Architecture Design Document (Revised)

Project: seda.fm

Document Type: ARCHITECTURE DESIGN

Version: 2

Created by: ARCHITECT

Created: Sat Sep 06 2025

Architecture Design Document seda.fm Recommendation Engine (Rooms, Playlists, Artists)
Prepared by: Principal Software Architect

Executive summary seda.fm's recommendation engine must deliver personalized, social, and diverse recommendations (rooms, playlists, artists) in near real time while respecting privacy, cost, and operational constraints. This design uses the approved MVP stack (Node/NestJS on Railway, Supabase Postgres + pgvector, Supabase Auth for identity, Socket.IO + Supabase Realtime for presence, BullMQ + Upstash Redis for background jobs, Firebase FCM for push, Supabase Storage for assets). The architecture is hybrid: daily batch for global ranking and model refresh, hourly/nightly re-vectorization for catalog drift, and real-time delta pipelines for social triggers (friend presence, room joins, votes) so notifications and feed updates feel immediate.

Key clarifications from product and engineering Q&A are integrated:

- Embeddings provider: OpenAI (text-embedding-3-small, 1536-dim) primary; Cohere standby via an EmbeddingsService adapter. Provider abstraction, metadata/versioning, circuit-breakers, and cost enforcement ($1,500/month cap) are required.
- Vector store strategy: Start with pgvector on Supabase (HNSW) — do not dual-write production traffic at launch. Implement VectorStore abstraction and support shadow/writer for small cohorts; migrate to managed vector stores only when defined thresholds are met.
- Auth: Supabase Auth (GoTrue) is canonical for production and sandbox; roles/authorization asserted via DB (users.role) and RLS using auth.uid().
- Apple Music: session/user-initiated import only for MVP; persist only the developer token server-side (encrypted) and do not persist per-user musicUserTokens.
- Spotify and other providers: persist refresh tokens encrypted in linked_accounts to allow background syncs where supported (Spotify).

- Onboarding: minimal (account creation + genre selection) with first meaningful feed within 60s. Provider linking is optional and deferred to progressive prompts.
- Notifications & feed: prioritize in-app feed cards, ephemeral in-app toasts for friend-driven triggers, and FCM push only as opt-in high-confidence events (1 push/day cap). Visual rules for friend avatars, dismissal semantics, and undo are specified.
- Privacy: provide explicit Export and Delete flows in Settings; deletion guarantees (immediate exclusion from ranking; physical deletion within 30 days) and clear messaging.

This document specifies components, schema updates, processing flows, ranking algorithm blueprint, SLOs, security/privacy controls, scalability plans, testing & rollout recommendations, risks and mitigations, and operational runbooks. It incorporates the Q&A clarifications where relevant.

1. Goals, constraints, and acceptance criteria

- Business goals: increase engagement, retention and growth via discovery; maximize social lift (friend-driven CTR).
- Non-functional constraints:
    - Managed services preferred, cost-conscious.
    - Privacy: opt-in linking, minimal stored data (summaries not raw logs), GDPR/CCPA compliance, EU-only processing for certain cohorts.
    - Scale targets (MVP Year 1): 10–25k DAU, peak concurrent ~1k–2.5k, 1k–5k events/sec sustained (burst to 10k), recommendation queries 10–50 QPS initially.
    - Latency targets: rec retrieval median <120ms, P95 <250ms; ANN-only P95 <120ms. Real-time notifications P99 <500ms (target 200ms).
- Measured success (from PRD): Engagement ≥20% sessions with rec activity, Quality thresholds (≥70% listens >30s, ≤15% skips), Growth & Delight metrics (detailed in instrumentation section).

2. High-level architecture Components and responsibilities (MVP):

- Auth/Identity: Supabase Auth (GoTrue) — JWTs used for authentication; authoritative role stored in users.role in DB for authorization and RLS.
- Backend/API: Node.js/NestJS services hosted on Railway — BFF API, ingestion adapters, recommendation service endpoints, admin services.
- Primary DB & vector store: Supabase Postgres with pgvector extension (canonical entities, embeddings, RLS). EU-residency partitions/projects used for EU-only users.
- Real-time transport: Socket.IO for chat/presence; Supabase Realtime for DB-change pub/sub.

- Background jobs: BullMQ workers (Railway) + Upstash Redis for job queues and orchestration.
- Push delivery: Firebase Cloud Messaging (FCM) for mobile/web push.
- Object storage: Supabase Storage for cover art, thumbnails, user-uploaded assets.
- Embeddings provider(s): OpenAI primary; Cohere configured as secondary in EmbeddingsService adapter. Ability to use EU self-hosted SentenceTransformers runner for EU-only processing.
- Observability & experiments: PostHog for analytics, LaunchDarkly (or Supabase feature flags) for experiments. Monitoring via Prometheus-compatible metrics and Sentry for errors.

3. Core design decisions (rationale)

- Single source of truth: Supabase Postgres simplifies ops, supports pgvector, RLS, backups and queries for metrics.
- Embeddings strategy: use hosted OpenAI embeddings for MVP (1536-dim), with provider abstraction and embedding_meta recorded for audit/versioning. Cohere standby. Do not dual-write production traffic at launch. Implement shadow writes for limited cohorts for migration testing.
- Hybrid processing cadence: batch nightly for global ranking, hourly for popularity shifts, real-time for social triggers to meet UX latency and cost goals.
- Summary signals only: store top-N items (artists, genres, playlists) and derived taste embeddings — do not persist raw play-by-play logs in production.
- Auth model: Supabase Auth primary; roles authoritative in DB (users.role). RLS policies use auth.uid(), users.role for access control.
- Apple Music: session-only import for MVP; persist server developer token only.

4. Data model (canonical) Key tables (Postgres). Preserve prior schema and add clarified fields (region, embedding provenance, deletion flags). Use snake_case; JSONB for flexible properties; vector columns for pgvector.

- users
    - id UUID PK
    - auth_user_id TEXT UNIQUE (supabase sub)
    - created_at TIMESTAMP
    - dob DATE
    - region TEXT
    - role TEXT (enum: user, artist, admin, super_admin)
    - preferences JSONB (genres, languages)

    - privacy_settings JSONB
    - deleted_at TIMESTAMP NULL
    - Indexes: idx_users_auth_user_id
- linked_accounts
    - id UUID PK
    - user_id UUID FK -> users(id)
    - provider TEXT ('spotify','apple','youtube',...)
    - external_user_id TEXT (pseudonymized)
    - scopes JSONB
    - refresh_token_encrypted BYTEA (KMS envelope-encrypted) — persisted for providers that support background syncs (Spotify). Apple per-user tokens are NOT persisted in MVP.
    - access_token_expires_at TIMESTAMP
    - last_synced_at TIMESTAMP
    - created_at TIMESTAMP
    - Indexes: idx_linked_accounts_user_provider
- artists
    - id UUID PK
    - external_ids JSONB
    - name TEXT
    - genres TEXT[] or JSONB
    - popularity_score FLOAT
    - embedding VECTOR(dim)
    - embedding_meta JSONB (provider, model, dim, region, version, computed_at)
    - last_updated TIMESTAMP
    - Indexes: gin on genres, vector index
- playlists
    - id UUID PK
    - external_ids JSONB
    - name TEXT
    - owner_user_id UUID NULL
    - provider TEXT
    - track_count INT

    - embedding VECTOR(dim)
    - embedding_meta JSONB
    - created_at TIMESTAMP
    - Indexes: idx_playlists_provider, vector index
- rooms
    - id UUID PK
    - name TEXT
    - created_by UUID
    - genre TEXT
    - cover_art_url TEXT
    - current_participants INT (cached)
    - embedding VECTOR(dim)
    - embedding_meta JSONB
    - created_at TIMESTAMP
    - Indexes: idx_rooms_genre, vector index
- user_profiles
    - user_id UUID PK
    - top_artists UUID[] (artist ids)
    - top_genres TEXT[]
    - top_playlists UUID[]
    - taste_embedding VECTOR(dim)
    - embedding_meta JSONB (provider, model, region, version, computed_at)
    - last_computed TIMESTAMP
    - privacy_flags JSONB
    - deleted_at TIMESTAMP NULL
- user_activity_summary
    - user_id UUID
    - listens JSONB (top N artists counts + recency buckets)
    - room_memberships JSONB (room_id -> last_joined_at)
    - follows JSONB (artists followed)
    - last_updated TIMESTAMP
- friends

    - user_id UUID
    - friend_user_id UUID
    - relationship_type TEXT
    - created_at TIMESTAMP
    - Indexes: idx_friends_user
- recommendations
    - id UUID PK
    - user_id UUID FK
    - entity_type TEXT ('room','playlist','artist')
    - entity_id UUID
    - score FLOAT
    - model_version TEXT
    - source TEXT ('batch','real-time')
    - created_at TIMESTAMP
    - expires_at TIMESTAMP
    - dismissed BOOLEAN
    - dismissed_until TIMESTAMP
    - context JSONB (surface, friend_presence_count, avatars_shown)
    - Indexes: idx_recs_user_created_at, idx_recs_user_entity
- embeddings_audit
    - entity_uuid UUID
    - entity_type TEXT
    - provider TEXT
    - model TEXT
    - dim INT
    - region TEXT
    - version TEXT
    - computed_at TIMESTAMP
    - deleted_at TIMESTAMP NULL
- fact_events (transient for analytics; short retention)
    - id SERIAL
    - user_id UUID

    - event_type TEXT (rec_click, listen_30s, rec_dismiss, track_skip, room_join)
    - props JSONB
    - created_at TIMESTAMP
    - Retention: purge older than X days (configurable)

Materialized views:

- mv_user_top_candidates(user_id) — precompute top N candidate entities per user for fast retrieval.
- mv_genre_top_rooms(genre) — cached top rooms per genre.

Notes: Embedding provenance now includes provider region and version. EU-only vectors must be stored in EU-region DB partition/project and flagged in embedding_meta.region='eu'.

5. Vector indexing & embedding strategy

- Dimensionality & provider:
    - Start with 1536-dim embeddings using OpenAI text-embedding-3-small as primary. Record provider, model_name, dim, version, region in embedding_meta for audit/rollback.
    - Cohere configured as secondary/standby in EmbeddingsService; do not full dual-write at launch.
- Index: use pgvector HNSW for ANN (suitable up to ~500k–1M vectors per table). HNSW parameters: M=12–16 (start M=16), ef_construction=100–200, ef_search default 64; tune ef_search at query time.
- Storage & partitioning:
    - Store vectors partitioned by entity_type (artists_vectors, playlists_vectors, rooms_vectors, user_vectors).
    - For EU-only users, persist vectors in EU-region Supabase project/table (separate physical storage).
- Refresh cadence:
    - Catalog entities: re-embed on metadata change or every 30–60 days.
    - User taste vectors: incremental recompute hourly; full recompute nightly; immediate compute on provider-link or major import.
    - Batch size: start with 64–256; default 128 with batching window ~200ms.
- EmbeddingsService:

    - Provider adapter pattern with per-provider rate-limiter, batching, circuit-breaker, retries, and cost-metering hooks.
    - Circuit-breaker policy: open if provider error rate >2% or P95 latency >1s sustained for 60s; open duration 60s; half-open probes at low traffic.
    - Timeouts: connect ≤300ms, read ≤2.5–3s; total per-request budget ≤6s including retries.
    - Shadow-write capability: background shadow writes for a small % (1–5%) of vectors for QA/migration only.
- EU-only processing:
    - Per-user region flag; EmbeddingsService routes EU users to EU provider endpoints when available, else to EU self-hosted SentenceTransformers runner deployed in EU.
    - Store EU-user vectors in EU-region Supabase project and record embedding_meta.region='eu'.
    - Validators prevent cross-region writes/exports.

6. Recommendation pipeline and ranking Hybrid pipeline blending content-based, collaborative, and social signals.

Candidate generation:

- Content-based via ANN: user taste_embedding -> nearest entity embeddings restricted by metadata (genre, language) to narrow search space.
- Collaborative filtering: offline item-item similarity (ALS/implicit MF) and precomputed item embeddings stored for re-rank.
- Social graph boost: friends' presence, recent follows, friend_count.
- Trending & freshness: popularity signals, recency windows.

Re-ranking:

- Features: embedding_similarity, collaborative_score, friend_presence_count, recency_score, popularity, diversity_penalty, user_privacy_flags, provider_affinity.
- Model: weighted linear model initially; plan lightweight GBDT or learning-to-rank later. Include diversity penalty to enforce 30% novel content.
- Business filters: respect user opt-outs, RLS privacy, provider link preferences, notification caps.
- Diversity: implement cluster-based de-duplication and simple determinantal heuristics to ensure at least 30% new content.

Outputs:

- Top-N per user stored in recommendations table with TTL and model_version.
- Feed retrieval: cached MV or Redis per-user top-50 for sub-120ms retrieval.
- Daily batch: full recompute; Real-time: small incremental re-ranks on social events.

Personalization weights & gating:

- Default initial weights: content 0.5, collaborative 0.25, social 0.15, popularity 0.1 (tunable). A/B experiments must confirm and adjust.
- Notification thresholds: send only when score delta exceeds push_threshold and quality rules satisfied.

7. Real-time and event processing

- Transport & presence:
    - Socket.IO clustered with Redis adapter for presence and chat.
    - Supabase Realtime used for DB-change pub/sub where appropriate.
- Event bus:
    - BullMQ + Upstash Redis for job queuing. Plan Kafka/Redpanda if throughput grows (50k+ events/sec).
- Real-time flow:
    - Presence events are aggregated server-side (250–500ms coalescing) for fanout efficiency.
    - For friend-driven events, broadcast to friends-of-actor only, not whole room.
    - Immediate in-app ephemeral Toasts via Socket.IO for high-signal friend events (first friend joins, threshold-crossing). Respect cooldowns and session limits (max 3 toasts/session, min 10 min cooldown).
    - Recommender computes delta re-ranks for affected users asynchronously and updates MV/cache.
- Idempotency: event ids, idempotency keys, worker dedupe logic.

Presence and UI UX specifics:

- Update cadence: client coalesces socket events and re-renders at up to 1 update/sec (500–1000ms window) to avoid thrash.
- Avatar rendering: show up to 4 avatars + +N overflow badge; if privacy hides avatars, show count only.
- Optimistic join: immediate local UI with server ack reconciliation; rollback on failure.

8. Notification & delivery model Surfaces & priority:

- Primary: In-app discovery feed card (“Recommended Rooms for You”) — persistent.
- Secondary: In-app ephemeral toasts via Socket.IO — session-level friend-driven triggers.
- Tertiary: FCM push — opt-in, high-confidence events only. Hard cap: 1 strong recommendation push per user per 24h.

Friend-driven visual affordances:

- Avatars: up to 4 inline (32px); overflow shown as +N.
- Copy: personalized (“Alex is here — join them”) when one friend; “3 of your friends are in #electronic right now” for >1 friend.
- Privacy: if friend hides presence, show count-only without names/avatars.

Delivery guardrails:

- Push send conditions:
    - User opted into push AND
    - Score >= push_threshold AND
    - friend_presence_count >=2 OR (friend_presence_count ==1 AND affinity_high) OR novelty condition met.
    - Respect quiet hours and per-user preferences.
- Ephemeral toasts:
    - Max 3 per session; cooldown 10 minutes per user between toasts; do not count against daily push cap.
- Feed:
    - No per-day hard limit, but re-ranker applies diversity constraints and suppresses previously dismissed items.

Dismissal & undo semantics:

- Quick dismiss: removes from view with 10s Snackbar “Undo” option.
- Durable suppression:
    - Default dismiss = hide for 24 hours (dismissed_until = now + 24h).
    - “Not interested / Don’t show again” = downweight/avoid for 30 days.
    - “Never show” explicit block persists until user reverses in Settings.

- All dismiss/feedback events are recorded (fact_events) and influence re-ranker immediately and in nightly batch.

9. Privacy, security & compliance

- Consent & onboarding:
    - Explicit opt-in for provider linking. Provider linking is optional and deferred from onboarding critical path.
- Auth & roles:
    - Supabase Auth as canonical provider. auth.uid() used in RLS policies. Authoritative roles stored in users.role. JWTs kept minimal; do not rely on mutable role claims for authorization.
- Token policies:
    - Spotify: persist refresh tokens encrypted (linked_accounts.refresh_token_encrypted) using KMS envelope encryption. Delete on unlink.
    - Apple: do NOT persist per-user musicUserTokens in MVP. Persist developer token encrypted; rotate proactively (recommend 90-day rotation cadence).
- Data minimization:
    - Store only mapped entities and summary signals. Embeddings and summaries are personal data — deletable within 30 days on request.
- Deletion & export flows:
    - Settings must surface “Export my data” and “Delete my imported data” actions. Export produces a secure ZIP; delete excludes data from ranking immediately and schedules physical purge within 30 days (background job).
    - On delete request: mark user_profiles.deleted_at or set flags; invalidate caches; run purge jobs to remove vectors, summaries, recommendations, and stored assets per user.
- Encryption & secrets:
    - KMS-managed keys for token encryption; rotate keys every 90 days. TLS 1.2+ for all transport.
- Access controls:

    - Postgres RLS enforced for rows; service_role key used only server-side. Least privilege for service accounts. Audit logs for sensitive operations.

RLS policy guidance (implementation-ready):

- Use auth.uid() in policies and query users.role in DB for role checks (examples in Appendix).

10. Observability, metrics, and instrumentation

- Telemetry:
    - Metrics: per-service latency, error rates, queue depths, vector-store latencies, embedding provider success & latency, cache hit rates.
    - Tracing: OpenTelemetry end-to-end tracing (API Recommender pgvector).
    - Logs: structured JSON; Sentry for errors. Scrub PII.
- Business metrics:
    - Event instrumentation for rec_click, listen_30s, track_skip, rec_dismiss; forward to PostHog & warehouse for analysis.
- Embeddings cost telemetry:
    - Per-job & per-provider cost attribution. Daily run-rate projections with alerts at 80% ($1,200) and 100% ($1,500) caps.
    - Enforcement: automatic throttles on non-critical background embeddings at 80%; emergency stop at 100%.
- SLOs & alerts:
    - Recommendation API: P95 <250ms; alert if P95 >400ms for 5 minutes.
    - ANN-only: P50 <50ms; P95 <120ms; P99 <250ms.
    - Real-time notifications: P99 <500ms (target 200ms).
    - Embedding provider error >2% -> failover/circuit.

11. Deployment, testing, and rollout

- Environments: local/dev, QA, sandbox (staging with synthetic data), production. Separate Supabase projects per environment; Supabase Auth used for all.
- CI/CD:
    - Build/test pipeline (unit, integration, contract tests). Deploy to Railway; canary/promote with feature flags.

- Canary & rollout:
    - Canary 5–10% traffic with LaunchDarkly or Supabase feature flags. Monitor hard-gates (operational & quality). Automatic rollback on gate failure.
- Testing:
    - Unit tests (Jest), E2E (Cypress), integration tests for provider connectors using mock/sandbox keys.
    - Load & chaos tests in sandbox using Event Replayer and Mock Provider Service.
- Staging capabilities:
    - Provide seeded sandbox with realistic friend graphs, catalog sizes, synthetic embeddings, and event replayer for load/chaos testing.
    - Provide mock/local EmbeddingsService and an optional small EU self-hosted SentenceTransformers instance for EU routing tests.

12. Scalability & evolution plan Short-term (MVP):

- pgvector on Supabase with HNSW for up to ~100k–1M vectors total; partition by entity_type and tune ef_search for recall/latency tradeoff.
- Socket.IO with Redis adapter; scale horizontally.
- BullMQ workers with Upstash Redis; monitor queue depths to scale workers.

Medium-term:

- Offload event streams to Kafka/Redpanda when throughput demands exceed BullMQ capability.
- Evaluate migration to managed vector stores (Pinecone/Qdrant) when thresholds met (per-index >500k vectors, ANN P95 >120ms after tuning, sustained ANN QPS >200–500).
- Implement dual-write only as staged migration (shadow writes read-side experiments staged cut-over).

Long-term:

- Microservices split by domain (ingestion, recommender, notifications, social graph).
- Feature store & model serving layers (Feast, Redis + model manager) for low-latency ranking.

13. Cost & procurement considerations

- Enforce embedding spend cap: $1,500/month. Alerts at 80% & 100% and automated throttles on non-critical jobs.

- Prefer Supabase-managed Postgres initially; plan for dedicated DB (RDS/Aurora) when data grows.
- Evaluate vector-store costs vs engineering overhead when scaling.
- Use managed SaaS (Supabase, LaunchDarkly, Upstash, Railway) to reduce ops burden.

14. Risks & mitigations

- Provider rate limits/outages:
    - Circuit-breakers, cache-first design, backoff respecting Retry-After, user-visible messages. Shadow-write only for limited cohorts.
- Privacy/regulatory issues:
    - Opt-in linking, per-user region routing for EU-only processing, deletion/export flows, and audit logs.
- Embedding cost & vendor lock-in:
    - EmbeddingsService abstraction, shadow writes, per-job cost attribution, and cap enforcement.
- Real-time scale (concert-level spikes):
    - Aggregate presence server-side, throttle fan-out, convert avatar lists to counts, use message brokers for buffering.
- Model bias & echo chamber:
    - Diversity penalty, rotate new artists, monitor NDCG/recall, and use product-level constraints (30% novelty).
- Operational complexity of pgvector at scale:
    - Partitioning, read replicas, index rebuild runbooks, and migration path to managed vector stores.

15. Sequence flows (high level) A. Linking and initial ingest
16. User authenticates (Supabase Auth) and optionally links Spotify (OAuth). Refresh token stored encrypted; Apple handled as session import (developer token persisted server-side).
17. Background job pulls summary data (top 50 artists, genres, playlists) and normalizes into canonical schema.

18. Compute user taste vector (EmbeddingsService) and store in user_profiles.taste_embedding with embedding_meta.

B. Onboarding & first feed

1. Minimal onboarding: account creation + choose ≥3 genres.
2. Server uses genre-seeded cached top-N (mv_genre_top_rooms) and quick seed taste vector to deliver initial feed within 60s.
3. If user links provider, enqueue import/embedding jobs; update feed when complete, notify via Socket.IO.

C. Daily batch recompute

1. Nightly job extracts candidates via ANN (pgvector), CF, popularity.
2. Re-ranker combines features; writes top-N into recommendations table and caches MV/Redis.

D. Real-time social trigger

1. Friend enters room presence event published (aggregated & coalesced).
2. NotificationService pushes ephemeral in-app toast to relevant friends; optionally enqueue push respecting daily cap.
3. Recommender computes delta boost and updates cached recommendations for affected users.
4. Operational runbooks & maintenance

- Provider outage: circuit-breaker open fallback to cached vectors and metadata heuristics; resume after half-open probe success.
- Token compromise: rotate KMS keys & provider tokens, notify affected users, follow incident runbook.
- Vector index rebuild: offline rebuild with blue-green swap; schedule in low-traffic windows.
- Deletion workflow: immediate exclusion from models, background physical purge within 30 days; audit and email confirmation on completion.
- Embedding cost spike: automated throttles at thresholds; manual override requires executive approval.

17. Staging & QA capabilities (detailed)

- Sandbox Supabase project with canonical schema, RLS, and pgvector enabled.

- Data seeding scripts with size profiles (Smoke / Medium / Large) to simulate production-like catalogs and friend graphs.
- Mock Provider Service: OAuth flows and deterministic provider responses with error/latency injection.
- Mock EmbeddingsService: deterministic embeddings (1536d), configurable latency/failure profiles, batch semantics matching production.
- Event Replayer: configurable headless Socket.IO clients and server-side event injection to simulate joins, votes, listens at controlled QPS and bursts.
- Optional EU self-hosted SentenceTransformers runner for EU-only processing validation.

18. Acceptance gates & rollout criteria Operational hard gates (must pass for canary promotion):

- Recommendation API: P95 <250ms (fail if P95 >350ms).
- Real-time notifications: P99 <500ms (fail if >1s).
- Error rate: rec endpoints 5xx <0.1% (fail if >0.5%).
- Privacy checks: RLS, deletion flows validated.

Business/Quality gates (no-regressions during canary):

- Recommendation CTR: no >10% relative drop; absolute safety floor defined per baseline.
- listen_30s: not drop >5pp relative and remain ≥60%.
- skip rate and rec_dismiss within thresholds (see instrumentation).

Experiment tooling:

- LaunchDarkly preferred for feature gating; Supabase feature flags acceptable for early experiments. PostHog for analytics and funnel analysis. Statistical thresholds: $p<0.05$, 80% power; min impressions ~5k per variant where applicable.

19. Open questions & next immediate decisions

- Confirm launch resource allocations for embedding spend cap enforcement and emergency manual override process.
- Provision EU-hosted container for SentenceTransformers to validate EU-only processing early.
- Confirm preference for LaunchDarkly vs Supabase flags for canary management (LaunchDarkly recommended for production-scale experimentation).

Appendix A — Operational knobs, thresholds, and defaults

- Embedding defaults: dim=1536; batch_size=128; batch_window=200ms.

- Circuit-breaker: error_rate_threshold=2%; latency_threshold(P95)=1s; open_duration=60s.
- Embedding cost guardrails: alert_at=80% ($1,200/mo), emergency_throttle_at=100% ($1,500/mo).
- pgvector HNSW params: M=16, ef_construction=200, ef_search default=64.
- ANN SLOs: P50 <50ms, P95 <120ms, P99 <250ms.
- Notification caps: max_push_per_user_per_24h = 1 (strong rec); ephemeral_toasts_per_session = 3 (cooldown 10m).
- Deletion window: immediate exclusion; physical purge within 30 days.

Appendix B — RLS policy sketches (implementation-ready)

- Use auth.uid() mapping to users.id and check users.role in policies. Example pseudocode:

-- Allow users to read own profile CREATE POLICY users_select_own ON public.users FOR SELECT USING (auth.uid() = id);

-- Allow rooms update by owner or admin CREATE POLICY rooms_update_owner_admin ON public.rooms FOR UPDATE USING (EXISTS (SELECT 1 FROM public.users u WHERE u.id = auth.uid() AND ((u.role = 'artist' AND u.id = created_by) OR u.role = 'admin')));

Appendix C — Privacy & deletion workflow (API contract summary)

- POST /privacy/export-data: enqueues export job, produces signed URL to download ZIP; artifact expiry 7 days.
- POST /privacy/delete-imports: requires re-auth; immediate exclusion from ranking; enqueue background purge; user notified on completion; physical purge within 30 days.
- Audit: privacy_jobs table with statuses (queued, in_progress, completed, failed).

Closing This architecture balances product needs (fast, social, privacy-preserving recommendations) with operational constraints (managed services, cost control). It provides an explicit embedding strategy with cost enforcement, an EU-only processing path for compliant cohorts, a minimal yet effective onboarding and initial feed plan, and a clear migration path from pgvector to managed vector stores when scale requires. Immediate next steps (implementation milestones and API contracts) can be produced on request to convert this architecture into a tracked implementation roadmap.
