# Leep Audio MVP – Supabase Validation & Backend Implementation To-Do

This document outlines two major responsibilities for the backend developer role on Leep Audio MVP:  
1) Supabase validation and verification of implemented features.  
2) Backend implementation tasks outside of Supabase to complete integration and ensure stability.

## 1) Supabase Validation Checklist

This checklist ensures that every feature implemented in Supabase works as expected. Each item should be checked off after successful verification.

### Environment & Keys

* ☐ Verify Supabase Project URL, anon key, and service role key are stored in a secure environment.
* ☐ Confirm Data API is enabled in Supabase (Settings → API).
* ☐ Regenerate keys only if previously exposed and update .env files across all environments.

### Authentication & Profiles

* ☐ Create a test user in Supabase Auth and verify profile auto-creation in public.profiles.
* ☐ Run query: SELECT id, role FROM public.profiles ORDER BY created\_at DESC LIMIT 1;
* ☐ Delete the test user and confirm cascade deletion in profiles table.

### Roles & Permissions

* ☐ Promote one profile to admin manually and verify it persists.
* ☐ Run SELECT enum\_range(NULL::user\_role); to confirm all expected roles exist.

### Songs Table

* ☐ Insert a song as a logged-in user and verify ownership-based RLS enforcement.
* ☐ Update the song’s title and confirm success (auth.uid() == artist\_id).
* ☐ Attempt update with mismatched UUID → expect RLS denial.
* ☐ Call RPC: SELECT publish\_song(<song\_id>); → expect is\_published = true.

### Comments & Realtime

* ☐ Insert a comment with author\_id == auth.uid() → success.
* ☐ Try mismatched author\_id → fail (RLS).
* ☐ Enable Realtime and confirm new comments trigger client event subscriptions.
* ☐ Delete a comment as artist → success.

### Reviews & Tips

* ☐ Insert review with valid rating (1–5) → success.
* ☐ Insert invalid rating → CHECK constraint failure.
* ☐ Insert tip (amount\_cents > 0) → success; invalid amount → fail.

### Events (Analytics)

* ☐ Insert events (play, view) for test songs.
* ☐ Run SELECT \* FROM artist\_dashboard(<artist\_id>); → verify returned counts.

### Storage Buckets

* ☐ Upload audio under audio/<user\_id>/<uuid>.mp3 → success.
* ☐ Upload artwork under artwork/<user\_id>/<uuid>.jpg → success.
* ☐ Upload invalid MIME or oversized file → blocked.
* ☐ Attempt cross-user folder access → fail (RLS).

### Admin RPCs

* ☐ As admin, run admin\_takedown\_song(<song\_id>) → is\_published = false.
* ☐ As non-admin, expect forbidden exception.
* ☐ Run admin\_delete\_comment(<comment\_id>) → row deleted.

### Backups & Logs

* ☐ Verify backups under Settings → Database → Backups.
* ☐ Trigger manual backup before new migrations.
* ☐ Check logs under Logs → Database for failed policies or RPC calls.

## 2) Backend Implementation Responsibilities (Outside Supabase)

These tasks complete your backend responsibilities for integrating the Supabase implementation with the Go API service.

### Environment Setup

* ☐ Initialize Go module: go mod init leepaudio/backend.
* ☐ Store secrets securely (SUPABASE\_URL, SUPABASE\_ANON\_KEY, SUPABASE\_SERVICE\_ROLE\_KEY).
* ☐ Configure local .env and production environment variables.

### Authentication Middleware

* ☐ Implement middleware to validate Supabase JWTs from the Authorization header.
* ☐ Verify signatures against Supabase JWKS or via /auth/v1/user endpoint.
* ☐ Extract sub (UUID) from JWT and attach it to request context.

### Role-Based Access

* ☐ Add helper to retrieve user role from public.profiles (cache for short TTL).
* ☐ Restrict admin endpoints to role == 'admin'.
* ☐ Ensure other routes enforce role-appropriate access (artist, producer, fan).

### Supabase Client Integration

* ☐ For user-scoped actions, forward user JWT to Supabase REST endpoints using anon key.
* ☐ For admin actions (e.g., moderation), use service role key.
* ☐ Map HTTP errors (401, 403, 409, 422) to consistent Go responses.

### API Endpoints to Implement

* ☐ POST /songs → inserts new song (user JWT, RLS enforced).
* ☐ POST /songs/:id/publish → calls publish\_song RPC.
* ☐ POST /projects → inserts new project (owner\_id = auth.uid()).
* ☐ POST /projects/:id/invite → inserts into project\_invitations (owner only).
* ☐ POST /stems → handles file upload + database insert.
* ☐ POST /tips → inserts tips.
* ☐ POST /events → logs play/view event.
* ☐ GET /analytics/artist/:id/summary → calls artist\_dashboard function.

### Storage Handling

* ☐ Generate signed URLs for playback via Supabase Storage API using service role key.
* ☐ Ensure client uploads only valid MIME and size-limited files.
* ☐ Add optional verification for audio MIME (magic bytes check).

### Error Handling & Validation

* ☐ Map unique constraint (409), RLS failures (403), check constraints (422).
* ☐ Return clear messages for failed inserts, updates, or RPC calls.
* ☐ Implement retries for transient network errors.

### Monitoring & Observability

* ☐ Implement structured logging with user\_id, route, and latency.
* ☐ Add Prometheus-compatible metrics (request count, errors, latency).
* ☐ Expose /healthz endpoint to validate Supabase connectivity.

### Security & Compliance

* ☐ Never expose service role key to client or frontend build.
* ☐ Enforce HTTPS and HSTS headers in production.
* ☐ Rotate keys if compromised and update all environments.
* ☐ Use short-lived signed URLs (< 1 hour) for secure media playback.

### Testing & QA

* ☐ Unit test all Go endpoints using mock Supabase responses.
* ☐ Integration test full flow: create user → upload song → publish → comment → view analytics.
* ☐ Verify all RLS and admin functions from backend API return expected Supabase responses.