M2DG v1 — Technical Specification

Backend: Supabase (Postgres + Auth + Storage + Edge Functions) • Client: Flutter

Core rules: 100m radius • 30-min cooldown • daily streak (1/day) • Called Next queue

1) System Architecture

- **Flutter app** handles UI, QR scan, location permission, local notifications, and countdown timers.
- **Supabase Postgres** stores users, courts, sessions, check-ins, streaks, leaderboards, and queue.
- **RLS** prevents spoofing: clients can’t mark a check-in verified or modify leaderboards directly.
- **Edge Functions** verify distance + cooldown and perform atomic writes (check-in + session + streak).
- **Scheduled jobs** (cron) refresh leaderboard snapshots for fast reads.

2) Data Model (Tables)

2.1 Required Extensions

```
create extension if not exists postgis;
create extension if not exists pgcrypto;
```

2.2 Tables Overview

Table	Purpose	Key Columns
profiles	public user profile	id (uuid auth), username (unique), display_name, home_court_id
courts	court directory	id, name, city, state, geom (geography point), qr_code_id
court_qr_codes	QR tokens per court	id, court_id, token_hash, active
checkins	every attempt	id, user_id, court_id, attempted_at, verified, reject_code, distance_
court_sessions	active/ended session state	id, user_id, court_id, started_at, ended_at, cooldown_until
streaks	daily streak tracking	user_id, last_day (date), current_streak, best_streak
called_next_queue	per-court queue	id, court_id, user_id, position, created_at, expires_at
leaderboard_snapshots	fast reads	scope (court/city/global), scope_id, rank, user_id, score, as_of

2.3 SQL DDL (Copy/Paste)

```
-- PROFILES
create table if not exists public.profiles (
  id uuid primary key references auth.users(id) on delete cascade,
  username text unique not null,
  display_name text,
  avatar_url text,
  home_court_id uuid null,
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now()
);

-- COURTS
create table if not exists public.courts (
  id uuid primary key default gen_random_uuid(),
  name text not null,
  city text not null,
  state text not null,
  address text,
  geom geography(point, 4326) not null,
  is_active boolean not null default true,
  created_at timestamptz not null default now()
);

-- QR CODES (hashed tokens)
create table if not exists public.court_qr_codes (
  id uuid primary key default gen_random_uuid(),
  court_id uuid not null references public.courts(id) on delete cascade,
  token_hash text not null,
  active boolean not null default true,
  created_at timestamptz not null default now(),
  unique (court_id)
);

-- CHECKINS (every attempt)
create table if not exists public.checkins (
  id uuid primary key default gen_random_uuid(),
  user_id uuid not null references auth.users(id) on delete cascade,
  court_id uuid not null references public.courts(id) on delete cascade,
  attempted_at timestamptz not null default now(),
  verified boolean not null default false,
  reject_code text,
  distance_m integer,
  metadata jsonb not null default '{}'::jsonb
);

create index if not exists idx_checkins_user_time on public.checkins(user_id, attempted_at desc);
create index if not exists idx_checkins_court_time on public.checkins(court_id, attempted_at desc)

-- COURT SESSIONS
create table if not exists public.court_sessions (
  id uuid primary key default gen_random_uuid(),
  user_id uuid not null references auth.users(id) on delete cascade,
  court_id uuid not null references public.courts(id) on delete cascade,
  started_at timestamptz not null default now(),
  ended_at timestamptz,
  cooldown_until timestamptz,
  is_active boolean not null default true
);

create index if not exists idx_sessions_user_active on public.court_sessions(user_id, is_active);

-- STREAKS (1 row per user)
create table if not exists public.streaks (
  user_id uuid primary key references auth.users(id) on delete cascade,
  last_day date,
  current_streak integer not null default 0,
  best_streak integer not null default 0,
  updated_at timestamptz not null default now()
);

-- CALLED NEXT QUEUE
create table if not exists public.called_next_queue (
  id uuid primary key default gen_random_uuid(),
  court_id uuid not null references public.courts(id) on delete cascade,
  user_id uuid not null references auth.users(id) on delete cascade,
  position integer not null,
  created_at timestamptz not null default now(),
  expires_at timestamptz not null
);

create unique index if not exists uq_queue_court_position on public.called_next_queue(court_id, pos
create unique index if not exists uq_queue_court_user on public.called_next_queue(court_id, user_ic

-- LEADERBOARD SNAPSHOTS
create table if not exists public.leaderboard_snapshots (
  id uuid primary key default gen_random_uuid(),
  scope text not null check (scope in ('court','city','global')),
  scope_id text not null, -- court_id for court, city|state for city, 'global' for global
  rank integer not null,
  user_id uuid not null references auth.users(id) on delete cascade,
  score integer not null,
  as_of timestamptz not null default now()
);

create index if not exists idx_lb_scope on public.leaderboard_snapshots(scope, scope_id, rank);
```

3) Row Level Security (RLS)

3.1 Principles

- Client can read public courts and leaderboard snapshots.
- Client can read/write its own profile.
- Client can insert check-in attempts, but **only Edge Function** can set verified=true and create sessions/streak updates.
- Queue actions are mediated via Edge Function to avoid spam and guarantee ordering.

3.2 Enable RLS + Policies (Copy/Paste)

```
-- Enable RLS
alter table public.profiles enable row level security;
alter table public.courts enable row level security;
alter table public.court_qr_codes enable row level security;
alter table public.checkins enable row level security;
alter table public.court_sessions enable row level security;
alter table public.streaks enable row level security;
alter table public.called_next_queue enable row level security;
alter table public.leaderboard_snapshots enable row level security;

-- PROFILES: user can select/update own
create policy "profiles_select_own" on public.profiles
for select using (auth.uid() = id);

create policy "profiles_update_own" on public.profiles
for update using (auth.uid() = id);

create policy "profiles_insert_own" on public.profiles
for insert with check (auth.uid() = id);

-- COURTS: public read
create policy "courts_select_public" on public.courts
for select using (is_active = true);

-- QR CODES: no public access (Edge Function uses service role)
-- (No select policy -> blocked)

-- CHECKINS:
-- users can insert their own attempts but cannot verify them
create policy "checkins_insert_own_unverified" on public.checkins
for insert with check (auth.uid() = user_id and verified = false);

-- users can read their own checkins
create policy "checkins_select_own" on public.checkins
for select using (auth.uid() = user_id);

-- disallow client updates/deletes (no update/delete policies)

-- SESSIONS: user can read own session records
create policy "sessions_select_own" on public.court_sessions
for select using (auth.uid() = user_id);

-- STREAKS: user can read own streak
create policy "streaks_select_own" on public.streaks
for select using (auth.uid() = user_id);

-- QUEUE: user can read queue for a court (public view)
create policy "queue_select_public" on public.called_next_queue
for select using (true);
-- no direct insert/update/delete from client (Edge Function only)
-- LEADERBOARD SNAPSHOTS: public read
create policy "lb_select_public" on public.leaderboard_snapshots
for select using (true);
```

4) Edge Functions (Supabase)

4.1 Functions list

- **verify_checkin**: QR token + GPS -> verify distance, enforce cooldown, write checkin + session + streak atomically.
- **clock_out** (optional v1): end session when user leaves (also can auto-end via background/geofence).
- **call_next**: add user to court queue if eligible; assign next available position; rate-limit.
- **get_court_state**: returns court queue + Court Champ + local leaderboard preview (optional).

4.2 verify_checkin — Contract

```
POST /functions/v1/verify_checkin
Headers: Authorization: Bearer <user_jwt>
Body:
{
  "qrToken": "<raw token from QR>",
  "lat": 32.0809,
  "lng": -81.0912,
  "clientTime": "2026-01-23T03:00:00Z"
}

Response 200:
{
  "verified": true,
  "courtId": "<uuid>",
  "distanceM": 42,
  "cooldownUntil": "2026-01-23T03:30:00Z",
  "streak": { "current": 5, "best": 9, "countedToday": true }
}

Response 409 (cooldown):
{ "verified": false, "reason": "COOLDOWN", "remainingSeconds": 812 }
```

4.3 verify_checkin — Server Logic (Pseudo)

```
1) auth: get userId from JWT
2) lookup court via QR token hash (secure compare)
3) compute distance = ST_Distance(court.geom, makePoint(lat,lng)::geography)
4) if distance > 100m -> reject_code = OUT_OF_RANGE
5) check cooldown:
   - find latest active session or latest cooldown_until for user
   - if now < cooldown_until -> reject_code = COOLDOWN + remainingSeconds
6) insert checkins row (verified=false initially)
7) if verified:
   - update that checkin verified=true, distance_m
   - upsert court_sessions: set is_active=true, started_at=now, cooldown_until=now + 30 min
   - update streaks (1/day):
       day = (now at user_tz)::date
       if last_day is null -> current=1
       elif day == last_day -> no increment
       elif day == last_day + 1 -> current += 1
       else -> current=1
       best = max(best, current)
8) return payload with cooldownUntil, streak info
```

4.4 call_next — Contract

```
POST /functions/v1/call_next
```

```
Body: { "courtId": "<uuid>" }

Response 200:
{ "ok": true, "position": 3, "expiresAt": "2026-01-23T04:20:00Z" }

Response 403:
{ "ok": false, "reason": "NOT_CHECKED_IN" }
```

5) Leaderboards Job

Approach

- Compute scores from verified check-ins (e.g., total verified check-ins last 30 days, or all-time for MVP).
- Write top N into leaderboard_snapshots per scope.
- Run on schedule (e.g., every 15 minutes) to keep reads instant.

Score definition (MVP)

MVP scoring recommendation: **All-time verified check-ins** (simple + stable). Later add seasonal/monthly boards.

6) Flutter App — Screen Map + Navigation

6.1 Screens (MVP)

- **Auth**: Sign In / Sign Up
- **Courts**: Map + List (tabs), Search/Filter
- **Court Detail**: distance, “Check In”, queue, Court Champ, leaderboard preview
- **QR Scan**: scan flow + confirm
- **Active Session**: countdown timer, “Clock Out” messaging, status
- **Profile**: username, home court, streak, stats
- **Leaderboards**: Court / City / Global tabs

6.2 Navigation

```
Bottom Nav (4):
1) Courts
2) Check-In (opens QR Scan)
3) Leaderboards
4) Profile
```

6.3 Suggested Flutter Folder Structure

```
lib/
  app/
    router.dart
    theme/
    di/
  features/
    auth/
      screens/
      data/
    courts/
      screens/ (courts_map.dart, courts_list.dart, court_detail.dart)
      data/
      widgets/
    checkin/
      screens/ (qr_scan.dart, active_session.dart)
      data/
    leaderboards/
      screens/
      data/
    profile/
      screens/
      data/
  shared/
```

```
    supabase/
    location/
    notifications/
    widgets/
  main.dart
```

7) Automation Setup (Developer-Friendly)

Recommended workflow

- Use a mono-repo: /app (Flutter) + /supabase (migrations + functions).
- Use Supabase CLI to version DB schema & Edge Functions in Git.
- Use migrations for every DB change; never edit production tables manually without a migration.
- Create make-like scripts for repeatable commands (setup, start, deploy).

```
repo/
  app/                # Flutter
  supabase/
    migrations/       # SQL migrations
    functions/        # Edge Functions (Deno)
    seed.sql
  scripts/
    setup.sh / setup.ps1
    deploy.sh / deploy.ps1
  CHANGELOG.md
```

8) Implementation Checklist (Click-by-click)

- Supabase → Create project → Project Settings → Database → enable extensions postgis + pgcrypto.
- SQL Editor → run DDL block (tables) → then run RLS block (policies).
- Auth → enable Email (and optionally magic link).
- Edge Functions → create verify_checkin + call_next → deploy with service role access.
- Flutter → add supabase_flutter + permissions for location + notifications → build screens in MVP order.
- Git: commit after each stable milestone; tag releases (v0.1.0 etc.).