NDYRA Platform vNext

Social + Member + Gym + Business OS
Soup-to-Nuts Engineering Blueprint (Aelric Edition)

Audience: Aelric Architect (Lead Dev)
Owner: William Davis Moore
Version: vNext — Aelric Perfection Build (v7, QC++ Anti-Drift Locked)
Date: 2026-02-17
Tagline: Where Discipline Becomes Identity.

How to Use This Blueprint

This document is the single source of truth for NDYRA. It is written to prevent build ambiguity and UI hiccups. If a feature is not in this doc, it is not in scope; if it is in this doc, it must be implemented exactly as specified.

- Operating rules:
- Database Row Level Security (RLS) is the source of truth. Frontend hiding buttons is not security.
- No page-specific hacks. If a UI element is needed, it becomes a component or token.
- Client is allowed to read/write directly to Supabase only where RLS guarantees correctness.
- Anything money/ledger/biometrics tokens are server-side only (service role) and fully auditable.
- All new routes must be declared in the Route Map section and must match the JS module naming convention.

Anti-Drift Protocol

ADP-0 Canonical Sources (Single Source of Truth)

- This blueprint is the canonical spec for product surface + architecture + data model.
- CP27 migrations SQL is the canonical schema implementation for Social Core.
- CP27 RLS tests are the canonical regression harness. If tests are not updated, the change is not real.
- CP27 Build Order Manifest is the canonical execution sequence. Do not reorder unless the blueprint is updated first.
- If it isn't written, it doesn't exist. If it exists, it must be written.

ADP-1 Structure Freeze (No New Patterns)

- No new frontend framework (React/Vue/etc.) during CP27–CP29. Do not rewrite. Extend via ES modules only.
- No new root-level folders. Allowed roots: /site, /netlify/functions, /supabase (if using CLI), /docs (for ADRs).
- Every new route MUST be added to Route Map + CP27 UI Build Spec before code is written.
- One pattern for pages: <body data-page="..."> + page module loaded via <script type="module">. No page-specific inline JS.
- One pattern for shared UI: tokens + components. If a UI rule repeats 2+ times, it becomes a component.

ADP-2 Database + RLS Freeze (RLS is Law)

- Default deny. RLS enabled on every public table (no exceptions without explicit written justification).
- No policy may use `using (true)` or `with check (true)` unless the table is explicitly approved as harmless-public AND recorded in the allowlist in the Anti-Drift Audit script.
- Post-adjacent tables MUST route through `can_view_post(post_id)` (posts, media, reactions, comments, stats, notifications visibility).
- Blocks MUST be enforced at the database layer (via `is_blocked_between()` used in `can_view_post()`).
- Money/ledger tables: client cannot insert/update/delete. Only server (service role / RPC) may write, with audit logging.
- Health/device connections: never readable by anon clients; tokens/refresh tokens stored server-side only.

ADP-3 Storage + Media Freeze (No Private Media Leaks)

- All uploads go direct-to-Supabase Storage (never through Netlify Functions).
- Bucket policies must be visibility-aware: public posts readable; non-public posts not publicly readable.
- No permanent public URLs for private media. If a post becomes private, its media must become non-public by policy.
- Avatars are separate from post media. Avatar reads may be public; writes are owner-only.

ADP-4 Feed/Query Discipline (Scale-Safe by Default)

- Seek pagination only (created_at/id cursor). No OFFSET pagination on feeds.
- No COUNT(*) per request in hot paths. Use cached `post_stats` / incremental triggers.
- Every query used in feeds must have an index plan documented (table + index) in the blueprint.
- No N+1 patterns in feed rendering: all required fields fetched in one query (joins/views ok).

ADP-5 Review Gates (Merge Blockers)

- Any PR touching DB must include: migration diff, RLS audit output, and updated RLS tests (if policies changed).
- Any PR adding a route must include: Route Map update + UI Build Spec update + screenshot/mock reference of intended layout.
- Any PR adding a table must include: RLS enabled, policies, indexes, and (if post-adjacent) `can_view_post` gating.

- Any PR touching tokens/ledger must include: RPC transaction proof (single-transaction spend/refund), audit_log entry, and idempotency proof.
- If a PR introduces a new pattern, it must be rejected unless the blueprint is updated and the pattern is explicitly standardized.

ADP-6 Anti-Drift Checkpoints (Hard Stops)

- Checkpoint AD-1 (CP27 Start): Run migrations on staging, run Anti-Drift Audit SQL, snapshot RLS fingerprint, commit fingerprint to repo.
- Checkpoint AD-2 (CP27 Mid): PostCard + FeedQuery finalized. No new UI variants without updating Component Spec.
- Checkpoint AD-3 (CP27 Pre-Merge): RLS tests pass with 2+ test accounts; audit returns zero failures; storage policies validated against private post media leak attempts.
- Checkpoint AD-4 (CP27 Exit): Cross-platform QA pass (iOS Safari + Android Chrome + Desktop PWA) and abuse smoke test (rate limits, report/block flows).

ADP-7 Drift Detection (Automated)

- Run the Anti-Drift Audit SQL in CI (or at minimum before each merge). It must fail the build on violations.
- Store and track an RLS fingerprint per checkpoint. Any fingerprint change requires an explicit approval note in ADR.
- Maintain a short ADR log: /docs/adr/0001-rls-guardrails.md, etc. Every structural change adds an ADR.

Table of Contents

- Forward-compat scaffolding added: feature flags, audit log, privacy/export/delete ops, and async job patterns for device sync and moderation.
- Cross-platform compliance notes added: iOS/Android billing rules, PWA push constraints, and HealthKit/Health Connect privacy expectations.
- Booking/token ledger guidance updated: all token spend/refund + capacity enforcement must be transactional (RPC + FOR UPDATE).
- post_stats triggers rebuilt to be incremental (no COUNT(*) fan-out on every reaction/comment).
- CP27 DDL appendix completed: copy/paste-ready schema + indexes + policies (no “continue later” placeholders).
- Post visibility rules centralized via can_view_post() helper and reused in all dependent table policies.
- Blocks enforced everywhere: if either party blocks, content becomes mutually invisible across feeds, profile walls, and notifications.
- RLS hardened: no private-post leaks via comments, reactions, media, or stats.

This edition is the same soup-to-nuts blueprint, but with a full security/performance audit applied. Use this as the source of truth going forward.

v6 Verification Summary (Double-Checked + OS/Store Verified)

0. NDYRA Product Thesis + Non-Negotiables

0.1 The Big Idea

NDYRA is a fitness operating system: a social fitness network + a gym business platform in one product. It replaces scattered tools and makes training identity actionable (bookings, memberships, tokens, biometrics, programming, events).

0.2 The Differentiator (Must Not Be Compromised)

- A profile is not bio + photos. A profile is a living performance card:
- Training identity (goals, style tags, clubs, accountability).
- Biometric snapshot (resting HR, HRV, VO2 max estimate, HR zones, recovery signals when available).
- Workout history + streaks + weekly minutes (discipline signal).
- Gym memberships + token wallets (actionable access).
- Achievements (global + gym-defined).
- Privacy-controlled sharing (public vs followers vs clubs vs gyms vs staff).

0.3 Platform Vibe: Gym-Floor Social (No Negativity)

NDYRA's social layer must feel like a gym floor, not a comment war zone. We enforce this through product design (encouragement-forward reactions, comment friction, slow-mode), clear community guidelines, and a strong moderation pipeline (gym-level + platform-level).

0.4 Brand Locks (Decisions Final)

Lock	Decision
Platform Name	NDYRA
Tagline	Where Discipline Becomes Identity.
Core UI Theme	Black canvas + white text + red accent. Cinematic but airy.
Feed Orientation	Portrait-first vertical feed. Default media ratio 4:5.
Clubs	Poster/cover image required; members subpage shows clickable faces; streak is the primary stat.
Reactions	Encouragement-only reactions: 🔥 💪 ⚡ 👏 🧠 (no downvote, no laugh).
Comments	Max depth = 2. First comment requires reminder checkbox. Repeat offenders enter slow-mode.

1. Roles, Relationships, Permissions

1.1 Global Roles (Platform-Wide)

Role	Meaning
Guest	No account. Can browse limited public content (public gym profiles + limited public posts).
User	Account created. Can follow gyms/users, post, react, join clubs, and hold tokens.
Member	User with active NDYRA/Timer subscription (e.g., $14.99) unlocking premium timer/builder and other premium features.
Platform Admin	Master admin. Full tenant visibility, moderation queue, support tools, feature flags.

1.2 Tenant Roles (Per Gym / Tenant)

- Tenant roles are stored in tenant_users.role and are enforced by RLS.

Tenant Role	Capabilities
Gym Admin	Owns tenant settings, staff roles, memberships, pricing, scheduling, posts, templates, payroll exports, ads, events.
Gym Staff	Coaches/trainers. Can view check-ins, run class timers, manage attendance, create posts in gym context, moderate gym space.
Trainer	A staff subtype. Has a public trainer profile + availability blocks + private session offerings.

1.3 Relationship Status (User <-> Gym)

Relationships drive content gating and commerce. These are not tenant roles.

Relationship	Unlocks
Follower	Free. Can view gym public posts/announcements; can receive updates; can buy tokens if enabled.
Gym Member	Paid/comp tier. Can book classes/training per plan; can see members-only posts + templates.
Token Holder	Has token wallet for that gym. Can book token-cost sessions even without a membership tier.

1.4 Permission Enforcement Rules

- Every table in public schema must have RLS enabled.

- RLS policies must reference helper functions (is_platform_admin, is_tenant_staff, is_tenant_member) to keep policies consistent.
- Money/ledger writes are service-role only (Netlify function or Supabase Edge function). Clients never directly update token balances or create token_transactions.
- Device connection tokens (OAuth refresh tokens) are service-role only and must be encrypted at rest.
- Storage policies must align with post visibility (public posts readable by anyone; private/member/staff posts require auth and relationship checks).

2. Route Map (Public, Member, Business, Admin)

2.1 Public (Marketing + Discovery)

Route	Purpose
/	Landing page.
/pricing	NDYRA member subscription (timer/builder premium).
/for-gyms	Gym owner sales page.
/gyms	Gym directory preview (public).
/gym/{slug}	Gym public profile (about, preview posts, schedule preview, follow/join CTAs).
/login	Auth.
/join	Create account + onboarding.

2.2 Member App (Core)

Route	Purpose
/app/fyp	For You feed (ranked blend).
/app/following	Following feed (chronological).
/app/profile	My profile (performance card + wall).
/app/gyms	Discover/manage gyms, memberships, tokens.
/app/clubs	Clubs list.
/app/clubs/{club_id}	Club main page (poster + feed + goals).
/app/clubs/{club_id}/members	Club members grid (faces + performance stats).
/app/wallet	Token wallet (tenant-aware).
/app/notifications	Notifications.
/app/timer	Timer system (existing HIIT56 tech).
/app/book/class/{class_session_id}	Book class flow (tokens/membership logic).
/app/post/{post_id}	Post detail + comments.

2.3 Business Portal (Gym Tenant)

Route	Purpose
/biz/dashboard	KPIs + today schedule + alerts.
/biz/check-in	Scan/search member; Member Readiness Card; attendance mark.
/biz/classes	Class types, sessions, rosters, waitlist.
/biz/trainers	Trainer profiles, availability, private sessions.
/biz/members	CRM-lite members list, tags, notes.
/biz/posts	Gym announcements + posts + moderation tools.
/biz/timer-templates	Gym timer programming; public/private templates.
/biz/events	Host events; affiliate with Group56.
/biz/payroll	Timesheets + export CSV.
/biz/ads	Boost posts (local targeting).

/biz/settings	Tenant settings, tiers, tokens, staff, branding.

2.4 Platform Admin

Route	Purpose
/admin/tenants	Create/suspend gyms; support.
/admin/users	Support tools; moderation; account actions.
/admin/moderation	Reports queue; takedowns; bans.
/admin/billing	Stripe event log + subscription states.
/admin/feature-flags	Rollout control.
/admin/audit	Audit log browsing.

3. UI System (Tokens, Grid, Typography, Components)

3.1 Visual Intent

NDYRA UI is cinematic + airy: dark canvas, bright clarity, and red only as intention. The feed feels like a performance gallery, not a chaotic scroll pit.

3.2 Layout Grid + Breakpoints (Locked)

- Grid rules:
- 8px spacing system only (8/16/24/32/40/48). No random margins.
- Mobile-first. Portrait-first feed. One-handed navigation.
- Max content widths: mobile full width minus 24px padding; desktop centered column 720px for feed.
- Breakpoints:
- Mobile: 375–430
- Tablet: 768
- Desktop: 1280
- Wide: 1440

3.3 Color Tokens (CSS Variables)

- Use these tokens only. Do not invent new colors.

```
:root {
  --ndyra-bg-900: #07070A;
  --ndyra-surface-800: #0F1016;
  --ndyra-surface-700: #141522;
  --ndyra-stroke-700: #242636;

  --ndyra-text-100: #FFFFFF;
  --ndyra-text-300: #B9BBC7;
  --ndyra-text-500: #7C7F92;

  --ndyra-red-500: #E10600;
  --ndyra-red-600: #B90400;
}
```

3.4 Typography

- Font family:
- Inter (preferred). Fallback: system-ui, -apple-system, Segoe UI.
- Type scale (mobile-first):

Token	Size/Line	Weight
H1	32/38	700
H2	24/30	700
H3	18/24	700
Body	16/24	400
Caption	13/18	400
Micro	11/14	400

3.5 Components (Build Once; Pages Are Assemblies)

- AppShell (TopBar + BottomNav mobile; LeftRail desktop later)
- Button (primary red, secondary outline, ghost, destructive, loading)
- Pill/Chip (filters, tags, visibility, membership badges)
- Card (standard, feed card, metric card, gym card)
- Input (text, search, textarea, select, toggles, segmented controls)
- Modal/Drawer (bottom sheet mobile; side drawer desktop)
- MediaTile (FYP wall tiles, profile grid)
- PostCard (header + discipline strip + media + caption + reactions)
- ReactionBar (🔥 💪 ⚡ 👏 🧠 + counts)
- CommentList + CommentComposer (depth 2, slow-mode)
- Avatar (sizes: 28/36/48/64/96)
- MetricStrip (biometrics + streak + weekly minutes)
- GymCTA (Follow / Join / Buy Tokens)
- Empty/Loading/Error States + Skeletons
- Toast system

3.6 Media Rules (Portrait-First)

- Default image ratio: 4:5 (portrait).
- Allowed: 1:1 square; 9:16 vertical video.
- Avoid landscape in the main feed; allow landscape only in post detail view.
- All images must be stored in Supabase Storage with generated signed/public URLs based on visibility.

3.7 Accessibility + Usability (Non-Negotiable)

- Contrast: text must pass WCAG AA on dark background.
- Tap targets: minimum 44px height.
- Keyboard navigation: focus states visible for desktop.
- Reduced motion: honor prefers-reduced-motion for animations.
- No red-only meaning: pair red with icons/labels for status (e.g., streak).

3.8 Platform Targets + OS Support Matrix (2026 Best-Practice Lock)

- Primary delivery = Responsive Web App + PWA (installable). This keeps the existing Netlify + Supabase architecture intact and is the fastest path to market.
- Mobile delivery = iOS + Android store apps as thin wrappers (Capacitor recommended) around the same web UI, with native bridges only for what the browser cannot do well (HealthKit / Health Connect, background delivery, camera QR scan reliability, native push).
- Desktop delivery = Installable PWA on Windows/macOS (Edge/Chrome/Safari). Optional packaging for Microsoft Store later if we want discoverability and enterprise-friendly deployment.
- Browser baseline = Safari (iOS + macOS), Chrome, Edge. Design for modern evergreen browsers; degrade gracefully on older devices (no hard blockers for Boca pilot).
- Golden rule: one design system + one routing map; wrappers add capabilities, not a second UI.

3.9 PWA Manifest + App Install + Icon System (Apple / Google / Windows)

- Add /site/manifest.webmanifest (or /manifest.webmanifest) and lock these fields: name, short_name, start_url, scope, display=standalone, background_color, theme_color, icons[].
- Icons must be generated from one master vector and exported into required platform sizes (see Appendix: Brand Export Checklist).
- Apple: add apple-touch-icon(s) and follow Apple Human Interface Guidelines for app icons.
- Android: provide an adaptive icon (foreground + background layers). Vectors are preferred for the layers.
- Windows/PWA: include multi-size PNG icons in the manifest; Edge/Microsoft guidance recommends specific tile sizes.

3.10 Notifications + Background Execution (Web Push First, Native Later)

- MVP notification channel = Web Push for the PWA (works on Android/desktop; supported on iOS Home Screen web apps on iOS 16.4+).
- Store push subscriptions in DB (push_subscriptions table) keyed by user_id + device + endpoint; send from server-side function only.
- Only request notification permission after a user action (button). Never on first page load.
- Phase-later (native wrappers): add APNs (iOS) + FCM (Android) via Capacitor plugins for higher delivery reliability and richer notification actions.
- Background work rules: keep heavy jobs off the client; use server-side scheduled jobs/queues for feed precompute, notification fan-out, and device sync.

4. CP27 UI Build Spec (Dev-Executable)

4.0 CP27 Scope + Exit Criteria

CP27 is the Social Core MVP. The goal is to ship a usable social experience that feels NDYRA (positive, high-signal, performance-first) and that can support Boca pilot gyms immediately.

- CP27 Exit Criteria:
- Profiles upgraded: avatar upload + handle + privacy settings shell.
- Posts: create (text + photos), view (feed + detail).
- Reactions: encouragement-only reactions (🔥 💪 ⚡ 👏 🧠).
- Comments: create + list; max depth 2; comment friction for first comment.
- Follow system: follow/unfollow users and gyms.
- Feeds: For You (ranked blend) + Following (chronological).
- Moderation primitives: report, block, mute, takedown (platform admin) + basic gym moderation.
- All above enforced by RLS (no reliance on UI hiding).

4.1 Shared UI Contracts (Components)

- These components are required and must be used consistently across screens.

4.1.1 PostCard Contract

- PostCard layout (portrait, airy):
- Header: avatar + display name + handle + optional gym badge + timestamp + visibility pill.
- DisciplineStrip: workout type + duration + streak (if available). Always present; if unknown, show type 'Training' only.
- Body: text (optional), media carousel (optional), workout card (optional).
- ReactionBar: five reactions + counts.
- Comments preview: 0–2 most recent comments.
- PostCard props/state contract (frontend):

```
PostCardProps {
  postId: string
  author: { type: 'user'|'tenant', id: string, name: string, handleOrSlug:
string, avatarUrl?: string }
  visibility: 'public'|'followers'|'members'|'club'|'private'|'staff'
  createdAt: string
  contentText?: string

  discipline: { workoutType?: string, durationMin?: number, streakDays?:
number, weeklyMinutes?: number }
  media: Array<{ id: string, type: 'image'|'video', url: string, width?:
number, height?: number }>

  reactions: { fire: number, strong: number, bolt: number, clap: number, mind:
number }
  viewerReaction?: 'fire'|'strong'|'bolt'|'clap'|'mind'|null
  commentsPreview: Array<{ id: string, user: {id, handle, avatarUrl}, body:
string }>
}
```

4.1.2 Query + Pagination Rules (All Feeds)

- Rules:
- Use seek pagination (created_at + id) never offset pagination.
- Default page size: 25 posts; comments page size: 50.
- Always request the same post fields to keep components consistent.

- Counts: use post_stats (required in CP27) to avoid COUNT() on hot feeds and to enable trending.
- Cursor pagination contract:

```
Cursor = { createdAt: string, id: string }

First page: no cursor.
Next page: fetch posts where (created_at, id) < (cursor.createdAt, cursor.id)
Order: created_at desc, id desc
```

4.1.3 Base Select Shape (Supabase)

Every screen that renders posts uses this select shape. Do not fork shapes per page.

```
const postSelect = `
  id, created_at, visibility, content_text, workout_ref,
  author_user_id, author_tenant_id, tenant_context_id,
  author_user:profiles!posts_author_user_id_fkey ( id, handle, display_name,
avatar_url ),
  author_tenant:tenants!posts_author_tenant_id_fkey ( id, name, slug,
logo_url ),
  media:post_media ( id, media_type, storage_path, width, height ),
  stats:post_stats ( fire_count, strong_count, bolt_count, clap_count,
mind_count, comments_count, score_48h )
`;
```

4.2 Screen Spec — For You Feed (FYP)

Route: /app/fyp

Goal

Deliver a high-signal, positive, addictive feed that blends your world + discovery, without negativity spirals.

Permissions

- Auth required (User). Guests redirected to /login.
- Feed content filtered by RLS: viewer only sees posts they are allowed to see based on visibility + relationships + blocks.

Primary Components

- AppShell.TopBar (title NDYRA + search icon + notifications icon).
- FeedFilterChips (All, Local, Gyms, Clubs, Events) — MVP can ship with All + Local only; chips remain in UI for later expansion.
- PostCard (see contract) in a vertical list with seek-pagination.
- FAB: CreatePost (floating + icon).
- SkeletonPostCard x3 (loading).
- EmptyState (no posts) with CTA to follow gyms and join clubs.

Data Sources

- profiles (viewer profile, location_city/region).
- follows_users, follows_tenants (viewer follow graph).
- gym_memberships (viewer memberships for members-only gating).
- tenant_locations (local tenant ids).
- posts + post_media + post_reactions + post_comments (feed content).
- blocks (exclude blocked users).

Supabase Queries (Exact)

```
// 0) Required: viewer id
const { data: { user } } = await supabase.auth.getUser();
const uid = user.id;

// 1) Viewer profile (location, handle)
const { data: me, error: meErr } = await supabase
  .from('profiles')
  .select('id, handle, display_name, avatar_url, location_city,
location_region')
  .eq('id', uid)
  .single();

// 2) Follow graph
const [{ data: fu }, { data: ft }] = await Promise.all([
  supabase.from('follows_users').select('followee_id').eq('follower_id', uid),
  supabase.from('follows_tenants').select('tenant_id').eq('follower_id', uid),
]);

const followedUserIds = (fu ?? []).map(r => r.followee_id);
const followedTenantIds = (ft ?? []).map(r => r.tenant_id);
```

```
// 3) Memberships (for members-only feed)
const { data: gms } = await supabase
  .from('gym_memberships')
  .select('tenant_id')
  .eq('user_id', uid)
  .eq('status', 'active');

const memberTenantIds = (gms ?? []).map(r => r.tenant_id);

// 4) Local gyms (city/region match)
const { data: locs } = await supabase
  .from('tenant_locations')
  .select('tenant_id')
  .eq('city', me.location_city);

const localTenantIds = [...new Set((locs ?? []).map(r => r.tenant_id))];

// 5) WORLD slice (following + memberships)
const worldOr = [
  followedUserIds.length ? `author_user_id.in.(${followedUserIds.join(',')})` :
null,
  followedTenantIds.length ? `author_tenant_id.in.($
{followedTenantIds.join(',')})` : null,
  memberTenantIds.length ? `tenant_context_id.in.($
{memberTenantIds.join(',')})` : null,
].filter(Boolean).join(',');

const { data: worldPosts } = await supabase
  .from('posts')
  .select(postSelect)
  .or(worldOr)
  .order('created_at', { ascending: false })
  .limit(25);

// 6) LOCAL slice (discovery)
const { data: localPosts } = await supabase
  .from('posts')
  .select(postSelect)
  .in('author_tenant_id', localTenantIds)
  .eq('visibility', 'public')
  .order('created_at', { ascending: false })
  .limit(15);

// 7) TRENDING slice (requires post_stats table; see Section 11)
const { data: trending } = await supabase
  .from('post_stats')
  .select(`post_id, score_48h, post:posts (${postSelect})`)
  .gt('score_48h', 0)
  .order('score_48h', { ascending: false })
  .limit(15);
```

Client Merge + Ranking (MVP)

- Merge algorithm:
- Combine worldPosts + localPosts + trending.post (flatten).
- Deduplicate by post.id.
- Compute score per post; sort desc; take first 25.
- Score formula (deterministic, no ML yet).

```
function scorePost(post, ctx) {
  const ageHours = (Date.now() - Date.parse(post.created_at)) / 36e5;
  const recency = Math.max(0, 48 - ageHours); // 0..48

  const rel =
    (ctx.followedUserIds.includes(post.author_user_id) ? 12 : 0) +
    (ctx.followedTenantIds.includes(post.author_tenant_id) ? 10 : 0) +
    (ctx.memberTenantIds.includes(post.tenant_context_id) ? 14 : 0) +
    (ctx.localTenantIds.includes(post.author_tenant_id) ? 6 : 0);

  const stats = Array.isArray(post.stats) ? post.stats[0] : post.stats;
  const reacts =
    (stats?.fire_count ?? 0) +
    (stats?.strong_count ?? 0) +
    (stats?.bolt_count ?? 0) +
    (stats?.clap_count ?? 0) +
    (stats?.mind_count ?? 0);

  const comments = stats?.comments_count ?? 0;
  const engagement = Math.log1p(reacts * 0.6 + comments * 1.2) * 8;

  return recency + rel + engagement;
}
```

Loading / Empty / Error States

- Loading:
- Show SkeletonPostCard x3. Disable filter chips until first load.
- Empty:
- Show EmptyState with actions: Follow a gym, Discover gyms near you, Join a club.
- Error:
- Toast: 'Feed failed to load' + Retry button. Log to telemetry endpoint.

User Actions + Mutations

- React to post: upsert into post_reactions (one reaction per user per post).
- Remove reaction: delete from post_reactions.
- Comment: insert into post_comments (RLS + slow-mode).
- Follow/unfollow: insert/delete follows_users / follows_tenants.
- Report: insert into reports.
- Block: insert into blocks.

Telemetry Events (Required)

- feed_open { feed: 'fyp' }
- post_impression { post_id } (debounced; only when 60% visible for >500ms)
- reaction_set { post_id, reaction }
- comment_create { post_id }
- follow { target_type, target_id }
- report { target_type, target_id, reason }

4.3 Screen Spec — Following Feed

Route: /app/following

Goal

A clean chronological feed showing only content from followed gyms/users/clubs (no discovery).

Permissions

- Auth required.
- RLS enforces visibility; following feed must not show non-followed content.

Primary Components

- TopBar title 'Following'.
- Feed list of PostCard.
- Empty state: 'Follow gyms and people to build your feed.' CTA to /app/gyms.

Supabase Query (Exact)

```
// Preload followedUserIds and followedTenantIds as in FYP
const followOr = [
  followedUserIds.length ? `author_user_id.in.(${followedUserIds.join(',')})` :
null,
  followedTenantIds.length ? `author_tenant_id.in.($
{followedTenantIds.join(',')})` : null,
].filter(Boolean).join(',');

const { data: posts, error } = await supabase
  .from('posts')
  .select(postSelect)
  .or(followOr)
  .order('created_at', { ascending: false })
  .limit(25);

// Pagination:
const { data: next } = await supabase
  .from('posts')
  .select(postSelect)
  .or(followOr)
  .lt('created_at', cursor.createdAt)
  .order('created_at', { ascending: false })
  .limit(25);
```

States

- Loading: skeleton x3.
- Empty: CTA to follow gyms/people.
- Error: toast + retry.

4.4 Screen Spec — My Profile (Performance Card)

Route: /app/profile

Goal

Make the user feel like NDYRA knows them. This is where discipline becomes identity: face + stats + memberships + history.

Primary Layout (Locked)

- Header: avatar (96), display name, handle, Edit button, Privacy button.
- Badges row: NDYRA premium badge (if timer subscription active) + gym membership badges (logos).
- Performance Strip (always visible): Resting HR, HRV, VO2, Weekly Minutes, Streak.
- Training Identity: goals + style tags.
- Injuries/modifications: private by default; shareable with gyms (staff view only).
- My Gyms list: each card shows tier + next booking + token balance + QR check-in button.
- Content wall: toggle (Posts | Workouts | Achievements). Default Posts grid.

Permissions

- Auth required.
- User can edit only their own profile.
- Biometrics + injuries visibility controlled by privacy_settings.

Supabase Queries (Exact)

```
// 1) Profile + privacy
const [{ data: me }, { data: privacy }] = await Promise.all([
  supabase.from('profiles')
    .select('id, handle, display_name, avatar_url, bio, location_city,
location_region')
    .eq('id', uid).single(),
  supabase.from('privacy_settings')
    .select('*')
    .eq('user_id', uid).single(),
]);

// 2) Latest biometrics (MVP: latest value per metric)
const { data: biometrics } = await supabase
  .from('biometric_snapshots')
  .select('metric, value, unit, measured_at')
  .eq('user_id', uid)
  .order('measured_at', { ascending: false })
  .limit(50);

// 3) Gyms: memberships + tokens
const { data: myGyms } = await supabase
  .from('gym_memberships')
  .select(`
    tenant_id, plan_id, status,
    tenant:tenants ( id, name, slug, logo_url ),
    plan:membership_plans ( id, name )
  `)
  .eq('user_id', uid)
  .eq('status', 'active');
```

```
const { data: wallets } = await supabase
  .from('token_wallets')
  .select('tenant_id, balance')
  .eq('user_id', uid);

// 4) My posts
const { data: posts } = await supabase
  .from('posts')
  .select(postSelect)
  .eq('author_user_id', uid)
  .order('created_at', { ascending: false })
  .limit(25);
```

Mutations (Exact)

```
// Update profile
await supabase.from('profiles').update({
  display_name, bio, location_city, location_region
}).eq('id', uid);

// Avatar upload (Storage bucket: avatars)
const path = `u/${uid}/${crypto.randomUUID()}.jpg`;
await supabase.storage.from('avatars').upload(path, file, { upsert: false });
const { data: pub } = supabase.storage.from('avatars').getPublicUrl(path);
await supabase.from('profiles').update({ avatar_url: pub.publicUrl }).eq('id',
uid);

// Update privacy settings
await supabase.from('privacy_settings').upsert({
  user_id: uid,
  biometrics_visibility,
  workouts_visibility,
  injuries_visibility,
  location_visibility,
});
```

States

- Loading: skeleton for header + metrics + gym cards.
- Empty gyms: show 'Follow gyms' and 'Discover gyms' buttons.
- No biometrics: show 'Add biometrics' CTA + 'Connect device' CTA.
- Error: show toast + retry; never white-screen.

4.5 Screen Spec — Post Detail

Route: /app/post/{post_id}

Goal

Deep view for one post: full media, full caption, full comments, and safe interaction.

Primary Components

- PostCard (single) in detail mode (shows full caption).
- CommentsList (paginated, depth max 2).
- CommentComposer (with community reminder friction on first-ever comment).
- Report / Block actions in overflow menu.

Supabase Queries (Exact)

```
const { data: post, error: postErr } = await supabase
  .from('posts')
  .select(postSelect)
  .eq('id', postId)
  .single();

const { data: comments, error: comErr } = await supabase
  .from('post_comments')
  .select(`
    id, post_id, user_id, parent_id, body, created_at,
    user:profiles ( id, handle, display_name, avatar_url )
  `)
  .eq('post_id', postId)
  .is('deleted_at', null)
  .order('created_at', { ascending: true })
  .limit(50);
```

Mutations

```
// Add comment (top-level)
await supabase.from('post_comments').insert({
  post_id: postId,
  user_id: uid,
  body: text,
});

// Reply (depth 2)
await supabase.from('post_comments').insert({
  post_id: postId,
  user_id: uid,
  parent_id: parentCommentId,
  body: text,
});

// Soft-delete own comment
await supabase.from('post_comments')
  .update({ deleted_at: new Date().toISOString() })
  .eq('id', commentId)
  .eq('user_id', uid);
```

States

- Loading: skeleton PostCard + comment placeholders.
- Post not found / forbidden: show 'This post is unavailable' (do not leak whether it exists).
- Comments empty: show 'Be the first to encourage'.

4.6 Screen Spec — Create Post (Modal/Sheet)

Route: FAB opens modal on any feed/profile

Goal

Fast post creation that still enforces positivity + correct privacy gating.

UI Components

- ComposerHeader: avatar + 'Post as' selector (Me / Gym if staff).
- Textarea for caption (with positivity hint: 'Ask, encourage, or share something useful').
- MediaPicker: up to 10 images (MVP).
- VisibilitySelector (Public, Followers, Members, Club, Private) — options depend on context.
- CTA: Post (disabled until valid).

Validation Rules

- At least one of: text OR media OR workout_ref.
- Max images: 10.
- Max text length: 2,000 chars.
- If posting as gym: must be tenant staff and select tenant_context_id = that tenant.

Upload + Insert Flow (Exact)

```
// 1) Create post row first (so we have post_id)
const { data: post, error } = await supabase
  .from('posts')
  .insert({
    author_user_id: postingAs === 'user' ? uid : null,
    author_tenant_id: postingAs === 'tenant' ? tenantId : null,
    tenant_context_id: tenantContextId ?? null,
    visibility,
    content_text,
    workout_ref: workoutRef ?? {},
  })
  .select('id')
  .single();

const postId = post.id;

// 2) Upload media directly to Storage (bucket: post-media)
for (const file of files) {
  const path = `p/${postId}/${crypto.randomUUID()}.jpg`;
  await supabase.storage.from('post-media').upload(path, file, { upsert:
false });

  await supabase.from('post_media').insert({
    post_id: postId,
    media_type: 'image',
    storage_path: path,
  });
}

// 3) Navigate to post detail (or optimistic insert into feed)
navigate(`/app/post/${postId}`);
```

States

- Loading: show progress bar per upload + disable Post CTA.
- Error upload: show which file failed; allow retry; do not create half-visible posts (see cleanup job).
- Cleanup: if post created but media upload fails, mark post as draft=true (or delete) to avoid broken posts.

4.7 Screen Spec — Notifications

Route: /app/notifications

Goal

Fast, clean notification list for reactions, comments, follows, bookings, and gym announcements.

Data Model (Required)

- notifications table (see Section 11) with:
- id, user_id, type, actor_user_id, actor_tenant_id, entity_type, entity_id, title, body, is_read, created_at

Supabase Query (Exact)

```
const { data: notifs } = await supabase
  .from('notifications')
  .select(`
    id, type, title, body, is_read, created_at,
    actor_user:profiles!notifications_actor_user_id_fkey ( id, handle,
display_name, avatar_url ),
    actor_tenant:tenants!notifications_actor_tenant_id_fkey ( id, name, slug,
logo_url ),
    entity_type, entity_id
  `)
  .eq('user_id', uid)
  .order('created_at', { ascending: false })
  .limit(50);
```

Mutations

```
// Mark read (single)
await supabase.from('notifications')
  .update({ is_read: true })
  .eq('id', notifId)
  .eq('user_id', uid);

// Mark all read
await supabase.from('notifications')
  .update({ is_read: true })
  .eq('user_id', uid)
  .eq('is_read', false);
```

States

- Empty: 'No updates yet'.
- Error: toast + retry.

4.8 Screen Spec — Onboarding (First-Run Flow)

Route: /join → /app/onboarding

Goal

Turn a new user into an activated user in <3 minutes: handle + face + goals + first follows. This is critical for retention and feed quality.

Steps (Locked)

19. Step 1: Create account (email/password or magic link).
20. Step 2: Choose handle (unique), display name, upload avatar (required).
21. Step 3: Set location (city/region) + privacy default (city-level).
22. Step 4: Pick goals (1–3) + training style tags (3–7).
23. Step 5: Follow 3 gyms (suggest local) and 3 people (optional).
24. Step 6: Join or create first club (optional but strongly recommended).
25. Step 7: Device connect prompt (skip allowed in MVP; shows 'connect later').

Supabase Writes

```
// After auth signup:
await supabase.from('profiles').upsert({
  id: uid,
  handle,
  display_name,
  location_city,
  location_region
});

await supabase.from('privacy_settings').upsert({
  user_id: uid,
  biometrics_visibility: 'followers',
  workouts_visibility: 'followers',
  injuries_visibility: 'my_gyms',
  location_visibility: 'city'
});

// Follow gyms selected
await supabase.from('follows_tenants').insert(selectedTenantIds.map(tid => ({
  follower_id: uid,
  tenant_id: tid
})));
```

4.9 CP27 Acceptance Checklist (Aelric Test Script)

- Create new user; complete onboarding; handle unique enforced.
- Upload avatar; profile displays correctly.
- Follow a gym; following feed populates with gym posts.
- Create a public photo post; verify guest can see it on gym/profile preview (public surfaces).
- Create a followers-only post; verify non-followers cannot access (direct URL).
- React to a post; reaction appears; cannot react twice; changing reaction updates counts.
- Comment and reply; depth limited to 2; slow-mode engaged when flagged.
- Block a user; verify mutual invisibility in feed and profile.
- Report a post; platform admin sees it in moderation queue.

5. Social System (Posts, Feeds, Anti-Negativity, Moderation)

5.1 Post Types (MVP → Later)

- MVP post types (ship first):
- Text post (content_text only).
- Photo post (1–10 images).
- Workout share post (auto-generated workout card from timer/class/workout_history).
- Phase-later:
- Video posts (short clips).
- Stories (24h).
- Live (very later).

5.2 Visibility + Context Model

- Visibility values (use enum post_visibility):
- public — visible to anyone (including guests).
- followers — visible to followers of the author (user or gym).
- members — visible to members of tenant_context_id.
- club — visible to members of club_id.
- private — only the author.
- staff — staff/admin of tenant_context_id only.
- Context rules:
- author_user_id XOR author_tenant_id (exactly one set).
- tenant_context_id optional; if set, the post is 'inside' that gym space and can be gated members/staff.
- club_id optional; if set, visibility must be club (enforced by DB check constraint).

5.3 Reactions System (Encouragement Only)

- Reactions (locked): 🔥 💪 ⚡ 👏 🧠
- Data model:
- post_reactions table with (post_id, user_id, reaction_type, created_at).
- Primary key (post_id, user_id) so each user has at most one reaction per post.
- Changing reaction is an UPDATE of reaction_type (not multiple rows).

5.4 Comments + Friction (Anti-Negativity by Design)

- Max thread depth = 2 (parent + one level of replies).
- First comment ever from an account triggers a reminder checkbox ('Be a good training partner'). Store ack timestamp on profile.
- Slow-mode: users with strikes can only comment once per X minutes (default 10).
- Gym option: members-only comments on gym posts.

5.5 Feed Types + Ranking

- Following feed:

- Chronological. Only content from followed users/gyms/clubs.
- For You feed:
- Algorithmic blend. MVP uses deterministic scoring (Section 4.2). Later add embeddings/personalization.

5.6 Moderation Pipeline (MVP Required)

- Report post/comment/profile/tenant with reason codes.
- Mute user (client-side + DB optional).
- Block user (DB-enforced mutual invisibility).
- Gym-level moderation: tenant staff can hide posts inside their tenant_context_id.
- Platform moderation: platform admins can remove any content; all actions write to audit_log.

5.7 Missing-But-Required for Real Scale (Approved)

- These are mandatory to avoid spam/abuse at scale, and are included by default:
- user_moderation_state table (strikes, slow_mode_until, shadow_banned_at).
- post_stats table (reaction counts per type + comments_count + score_48h).
- content_moderation table (image/text safety verdicts).
- rate limiting for /api/* and DB writes (per user + per IP).

6. Gyms (Profiles, Memberships, Tokens, Ads, Events)

6.1 Gym Profile (Public + Gated)

- Public view (guest allowed):
- About, photos, locations, contact, website.
- Public posts preview (public visibility only).
- Schedule preview (public sessions only).
- CTA: Follow (free), Join (membership tiers), Buy Tokens (if enabled).
- Member view (unlocked):
- Full schedule with booking.
- Members-only posts.
- Members-only timer templates.
- Club recommendations.
- Staff/Admin view:
- Member roster + CRM-lite.
- Check-in dashboard (Member Readiness Card).
- Class rosters + attendance + waitlist.
- Gym moderation tools.

6.2 Membership Tiers (Per Tenant)

- Each tenant defines membership_plans (name, price, booking_rules, perks).
- gym_memberships links user to tenant and plan; status active/paused/canceled.
- Booking_rules stored as JSONB initially (fast iteration), later normalized if needed.

6.3 Tokens (ClassPass-like but Tenant-Aware)

- Core token rules:
- Tokens stored per tenant in token_wallets (tenant_id, user_id, balance).
- All token balance changes are derived from token_transactions ledger.
- Bookings that require tokens call server-side RPC spend_tokens (transactional).
- Refunds and cancellations create offsetting token_transactions rows; never delete ledger rows.

6.4 Ads / Boosted Posts (MVP Later, Approved)

- Gyms can boost posts to local audiences; boosted content is clearly labeled 'Sponsored'.
- Ads table stores campaign, budget, targeting (city/region + interests tags).
- Impressions/clicks tracked in ad_events table.

6.5 Events Network (Group56)

- Group56 is the parent race / umbrella event brand (global).
- Gyms can host local events and optionally affiliate them with Group56.
- Events generate posts, badges, and leaderboards.

7. Scheduling + Booking + Check-In + Payroll Export

7.1 Class Scheduling (Tenant)

- class_types: defines class name, default token cost, description.
- class_sessions: defines date/time, coach, capacity, visibility (public/members), token_cost override.
- class_bookings: user booking state (booked/canceled/attended/no_show/waitlist).

7.2 Booking Rules (Hard)

- Capacity must be enforced transactionally (no oversells).
- Waitlist auto-promotion when a spot opens.
- Cancellation cutoff per tenant (default 2h).
- Token spend/refund rules must be deterministic and auditable.

7.3 Private Training Sessions

- trainer_profiles: ties a user (staff) to tenant trainer details (bio, specialties, pricing).
- trainer_availability: availability blocks (recurrence supported later).
- private_sessions: booking records (requested/confirmed/canceled/completed).

7.4 Check-In Flow (The 'Gym Already Knows You' Moment)

- At check-in (scan QR or lookup), show Member Readiness Card:
- Member name + profile photo.
- Membership status + token balance (allowed in?).
- Injury/modification flags (only if user consented for that gym).
- Coach notes (internal; staff only).
- Last visit + streak + today's booking.
- Quick toggles: 'Needs Modifications', 'First Timer', 'Returning Injury' (creates internal notes).

7.5 Payroll (MVP = Tracking + Export)

- MVP payroll is accounting-friendly, not payout automation:
- Track classes coached + attendance count.
- Track private sessions completed.
- Apply pay rules (flat rate, per head, per session).
- Export CSV per pay period.

8. Clubs + Accountability

8.1 Club Concept

Clubs are micro-communities for accountability (running groups, workout groups, race prep squads). They are a key retention engine and must feel performance-first, not chatty.

8.2 Club Pages (Locked UI)

- Club main page includes a poster/cover image (editable).
- Club includes goal definition (e.g., 4 workouts/week) and group progress bar.
- Club members page is a clickable grid of faces (real-photo style) with performance-first stats.
- Primary stat under avatar: streak days (large). Secondary: sessions/week or weekly minutes.

8.3 Privacy + Sharing Rules

- A user can choose to share biometrics with clubs (privacy_settings).
- Club leaderboards default to discipline metrics (streak, weekly minutes) not sensitive biometrics.
- Clubs can be public, private, or invite-only.

8.4 Club Data Model (Required)

- clubs (id, owner_user_id, name, slug, poster_url, visibility, goal_json, created_at).
- club_members (club_id, user_id, role, joined_at).
- club_posts via posts.club_id with visibility='club'.

9. Biometrics + Device Integrations (MVP → Native)

9.1 Biometrics Sources (Reality)

- Manual entry (MVP fallback; required for launch).
- OAuth providers (Strava, Fitbit, Garmin, Whoop, Oura, etc.).
- OS-level frameworks: Apple HealthKit (iOS/watchOS) and Android Health Connect.
- Android note (2026+): Google Fit APIs are being deprecated; do not build new core integrations on Google Fit. Use Health Connect as the OS-level aggregator on Android.
- Apple note (HealthKit): treat all health metrics as sensitive data. HealthKit policies prohibit using HealthKit data for advertising/data brokerage; require clear consent and a privacy policy.

9.2 'Living Profile' Biometrics Sync

- MVP behavior:
- User selects device(s) used (stored in profiles.device_hint_json).
- Biometric snapshots are stored whenever user manually enters or after a workout completes (timer/class).
- Profile performance strip always renders; missing data displays 'Connect' or '--', never fake values.
- Phase 2 behavior (OAuth connectors):
- User completes OAuth connect.
- Backend stores refresh token encrypted (service role only).
- Scheduled sync updates daily summary + latest biometrics + workouts.
- Phase 3 behavior (Native/Wrapper):
- Capacitor/React Native wrapper reads HealthKit/Health Connect data locally and uploads to Supabase.
- Background delivery respects OS constraints; no invasive polling.

9.3 Privacy Rules (Health Data = Bank Data)

- Biometrics visibility controlled per user (public/followers/clubs/private).
- Injuries/modifications default private; share only with gym staff for gyms where user is active member (consent required).
- No advertising targeting based on HealthKit data.
- Provide user controls: revoke access, export data, delete account.

10. Payments (Stripe) + Ledger Architecture

10.1 Payment Realities

- There are two distinct payment domains:
- A) NDYRA Premium (global) — subscription unlocking timer/builder and premium features.
- B) Gym commerce (tenant-specific) — gym memberships and token packs.

10.2 Marketplace Strategy (Approved)

- Phase 1 (Boca pilot): Platform as Merchant of Record (fastest).
- Phase 2 (scale): Stripe Connect (gyms as connected accounts).

10.3 Ledger Rules (Non-Negotiable)

- token_wallets balance is derived from ledger; direct client writes forbidden.
- token_transactions rows are immutable (no deletes). Adjustments are new rows.
- All ledger mutations happen in a single DB transaction (RPC).
- Every money-affecting change writes an audit_log row.

10.4 Stripe Webhooks (Idempotent)

- Always insert stripe_events first with unique constraint on event id.
- Process known events only; ignore others safely.
- If processing fails, store error and retry safely (idempotent).

NOTE: this is engineering guidance, not legal advice. Before public mobile store launches, do a final policy pass against current App Review Guidelines + Google Play Payments policy.

- Never hardcode pricing; use provider catalog (Stripe products/prices; StoreKit products; Play Billing products).
- All entitlements flow into the same subscriptions/entitlements tables so product gating is consistent across web/native.
- Payments capability is an isolated module: /core/billing (web Stripe) + /native/billing (StoreKit/Play Billing) — behind a feature flag.

Implementation guardrails:

- If we want the lowest-risk path, plan a parallel in-app subscription implementation (StoreKit for iOS, Play Billing for Android) OR keep purchases strictly on the web with compliant UX inside the app.
- Apple and Google generally require using their in-app purchase/billing systems for digital goods/services sold inside the app. Rules can vary by region and evolve (court rulings, DMA, etc.).

If NDYRA is distributed through the Apple App Store or Google Play:

- Keep a clean separation: NEVER use health/biometrics for ad targeting (especially if any HealthKit/Health Connect data is involved).
- If/when we ship native iOS/Android wrappers, treat them as companions unless we decide to implement StoreKit/Play Billing for in-app purchase flows.
- Ship Web-first as a PWA (Stripe checkout is fine on the web).

Recommended strategy for NDYRA:

10.5 App Store + Play Store Billing Compliance (2026 Reality)

11. Data Model (Supabase Postgres) + RLS + Storage Policies

11.1 Canonical Entities

- Core identity:
- profiles (extends auth.users).
- privacy_settings.
- user_moderation_state (strikes/slow-mode/shadow bans).

- Gyms / tenants:
- tenants, tenant_locations, tenant_users.
- membership_plans, gym_memberships.
- token_wallets, token_transactions.
- Social:
- posts, post_media, post_reactions, post_comments, post_stats.
- follows_users, follows_tenants.
- blocks, mutes (optional), reports, notifications.
- Business:
- class_types, class_sessions, class_bookings, attendance.
- trainer_profiles, trainer_availability, private_sessions.
- payroll_rules, payroll_exports (CSV generation).
- Biometrics:
- biometric_snapshots, device_connections (encrypted tokens), health_workouts (imported).
- Clubs + Events + Ads:
- clubs, club_members, club_events.
- events, event_hosts, event_registrations, event_results, tenant_affiliations (Group56).
- ads, ad_events.

11.2 Enums (Recommended to Prevent Bugs)

```
-- Example enums (create once, use everywhere)
create type public.post_visibility as enum
('public','followers','members','club','private','staff');
create type public.reaction_type as enum
('fire','strong','bolt','clap','mind');
create type public.tenant_role as enum ('admin','staff','trainer');
create type public.booking_status as enum
('booked','canceled','attended','no_show','waitlist');
```

11.3 RLS Principles

- Default deny: RLS enabled on every table.
- Helper functions: is_platform_admin(), is_tenant_staff(tid), is_tenant_member(tid).
- Posts: select allowed only if visibility rules pass AND author is not blocked.
- Ledger tables: select allowed (own or staff), but inserts/updates only via service role.
- Device tokens: service role only (no client reads/writes).

11.4 Storage Buckets + Policies

- Buckets:
- avatars (profile images) — public read, auth write own folder.
- tenant-logos (gym branding) — public read, tenant staff write.
- post-media (feed media) — read based on post visibility; write by post author only.
- club-posters (club covers) — read based on club visibility; write by club owner/admin.
- Policy rule:

- Storage read must not leak private/member content to guests. Either store private media in private bucket + signed URLs, or enforce storage.objects RLS tied to posts/visibility.

12. API Blueprint (Netlify Functions, RPCs, Webhooks)

12.1 Client-Direct vs Serverless (Rule)

- Client-direct (Supabase anon key + RLS):
- Read feeds/posts/comments.
- Create posts (text + media references) where RLS allows.
- Follow/unfollow.
- Book classes (insert booking) where no tokens are required.
- Serverless (service role only):
- Stripe checkout session creation.
- Stripe webhooks.
- Token purchases + ledger writes.
- Token spend/refund RPC wrappers.
- Device connector token storage/refresh + scheduled sync.
- High-impact admin actions (bans, takedowns, shadow bans).

12.2 Required Netlify Function Endpoints

Endpoint	Purpose
/.netlify/functions/stripe-create-checkout	Create Stripe checkout sessions (subscription or token pack).
/.netlify/functions/stripe-webhook	Process Stripe events; idempotent logging in stripe_events.
/.netlify/functions/tokens-spend	Spend tokens for booking; calls RPC spend_tokens.
/.netlify/functions/tokens-refund	Refund tokens for cancellation; calls RPC refund_tokens.
/.netlify/functions/device-oauth-callback	Store device tokens encrypted.
/.netlify/functions/moderation-action	Platform admin actions (takedown/ban).

12.3 Core RPC Functions (Database)

- spend_tokens(tenant_id, user_id, amount, ref_type, ref_id) returns new_balance
- refund_tokens(tenant_id, user_id, amount, ref_type, ref_id)
- book_class_with_tokens(class_session_id) (optional wrapper that creates booking + token spend transactionally)
- set_user_moderation_state(user_id, strike_delta, slow_mode_until, shadow_banned_at) (platform admin only)

13. Scaling + Performance + Abuse Protection

13.1 What Scales Easily vs Pain Points

- Easy:
- Static pages via Netlify CDN.
- Supabase Storage for images (with proper caching).
- Hard:
- Feed ranking queries at high QPS.
- Counts (reactions/comments) on hot posts.
- Abuse/spam waves.
- Notification fan-out.

13.2 MVP Scaling Strategy

- Use post_stats to avoid COUNT() on every render.
- Index posts on (created_at desc), (author_user_id, created_at), (author_tenant_id, created_at), (tenant_context_id, created_at).
- Index post_reactions on (post_id) and (user_id) and maintain counts in post_stats via triggers.
- Seek pagination only.

13.3 Abuse Protection

- Netlify rate limiting for all /api/* endpoints.
- DB constraints: one reaction per user per post; unique follow rows; prevent spam duplicates.
- Account age limits: new accounts limited to X posts/day until verified (store in user_moderation_state).
- Shadow bans: hide content from everyone except the author while allowing them to keep using the app (reduces escalation).

14. Observability + Analytics + Audit

14.1 Audit Log (Required)

- audit_log captures admin actions and ledger actions: who, what, target, before/after JSON, timestamp, ip/user_agent.
- Every moderation takedown, ban, or token adjustment must create an audit_log row.

14.2 Telemetry Events (Client)

- feed_open, post_impression, reaction_set, comment_create, booking_create, token_purchase, checkin_success
- All events are POSTed to a telemetry endpoint (existing in repo) with sampling to control cost.

14.3 Monitoring

- Supabase: enable slow query logs; create dashboards for CPU, connections, cache hit rate.
- Netlify: monitor function error rate and cold starts.
- Stripe: monitor webhook retries and failures.
- Error tracking: add Sentry (or equivalent) for JS + serverless functions; capture release version, user_id (hashed), and route for fast triage.
- Distributed tracing: emit OpenTelemetry-style trace IDs from frontend requests and propagate through Netlify Functions to Supabase for correlation (even if full tracing comes later).
- Uptime: external synthetic monitoring for /api/health + key pages (login, feed). Alert on latency spikes and auth failures.
- Security monitoring: alert on unusual auth attempts, rate-limit triggers, and spikes in reports/spam flags.

15. Dev Workflow (Environments, Migrations, Testing, Release)

15.1 Environments (Required)

- Local dev (optional): Supabase local via CLI.
- Staging: separate Supabase project + Stripe test mode.
- Production: locked Supabase project + Stripe live mode.

15.2 Secrets + Env Vars (Netlify)

- SUPABASE_URL
- SUPABASE_ANON_KEY
- SUPABASE_SERVICE_ROLE_KEY (functions only)
- STRIPE_SECRET_KEY
- STRIPE_WEBHOOK_SECRET
- TELEMETRY_INGEST_KEY

15.3 Database Migrations

- All schema changes must be migrations committed to repo (no manual prod edits).
- Migrations must include: DDL + RLS policies + indexes.
- Every migration must have a rollback plan (at least documented).

15.4 Testing (Must Exist Before Scaling)

- RLS tests: verify forbidden access using anon key and random users.
- Unit tests: scoring functions and helper utilities.
- E2E tests: onboarding, create post, react, comment, follow, booking.
- Load test: feed read QPS + hot post reactions.

15.5 2026 Engineering Standards (Tooling + Security + Cross-Platform)

- Type safety: migrate new modules to TypeScript (or JS + JSDoc type checking as a stepping stone). Enforce strict mode for all new code.
- Bundling: use a modern build step (Vite recommended) for module bundling, code-splitting, and asset hashing; keep static HTML routes but compile JS modules.
- Code quality: ESLint + Prettier + EditorConfig; pre-commit hooks (Husky) to block broken formatting/tests.
- CI/CD: GitHub Actions (or equivalent) running: lint, unit tests, RLS smoke tests (against staging), and Playwright E2E. Auto-deploy preview builds to Netlify per PR.
- Security headers: enforce CSP, HSTS, X-Content-Type-Options, Referrer-Policy, and Permissions-Policy at the CDN edge. Lock down allowed image/media origins (Vimeo, Supabase Storage).
- Dependency hygiene: monthly dependency audit; pin major versions; SBOM generation for enterprise readiness (phase-later).
- Privacy compliance ops: implement data export + deletion workflows (user-controlled), and maintain App Store / Play Store privacy disclosures as features ship.

16. Checkpoint Roadmap (CP27–CP35+) + Acceptance Criteria

Checkpoint	Exit Criteria
CP27 — Social Core MVP	Profiles + posts + reactions + comments + follows + FYP/following + moderation primitives.
CP28 — Gyms Directory + Follow → Join	Public gym profiles + membership plans (comp mode) + gated member view + check-in scaffold.
CP29 — Tokens + Booking MVP	Token wallet + ledger + class sessions + booking + cancellation/refund rules.
CP30 — Trainer Booking + Payroll Export MVP	Trainer availability + private bookings + pay rules + CSV exports.
CP31 — Clubs + Accountability	Create/join clubs + club feed + leaderboards + share discipline metrics.
CP32 — Events Network (Group56)	Event creation + registrations + results + badges.
CP33 — Biometrics Integrations Phase 1	Manual snapshots + connect 1–2 providers + secure token storage + scheduled sync.
CP34 — Ads / Boosted Posts	Boost post + targeting + metrics + sponsored labeling.
CP35+ — Connect Payouts + Native Apps	Stripe Connect onboarding + payouts + HealthKit/Health Connect wrapper apps.

Appendix A — CP27 Database Schema (DDL Skeleton)

```
-- ============================================================
-- NDYRA CP27 (Social Core MVP) — Corrected v6
-- Supabase Postgres (RLS-safe, blocked-safe, no private leaks)
-- ============================================================

-- Extensions
create extension if not exists "uuid-ossp";
create extension if not exists "pgcrypto";

-- Optional: trigram for search/handles later
create extension if not exists "pg_trgm";



-- ------------------------------------------------------------
-- Enums
-- ------------------------------------------------------------
do $$ begin
  create type public.post_visibility as enum
('public','followers','members','club','private','staff');
exception when duplicate_object then null; end $$;

do $$ begin
  create type public.reaction_type as enum
('fire','strong','bolt','clap','mind');
exception when duplicate_object then null; end $$;



-- ------------------------------------------------------------
-- Helper functions (RLS building blocks)
-- ------------------------------------------------------------
create or replace function public.is_platform_admin()
returns boolean
language sql stable
as $$
  select exists(select 1 from public.platform_admins pa where pa.user_id =
auth.uid());
$$;

create or replace function public.is_tenant_admin(tid uuid)
returns boolean
language sql stable
as $$
  select exists(
    select 1 from public.tenant_users tu
    where tu.tenant_id = tid and tu.user_id = auth.uid() and tu.role = 'admin'
  );
$$;

create or replace function public.is_tenant_staff(tid uuid)
returns boolean
language sql stable
as $$
  select exists(
    select 1 from public.tenant_users tu
    where tu.tenant_id = tid and tu.user_id = auth.uid() and tu.role in
('admin','staff','trainer')
  );
$$;
```

```
create or replace function public.is_tenant_member(tid uuid)
returns boolean
language sql stable
as $$
  select exists(
    select 1 from public.gym_memberships gm
    where gm.tenant_id = tid and gm.user_id = auth.uid() and gm.status =
'active'
  );
$$;

-- Mutual block check (SECURITY DEFINER so RLS on blocks cannot prevent
enforcement)
create or replace function public.is_blocked_between(a uuid, b uuid)
returns boolean
language sql stable
security definer
set search_path = public
as $$
  select case
    when a is null or b is null then false
    else exists(
      select 1 from public.blocks bl
      where (bl.blocker_id = a and bl.blocked_id = b)
         or (bl.blocker_id = b and bl.blocked_id = a)
    )
  end;
$$;

-- Centralized visibility check used by RLS on every post-adjacent table.
-- IMPORTANT: keep this logic in sync with the product spec.
create or replace function public.can_view_post(p_post_id uuid)
returns boolean
language plpgsql stable
security definer
set search_path = public
as $$
declare
  v_viewer uuid := auth.uid();
  v_author_user uuid;
  v_author_tenant uuid;
  v_tenant_ctx uuid;
  v_visibility public.post_visibility;
  v_deleted boolean;
begin
  select p.author_user_id, p.author_tenant_id, p.tenant_context_id,
p.visibility, p.is_deleted
    into v_author_user, v_author_tenant, v_tenant_ctx, v_visibility, v_deleted
  from public.posts p
  where p.id = p_post_id;

  if v_author_user is null and v_author_tenant is null then
    return false;
  end if;

  if v_deleted then
    return false;
  end if;

  -- Platform admins can view everything (except: future sealed/legal holds)
  if public.is_platform_admin() then
    return true;
```

```
  end if;

  -- Anonymous users can only view public posts
  if v_viewer is null then
    return (v_visibility = 'public');
  end if;

  -- Blocked users cannot view each other (user-authored posts only)
  if v_author_user is not null and public.is_blocked_between(v_viewer,
v_author_user) then
    return false;
  end if;

  -- Visibility rules
  if v_visibility = 'public' then
    return true;
  elsif v_visibility = 'private' then
    return (v_author_user = v_viewer);
  elsif v_visibility = 'followers' then
    if v_author_user is not null then
      return exists(
        select 1 from public.follows_users fu
        where fu.follower_id = v_viewer and fu.followee_id = v_author_user
      );
    else
      return exists(
        select 1 from public.follows_tenants ft
        where ft.follower_id = v_viewer and ft.tenant_id = v_author_tenant
      );
    end if;
  elsif v_visibility = 'members' then
    return (v_tenant_ctx is not null) and
(public.is_tenant_member(v_tenant_ctx) or
public.is_tenant_staff(v_tenant_ctx));
  elsif v_visibility = 'staff' then
    return (v_tenant_ctx is not null) and public.is_tenant_staff(v_tenant_ctx);
  else
    -- club visibility shipped later
    return false;
  end if;
end $$;



-- ------------------------------------------------------------
-- Profiles (extend existing) + privacy settings
-- ------------------------------------------------------------
alter table public.profiles
  add column if not exists handle text unique,
  add column if not exists avatar_url text,
  add column if not exists display_name text,
  add column if not exists location_city text,
  add column if not exists location_region text,
  add column if not exists comment_acknowledged_at timestamptz,
  add column if not exists updated_at timestamptz not null default now();

create table if not exists public.privacy_settings (
  user_id uuid primary key references auth.users(id) on delete cascade,
  biometrics_visibility text not null default 'followers', -- public|followers|
clubs|private
  workouts_visibility text not null default 'followers',
  injuries_visibility text not null default 'my_gyms',
  location_visibility text not null default 'city',
```

```
  updated_at timestamptz not null default now()
);

alter table public.privacy_settings enable row level security;

create policy "privacy_select_own"
on public.privacy_settings for select
using (auth.uid() = user_id);

create policy "privacy_insert_own"
on public.privacy_settings for insert
with check (auth.uid() = user_id);

create policy "privacy_update_own"
on public.privacy_settings for update
using (auth.uid() = user_id)
with check (auth.uid() = user_id);



-- ------------------------------------------------------------
-- Follow system (users + gyms/tenants)
-- ------------------------------------------------------------
create table if not exists public.follows_users (
  follower_id uuid not null references auth.users(id) on delete cascade,
  followee_id uuid not null references auth.users(id) on delete cascade,
  created_at timestamptz not null default now(),
  primary key (follower_id, followee_id)
);

alter table public.follows_users enable row level security;

create policy "follows_users_select_follower_or_followee"
on public.follows_users for select
using (auth.uid() = follower_id or auth.uid() = followee_id or
public.is_platform_admin());

create policy "follows_users_insert_own"
on public.follows_users for insert
with check (auth.uid() = follower_id);

create policy "follows_users_delete_own"
on public.follows_users for delete
using (auth.uid() = follower_id);

create index if not exists follows_users_followee_idx on
public.follows_users(followee_id, created_at desc);


create table if not exists public.follows_tenants (
  follower_id uuid not null references auth.users(id) on delete cascade,
  tenant_id uuid not null references public.tenants(id) on delete cascade,
  created_at timestamptz not null default now(),
  primary key (follower_id, tenant_id)
);

alter table public.follows_tenants enable row level security;

create policy "follows_tenants_select_follower_or_tenant_staff"
on public.follows_tenants for select
using (
  auth.uid() = follower_id
  or public.is_tenant_staff(tenant_id)
```

```
  or public.is_platform_admin()
);

create policy "follows_tenants_insert_own"
on public.follows_tenants for insert
with check (auth.uid() = follower_id);

create policy "follows_tenants_delete_own"
on public.follows_tenants for delete
using (auth.uid() = follower_id);

create index if not exists follows_tenants_tenant_idx on
public.follows_tenants(tenant_id, created_at desc);



-- ------------------------------------------------------------
-- Blocks (mutual invisibility)
-- ------------------------------------------------------------
create table if not exists public.blocks (
  blocker_id uuid not null references auth.users(id) on delete cascade,
  blocked_id uuid not null references auth.users(id) on delete cascade,
  created_at timestamptz not null default now(),
  primary key (blocker_id, blocked_id)
);

alter table public.blocks enable row level security;

create policy "blocks_select_own"
on public.blocks for select
using (auth.uid() = blocker_id);

create policy "blocks_insert_own"
on public.blocks for insert
with check (auth.uid() = blocker_id);

create policy "blocks_delete_own"
on public.blocks for delete
using (auth.uid() = blocker_id);



-- ------------------------------------------------------------
-- Social posts + media + reactions + comments + stats
-- ------------------------------------------------------------
create table if not exists public.posts (
  id uuid primary key default uuid_generate_v4(),
  author_user_id uuid references auth.users(id) on delete cascade,
  author_tenant_id uuid references public.tenants(id) on delete cascade,
  tenant_context_id uuid references public.tenants(id) on delete set null,
  club_id uuid,
  visibility public.post_visibility not null default 'public',
  content_text text,
  workout_ref jsonb not null default '{}'::jsonb,
  is_deleted boolean not null default false,
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now(),
  check (
    (author_user_id is not null and author_tenant_id is null)
    or (author_user_id is null and author_tenant_id is not null)
  )
);
```

```
alter table public.posts enable row level security;

-- READ: use centralized can_view_post()
create policy "posts_select_can_view"
on public.posts for select
using (public.can_view_post(id));

-- WRITE: user-authored posts
create policy "posts_insert_user"
on public.posts for insert
with check (
  (author_user_id = auth.uid() and author_tenant_id is null)
  or (author_tenant_id is not null and
public.is_tenant_staff(author_tenant_id))
);

create policy "posts_update_owner_or_staff"
on public.posts for update
using (
  (author_user_id = auth.uid())
  or (author_tenant_id is not null and
public.is_tenant_staff(author_tenant_id))
  or public.is_platform_admin()
)
with check (
  (author_user_id = auth.uid())
  or (author_tenant_id is not null and
public.is_tenant_staff(author_tenant_id))
  or public.is_platform_admin()
);

create policy "posts_delete_owner_or_staff"
on public.posts for delete
using (
  (author_user_id = auth.uid())
  or (author_tenant_id is not null and
public.is_tenant_staff(author_tenant_id))
  or public.is_platform_admin()
);

create index if not exists posts_created_idx on public.posts(created_at desc);
create index if not exists posts_tenant_ctx_idx on
public.posts(tenant_context_id, created_at desc);


create table if not exists public.post_media (
  id uuid primary key default uuid_generate_v4(),
  post_id uuid not null references public.posts(id) on delete cascade,
  media_type text not null, -- image|video
  storage_path text not null,
  width integer,
  height integer,
  duration_ms integer,
  created_at timestamptz not null default now()
);

alter table public.post_media enable row level security;

create policy "post_media_select_can_view_post"
on public.post_media for select
using (public.can_view_post(post_id));

create policy "post_media_insert_owner"
```

```
on public.post_media for insert
with check (exists(
  select 1 from public.posts p
  where p.id = post_id
    and p.is_deleted = false
    and (
      (p.author_user_id = auth.uid())
      or (p.author_tenant_id is not null and
public.is_tenant_staff(p.author_tenant_id))
    )
));


create table if not exists public.post_reactions (
  post_id uuid not null references public.posts(id) on delete cascade,
  user_id uuid not null references auth.users(id) on delete cascade,
  reaction public.reaction_type not null,
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now(),
  primary key (post_id, user_id)
);

alter table public.post_reactions enable row level security;

create policy "post_reactions_select_can_view_post"
on public.post_reactions for select
using (public.can_view_post(post_id));

create policy "post_reactions_insert_own"
on public.post_reactions for insert
with check (auth.uid() = user_id and public.can_view_post(post_id));

create policy "post_reactions_update_own"
on public.post_reactions for update
using (auth.uid() = user_id)
with check (auth.uid() = user_id);

create policy "post_reactions_delete_own"
on public.post_reactions for delete
using (auth.uid() = user_id);

create index if not exists post_reactions_post_idx on
public.post_reactions(post_id, created_at desc);


create table if not exists public.post_comments (
  id uuid primary key default uuid_generate_v4(),
  post_id uuid not null references public.posts(id) on delete cascade,
  user_id uuid not null references auth.users(id) on delete cascade,
  parent_id uuid references public.post_comments(id) on delete cascade,
  body text not null,
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now(),
  deleted_at timestamptz
);

alter table public.post_comments enable row level security;

create policy "post_comments_select_can_view_post"
on public.post_comments for select
using (public.can_view_post(post_id));

-- Comment friction: require comment_acknowledged_at set before first comment.
```

```
create policy "post_comments_insert_own"
on public.post_comments for insert
with check (
  auth.uid() = user_id
  and public.can_view_post(post_id)
  and exists(select 1 from public.profiles pr where pr.id = auth.uid() and
pr.comment_acknowledged_at is not null)
);

create policy "post_comments_update_own"
on public.post_comments for update
using (auth.uid() = user_id)
with check (auth.uid() = user_id);

create index if not exists post_comments_post_idx on
public.post_comments(post_id, created_at asc);


-- Cached stats (for feed ranking + UI counts)
create table if not exists public.post_stats (
  post_id uuid primary key references public.posts(id) on delete cascade,
  fire_count integer not null default 0,
  strong_count integer not null default 0,
  bolt_count integer not null default 0,
  clap_count integer not null default 0,
  mind_count integer not null default 0,
  comments_count integer not null default 0,
  last_engaged_at timestamptz,
  score_48h numeric not null default 0,
  updated_at timestamptz not null default now()
);

alter table public.post_stats enable row level security;

create policy "post_stats_select_can_view_post"
on public.post_stats for select
using (public.can_view_post(post_id));



-- ------------------------------------------------------------
-- Moderation primitives + notifications (CP27 scope)
-- ------------------------------------------------------------
create table if not exists public.reports (
  id uuid primary key default uuid_generate_v4(),
  reporter_id uuid not null references auth.users(id) on delete cascade,
  target_type text not null, -- post|comment|profile|tenant
  target_id uuid not null,
  reason text not null,
  details text,
  status text not null default 'open', -- open|reviewing|resolved|rejected
  created_at timestamptz not null default now(),
  resolved_at timestamptz,
  resolved_by uuid references auth.users(id)
);

alter table public.reports enable row level security;

create policy "reports_insert_own"
on public.reports for insert
with check (auth.uid() = reporter_id);

create policy "reports_select_platform_admin"
```

```
on public.reports for select
using (public.is_platform_admin());

create policy "reports_update_platform_admin"
on public.reports for update
using (public.is_platform_admin())
with check (public.is_platform_admin());


create table if not exists public.notifications (
  id uuid primary key default uuid_generate_v4(),
  user_id uuid not null references auth.users(id) on delete cascade,
  type text not null, -- reaction|comment|follow|booking|announcement
  actor_user_id uuid references auth.users(id),
  actor_tenant_id uuid references public.tenants(id),
  entity_type text,
  entity_id uuid,
  title text,
  body text,
  is_read boolean not null default false,
  created_at timestamptz not null default now()
);

alter table public.notifications enable row level security;

create policy "notifications_select_own"
on public.notifications for select
using (auth.uid() = user_id);

create policy "notifications_update_own"
on public.notifications for update
using (auth.uid() = user_id)
with check (auth.uid() = user_id);

create index if not exists notifications_user_idx on
public.notifications(user_id, created_at desc);
```

Appendix B — Triggers (post_stats + notifications)

```
-- ============================================================
-- Appendix B — Triggers (Corrected v6)
-- post_stats is maintained incrementally (no COUNT(*) fan-out)
-- ============================================================

-- Ensure a stats row exists for every post
create or replace function public.on_post_insert_init_stats()
returns trigger language plpgsql as $$
begin
  insert into public.post_stats(post_id)
  values (new.id)
  on conflict (post_id) do nothing;
  return new;
end $$;

drop trigger if exists trg_posts_init_stats on public.posts;
create trigger trg_posts_init_stats
after insert on public.posts
for each row execute function public.on_post_insert_init_stats();
```

```
-- Helper: apply reaction delta (+1 / -1) and refresh score
create or replace function public.apply_reaction_delta(pid uuid, r
public.reaction_type, delta int)
returns void language plpgsql as $$
begin
  insert into public.post_stats(post_id) values (pid)
  on conflict (post_id) do nothing;

  update public.post_stats ps
  set
    fire_count   = ps.fire_count   + (case when r = 'fire'   then delta else 0
end),
    strong_count = ps.strong_count + (case when r = 'strong' then delta else 0
end),
    bolt_count   = ps.bolt_count   + (case when r = 'bolt'   then delta else 0
end),
    clap_count   = ps.clap_count   + (case when r = 'clap'   then delta else 0
end),
    mind_count   = ps.mind_count   + (case when r = 'mind'   then delta else 0
end),
    last_engaged_at = now(),
    updated_at = now(),
    score_48h = (
      (ps.fire_count   + (case when r = 'fire'   then delta else 0 end))
    + (ps.strong_count + (case when r = 'strong' then delta else 0 end))
    + (ps.bolt_count   + (case when r = 'bolt'   then delta else 0 end))
    + (ps.clap_count   + (case when r = 'clap'   then delta else 0 end))
    + (ps.mind_count   + (case when r = 'mind'   then delta else 0 end))
    + (ps.comments_count * 1.5)
    )
  where ps.post_id = pid;
end $$;


create or replace function public.on_post_reaction_ins()
returns trigger language plpgsql as $$
begin
  perform public.apply_reaction_delta(new.post_id, new.reaction, 1);
  return new;
end $$;

create or replace function public.on_post_reaction_del()
returns trigger language plpgsql as $$
begin
  perform public.apply_reaction_delta(old.post_id, old.reaction, -1);
  return old;
end $$;

create or replace function public.on_post_reaction_upd()
returns trigger language plpgsql as $$
begin
  if new.reaction <> old.reaction then
    perform public.apply_reaction_delta(old.post_id, old.reaction, -1);
    perform public.apply_reaction_delta(new.post_id, new.reaction, 1);
  end if;
  return new;
end $$;

drop trigger if exists trg_post_reactions_ins on public.post_reactions;
create trigger trg_post_reactions_ins after insert on public.post_reactions
for each row execute function public.on_post_reaction_ins();
```

```
drop trigger if exists trg_post_reactions_del on public.post_reactions;
create trigger trg_post_reactions_del after delete on public.post_reactions
for each row execute function public.on_post_reaction_del();

drop trigger if exists trg_post_reactions_upd on public.post_reactions;
create trigger trg_post_reactions_upd after update on public.post_reactions
for each row execute function public.on_post_reaction_upd();



-- Helper: apply comments delta (+1 / -1) and refresh score
create or replace function public.apply_comment_delta(pid uuid, delta int)
returns void language plpgsql as $$
begin
  insert into public.post_stats(post_id) values (pid)
  on conflict (post_id) do nothing;

  update public.post_stats ps
  set
    comments_count = greatest(0, ps.comments_count + delta),
    last_engaged_at = now(),
    updated_at = now(),
    score_48h = (
      ps.fire_count + ps.strong_count + ps.bolt_count + ps.clap_count +
ps.mind_count
      + (greatest(0, ps.comments_count + delta) * 1.5)
    )
  where ps.post_id = pid;
end $$;


create or replace function public.on_post_comment_ins()
returns trigger language plpgsql as $$
begin
  if new.deleted_at is null then
    perform public.apply_comment_delta(new.post_id, 1);
  end if;
  return new;
end $$;

create or replace function public.on_post_comment_del()
returns trigger language plpgsql as $$
begin
  if old.deleted_at is null then
    perform public.apply_comment_delta(old.post_id, -1);
  end if;
  return old;
end $$;

create or replace function public.on_post_comment_upd()
returns trigger language plpgsql as $$
begin
  -- soft delete toggles affect count
  if old.deleted_at is null and new.deleted_at is not null then
    perform public.apply_comment_delta(new.post_id, -1);
  elsif old.deleted_at is not null and new.deleted_at is null then
    perform public.apply_comment_delta(new.post_id, 1);
  end if;
  return new;
end $$;

drop trigger if exists trg_post_comments_ins on public.post_comments;
```

```
create trigger trg_post_comments_ins after insert on public.post_comments
for each row execute function public.on_post_comment_ins();

drop trigger if exists trg_post_comments_del on public.post_comments;
create trigger trg_post_comments_del after delete on public.post_comments
for each row execute function public.on_post_comment_del();

drop trigger if exists trg_post_comments_upd on public.post_comments;
create trigger trg_post_comments_upd after update on public.post_comments
for each row execute function public.on_post_comment_upd();



-- ============================================================
-- Notifications triggers (MVP)
-- ============================================================
create or replace function public.notify_post_author_on_reaction()
returns trigger language plpgsql as $$
declare
  author uuid;
begin
  select author_user_id into author from public.posts where id = new.post_id;

  -- tenant posts: notifications handled later
  if author is null then
    return new;
  end if;

  -- no self notification
  if author = new.user_id then
    return new;
  end if;

  -- respect blocks
  if public.is_blocked_between(author, new.user_id) then
    return new;
  end if;

  insert into public.notifications(user_id, type, actor_user_id, entity_type,
entity_id, title, body)
  values (author, 'reaction', new.user_id, 'post', new.post_id, 'New reaction',
'Someone reacted to your post.');

  return new;
end $$;

drop trigger if exists trg_notify_reaction on public.post_reactions;
create trigger trg_notify_reaction
after insert on public.post_reactions
for each row execute function public.notify_post_author_on_reaction();


create or replace function public.notify_post_author_on_comment()
returns trigger language plpgsql as $$
declare
  author uuid;
begin
  select author_user_id into author from public.posts where id = new.post_id;

  if author is null then
    return new;
  end if;
```

```
  if author = new.user_id then
    return new;
  end if;

  if public.is_blocked_between(author, new.user_id) then
    return new;
  end if;

  insert into public.notifications(user_id, type, actor_user_id, entity_type,
entity_id, title, body)
  values (author, 'comment', new.user_id, 'post', new.post_id, 'New comment',
'Someone commented on your post.');

  return new;
end $$;

drop trigger if exists trg_notify_comment on public.post_comments;
create trigger trg_notify_comment
after insert on public.post_comments
for each row execute function public.notify_post_author_on_comment();
```

Appendix C — Token Ledger RPC (Transactional)

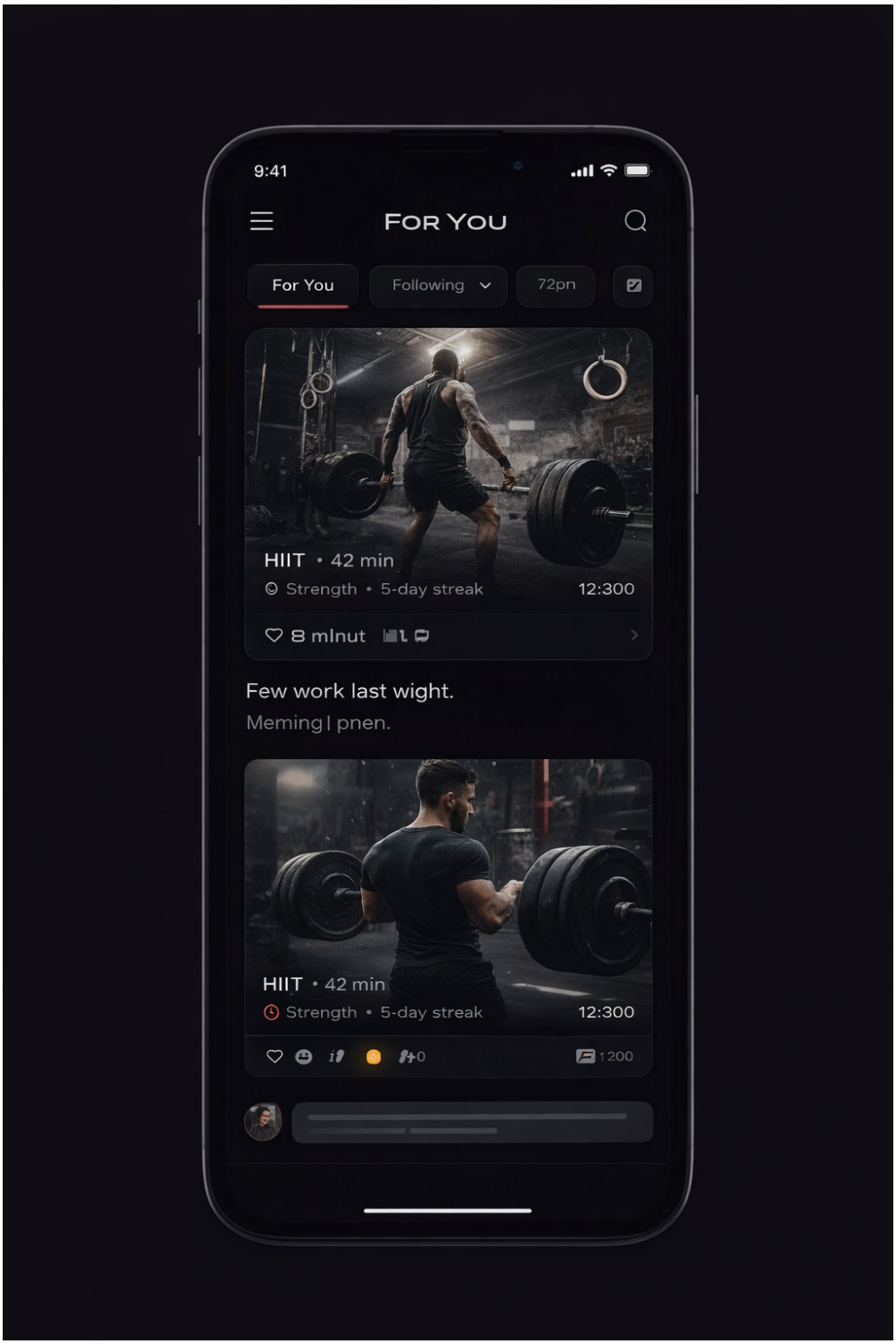
```
-- ============================================================
-- Tokens: spend_tokens / refund_tokens (ledger-based)
-- ============================================================

create or replace function public.spend_tokens(
  p_tenant_id uuid,
  p_user_id uuid,
  p_amount integer,
  p_ref_type text,
  p_ref_id uuid
)
returns integer
language plpgsql
security definer
as $$
declare
  bal integer;
begin
  -- Only service role should call this in production.
  -- In Supabase, enforce via function privileges and do NOT grant execute to
anon/authenticated.

  select balance into bal
  from public.token_wallets
  where tenant_id = p_tenant_id and user_id = p_user_id
  for update;

  if bal is null then
    insert into public.token_wallets(tenant_id, user_id, balance)
    values (p_tenant_id, p_user_id, 0);
    bal := 0;
  end if;

  if bal < p_amount then
    raise exception 'Insufficient tokens';
  end if;
```

```
  -- Ledger entry (negative spend)
  insert into public.token_transactions(tenant_id, user_id, kind, amount,
ref_type, ref_id, note)
  values (p_tenant_id, p_user_id, 'spend', -p_amount, p_ref_type, p_ref_id,
'Booking spend');

  update public.token_wallets
  set balance = balance - p_amount, updated_at = now()
  where tenant_id = p_tenant_id and user_id = p_user_id;

  select balance into bal
  from public.token_wallets
  where tenant_id = p_tenant_id and user_id = p_user_id;

  return bal;
end $$;
```

Appendix D — UI References (Locked Layout Screens)

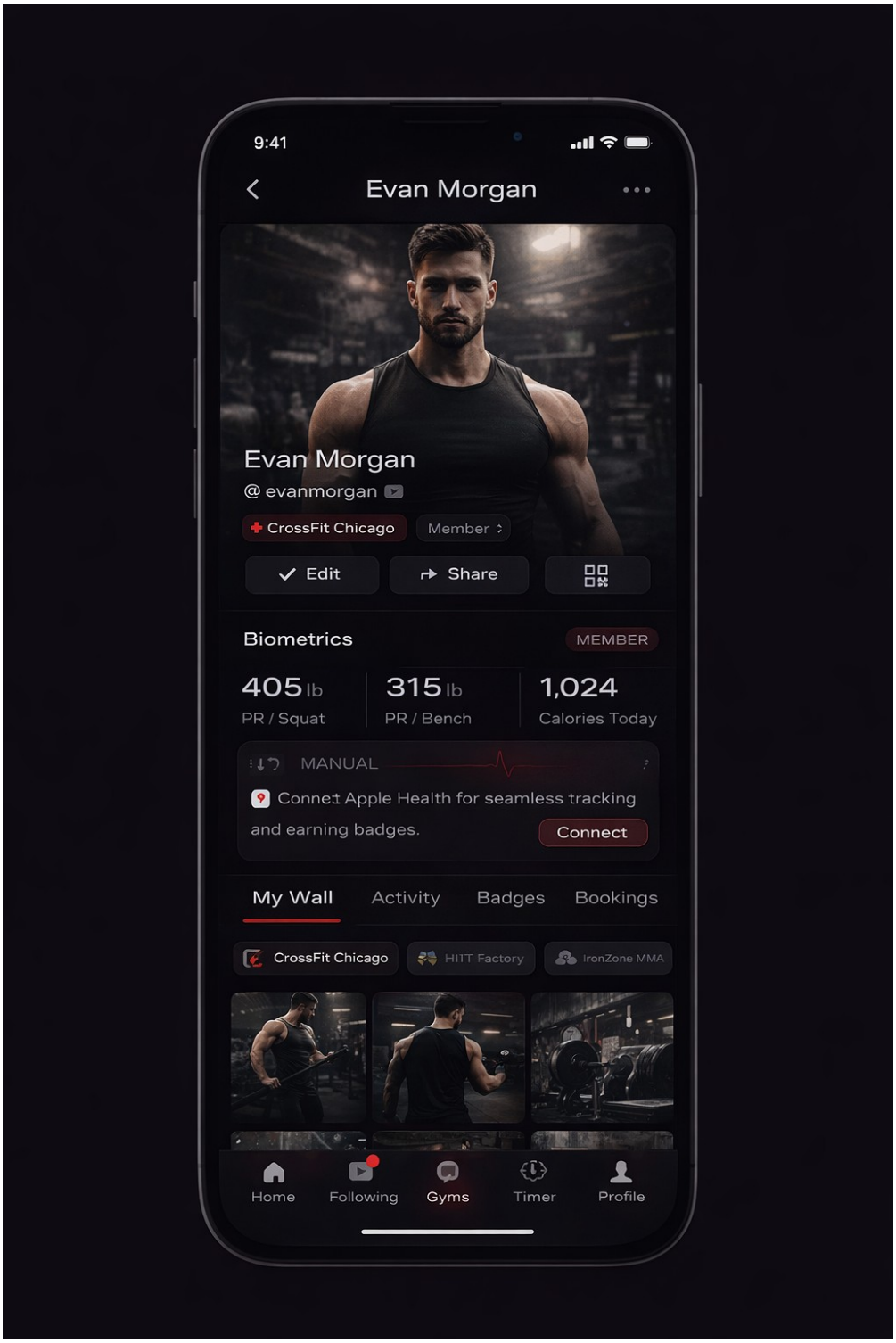
These are visual references to match the build. Layout and hierarchy are locked; styling must use tokens in Section 3.

FYP / Feed (portrait)



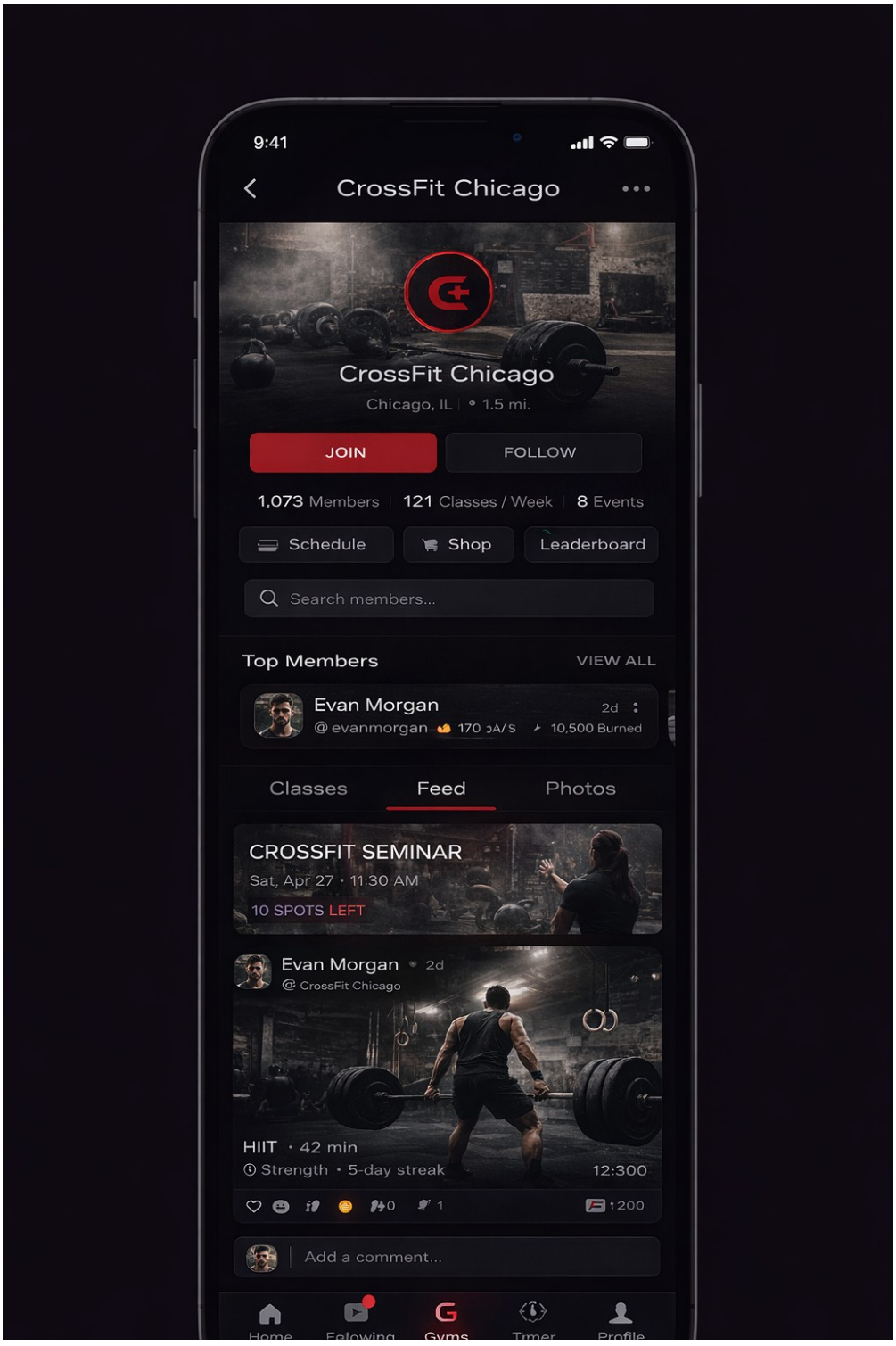
FYP / Feed (portrait) — reference mockup

Profile (populated)



Profile (populated) — reference mockup

Gym public profile (preview)



Gym public profile (preview) — reference mockup

Gym member view (gated/unlocked)

ELEVATION HIIT

UNLIMITED

Member Access Unlocked

Members-Only Challenge

6-Week Strength Cycle

Progress Tracking Enabled

Private Class Templates

Hybrid 45 • Strength 60 • Recovery Flow

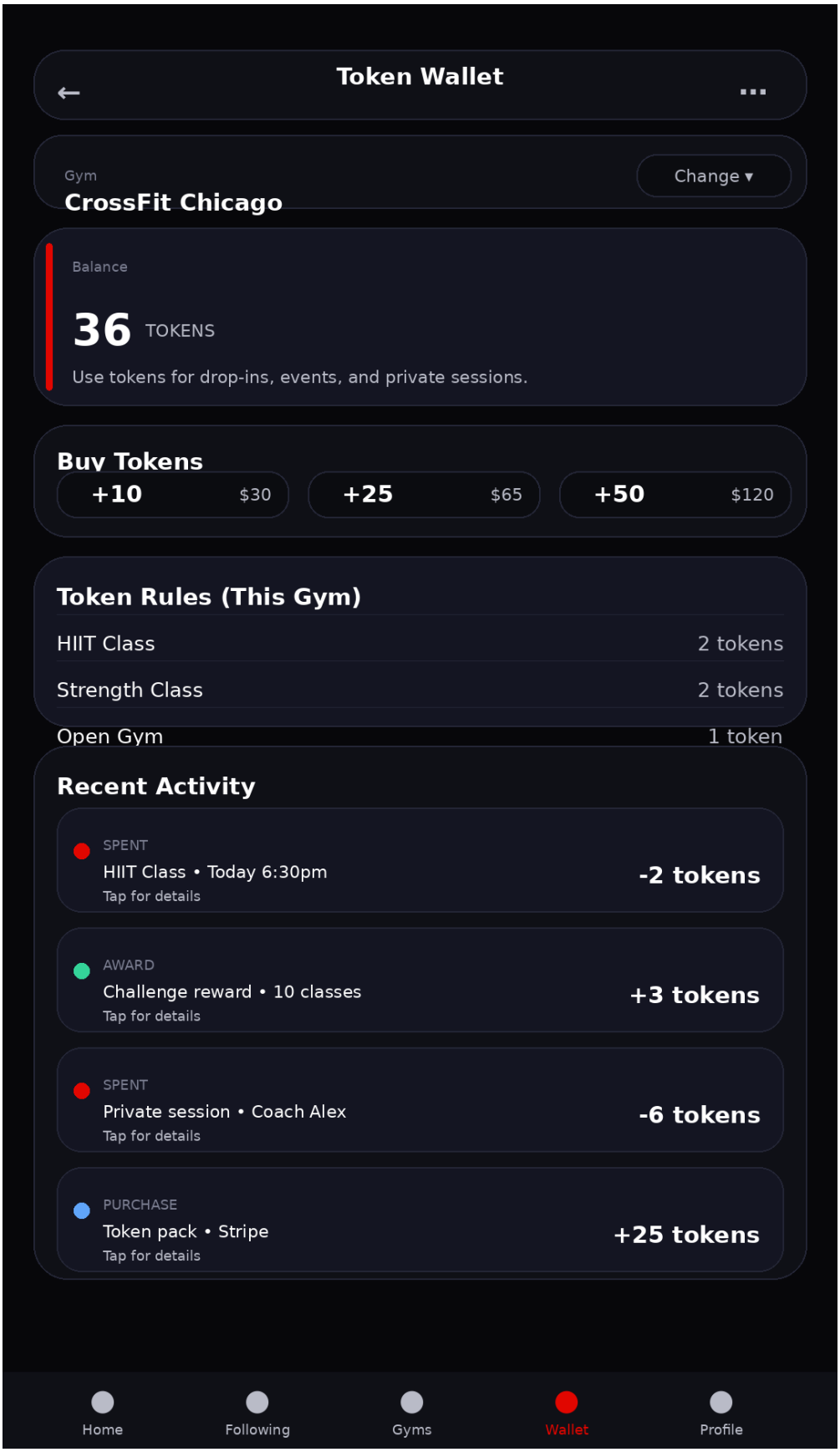
Member Leaderboard

1. Evan M. • 12 Streak •

2. Maya L. • 10 Streak •

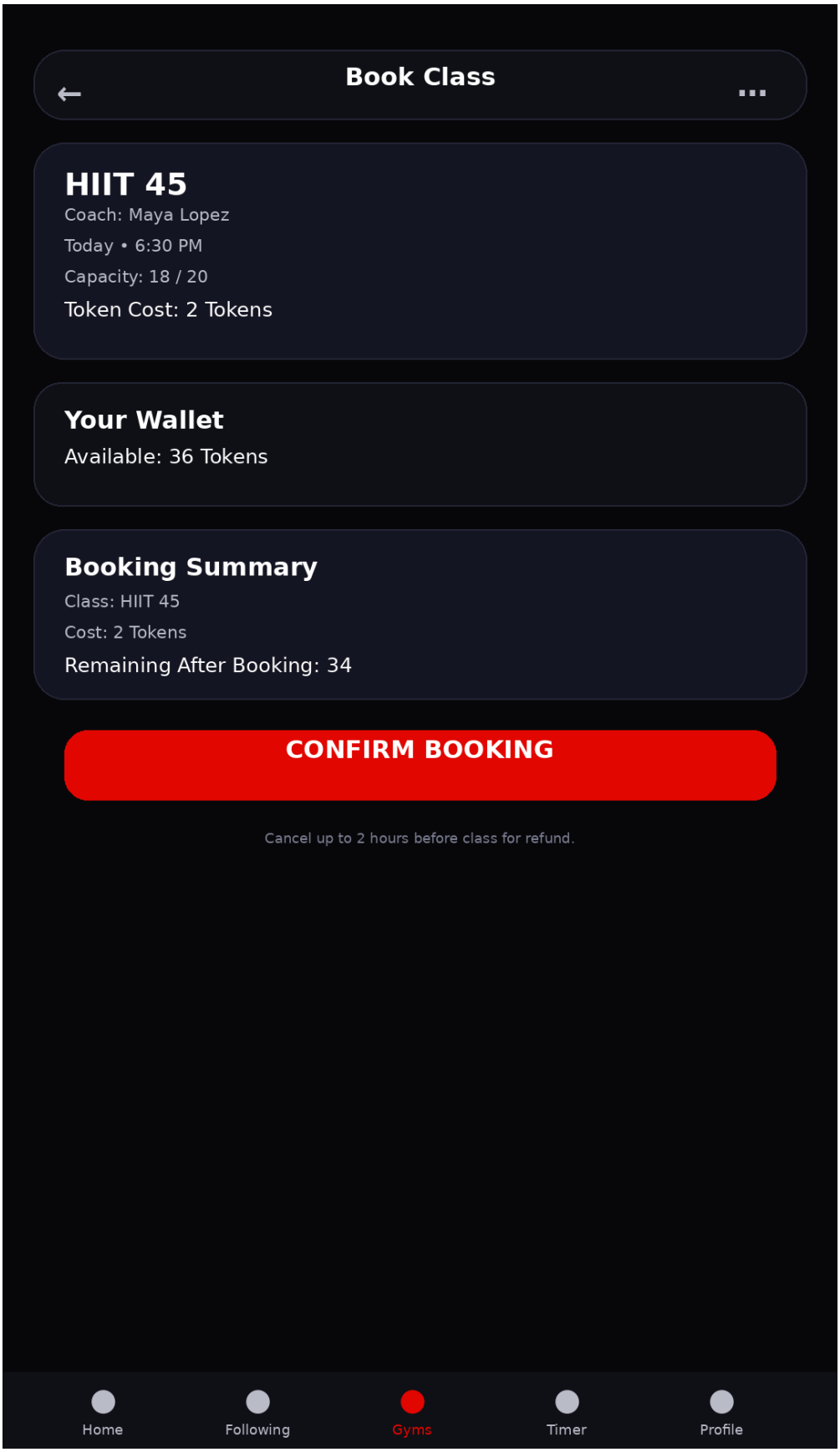
Gym member view (gated/unlocked) — reference mockup

Token wallet


Token Wallet
Gym
CrossFit Chicago
Change ▾
Balance
36 TOKENS
Use tokens for drop-ins, events, and private sessions.
Buy Tokens
+10 $30
+25 $65
+50 $120
Token Rules (This Gym)
HIIT Class 2 tokens
Strength Class 2 tokens
Open Gym 1 token
Recent Activity
SPENT
HIIT Class • Today 6:30pm
Tap for details
-2 tokens
AWARD
Challenge reward • 10 classes
Tap for details
+3 tokens
SPENT
Private session • Coach Alex
Tap for details
-6 tokens
PURCHASE
Token pack • Stripe
Tap for details
+25 tokens
Home
Following
Gyms
Wallet
Profile

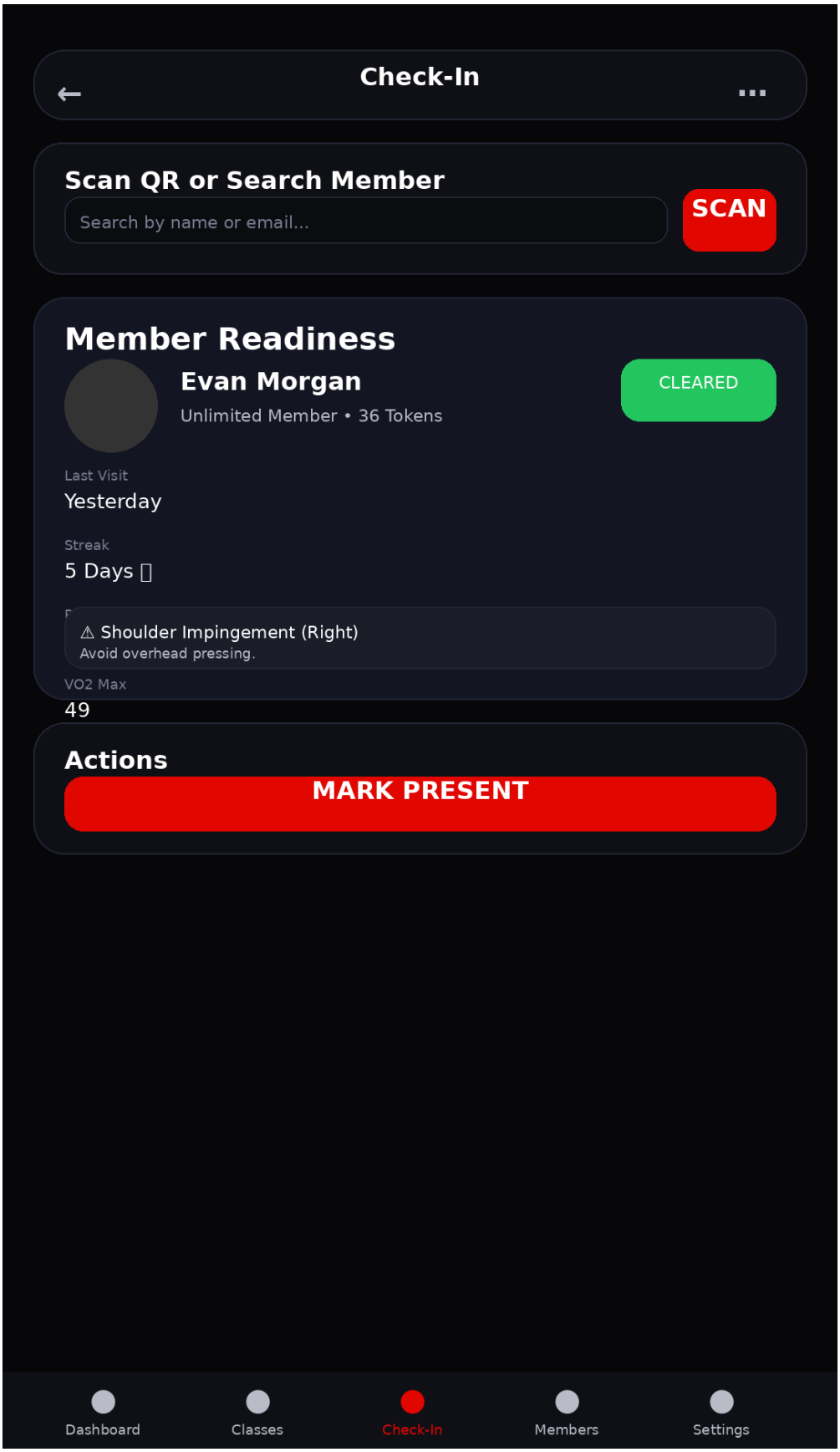
Token wallet — reference mockup

Class booking flow


Book Class
HIIT 45
Coach: Maya Lopez
Today • 6:30 PM
Capacity: 18 / 20
Token Cost: 2 Tokens
Your Wallet
Available: 36 Tokens
Booking Summary
Class: HIIT 45
Cost: 2 Tokens
Remaining After Booking: 34
CONFIRM BOOKING
Cancel up to 2 hours before class for refund.
Home
Following
Gyms
Timer
Profile

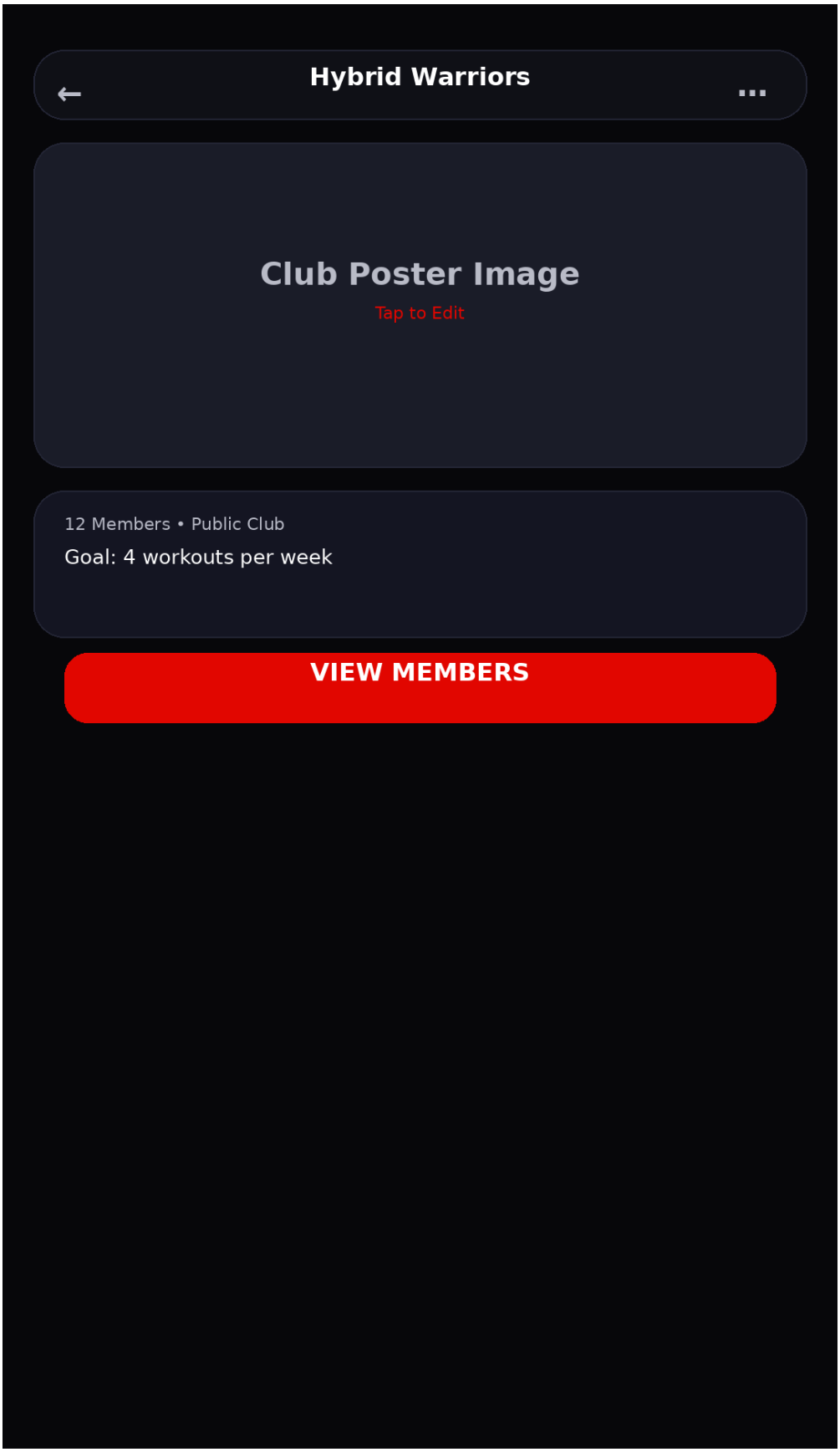
Class booking flow — reference mockup

Business check-in (Member Readiness Card)


Check-In
Scan QR or Search Member
Search by name or email...
SCAN
Member Readiness
Evan Morgan
Unlimited Member • 36 Tokens
CLEARED
Last Visit
Yesterday
Streak
5 Days
⚠ Shoulder Impingement (Right)
Avoid overhead pressing.
VO2 Max
49
Actions
MARK PRESENT
Dashboard
Classes
Check-In
Members
Settings

Business check-in (Member Readiness Card) — reference mockup

Club main with poster


Hybrid Warriors
Club Poster Image
Tap to Edit
12 Members • Public Club
Goal: 4 workouts per week
VIEW MEMBERS

Club main with poster — reference mockup

Club members (faces + performance-first)

Club members (faces + performance-first) — reference mockup

Appendix E — Missing Items Added (So Nobody Has to Ask Later)

- These items were not always explicit in earlier drafts, but are required for a real-world social + commerce platform. They are approved and included in this blueprint by default.
- Reactions table (post_reactions) supporting 5 encouragement reactions (not just likes).
- post_stats cached counts + trending score (prevents COUNT() overload and enables ranking).
- Notifications system (table + triggers) for reactions/comments/follows/bookings.
- user_moderation_state for slow-mode, strike tracking, and shadow bans.
- content_moderation pipeline for images/text (auto-hold content pending review).
- Onboarding activation flow (handle + face + goals + first follows).
- Storage bucket strategy aligned with visibility (avoid leaking private media).
- Audit logging for money + moderation actions (compliance + debugging).
- Seek pagination standard everywhere (no offset).
- Testing requirements (RLS tests + e2e) and environment separation (staging/prod).

If the team builds exactly what's in this blueprint, there should be zero ambiguity left at implementation time.

Appendix F — Cross-Platform OS + Store Best Practices (2026 Lock)

This platform ships Web-first (PWA), then wraps for iOS/Android where native capabilities matter (biometrics, background delivery, push reliability). The checklists below are the implementation rules so NDYRA behaves correctly across Apple / Google / Windows ecosystems.

F.1 iOS (Safari PWA + App Store Wrapper)

- PWA baseline: ensure all core member flows work in Safari (iOS) without native wrappers (feed, profile, booking, clubs).
- Web Push: supported for Home Screen web apps on iOS 16.4+; require Add to Home Screen + user-initiated permission prompt.
- Native wrapper (Capacitor): required for HealthKit reads + background updates + higher push reliability when we go App Store.
- App Store readiness: maintain App Privacy details, and never use health data for advertising targeting; keep explicit consent screens and revocation paths.
- Deep links: configure Universal Links so ndyra:// and https://ndyra.com links route into the app or PWA cleanly.

F.2 Android (Chrome PWA + Play Store Wrapper)

- Health platform: build on Health Connect for OS-level health aggregation; do not start new work on Google Fit APIs (deprecated).
- PWA install: ensure install prompt + offline caching behaves well in Chrome/Edge on Android.

- Native wrapper (Capacitor): required for Health Connect access + background work; optional for camera QR scan performance.
- Push: Web Push works well on Android; native FCM push later for richer actions and better delivery metrics.
- App Store readiness: keep Google Play Data Safety disclosure updated as features ship (health data is sensitive).

F.3 Windows + Desktop (Installable PWA)

- Primary desktop runtime: PWA installed via Edge/Chrome; ensure keyboard navigation + responsive layout works at 1200px+ widths.
- Manifest icons: include the recommended Windows tile icon sizes so the app looks 'native' when installed.
- Optional Store: package PWA for Microsoft Store later (PWABuilder/MSIX) if we want discovery + enterprise distribution.

F.4 Cross-Platform Compatibility Rules

- Camera: QR scanning must work in PWA (getUserMedia). Provide clear permission UI and fallback 'type code' if camera fails.
- Media: portrait-first feed; generate multiple image sizes and lazy-load to prevent memory spikes on mobile.
- Auth: support passwordless-ready architecture (email OTP now; passkeys later) without breaking existing accounts.
- Accessibility: WCAG AA contrast, 44px targets, reduced motion, and keyboard support on desktop.
- Performance: no blocking JS on initial paint; defer heavy modules until after auth/session resolves.

Appendix G — Brand Asset Export Checklist (NDYRA Logo + Icon System)

This is the production asset pipeline so designers can hand Aelric a 'single source of truth' and he can generate every required icon size without re-drawing anything.

G.1 Master Assets (Source of Truth)

- Master symbol: NDYRA 'Y' mark as pure vector (SVG + PDF).
- Master wordmark: NDYRA wordmark as pure vector (SVG + PDF).
- Master lockup: wordmark + orbit/star lockup as pure vector (SVG + PDF).
- Color tokens: Black (#000000), White (#FFFFFF), NDYRA Red (#e40001), optional 'Glow Red' for marketing only (never required for UI).
- Do not ship glow/gradients as the only version. Always include flat versions for scaling and accessibility.

G.2 Required Exports (Web + iOS + Android + Windows)

- Web/PWA manifest icons: 192x192 PNG and 512x512 PNG (from flat icon).
- iOS App Store icon: 1024x1024 PNG (no transparency). Xcode will generate sizes from this master in the wrapper app.
- Android adaptive icon: foreground layer + background layer (vectors preferred) and export PNGs for 432x432 as a safe high-res baseline.
- Windows tiles: include 44x44, 71x71, 150x150, 310x150, 310x310 and store logo 50x50 in manifest (or let PWABuilder generate).
- Social/marketing: 1080x1080, 1920x1080, 1080x1920 in both flat + glow variants.

G.3 Naming Conventions (No Chaos)

- assets/brand/ndyra_symbol.svg
- assets/brand/ndyra_wordmark.svg
- assets/brand/ndyra_lockup.svg
- assets/icons/pwa/icon-192.png, icon-512.png
- assets/icons/ios/appstore-1024.png
- assets/icons/android/adaptive_foreground.png, adaptive_background.png

Appendix H — 2026 OS/Store Compliance + Privacy Manifests (Verified)

This appendix hardens the blueprint for 2026 platform realities across Apple / Google / Windows ecosystems. It is written as a checklist Aelric can implement without guesswork. Anything here is treated as required unless explicitly marked optional.

H.1 iOS/iPadOS — PWA + Native Wrapper Rules

- PWA first: every CP27 member flow must work in Safari iOS (FYP, Following, Post Detail, Create Post, Profile, Notifications).
- Web Push: implement Push API + Service Worker push handler; only prompt permission after a user action (e.g., 'Enable Notifications') and only after the app is installed to Home Screen for iOS-capable web push.
- Native wrapper (Capacitor): required for HealthKit reads, background delivery (where allowed), and the most reliable push workflows; keep web UI shared.
- HealthKit policy: treat all HealthKit/Apple Health-derived data as sensitive; never use it for advertising targeting, data brokerage, or resale. Build explicit consent and revocation flows.
- Apple privacy manifest: include and maintain a privacy manifest file and required-reason API declarations as the app evolves; keep third-party SDK manifests/signatures up to date.
- App privacy disclosures: keep App Store privacy details aligned with reality (health/fitness data collection, identifiers, analytics).

H.2 Android — PWA + Play Store Wrapper Rules

- Health platform: implement OS-level health aggregation via Health Connect. Avoid new Google Fit API work; plan migration for any legacy Fit usage.
- Permissions: implement runtime permission UX for camera (QR), notifications, and activity/health connectors (native wrapper).
- Adaptive icons: ship foreground/background layers for the wrapper app; use flat icon source-of-truth.
- Background limits: assume strict background execution limits; prefer OS-supported scheduled jobs and user-initiated sync.

H.3 Web/Desktop — Windows/macOS/Linux Browsers

- Primary desktop runtime: installed PWA via Chrome/Edge/Safari (macOS) with full keyboard navigation.
- Service Worker caching: cache app shell + static assets; never cache authenticated API responses unless encrypted/local-first explicitly.
- Security headers: enforce CSP, HSTS, X-Content-Type-Options, Referrer-Policy, Permissions-Policy, and strict cookie settings for any non-Supabase cookies.
- Media: generate responsive image variants and lazy-load; enforce portrait-first feed rendering at 9:16 with safe-area spacing.

H.4 Store Review, Legal, and Compliance Checklist

- Terms + Privacy: ship platform Terms of Service + Privacy Policy before public beta; include health data handling and user rights (export/delete).
- User rights ops: implement account deletion + data export workflows (profile, posts, media, biometrics, bookings) with audit logging.
- Copyright/IP: implement DMCA-style takedown workflow for user media. Disallow reposting copyrighted content without rights.
- Child safety: block underage accounts where required, and enforce safe defaults; maintain reporting tools.
- Incident response: define a security incident process (breach triage, user notification, rollback).

Appendix I — Missing Tables + Migrations Addendum (Must Add)

During verification we identified several 'must-have' tables referenced in the blueprint but not fully defined in the CP27 skeleton. They are included here to eliminate implementation ambiguity.

I.1 user_consent_tenants (biometrics + modifications sharing per gym)

Purpose: user-controlled consent per tenant. This enforces the rule: gyms only see what the member explicitly shares.

SQL (paste into Supabase SQL editor):

```
create table if not exists public.user_consent_tenants (
  tenant_id uuid not null references public.tenants(id) on delete cascade,
  user_id uuid not null references auth.users(id) on delete cascade,
  share_biometrics boolean not null default false,
  share_modifications boolean not null default false,
  share_checkin_photo boolean not null default true,
  updated_at timestamptz not null default now(),
  primary key (tenant_id, user_id)
);

alter table public.user_consent_tenants enable row level security;

create policy "consent_select_own_or_staff"
on public.user_consent_tenants for select
using (
  auth.uid() = user_id
  or public.is_tenant_staff(tenant_id)
  or public.is_platform_admin()
);

create policy "consent_upsert_own"
on public.user_consent_tenants for insert
with check (auth.uid() = user_id);

create policy "consent_update_own"
on public.user_consent_tenants for update
using (auth.uid() = user_id)
with check (auth.uid() = user_id);
```

I.2 user_modifications (member-managed injuries/modifications)

Purpose: store member-reported injuries/modifications. Visibility is controlled by privacy settings + per-tenant consent.

SQL:

```
create table if not exists public.user_modifications (
  id uuid primary key default uuid_generate_v4(),
  user_id uuid not null references auth.users(id) on delete cascade,
  title text not null,
  details text,
  severity text, -- mild|moderate|severe
  is_active boolean not null default true,
  updated_at timestamptz not null default now(),
  created_at timestamptz not null default now()
);

alter table public.user_modifications enable row level security;

create policy "mods_select_own"
on public.user_modifications for select
using (auth.uid() = user_id or public.is_platform_admin());

create policy "mods_write_own"
on public.user_modifications for all
using (auth.uid() = user_id or public.is_platform_admin())
with check (auth.uid() = user_id or public.is_platform_admin());
```

I.3 tenant_member_notes (staff-only coaching notes)

Purpose: staff internal notes at the gym level (not visible to other members). Supports the Readiness Card.

SQL:

```
create table if not exists public.tenant_member_notes (
  id uuid primary key default uuid_generate_v4(),
  tenant_id uuid not null references public.tenants(id) on delete cascade,
  user_id uuid not null references auth.users(id) on delete cascade,
  created_by uuid not null references auth.users(id),
  note text not null,
  tags text[] not null default '{}',
```

```
  created_at timestamptz not null default now()
);

alter table public.tenant_member_notes enable row level security;

create policy "notes_select_staff"
on public.tenant_member_notes for select
using (public.is_tenant_staff(tenant_id) or public.is_platform_admin());

create policy "notes_write_staff"
on public.tenant_member_notes for insert
with check (public.is_tenant_staff(tenant_id) or public.is_platform_admin());

create policy "notes_delete_admin"
on public.tenant_member_notes for delete
using (public.is_tenant_admin(tenant_id) or public.is_platform_admin());

create index if not exists tenant_member_notes_tenant_user_idx on
public.tenant_member_notes(tenant_id, user_id, created_at desc);
```

I.4 user_moderation_state + content_moderation_queue (anti-abuse primitives)

Purpose: implement slow-mode, strike tracking, shadow bans, and auto-hold queues for moderation. This is required to keep NDYRA from becoming a negativity dumping ground.

SQL:

```
create table if not exists public.user_moderation_state (
  user_id uuid primary key references auth.users(id) on delete cascade,
  strikes integer not null default 0,
  slow_mode_until timestamptz,
  shadow_banned_at timestamptz,
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now()
);

alter table public.user_moderation_state enable row level security;

create policy "ums_select_platform_admin"
on public.user_moderation_state for select
using (public.is_platform_admin());
```

```
create policy "ums_update_platform_admin"
on public.user_moderation_state for update
using (public.is_platform_admin())
with check (public.is_platform_admin());

create table if not exists public.content_moderation_queue (
  id uuid primary key default uuid_generate_v4(),
  target_type text not null, -- post|comment|profile|media
  target_id uuid not null,
  detected_by text not null, -- automated|user_report|gym_admin
  reason text not null,
  status text not null default 'queued', -- queued|reviewing|approved|rejected
  created_at timestamptz not null default now(),
  reviewed_at timestamptz,
  reviewed_by uuid references auth.users(id),
  notes text
);

alter table public.content_moderation_queue enable row level security;

create policy "cmq_select_platform_admin"
on public.content_moderation_queue for select
using (public.is_platform_admin());

create policy "cmq_write_platform_admin"
on public.content_moderation_queue for all
using (public.is_platform_admin())
with check (public.is_platform_admin());
```

Appendix J — Verification Checklist (No-Hiccup Build)

This is the final ‘double check everything’ list. Aelric must run this before CP27 signoff and before any Boca pilot onboarding.

- Schema: RLS enabled on every public table; no policy has `using (true)` unless the data is intentionally public and harmless.
- Schema: can_view_post() is used on post_media / post_reactions / post_comments / post_stats policies (no private leaks).
- Storage: post-media bucket policies are visibility-aware (public posts readable publicly; non-public posts not readable via direct URL).
- Auth: handle is unique, validated, and immutable after 30 days (optional lock) to prevent impersonation.
- Feed: seek pagination only; never offset pagination on posts.
- Rate limits: server endpoints for media upload signing, Stripe checkout, token ledger, and moderation are rate limited.
- Abuse: new accounts have conservative posting limits until verified; strikes/slow-mode pipeline exists.
- Observability: webhook idempotency log exists; errors are recorded; alerts are configured for failures.
- Anti-Drift: run `NDYRA_CP27_AntiDrift_Audit_v7.sql` (enforce=true) and confirm PASS.
- Anti-Drift: capture and commit the RLS fingerprint printed by the audit script; any change requires an ADR note.
- Cross-platform: PWA manifest icons correct; iOS safe-area respected; Android back button behavior correct; desktop keyboard navigation works.
- Privacy: data export + deletion flows verified end-to-end.