# Product Requirements Document

## Overview

A basketball league management system for community center leagues. Supports players, team managers, admins, and super admins. Priorities: **registration → scheduling → standings**.

---

## Goals

- Simplify team creation and registration.
- Centralize schedule visibility (league-wide and team-specific).
- Support fair competition with standings and results tracking.

---

## Scope

### In-Scope (MVP)

- Firebase Authentication
- League/Team browsing (public)
- Team creation & join flows
- Manager-in-team functionality (edit team, invite players)
- Payment status tracking (boolean)
- Team approval state (set by admin)
- Schedule PDFs (per-league, uploaded by admin)
- Team schedules (filtered from league schedule)

### Out-of-Scope (MVP)

- Real payment integration (Stripe, PayPal)
- Automated bracket generation
- Notification systems (email, SMS)
- Real-time scheduling UI

---

## User Roles & Permissions

| Role | Capabilities |
|---|---|
| Player | Join one team per league; view team page (roster, schedule, standings); see dues status. |
| Manager (team-scoped) | Edit team name/description; invite players; visually distinct in roster. |
| Admin | Manage leagues: approve teams, upload schedules, record results, view master roster, inspect teams & players. |
| Super Admin | Oversight: view all leagues, teams, players. Limited management for developers/department heads. |

---

## User Journeys

### Public

- View leagues → teams list (names, descriptions).
- View schedule (PDFs).
- View standings (placeholder).
- Prompt to log in for deeper features.

### Player

- Log in → see dashboard with their teams.

- Create Team → choose league, name, description → become manager.
- Join Team → code or invite link.
- Team Page → details, roster, invite (if manager), schedule, standings.

### Admin

- Log in → leagues managed by them.
- League Page → manage schedule (PDF), standings, results, master roster.
- Team Page → same as player + approve/unapprove, view player profiles.

### Super Admin

- Log in → all leagues → league manager + teams → rosters.
- Oversight only.

---

## Functional Requirements

1. **Authentication**
    - Firebase email/password and social providers.
    - Logged-in state required for create/join team.
2. **Leagues**
    - Public: view all leagues.
    - Admin: add schedules (PDFs), standings.
3. **Teams**
    - Create team (manager = creator).
    - Join team (invite code/link).

    - Team roster (players, manager marked).
    - Approval status (boolean).
    - Manager can edit team name/description.
4. **Membership**
    - One team per league per user.
    - Manager is a **membership-level attribute**, not global.
5. **Scheduling**
    - MVP: schedule PDF upload (league-level).
    - Team pages auto-filter schedule.
6. **Standings**
    - Admin can input results (scores).
    - League standings auto-update teams.

---

## Non-Functional Requirements

- Deployable on Vercel.
- Use Firebase Auth.
- Store state in Vercel KV (no-SQL).
- Initial schedules = PDFs (simple to implement).

---

## Roadmap

- **Phase 1 (MVP):** Registration flows, Firebase auth, rosters, dues/approval.
- **Phase 2:** League schedules (PDF uploads → team schedule views).

- **Phase 3:** Standings & scoring logs.
- **Phase 4:** Payments integration, notifications, bracket builder.

# PRD Addendum: Vercel KV Key Schema

## 0) Conventions

- **Notation:** `namespace:{id}:subkey`
- **IDs:** `userId` comes from Firebase Auth `uid`. `leagueId`, `teamId`, `gameId` are UUIDs (or slugs).
- **JSON values:** All values are JSON unless noted.
- **Edge-safety:** All keys compatible with Edge via `@vercel/kv`.
- **TTL:** Only ephemeral objects (invites) use TTL; everything else persists.
- **Denormalization:** Prefer small, focused lists that can be read in one round-trip for each screen.

---

## 1) Users & Profiles

### Keys

**`user:{userId}`** → `UserProfile`

```
 {
   "id": "firebase-uid",
   "email": "a@b.com",
   "name": "Raiyan G",
   "phone": "555-5555",
   "address": "…",
   "createdAt": "ISO"
}
```

**`user:{userId}:memberships`** → `MembershipSummary[]`
A *list across leagues*—each record says which team they're on in that league and whether they're manager.

```
[
  { "leagueId": "5v5", "teamId": "team-123", "isManager": false },
  { "leagueId": "4v4b", "teamId": "team-789", "isManager": true }
]
```

### Common ops

- Add/update profile on first login.
- When joining/creating a team, **append** the new membership (enforce "one team per league" before write).

---

## 2) Leagues (Public Browsing & Admin)

### Keys

**`league:index`** → `LeagueLite[]` (for `/leagues` page)

```
[
  { "id": "5v5", "name": "5v5" },
  { "id": "4v4b", "name": "4v4B" },
  { "id": "4v4w", "name": "4v4 Women" }
]
```

**`league:{leagueId}`** → `League`

```
{
  "id": "5v5",

  "name": "5v5",

  "schedulePdfUrl": "https://…/5v5-fall-2025.pdf",

  "standingsVersion": 3,            // bump to invalidate caches

  "createdAt": "ISO",

  "updatedAt": "ISO"

}
```

**`league:{leagueId}:teams`** → `TeamCard[]` (names & descriptions only, public)

```
 [

  { "teamId": "t1", "name": "Rockets", "description": "…" },

  { "teamId": "t2", "name": "Spurs", "description": "…" }

]
```

**`league:{leagueId}:standings`** → `StandingRow[]` (admin writes, public reads)

```
 [

  { "teamId": "t1", "wins": 5, "losses": 2, "pointsFor": 400,
"pointsAgainst": 350, "rank": 1 },

  { "teamId": "t2", "wins": 4, "losses": 3, "pointsFor": 360,
"pointsAgainst": 355, "rank": 2 }

]
```

**`league:{leagueId}:games`** → `Game[]` (optional now; used when you leave PDFs)

```
 [

  {
    "id": "g1",
    "leagueId": "5v5",
    "startAt": "ISO",
    "location": "Main Gym",
    "homeTeamId": "t1",
    "awayTeamId": "t2",
    "status": "SCHEDULED",  // or FINAL
    "homeScore": null,
    "awayScore": null,
    "round": 1
  }
]
```

### Common ops

- Admin updates `schedulePdfUrl` and standings; bump `standingsVersion`.
- Public league page reads `league:{leagueId}`, `league:{leagueId}:teams`, `league:{leagueId}:standings`.
- When you add in-app scheduling, write to `league:{leagueId}:games` and (see §4) team-specific denorm lists.

---

## 3) Teams (Registration, Roster, Approval)

### Keys

**`team:{teamId}`** → `Team`

```
{
  "id": "t1",
  "leagueId": "5v5",
  "name": "Rockets",
  "description": "…",
  "managerUserId": "uid-a",
  "approved": false,              // set by admin
  "rosterLimit": 8,
  "createdAt": "ISO",
  "updatedAt": "ISO"
}
```

**`team:{teamId}:roster`** → `RosterMember[]` (public *names* only; payment status omitted)

```
[
  { "userId": "uid-a", "displayName": "Aliyah S.", "isManager": true,
"joinedAt": "ISO" },
  { "userId": "uid-b", "displayName": "Devon K.", "isManager": false,
"joinedAt": "ISO" }
]
```

**`team:{teamId}:roster:private:{userId}`** → `RosterPrivate` (per-member privacy fields)

```
{ "paymentStatus": "PAID", "notes": "" }
```

- Privacy: Payment data is **not** stored on the public roster list.

**`league:{leagueId}:teamIds`** → `string[]` (index for admins to iterate)

```
["t1","t2","t3"]
```

### Constraints & checks

- **One team per league per user:** Before adding a membership, scan `user:{userId}:memberships` for same `leagueId`.
- **Roster cap (8):** Read `team:{teamId}:roster`, reject if `length >= 8`.
- **Manager is team-scoped:** `managerUserId` on the team, and `isManager` on that user's roster record only for this team.

### Common ops

- Create team: write `team:{teamId}`, push to `league:{leagueId}:teams`, append to `league:{leagueId}:teamIds`, create roster with manager, and update `user:{userId}:memberships`.
- Edit team (manager-only): update `team:{teamId}.name/description`.
- Approve team (admin-only): set `team:{teamId}.approved = true`.

---

## 4) Scheduling (PDF now; in-app later)

### MVP (PDF only)

- Already covered via `league:{leagueId}.schedulePdfUrl`.
- Team page links to league PDF and optionally anchors to team if the PDF supports anchors.

### In-app (when you add it)

- **`league:{leagueId}:games`** (see §2) is authoritative.
- **Denormalized per-team list** for fast team page reads:

**`team:{teamId}:games`** → `Game[]` (subset filtered where team is home/away)

```
 [
   { "id": "g1", "startAt": "ISO", "location": "Main Gym", "opponentTeamId":
"t2", "status": "SCHEDULED" }
]
```

- On admin update (create/edit result), write to `league:{leagueId}:games` and update both teams' `team:{teamId}:games`.
- Team page calls only `team:{teamId}:games` for performance.

---

## 5) Standings & Results

### Keys

- **`league:{leagueId}:standings`** → `StandingRow[]` (see §2)

**`team:{teamId}:results`** → `TeamResult[]` (history on team page)

```
 [
   { "gameId": "g1", "opponentTeamId": "t2", "homeAway": "H", "scoreFor": 60,
"scoreAgainst": 55, "won": true, "playedAt": "ISO" }
]
```

### Write flow (admin)

- Input result once:
    1. Update `league:{leagueId}:games` entry (`FINAL`, scores).
    2. Update `league:{leagueId}:standings` table (recompute rank).
    3. Append to **both** teams' `team:{id}:results`.

4. (Optional) Update `team:{id}:games` to mark status `FINAL`.

---

## 6) Invites (Join via Link / Code)

### Keys

**`invite:token:{hash}`** → `{ teamId, createdBy, uses, maxUses?, expiresAt? }` (TTL set by `ex`)

```
{ "teamId": "t1", "createdBy": "uid-a", "uses": 0 }
```

- Stored by **hash** of the token (`sha256(base64url)`).
- **TTL**: 14 days (typical).
- On consume: validate, enforce roster cap & "one team per league", then **delete** (one-time) or increment `uses` (if multi-use later).

**`invite:code:{code}`** → `{ teamId, createdBy }` (TTL 14 days)

```
{ "teamId": "t1", "createdBy": "uid-a" }
```

- On consume: delete (one-time).
- Rate-limit attempts (see §10).

### Manager UI constraints

- Allow generating invites only while `roster.length < 8`.
- If 8/8 reached, return error on create invite.

---

## 7) Approvals & Dues (MVP boolean)

### Keys

- **`team:{teamId}.approved`** (inside team object, §3).
- **`team:{teamId}:roster:private:{userId}.paymentStatus`** → `"PAID" | "UNPAID"`

### Admin “Master Roster” view (league)

**`league:{leagueId}:players`** → `LeaguePlayerRow[]`

```
 [
   {
     "userId": "uid-a",
     "displayName": "Aliyah S.",
     "teamId": "t1",
     "teamName": "Rockets",
     "isManager": true,
     "paymentStatus": "PAID"
   }
]
```

- This is a **derived index** the admin page reads in one trip. Update it whenever memberships or payment statuses change.

---

## 8) Admin & Super Admin Indexes

### Keys

**`admin:{userId}:leagues`** → `string[]` (IDs this admin manages)

```
 ["5v5","4v4b"]
```

- **`superadmin:index`** → `string[]` (all leagues) or reuse `league:index`.

---

## 9) Search & Display Helpers

Optional helper lists (denormalizations) to keep reads fast:

- **`team:index`** → `{ teamId, leagueId, name }[]` (for super admin quick lookup)
- **`user:{userId}:teams`** → `{ teamId, leagueId, name, isManager }[]` (redundant with `memberships` but useful for UI)
- **`league:{leagueId}:managers`** → `{ userId, teamId, teamName }[]` (admin-only index)

Keep all helper indexes **authoritative to the base** (`team`, `roster`, `memberships`). Rebuild on write or use small transactional updates.

---

## 10) Rate Limiting (recommended)

To prevent brute-force on codes:

- **`ratelimit:join-code:{ip}`** → count (TTL 60s)
  - Check / increment on each **join-by-code** attempt; reject if > N/min.
- **`ratelimit:invite-create:{userId}`** → count (TTL 60s)
  - Throttle invite generation.

You can store integers via `kv.incr(key)` with expiry via `ex`.

---

## 11) Typical Read/Write Patterns (pseudo-code)

### Create Team (Player → Manager)

1. Assert user **not** in a team for this league: scan `user:{uid}:memberships`.

2. Create `team:{teamId}`.
3. Push `{ teamId, name, description }` to `league:{leagueId}:teams`.
4. Append `teamId` to `league:{leagueId}:teamIds`.
5. Create `team:{teamId}:roster` with manager member.
6. Append to `user:{uid}:memberships`.
7. Update `league:{leagueId}:players` (derived).

### Join by Token/Code

1. Resolve token/code → `teamId`.
2. Check one-team-per-league (via `user:{uid}:memberships`).
3. Check `team:{teamId}:roster.length < 8`.
4. Append roster member; write `team:{teamId}:roster:private:{uid}` with `{ paymentStatus: "UNPAID" }`.
5. Append to `user:{uid}:memberships`.
6. Update `league:{leagueId}:players` (derived).
7. Delete invite record (one-time).

### Approve Team (Admin)

1. Set `team:{teamId}.approved = true`.
2. (Optional) Recompute `league:{leagueId}:standings` if approval gates inclusion.

### Record Result (Admin)

1. Update game in `league:{leagueId}:games` (FINAL, scores).
2. Update `team:{home}:results` & `team:{away}:results`.

3. Recompute & write `league:{leagueId}:standings`.

---

## 12) Types (reference)

```
type RosterMember = {
  userId: string;
  displayName: string;
  isManager: boolean;
  joinedAt: string; // ISO
};

type RosterPrivate = {
  paymentStatus: 'PAID' | 'UNPAID';
  notes?: string;
};

type League = {
  id: string;
  name: string;
  schedulePdfUrl?: string;
  standingsVersion?: number;
  createdAt: string;
  updatedAt: string;
};
```

```
type Team = {
  id: string;
  leagueId: string;
  name: string;
  description?: string;
  managerUserId: string; // team-scoped authority
  approved: boolean;
  rosterLimit: number; // 8
  createdAt: string;
  updatedAt: string;
};

type StandingRow = {
  teamId: string;
  wins: number;
  losses: number;
  pointsFor: number;
  pointsAgainst: number;
  rank: number;
};

type Game = {
  id: string;
  leagueId: string;

  startAt: string; // ISO

  location: string;

  court?: string;

  status: 'SCHEDULED' | 'FINAL';

  homeTeamId: string;

  awayTeamId: string;

  homeScore?: number;

  awayScore?: number;

  round?: number;

};
```

---

## 13) Security & Privacy Notes

- **Payment status is private**: keep it **only** in `team:{teamId}:roster:private:{userId}` and in admin aggregated lists (not public roster).
- **Manager capability** is enforced by `team:{teamId}.managerUserId === userId` and/or `isManager` flag on that roster member.
- **One-team-per-league** is enforced **before** writes by checking `user:{userId}:memberships`.

---

## 14) Migration Thoughts (later)

- If/when you move to SQL, these keys map 1:1 to relational tables: `users`, `leagues`, `teams`, `memberships`, `games`, `standings`, plus materialized views for admin summaries.

- Keep ID shapes stable now (UUIDs) to ease migration.