retourne le profil d'un joueur (note, palier, historique, statistiques).
- `GET /ranking?region=X&page=1` : retourne une liste triée par note Elo, filtrable par région/palier.
- `GET /matches/{id}` : renvoie les détails d'un match (joueurs, scores, statut, commentaires).
- `POST /matches` : enregistre un résultat de match. Le corps contient les identifiants des paires, le score et le consentement des deux équipes. Le backend vérifie que les deux équipes ont confirmé avant d'enregistrer le match et de calculer la mise à jour de l'Elo.

1.5 Comportements d'interface

- Le site et l'application affichent les paliers sous forme de **badges colorés** (Bronze, Argent…). La montée/descente de palier déclenche une **animation** et une **notification** (avec son). Les barres de progression montrent les points restants avant le palier suivant.
- La page de profil présente des **graphes d'évolution** (par exemple via la bibliothèque Chart.js). Les changements importants (gain/perte de plus de 100 pts) peuvent déclencher des "highlights" (confettis ou effets lumineux).
- Lorsqu'un joueur consulte la fiche d'un adversaire, un bouton « Défier » permet de lancer un matchmaking direct avec ce joueur en créant une demande de match.

2 Matchmaking intelligent

2.1 Concept

Le système de matchmaking doit proposer automatiquement une paire adverse en fonction du niveau et des disponibilités. Chaque recherche repose sur trois critères :

1. **Équilibre de niveau :** l'algorithme cherche des joueurs dont la différence de note Elo est faible afin de maximiser l'intérêt du match. La différence de 100 points correspond à 64 % de chances

de victoire pour le joueur mieux classé [10] ; le moteur peut définir un seuil maximum (ex. ±150 points) pour garantir des rencontres équilibrées.

2. **Proximité géographique :** les joueurs saisissent leur localisation et un rayon de recherche (ex. 20 km). La distance entre deux joueurs est calculée via la **formule de Haversine**, qui mesure la distance entre deux coordonnées géographiques [8] . Les courts sont aussi géolocalisés pour ajuster la distance moyenne.
3. **Disponibilités compatibles :** chaque joueur enregistre ses créneaux libres dans un agenda (jour/horaire). Le système cherche des intersections entre disponibilités des quatre joueurs potentiels et propose des créneaux communs.

2.2 Flux utilisateur

1. **Définition des préférences :** dans son profil, chaque joueur indique :
2. Rayon maximum (en km) pour rencontrer d'autres joueurs.
3. Plages horaires disponibles (ex. lundi 18–21 h, mercredi 20–22 h). Un calendrier simple (type date-picker) permet de gérer ces créneaux.
4. **Recherche de match :**
5. Le joueur choisit s'il souhaite trouver une partie **amicale** ou **compétitive**. Dans le cas d'une partie compétitive, les scores influenceront l'Elo.
6. Le système parcourt la base de données pour trouver des joueurs de niveau similaire disponibles dans le rayon défini. Il forme des paires en combinant les joueurs par proximité de score et de localisation.
7. Un **score d'appariement** est calculé pour chaque paire potentielle (pondération du différentiel de note, distance moyenne et compatibilité horaire). Les 3 meilleures propositions sont renvoyées.
8. **Présentation des propositions :** l'utilisateur voit :
9. Les joueurs proposés, leur palier et la distance par rapport au joueur.
10. Les créneaux de matchs possibles et les courts disponibles (voir section 3).
11. La probabilité de victoire (selon l'Elo) pour pimenter l'expérience.
12. **Acceptation :** quand les quatre joueurs acceptent une proposition (match et horaire), l'application crée un **match** en statut « proposé » et lance la réservation du court.

2.3 Modèle de données & API

Entité	Attributs clés	Description
AvailabilitySlot	`id`, `player_id`, `day_of_week`, `start_time`, `end_time`	Créneau de disponibilité hebdomadaire d'un joueur.
MatchSearchRequest	`id`, `player_id`, `match_type` (amical/compétitif), `radius_km`, `preferred_dates[]`	Objet temporaire pour initier une recherche de match.
MatchProposal	`id`, `request_id`, `pair_a_id`, `pair_b_id`, `suggested_time`, `distance_avg_km`, `rating_difference`, `score`	Proposition générée par l'algorithme. Contient un classement de pertinence.

Endpoints principaux :

- `POST /matchmaking/search` : enregistre une requête de matchmaking et renvoie une liste de propositions. Le calcul se fait côté serveur avec un script planifié ou un service en file d'attente.
- `POST /matchmaking/proposals/{proposalId}/accept` : un joueur accepte une proposition. Une fois les quatre joueurs ayant accepté, l'état passe à « confirmé » et une entrée `Match` est créée.
- `POST /matchmaking/proposals/{proposalId}/cancel` : refuse la proposition.

2.4 Interface

- Recherche assistée : l'utilisateur voit un **compteur** indiquant « Recherche d'adversaires… » avec des icônes animées. Une fois les propositions trouvées, elles apparaissent sous forme de **cartes** affichant les infos clés (photos de profils, niveau, distance, créneau). L'utilisateur peut glisser vers la droite pour accepter ou vers la gauche pour rejeter.
- Notifications en temps réel : lorsqu'une proposition est acceptée par tous, une notification (WebSocket et FCM) alerte les joueurs avec un son inspiré des jeux vidéo.

3 Réservation optimisée de courts

3.1 Contexte

La réservation de courts de padel est souvent fragmentée : des fournisseurs comme Playtomic, Matchi ou Padelmates proposent chacun leur propre interface et il est difficile de comparer les disponibilités [13]. Certaines initiatives agrègent ces informations en un seul calendrier [14]. Pour limiter les coûts, deux stratégies sont envisagées :

1. **Intégration d'API externes :** si des fournisseurs de réservation (Playtomic, Matchi, Playskan, etc.) proposent des API publiques ou accessibles sous conditions, l'application peut interroger ces API pour récupérer les créneaux et effectuer les réservations. Cette option permet de bénéficier d'un catalogue étendu mais suppose de négocier des accords ou de respecter des quotas.
2. **Back-office interne :** dans les zones où aucune API n'est disponible, la plateforme offre aux clubs un back-office simple pour saisir les disponibilités de leurs courts et accepter les réservations. L'application devient alors un canal de réservation pour ces clubs (modèle de marketplace). Le back-office peut être développé avec une interface administrative (ex. React Admin).

3.2 Algorithme de suggestion de créneau

Lorsqu'un match est confirmé via le matchmaking, l'algorithme de réservation recherche automatiquement un créneau optimal en se basant sur :

- **Distance moyenne des joueurs :** on calcule la distance entre chaque joueur et chaque court disponible via la formule de Haversine [8] et on en déduit la **distance moyenne**. Le court minimisant cette moyenne est favorisé.
- **Disponibilité des courts :** on interroge l'API externe ou la base interne pour connaître les courts libres à la date et à l'horaire proposés.
- **Préférences** : certains joueurs peuvent indiquer des clubs favoris ou des critères (terrain couvert, prix). Ces paramètres ajustent la pondération du choix.

L'algorithme génère une liste de `ReservationSuggestion` avec :

Attribut	Description
`court_id`	identifiant du court.
`address`, `lat`, `lon`	géolocalisation et adresse pour affichage sur carte.
`time_slot`	date et horaire proposé.
`distance_average_km`	distance moyenne des joueurs jusqu'au court.
`cost`	tarif du créneau (si disponible).
`score`	note de pertinence calculée à partir de la distance, du prix et des préférences.

3.3 API et flux utilisateur

- `GET /courts?lat=...&lon=...&radius=...` : retourne la liste des courts dans un rayon donné, avec leurs disponibilités.
- `POST /reservations/suggest` : prend en entrée les identifiants des joueurs et la date souhaitée, et renvoie une liste de suggestions (`ReservationSuggestion`).
- `POST /reservations/{id}/confirm` : confirme la réservation auprès du fournisseur (API externe ou base interne). L'état de la réservation passe à « confirmé » et un paiement peut être initié si nécessaire (intégration d'un service de paiement comme Stripe avec un mode test pour les petites structures).

3.4 Interface

- La page de proposition de créneaux affiche les courts sur une **carte interactive** Leaflet. Chaque marqueur indique le nom du club, le tarif et le temps de trajet estimé.
- Une liste à droite affiche les suggestions triées. Le joueur peut cliquer sur une suggestion pour zoomer sur la carte et confirmer. L'interface doit être intuitive pour éviter la complexité de la réservation multi-plateformes.

4 Exploration des tournois

4.1 Fonctionnalités

Les joueurs veulent participer à des tournois adaptés à leur niveau. La plateforme propose une page dédiée où l'on peut :

- Consulter **les tournois ouverts** selon son niveau Elo (un filtre "min Elo – max Elo"). Un joueur Bronze n'aura accès qu'aux tournois <1200, par exemple.
- Filtrer par date, format (simple/double), type (open, mixte), frais d'inscription.
- Afficher les tournois sous forme de **cartes** listant le nom, la localisation, la date, le niveau requis et un bouton « S'inscrire ».
- Visualiser une **carte interactive** (Leaflet) montrant l'emplacement des tournois. Les marqueurs sont colorés par niveau requis, et un clic affiche un pop-up avec les détails.
- S'inscrire directement si le nombre de places le permet. Le système vérifie que la note du joueur est dans la plage requise et qu'il n'est pas déjà inscrit à un autre tournoi à la même date.

4.2 Modèle de données & API

Entité	Attributs clés	Description
Tournament	`id`, `name`, `description`, `lat`, `lon`, `address`, `date_start`, `date_end`, `min_elo`, `max_elo`, `format` (simple/double), `organizer_id`	Représente un tournoi.
TournamentRegistration	`id`, `tournament_id`, `player_or_pair_id`, `registered_at`	Enregistrement d'un joueur ou d'une paire dans un tournoi.
Match (Tournoi)	`id`, `tournament_id`, `round`, `pair_a_id`, `pair_b_id`, `score_a`, `score_b`, `winner_id`	Permet de suivre l'arbre du tournoi.

Endpoints :

- `GET /tournaments?minElo=x&maxElo=y&dateRange=start,end` : liste des tournois filtrés.
- `GET /tournaments/{id}` : détails d'un tournoi et liste des participants inscrits.
- `POST /tournaments/{id}/register` : inscription d'un joueur ou d'une paire. Vérifie la plage Elo et la disponibilité.
- `GET /tournaments/map` : renvoie un GeoJSON des tournois pour l'intégration avec Leaflet.

4.3 Interface

- Un système d'onglets permet d'afficher les **prochains tournois** (par date) et les **tournois passés** où l'on peut consulter les résultats. Chaque carte de tournoi affiche un visuel (logo du club ou affiche du tournoi), des informations essentielles et un bouton d'inscription.
- La **carte interactive** Leaflet est située en haut ou sur le côté et se met à jour en fonction des filtres. Les marqueurs ont des infobulles claires et, en cliquant sur un tournoi, l'utilisateur voit les détails et peut s'inscrire.
- Une fonction « Suggérer un tournoi » permet aux organisateurs d'ajouter leurs tournois via un formulaire (champ d'adresse géocodé, date/heure, niveau requis, nombre de places). Un back-office admin peut valider ces propositions.

5 Système de recommandation et reporting post-match

Après chaque match, la plateforme doit encourager le fair-play et garantir l'intégrité des scores.

5.1 Flux de validation du résultat

1. **Saisie du score :** un joueur (capitaine) saisit le score dans l'application via un formulaire simple (nombre de sets et jeux). Ce score est stocké provisoirement dans l'entité `Match` avec le statut « en attente de confirmation ».
2. **Notification** : les adversaires reçoivent une notification les invitant à **valider ou contester** le score.
3. **Validation :** si tous confirment, le match passe à l'état « validé » et les mises à jour Elo sont appliquées. Si un joueur conteste le score, le statut devient « en litige » et l'équipe support ou l'administrateur doit intervenir (possibilité de joindre une preuve photo).

5.2 Notation des adversaires et recommandations

- **Notation** : après validation, chaque joueur peut attribuer des notes de 1 à 5 sur :
- Le **fair-play** (respect des règles, comportement sur le terrain).
- Le **niveau** (compétitivité, placement, technique).
- L'**envie de rejouer** avec cette personne.
- Ces notes sont moyennées et visibles sur le profil du joueur ; elles n'influencent pas directement le classement Elo mais peuvent être utilisées dans l'algorithme de matchmaking (par exemple, éviter d'associer des joueurs ayant un score de fair-play très bas).
- **Commentaires** : possibilité de laisser un commentaire public ou privé (pour l'admin). Les commentaires publics apparaissent sur la fiche du joueur (comme des avis).

5.3 Modèle de données & API

Entité	Attributs clés	Description
MatchResult	`id`, `match_id`, `submitted_by_id`, `score_a`, `score_b`, `submitted_at`	Score proposé en attente de confirmation.
MatchConfirmation	`id`, `match_id`, `player_id`, `status` (confirmé/contesté), `comment`	Enregistre la réponse de chaque joueur.
RatingReview	`id`, `match_id`, `reviewer_id`, `reviewed_player_id`, `fair_play`, `skill`, `replay_desire`, `comment`, `created_at`	Avis laissé après le match.

Endpoints :

- `POST /matches/{id}/submit-score` : soumission du score par un joueur.
- `POST /matches/{id}/confirm-score` : un joueur confirme ou conteste un score.
- `POST /matches/{id}/review` : enregistre une note et un commentaire pour chaque adversaire.

5.4 Interface

- **Double confirmation** : un bandeau rouge/orange indique qu'un match est en attente de confirmation. Les joueurs sont invités à valider, et un rappel apparaît après 24 h.
- **Notations rapides** : l'interface présente des curseurs ou étoiles pour évaluer les adversaires. Des feedbacks anonymes incitent au fair-play et renforcent la communauté.
- **Gestion des litiges** : si un litige est déclaré, un formulaire permet de joindre une preuve et un message. Les administrateurs peuvent arbitrer et ajuster le résultat.

6 Gamification et expérience utilisateur (UX)

La réussite de la plateforme repose sur une expérience ludique et motivante. L'inspiration vient des jeux vidéo compétitifs : missions, niveaux, récompenses visuelles, sons et feedback immédiats.

6.1 Missions et XP

- **Missions quotidiennes et hebdomadaires :** liste de tâches variant chaque jour/semaine (ex. « Joue 3 matchs », « Affronte un joueur Platine », « Participe à un tournoi »). Chaque mission a un niveau de difficulté, un nombre de points d'expérience (XP) et parfois une récompense cosmétique.
- **XP et niveaux :** le joueur accumule de l'XP en jouant, en remportant des matchs, en accomplissant des missions et en recevant de bonnes évaluations. Un niveau global reflète l'engagement sur la plateforme et débloque des **avatars personnalisés**, des **titres temporaires** ou des **skins** (couleurs de raquette, fonds de profil). La progression du niveau n'impacte pas l'Elo mais offre un sentiment d'accomplissement.
- **Titres temporaires et classements régionaux :** des classements hebdomadaires ou mensuels par région identifient les joueurs les plus actifs/performants. Les premiers obtiennent un titre spécial (ex. « Champion régional d'août »). Ces titres sont visibles pendant un mois.
- **Effets audio et visuels :** à chaque réussite (nouveau palier, mission accomplie), un son inspiré des jeux vidéo retentit et une animation s'affiche. Les notifications sont gérées via FCM ou WebSockets, avec la possibilité de les désactiver.

6.2 Modèle de données & API

Entité	Attributs clés	Description
Mission	`id`, `name`, `description`, `type` (quotidienne/hebdomadaire), `criteria` (JSON décrivant la condition), `reward_xp`, `reward_item_id`	Mission disponible pour les joueurs.
PlayerMission	`id`, `player_id`, `mission_id`, `progress`, `completed_at`	Suit la progression d'un joueur sur une mission.
Item	`id`, `name`, `type` (avatar, skin, titre), `image_url`, `rarity`	Récompenses cosmétiques.
PlayerItem	`id`, `player_id`, `item_id`, `owned_at`, `equipped`	Stocke les objets obtenus et ceux équipés.

Endpoints :

- `GET /missions/today` : renvoie les missions quotidiennes/hebdomadaires disponibles pour l'utilisateur.
- `POST /missions/{id}/progress` : le client signale la progression (par exemple, après chaque match, l'API incrémente la progression des missions correspondantes).
- `POST /missions/{id}/claim` : validation d'une mission et attribution de la récompense.
- `GET /items` : liste des éléments cosmétiques disponibles ou obtenus.
- `POST /items/{id}/equip` : permet de sélectionner un avatar/skin/titre.

6.3 Interface et UX

- **Tableau de bord** : affiche les missions en cours avec une jauge de progression et un bouton pour réclamer la récompense. Un compteur de temps indique la fin de la mission quotidienne/hebdomadaire.

- **Inventaire** : section permettant de visualiser et équiper les avatars, skins ou titres. Les objets rares sont mis en valeur (fond brillant, effet halo).
- **Effets sonores** : un catalogue de sons courts (non protégés par copyright) sera intégré ; les utilisateurs peuvent activer/désactiver ces sons dans les paramètres.
- **Accessibilité** : proposer un thème clair/sombre, adapter les animations pour les personnes sensibles (option de réduction des mouvements), et rendre l'interface responsive.

7 Choix techniques détaillés

7.1 Stack Front-end

- **React et Next.js** (web) : gestion du routage côté serveur, optimisation SEO et génération de pages statiques. React fournit une architecture de composants réutilisables.
- **React Native** (mobile) : permet de créer des applications natives Android/iOS en JavaScript tout en accédant aux API natives [2]. Grâce au partage de logique avec React, l'équipe gagne du temps et peut mutualiser des composants et librairies.
- **Bibliothèques UI** : utilisation de Tailwind CSS ou [Material UI] pour le web, et [React Native Paper] pour le mobile. Ces bibliothèques offrent des composants modulaires et cohérents.
- **Leaflet** pour la carte : open-source, léger et mobile-friendly [5] ; possibilité d'ajouter des marqueurs, des pop-ups et des calques GeoJSON [6].
- **Chart.js** ou **Recharts** pour les courbes d'évolution et statistiques.

7.2 Backend et API

- **Node.js + Express** : Express est un framework web rapide et minimaliste fournissant des fonctionnalités robustes pour les applications Web et les API [3]. Il offre un système de routage et un middleware flexible.
- **ORM (Object-Relational Mapper)** : utilisation de **Prisma** ou **TypeORM** pour mapper les entités vers PostgreSQL et générer des migrations. Cela réduit le risque d'incohérences et facilite les évolutions du schéma.
- **Validation et documentation** : utilisation de **Zod** ou **Joi** pour valider les requêtes ; génération automatique de la documentation API via **Swagger**.
- **Authentification** : mise en place d'un serveur OAuth2 via **Keycloak** (self-hosted et open-source) ou utilisation de **Firebase Authentication** en mode Spark (gratuit) qui offre une authentification par e-mail et réseaux sociaux [9]. La solution la plus économique est Keycloak (hébergement sur le même serveur) mais Firebase simplifie l'implémentation au prix d'une dépendance externe.
- **Notifications** : **WebSockets (Socket.io)** pour les notifications en temps réel sur le web ; **Firebase Cloud Messaging** pour les notifications push sur mobile, disponible sans coût initial dans le plan Spark [9].
- **Tests** : écriture de tests unitaires et d'intégration via **Jest** et **Supertest**. Mise en place d'une intégration continue (GitHub Actions) pour exécuter les tests et les lintings à chaque commit.

7.3 Base de données

- **PostgreSQL** : SGBDR open-source réputé pour sa fiabilité et sa robustesse [4]. L'extension **PostGIS** permettra de stocker des coordonnées géographiques et de calculer des distances (mise en œuvre du Haversine via `ST_DistanceSphere`).
- **Schéma** : tables normalisées (cf. sections précédentes) avec clés étrangères et index appropriés (index B-Tree pour l'ID, index GiST pour les colonnes géographiques). Les champs JSON (ex. `criteria` des missions) utilisent le type `jsonb`.

- **NoSQL complémentaire :** si l'on souhaite enregistrer des événements de jeu (logs d'actions, streams de chat), un moteur NoSQL comme **MongoDB** ou **Redis Streams** peut être ajouté. Toutefois, PostgreSQL avec la table `jsonb` peut suffire pour un budget limité.

7.4 Géolocalisation et calcul de distance

- **Nominatim** : moteur de géocodage open-source basé sur OpenStreetMap qui permet de rechercher des adresses et d'effectuer du reverse-geocoding [7]. Il est possible d'installer un serveur Nominatim local ou d'utiliser des fournisseurs tiers (Geoapify, LocationIQ) qui offrent des quotas gratuits.
- **Haversine** : la formule de Haversine calcule la distance sur une sphère à partir des latitudes et longitudes [8]. Dans PostgreSQL, la fonction `ST_DistanceSphere` (PostGIS) implémente cette formule ; côté JavaScript, des librairies comme `geolib` peuvent être utilisées.

7.5 Hébergement et déploiement

- **Conteneurisation** : chaque service (web, API, base de données, authentification) est empaqueté dans un conteneur Docker. Le déploiement peut se faire sur un **VPS** (DigitalOcean, Hetzner) ou des plateformes PaaS avec un free tier (Railway, Render, Fly.io). L'utilisation d'une architecture micro-services permet une montée en charge progressive.
- **CI/CD** : pipeline GitHub Actions pour le déploiement automatisé. Les migrations de base se lancent automatiquement en production après validation. Pour un budget très limité, les déploiements peuvent être manuels via `docker-compose`.
- **Sauvegardes et monitoring** : mise en place de sauvegardes quotidiennes de la base de données et de logs (journalisation avec Winston). Outils de monitoring open-source comme **Prometheus** et **Grafana** pour suivre l'usage et les performances.

8 Modèle relationnel global (synthèse)

Ci-dessous un résumé des principales entités et relations (les clés primaires sont en gras) :

Table	Clés et champs principaux	Relations
players	**id**, username, email, password_hash, home_lat, home_lon, region, elo_score, xp, level, tier, avatar_url	Un `player` peut appartenir à plusieurs `pairs` et possède plusieurs `availabilities`, `reviews`, `missions`.
pairs	**id**, player1_id, player2_id, elo_score	Clé unique (player1_id, player2_id). Relié à `matches` et `match_proposals`.
matches	**id**, pair_a_id, pair_b_id, score_a, score_b, court_id, played_at, validated, status	Lié à `rating_history`, `match_results`, `match_confirmations`, `rating_reviews`.
courts	**id**, name, address, lat, lon, club_id	Utilisé pour les réservations ; relié à `reservations` et `tournaments`.

Table	Clés et champs principaux	Relations
reservations	**id**, match_id, court_id, time_slot, status, cost	Une réservation est associée à un match et à un court.
tournaments	**id**, name, description, lat, lon, date_start, date_end, min_elo, max_elo, format	Contient les données de chaque tournoi.
tournament_registrations	**id**, tournament_id, player_or_pair_id	Enregistre la participation des joueurs aux tournois.
missions, **player_missions**, **items**, **player_items**	Tables liées à la gamification (voir section 6).	
rating_history	**id**, player_id, match_id, rating_before, rating_after, rating_change, created_at	Historique des variations Elo.
rating_reviews	**id**, match_id, reviewer_id, reviewed_player_id, fair_play, skill, replay_desire, comment	Avis post-match.
availability_slots	**id**, player_id, day_of_week, start_time, end_time	Disponibilités hebdomadaires pour le matchmaking.
match_proposals, **match_confirmations**	Tables d'organisation du matchmaking intelligent et de la validation des scores.	

Cette modélisation relationnelle peut évoluer selon les besoins (par exemple, ajouter des tables pour les salons de discussion ou la gestion des clubs).

9 Conclusion

Cette spécification technique définit les fonctionnalités clés pour une plateforme de padel gamifiée couvrant le **classement Elo**, le **matchmaking intelligent**, la **réservation optimisée**, l'**exploration de tournois**, la **validation des scores et la notation des adversaires**, ainsi qu'un puissant **système de gamification**. Les choix techniques s'appuient sur des technologies open-source et gratuites : Express/Node.js pour le backend [3], PostgreSQL pour la base de données [4], React/React Native pour les interfaces [2], Leaflet pour les cartes interactives [5] et Nominatim pour le géocodage [7]. Les algorithmes de distance utilisent la formule de Haversine [8], et le système de classement suit les principes du Elo où la différence de note détermine la probabilité de victoire [10] et les transferts de points [15].

En suivant cette architecture et ces spécifications, une équipe produit/développement dispose d'un plan détaillé pour lancer un site et une application mobile de padel gamifiés, avec des fonctionnalités complètes et une expérience utilisateur similaire à celle des jeux vidéo tout en restant respectueuse du budget.

[1] [2] React Native · Learn once, write anywhere
https://reactnative.dev/

[3] Express - Node.js web application framework
https://expressjs.com/

[4] PostgreSQL: The world's most advanced open source database
https://www.postgresql.org/

[5] [6] Leaflet - a JavaScript library for interactive maps
https://leafletjs.com/

[7] Nominatim - OpenStreetMap Wiki
https://wiki.openstreetmap.org/wiki/Nominatim

[8] Haversine formula - Wikipedia
https://en.wikipedia.org/wiki/Haversine_formula

[9] Firebase pricing plans | Firebase Documentation
https://firebase.google.com/docs/projects/billing/firebase-pricing-plans

[10] [11] [12] [15] Elo rating system - Wikipedia
https://en.wikipedia.org/wiki/Elo_rating_system

[13] [14] We Built the World’s First Padel Booking Aggregator — Here’s How | by Henri Happonen | Jun, 2025 | Medium
https://medium.com/@henrihapponen/we-built-the-worlds-first-padel-booking-aggregator-33c6511e212e