**Vettale — Admin Booking Enablement (Unclaimed Admin‑Created Clients)**

**Architecture & Decision Log • Date: Aug 21, 2025 • Owner: Admin Booking WG**

**0) Executive summary**

**We enabled admin‑only bookings for unclaimed, admin‑created clients while preserving the existing client self‑service flow. The server now accepts \_client\_id in admin RPCs only when clients.admin\_created = true and clients.user\_id IS NULL. Otherwise, admin flows continue to resolve the client via \_client\_user\_id → clients.user\_id.**

**We corrected availability to be per‑staff profile, added atomic, set‑based locks to prevent double‑booking, and ensured appointment\_staff.role is always set. UI now branches payloads (claimed vs unclaimed). No changes to client self‑service.**

**Canonical naming: we do not use “provider id.” The system uses staff\_profile\_id everywhere. Frontend still sends \_provider\_ids for backward compatibility, which we alias internally to staff profile IDs.**

**1) Problem statement**

* **Admins must be able to book for clients created in the admin UI that have not claimed their account.**
* **Prior design only allowed resolving clients by user\_id (claimed users). Unclaimed admin‑created clients lacked a supported path.**
* **Availability checks were not scoped per staff and could double‑book under concurrency.**
* **appointment\_staff.role is NOT NULL and was not always set.**

**2) Investigation findings (from repo + Cursor)**

**Admin booking RPCs in use (SECURITY INVOKER):**

* **create\_booking\_admin**
* **create\_booking\_admin\_override**
* **create\_admin\_booking\_with\_dual\_services**

**Callsites:**

* **AdminBookingPage.tsx (dual‑service)**
* **AdminManualBooking.tsx + adminBookingUtils.ts (single + override)**

**Clients table (before): had user\_id, needs\_registration; no admin\_created / created\_by.**

**Admin client creation flow (before): direct supabase.from('clients').insert(...) in AdminClients.tsx (no markers set besides needs\_registration).**

**Terminology: availability/assignments use staff\_profile\_id; there is no provider\_id in the schema.**

**3) Changes implemented**

**3.1 Schema markers for admin‑created clients**

* **Migration: add columns on public.clients**
  + **admin\_created boolean NOT NULL DEFAULT false**
  + **created\_by uuid NULL (+ FK to auth.users(id) with ON DELETE SET NULL)**
  + **Indexes: (admin\_created), (user\_id)**
* **AdminClients.tsx: ensure admin UI inserts set:**
  + **admin\_created = true, created\_by = user.id, needs\_registration = true, and leave user\_id = NULL until claim.**

**3.2 Admin booking RPCs: \_client\_id enablement + guardrails**

**Patched functions:**

* **create\_booking\_admin**
* **create\_booking\_admin\_override**
* **create\_admin\_booking\_with\_dual\_services**

**Add parameter: \_client\_id uuid DEFAULT NULL (back‑compatible; UI unchanged initially)**

**Guardrails (top of each function):**

* **Enforce admin: IF NOT is\_admin(auth.uid()) THEN RAISE ... END IF;**
* **If \_client\_id provided → resolve client and require:**
  + **clients.admin\_created = true**
  + **clients.user\_id IS NULL**
  + **Else RAISE Explicit \_client\_id allowed only for admin‑created, unclaimed clients.**
* **Else (no \_client\_id) → existing \_client\_user\_id → clients.user\_id resolution.**
* **Pet ownership check: pet must belong to the resolved client.**

**3.3 Staff‑scoped availability (no “provider”)**

* **Alias UI param \_provider\_ids uuid[] → v\_staff\_profile\_ids uuid[].**
* **Validate at least one staff profile was provided.**
* **Atomic, set‑based locking to prevent double‑booking:**
  + **Compute end\_time from service duration; compute expected\_minutes.**
  + **For each staff\_profile\_id, perform a single UPDATE over a generated minute span for that date/time to set available = FALSE and count affected rows.**
  + **If updated\_count < expected\_minutes and not overriding, RAISE “not fully available.”**
  + **If overriding, continue and track deficit for audit logging.**
* **Time series safety: use timestamp‑anchored generate\_series((\_booking\_date::timestamp+\_time\_slot), ...) and cast back to time for the join (some Pg versions reject generate\_series(time, ...)).**
* **Index: CREATE INDEX IF NOT EXISTS idx\_staff\_availability\_key ON public.staff\_availability (staff\_profile\_id, date, time\_slot);**

**3.4 Appointment/staff role**

* **appointment\_staff.role is NOT NULL → always insert with role = 'assigned' (simple, consistent default). Dual‑service still uses the single 'assigned' role (segmented roles can be added later).**

**3.5 Dual‑service behavior**

* **Keeps prior behavior: always allows override and logs admin override slots. (We can later add \_override\_conflicts parity if needed.)**

**3.6 Admin UI payload branching (Prompt 4)**

* **Helper: buildAdminClientFields(client) returns either { \_client\_user\_id } (claimed) or { \_client\_id } (admin‑created & unclaimed). If unclaimed but not admin‑created → block.**
* **Update client fetches to include id, user\_id, admin\_created.**
* **Apply branching in AdminBookingPage.tsx and AdminManualBooking.tsx payload builders for all admin RPC calls.**

**4) Invariants & guardrails (what must stay true)**

1. **No “provider” concept. Use staff\_profile\_id everywhere. UI param \_provider\_ids is just legacy naming, aliased internally.**
2. **Admin‑only \_client\_id path requires both: admin\_created = true and user\_id IS NULL.**
3. **Pet ownership: \_pet\_id must belong to the resolved client\_id.**
4. **Self‑service unaffected. Client RPCs keep resolving by auth.uid().**
5. **Atomic availability locks per staff using date‑anchored minute series updates.**
6. **appointment\_staff.role always set (currently 'assigned').**
7. **RLS/SECURITY: admins must have the necessary privileges; functions remain SECURITY INVOKER unless we identify policy gaps.**

**5) RLS & security checklist**

**Confirm admins can:**

* **SELECT/UPDATE public.staff\_availability**
* **INSERT public.appointments, public.appointment\_staff, public.appointment\_services, public.appointment\_addons, public.admin\_actions  
  If any gap, either:**
* **add explicit admin RLS policies, or**
* **convert affected RPCs to SECURITY DEFINER (with least‑privileged owner) and GRANT EXECUTE to admin role only. *(Not done yet.)***

**6) Performance notes**

* **Added index on staff\_availability(staff\_profile\_id, date, time\_slot) to speed the atomic updates.**
* **Minute granularity retained (matches current system). If we adopt 10‑minute slots, we can switch to 10‑minute series and reduce row pressure.**

**7) Test plan (staging)**

**A. Concurrency (no override):**

* **Two simultaneous bookings for the same staff\_profile\_id, same date/time window.**
* **Expect: one succeeds, one fails (updated\_count < expected\_minutes).**

**B. Claimed client path:**

* **Use \_client\_user\_id, one available staff → success; availability rows flip to available = FALSE.**

**C. Admin‑created, unclaimed path:**

* **Use \_client\_id where admin\_created = true and user\_id IS NULL → success; pet ownership enforced.**

**D. Unclaimed but NOT admin‑created:**

* **Attempt with \_client\_id → fail with explicit guardrail error.**

**E. Pet mismatch:**

* **Pet not owned by resolved client → fail with explicit error.**

**F. Dual‑service happy path:**

* **Two services, staff provided, override logs created (admin\_override\_slots), roles inserted ('assigned').**

***(Optional)* Add UI smoke tests to verify payload branching and error messages.**

**8) Known limitations / deferred work**

* **Dual‑service segmentation: Currently blocks the entire span for all selected staff and uses a single 'assigned' role. Future: primary/secondary segmentation (distinct staff/time windows and roles).**
* **Role validation: We infer required roles from service booleans but do not yet verify staff capabilities vs required roles. Future: enforce capability mapping.**
* **Cancellation path: Ensure cancellations/free‑ups restore staff\_availability.available = TRUE for the booked span. (Add triggers or service code if missing.)**
* **NOTICE noise: Consider gating RAISE NOTICE under a debug flag post‑QA.**

**9) Files & artifacts touched**

* **Migrations**
  + **add-clients-admin-markers.sql: add admin\_created, created\_by, FK, indexes.**
  + **Function replacements: create\_booking\_admin, create\_booking\_admin\_override, create\_admin\_booking\_with\_dual\_services (added \_client\_id, guardrails, atomic availability lock, role insertion, timestamp‑based generate\_series).**
  + **Index: idx\_staff\_availability\_key.**
* **Frontend**
  + **AdminClients.tsx: insert now sets admin\_created, created\_by; .select() returns these fields.**
  + **AdminBookingPage.tsx, AdminManualBooking.tsx: added buildAdminClientFields(); client query includes admin\_created; payloads use \_client\_id or \_client\_user\_id accordingly.**

**10) Glossary & canonical naming**

* **staff\_profile\_id: canonical staff identifier used in staff\_profiles, staff\_availability, appointment\_staff.**
* **\_provider\_ids (UI): legacy frontend param name; always pass staff profile IDs. Server aliases to v\_staff\_profile\_ids.**
* **admin‑created client: clients.admin\_created = true, created\_by = admin user id, user\_id IS NULL until claim.**
* **claimed client: clients.user\_id IS NOT NULL.**

**11) Decision log (DID / WHY)**

* **Add admin markers on clients — to distinguish admin‑created unclaimed accounts for safe admin booking.**
* **Allow \_client\_id in admin RPCs — enables admin booking without a user\_id while preserving structure.**
* **Guardrails on \_client\_id — prevent misuse; maintain data integrity.**
* **Per‑staff availability + atomic lock — eliminate double‑book race conditions; align with staff‑based scheduling.**
* **Always set appointment\_staff.role — satisfy NOT NULL, keep data consistent.**
* **UI payload branching — deterministic routing: claimed via \_client\_user\_id, admin‑created unclaimed via \_client\_id.**
* **Timestamp‑anchored generate\_series — portable across Postgres versions.**

**12) Ready‑to‑test checklist**

* **Timestamp‑anchored generate\_series in all three RPCs**
* **Index present on staff\_availability(staff\_profile\_id, date, time\_slot)**
* **Admin RLS permits required ops (or functions run under DEFINER where needed)**
* **AdminClients creates clients with admin\_created=true, created\_by, user\_id=NULL**
* **UI fetch returns admin\_created and branches payloads correctly**
* **Staging availability seeded for bookable minutes**

**13) Next steps (post‑merge)**

* **Add segmented dual‑service support (primary/secondary windows & roles).**
* **Enforce staff capability vs service requirements.**
* **Implement cancellation free‑up logic if not already present.**
* **Consider \_override\_conflicts parity in dual‑service RPC.**
* **Reduce NOTICE noise; add structured audit events if needed.**

**Canonical Snapshot:  
[LOG\_UPDATE]**

**date: 2025-08-21**

**by: system**

**area: Canonical Snapshot (Architecture & Rules)**

**change\_summary:**

**- Establish Vettale-LOG as the single source of truth for design and history.**

**- Set timezone to America/Sao\_Paulo (São Paulo, Brazil) for all dates/times.**

**- Confirm staff nomenclature: no “provider”; use staff profiles everywhere.**

**- Confirm admin booking guardrail: `\_client\_id` allowed only for admin-created & unclaimed clients.**

**- Confirm availability is atomic, per-staff, set-based UPDATE with row-count verification.**

**- Confirm `appointment\_staff.role` must be set (NOT NULL); using 'assigned' currently.**

**- Confirm UI payload branching: claimed → `\_client\_user\_id`, admin-created unclaimed → `\_client\_id`, unclaimed non-admin-created → blocked.**

**- Confirm dual-service (simple mode): allow override by default; all selected staff blocked for full duration.**

**- Confirm index: `staff\_availability(staff\_profile\_id, date, time\_slot)` exists/required.**

**rationale:**

**- Keep every new session aligned, reduce churn, and prevent regressions or legacy term bleed-through (e.g., “provider”).**

**- Make LOG the living contract; prompt stays lean and defers to LOG.**

**touch\_points:**

**- docs: Vettale-LOG (this entry)**

**- code/db (reference only): admin RPCs, staff availability, appointment\_staff inserts, UI payload branching**

**tests:**

**- Admin booking for admin-created+unclaimed client with `\_client\_id` succeeds; pet ownership enforced.**

**- Claimed client via `\_client\_user\_id` succeeds.**

**- Unclaimed but not admin-created → blocked with explicit error.**

**- Availability is locked per staff; concurrent double-book fails without override.**

**- appointment\_staff inserts include `role='assigned'`.**

**status: baseline recorded; outcome: pass**

**follow\_ups:**

**- If dual-service requires segmented staff/time windows, evolve roles (`primary`/`secondary`) + per-segment availability.**

**- Verify RLS grants for admin across `staff\_availability`, `appointments`, `appointment\_staff`, `appointment\_services`, `appointment\_addons`, `admin\_actions`; switch to SECURITY DEFINER for any gaps if needed.**

**[/LOG\_UPDATE]**

**[LOG\_UPDATE]  
date: 2025-08-21  
by: GPT-5 Thinking  
area: Calendars / Admin Booking UI / Availability / UI infra  
change\_summary:**

* **Hard-isolated calendars: created BookingCalendar (admin) and PetDobCalendar + PetDobPicker (pet); removed cross-coupling.**
* **Deprecated shared wrapper src/components/ui/calendar.tsx; removed default Caption and any module-level state; stopped barrel re-exports.**
* **Implemented arrow-only navigation for Admin; removed month/year dropdowns; blocked past navigation with fromMonth=today.**
* **Integrated staff availability into Admin calendar via SECURITY DEFINER RPC get\_staff\_availability\_summary(staff\_ids uuid[], start date, end date) → (date, has\_availability).**
* **Added month-scoped fetch on (staffProfileIds, visibleMonth); built enabledDates: Set<YYYY-MM-DD> for O(1) disabling.**
* **Final disabled predicate (Admin) = past date OR (selected staff AND !enabledDates.has(dateISO)) [Sunday rule retained where configured].**
* **Ensured staff selection is staff\_profiles.id[] only; stored as bookingData.staffProfileIds.**
* **Pet DOB kept past-only (popover with typing + dropdowns); future dates blocked; fully independent styling and logic.**
* **Moved all calendar styles to CSS modules: BookingCalendar.module.css, PetDobCalendar.module.css; removed global .rdp-\* leakage.**
* **Added temporary debug logs/tags during rollout; documented for later removal.  
  rationale:**
* **Prevent regressions from shared UI state; enforce LOG non-negotiables (per-staff availability, staff\_profile\_id), and make booking UX reflect true bookability before time selection.  
  touch\_points:**
* **code:**
  + **Added: src/components/calendars/admin/BookingCalendar.tsx, src/components/calendars/admin/BookingCalendar.module.css**
  + **Added: src/components/calendars/pet/PetDobCalendar.tsx, src/components/calendars/pet/PetDobPicker.tsx, src/components/calendars/pet/PetDobCalendar.module.css**
  + **Modified: src/pages/AdminBookingPage.tsx (calendar usage, availability fetch, disabled predicate, month state)**
  + **Modified: pet forms/pages to use PetDobPicker (AdminClients/AdminPets/PetForm/Profile as applicable)**
  + **Deprecated: src/components/ui/calendar.tsx and any barrels re-exporting it**
* **db: RPC get\_staff\_availability\_summary (SECURITY DEFINER, read-only); no schema changes  
  tests:**
* **Manual verification:**
  + **Admin: arrows work; past dates disabled; dates without overlap availability disabled; month change triggers fetch; staff change updates enabled dates.**
  + **Pet: future dates disabled; year/month dropdowns OK; admin changes do not affect pet UI.  
    status: staging outcome: pass  
    follow\_ups:**
* **Replace temporary debug logs/tags with analytics event(s) or remove in production.**
* **Add ESLint ban rule to prevent importing ui/calendar from pet/admin code.**
* **Prefetch availability for adjacent months to reduce arrow-click latency.**
* **Extend overlap logic to the time grid (minute-level intersection for selected services).**
* **Add lightweight E2E happy-path (admin: select staff → choose enabled day → pick slot → create appointment).**
* **Document guardrail: \_client\_id only for admin-created clients (admin\_created = true, user\_id IS NULL); else \_client\_user\_id.  
  [/LOG\_UPDATE]**

**[LOG\_UPDATE]**

**date: 2025-08-24**

**by: GPT-5 Thinking**

**area: Prompting Framework / Change Management**

**change\_summary:**

**- Added "SILO SCOPE" rule for all Cursor prompts: focus strictly on the current problem and code surface.**

**- Removed cross-domain guardrails from focused prompts (e.g., no staff/availability mentions in client-only tasks).**

**- Introduced STOP-ON-SCOPE-DRIFT: Cursor must pause and request approval before touching out-of-scope areas.**

**- Standardized LOG access: Cursor must read /Context/Vettale-LOG.md; it must NOT write/update the LOG.**

**- Reaffirmed minimal-diff principle: no refactors outside scope; small, testable changes only.**

**rationale:**

**- Reduce assistant confusion and accidental edits by keeping each PR tightly scoped.**

**- Ensure the LOG remains the single source of truth while staying human-controlled.**

**- Prevent ripple impacts across modules (e.g., staff/availability) during client-only changes.**

**touch\_points:**

**- docs: /Context/Vettale-LOG.md (process rules added)**

**- prompts: All future Cursor prompts should append a SILO SCOPE block**

**- code: no code changes in this update**

**tests:**

**- Process change only; verified by using the SILO SCOPE add-on in current “client claim invite” work.**

**status: staging outcome: pass**

**[/LOG\_UPDATE]**

**[LOG\_UPDATE]**

**date: 2025-08-24**

**by: GPT-5 Thinking**

**area: Auth / Clients / Claim Flow + Edge Function + Admin UI badges**

**change\_summary:**

**- Implemented invite→claim flow for admin-created clients (email invite + correct linking moment).**

**- Replaced INSERT-based linkage with confirmation-based linkage: link client ⇢ auth.user \*\*after\*\* `email\_confirmed\_at` is set.**

**- Added/used `claim\_invited\_at` and `claimed\_at` timestamps to drive UI states (not `user\_id`).**

**- Created Edge Function `send-client-invite` using Supabase Admin API; added CORS headers for browser calls.**

**- Wired Admin UI to call the function on create and via a “Send/Resend Invite” action; updated card badges.**

**- Standardized redirect target via `CLAIM\_REDIRECT` env (local vs prod).**

**- Finalized invite email template using Supabase magic link placeholder.**

**rationale:**

**- Supabase `inviteUserByEmail` creates an `auth.users` row immediately (unconfirmed). Our old INSERT trigger linked too early, showing “Conta vinculada” before the user confirmed. Moving linkage to the \*\*email-confirmed\*\* moment fixes status accuracy and prevents premature association.**

**- CORS headers unblock Admin UI → Edge Function calls without server-side proxies.**

**- Timestamps (`claim\_invited\_at`, `claimed\_at`) give clear, auditable state transitions.**

**touch\_points:**

**- code:**

**- `supabase/functions/send-client-invite/index.ts` (invite + CORS + `CLAIM\_REDIRECT`)**

**- Admin UI (clients list/card): invoke function on create and on “send invite”; badge logic keyed to timestamps**

**- Email templates (Supabase Auth → Invite User): CTA uses `{{ .ConfirmationURL }}` (or `{{ .ActionURL }}` per project setting)**

**- db:**

**- Dropped old trigger: `trg\_link\_client\_on\_auth\_signup` (INSERT on `auth.users`)**

**- New function: `public.link\_client\_when\_email\_confirmed()` (SECURITY DEFINER)**

**- New trigger: `trg\_link\_client\_on\_email\_confirmed` (AFTER UPDATE OF `email\_confirmed\_at` on `auth.users`)**

**- Columns (idempotent confirm): `public.clients.claim\_invited\_at timestamptz`, `public.clients.claimed\_at timestamptz`**

**- (Optional/no-op if present) `citext` for `public.clients.email`**

**tests:**

**- Create admin client (`admin\_created=true`, `user\_id=NULL`, email set) → Edge Function sends invite; `claim\_invited\_at` set; UI shows \*\*Convite Enviado\*\*.**

**- Supabase Auth shows user “Waiting for verification”; `public.clients.user\_id` remains NULL.**

**- Click invite → confirm email → trigger runs; `public.clients.user\_id` now set; `claimed\_at` stamped; UI flips to \*\*Conta Vinculada\*\*.**

**- Re-send invite on the same client returns success without duplicate side effects.**

**- Browser preflight no longer fails (CORS headers present).**

**- Redirect verified:**

**- local: `CLAIM\_REDIRECT=http://localhost:8080/claim`**

**- prod: `CLAIM\_REDIRECT=https://vettale.shop/claim`**

**status: staging outcome: pass**

**[/LOG\_UPDATE]**

**[LOG\_UPDATE]**

**date: 2025-08-24**

**by: GPT-5 Thinking**

**area: Prompting Framework / Cursor Memories & Rules**

**change\_summary:**

**- Installed evergreen Cursor Memories to reduce prompt boilerplate.**

**- Kept only high-signal rules: LOG-first, SILO scope discipline, Role isolation (Client vs Staff vs Admin), Admin booking guardrail, Naming (“staff\_profile\_id”, never “provider”), DB-first invariants, Secrets server-side, Timezone.**

**rationale:**

**- Make prompts concise while enforcing architectural guardrails and scope isolation.**

**touch\_points:**

**- docs/process: Cursor → Memories (8 short entries)**

**- repo prompting: future tasks use tiny “TASK + SILO” prompts; Cursor reads `/Context/Vettale-LOG.md` before coding.**

**tests:**

**- Used new memories in the current “client invite/claim” work; Cursor stayed within scope; diffs were minimal.**

**status: staging outcome: pass**

**follow\_ups:**

**- Add `/docs/prompting.md` with examples of SILO prompts (optional).**

**- Review memories quarterly; keep them short and role-separated.**

**[/LOG\_UPDATE]**

**Problem: Self-registration wasn't populating clients table properlyRoot Cause: Trigger function missing admin\_created and claimed\_at fieldsSolution: Updated handle\_unified\_registration() function + cleaned up existing dataResult: ✅ Client records now immediately populated when users click "Registrar"**

**🔧 Key Technical Details:**

* **Migration: fix\_client\_registration\_claimed\_at applied**
* **Database: Updated public.handle\_unified\_registration() trigger**
* **Data Cleanup: Fixed 5+ existing incomplete client records**
* **Admin Flows: ✅ Preserved (no conflicts)**
* **Timezone: ✅ America/Sao\_Paulo compliance maintained**

**📊 Verification:**

* **✅ Self-registration creates complete client records immediately**
* **✅ Admin-created clients still work via separate flow**
* **✅ All existing clients now have complete data**
* **✅ No security or business logic compromised**

**[/LOG\_UPDATE]**

**End of document.**