**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]**

**[LOG\_UPDATE]**

**date: 2025-08-24**

**by: Matheus (via Cursor)**

**area: Client Profile 2.0 — DB, RPCs, Micro-Wizard, UX polish**

**change\_summary:**

**- DB (clients): added nullable fields phone, is\_whatsapp, preferred\_channel, emergency\_contact\_name, emergency\_contact\_phone,**

**preferred\_staff\_profile\_id (FK), accessibility\_notes, general\_notes, marketing\_source\_code, marketing\_source\_other,**

**profile\_completion\_score, first\_visit\_setup\_at, last\_nudge\_dismissed\_at, birth\_date; index on preferred\_staff\_profile\_id.**

**- DB (consents): created public.client\_consents (append-only, LGPD), RLS (client read/insert own; no update/delete), helpful indexes.**

**- RPCs/SQL: f\_client\_profile\_progress(); client\_get\_profile\_progress(); client\_update\_profile() with**

**first-write-wins for marketing\_source\_\* + p\_birth\_date; client\_needs\_first\_visit\_setup(); client\_log\_consent();**

**client\_mark\_first\_visit\_setup(); client\_get\_consent\_snapshot().**

**- RLS: confirmed owner-only UPDATE on public.clients; consents policies enforced; functions SECURITY INVOKER.**

**- Wizard: 4-step micro-wizard (skippable) with Step 1 contact (+ required preferred\_channel & marketing\_source),**

**Step 2 required ToS/Privacy/Reminders (consents logged), Step 3 emergency contact, Step 4 preferences + finalize (stamps first\_visit\_setup\_at).**

**- Profile UI: left card read-only (email/type/registration + consent snapshot + read-only "Como nos conheceu");**

**right card editable sections ("Contato" and "Preferências"); BR phone mask; constants for channels & marketing;**

**progress bar driven by server; banner hides on edit and remains hidden ≥80%.**

**- UX polish: default to view mode; “Rascunho restaurado” only on unsaved edit restore; hide “Completude do Perfil” when 100%;**

**preferred\_channel persisted reliably; birth date added (optional, not counted in score).**

**rationale:**

**- Reduce signup friction; collect operationally critical data progressively; ensure LGPD-compliant consent logging;**

**keep admin-centric integrity (immutable attribution) and clean UX.**

**touch\_points:**

**- db: public.clients (columns, index), public.client\_consents (table/indexes/policies),**

**functions: f\_client\_profile\_progress, client\_get\_profile\_progress, client\_update\_profile, client\_needs\_first\_visit\_setup,**

**client\_log\_consent, client\_mark\_first\_visit\_setup, client\_get\_consent\_snapshot.**

**- ui: Profile.tsx (sections, progress, banner, consent snapshot, birth date, draft logic),**

**ClientMicroWizard.tsx (steps, masking, required consents, marketing source),**

**Navigation fix (remove render loop), shared constants for preferred channel & marketing.**

**tests:**

**- DB: verified columns exist; policies enforced; RPCs callable; consents append; profile score recomputes.**

**- UI/E2E: phone mask; wizard saves partials; Step 2 blocks until all 3 consents; “Como nos conheceu” set once & read-only;**

**preferred\_channel counted (no lingering “pending”); progress refreshes from server; completeness card hidden at 100%;**

**draft toast only on unsaved restore; console clean (no PGRST202/404).**

**status: staging outcome: pass**

**follow\_ups:**

**- Authorized Pickups CRUD (behind flag), quiet\_hours model/enforcement, admin report of latest consents,**

**birthday communications (human + pet), move constants to lookups when ready, analytics dashboards for marketing sources & wizard drop-offs.**

**[/LOG\_UPDATE]**

**[LOG\_UPDATE]**

**date: 2025-08-24**

**by: GPT-5 Thinking**

**area: Client Pets UI (PADS) / WhatsApp CTA / Typography**

**change\_summary:**

**- Prepared Cursor prompt to apply the Pagonia font family consistently across client UI (incl. profile dropdown and Pets pages).**

**- Prepared Cursor prompt to keep the branded background on the \*\*Add/Edit Pet\*\* screens (no gray fallback), via a reusable background wrapper.**

**- Defined required fields for Pet Create/Edit: \*\*Nome do Pet, Raça, Data de Nascimento, Porte\*\* with client-side validation (future birth dates blocked; submit disabled when invalid).**

**- Delivered WhatsApp CTA fix: bypass `wa.me` and use direct links (mobile deep link or `web.whatsapp.com`/`api.whatsapp.com`) with `encodeURIComponent` to preserve the prefilled message.**

**- Removed Bitly preview in proposed solution (no shorteners). User decision afterward: adopt \*\*rebrand.ly/VettaleWhats\*\* for now.**

**- Proposed first-party click tracking (GA4 event + Supabase `cta\_clicks` table + Edge Function); deferred by user for later.**

**rationale:**

**- Brand consistency (Pagonia) and cohesive visuals across Pets flows.**

**- Ensuring Pet core data exists for future booking/pricing logic.**

**- Prevent loss of the prefilled WhatsApp message caused by `wa.me` + intermediaries.**

**- User wants lightweight click tracking without building backend right now.**

**touch\_points:**

**- code (planned/updated): global CSS & tailwind font mapping; Pets list + Pet form component; small `BrandBackground` wrapper; WhatsApp helper (`utils/whatsapp.ts`) and CTA buttons.**

**- db: none (tracking table/Edge Function proposed but not implemented).**

**tests:**

**- Verify Pagonia renders on Meus Pets, Add/Edit Pet, and profile dropdown.**

**- Navigate Meus Pets → Add/Edit Pet: background matches landing (no gray).**

**- Attempt Pet save with any required field empty or birth date in future → blocked with inline PT-BR errors.**

**- Fill all required fields → save succeeds.**

**- WhatsApp CTA on desktop opens \*\*web.whatsapp.com\*\* with message prefilled; on mobile opens app or falls back to `api.whatsapp.com` with message prefilled.**

**status: staging outcome: pending QA (user chose rebrand.ly link; typography + PADS updates queued via Cursor prompts)**

**follow\_ups:**

**- If keeping rebrand.ly, set destination to \*\*`https://api.whatsapp.com/send?phone=5511996378518&text=<ENCODED\_MSG>`\*\* to preserve the message and avoid previews (some shorteners inject interstitials).**

**- When ready, implement the first-party click tracking (GA4 + Supabase `cta\_clicks` + Edge Function) to replace shorteners.**

**- After QA passes, update status to production and mark outcome pass.**

**[/LOG\_UPD**[LOG\_UPDATE]  
date: 2025-08-28  
by: GPT-5 Thinking  
area: UI Performance & Dev Experience (PADS/Profile/Global Logging)  
change\_summary:

* Dev “silent by default”: logger passa a ocultar logs por padrão; debug só por ?debug=1 ou localStorage.setItem('debug','1').
* Logger com redação de PII (emails, tokens, campos \*\_id, strings longas).
* Removida gambi de “console stubbing” no main.tsx; mantido drop: ['console','debugger'] no build.
* Limpeza de console.\* ruidosos: migrados para log.debug/error em PADS/Profile/Nav/Auth/AdminBooking.
* Regra ESLint no-console para evitar regressão.
* Documentado “como habilitar/limpar debug”.
* Preparado escopo de correção mobile (ainda não aplicado):
  + /services: prevenir overflow dos cards/CTAs; grid 1-col no mobile; min-w-0 + break-words.
  + /profile: banner “Complete seu perfil” empilhando ações no mobile, com min-h estável.  
    rationale:
* Reduzir travamentos nos PCs antigos da clínica (menos custo de render/layout por logs).
* Manter capacidade de diagnóstico sob demanda, com PII protegida.
* Deixar mapeado um ajuste simples de CLS/overflow em mobile sem tocar back-end.  
  touch\_points:
* code: src/utils/logger.ts, src/main.tsx, src/pages/Pets.tsx, src/pages/Profile.tsx, src/components/ClientMicroWizard.tsx, src/hooks/useAuth.tsx, src/components/Navigation.tsx, src/pages/AdminBookingPage.tsx, vite.config.ts, .eslintrc\*
* db: n/a  
  tests:
* Dev local: console limpo por padrão; debug liga/desliga por URL/localStorage; navegação em PADS/Profile sem erros.  
  status: staging outcome: pass  
  follow\_ups:
* Teste presencial nos PCs antigos da clínica (abrir /profile e /services, navegar/rolar; observar travas).
* Se OK, promover para produção.
* Opcional (quando for conveniente): aplicar o patch mobile descrito acima e medir CLS (alvo ≤0,10).  
  [/LOG\_UPDATE]

[LOG\_UPDATE]

date: 2025-08-30

by: GPT-5 Thinking

area: Admin Clients (Profile 2.0 alignment) + Modal stability

change\_summary:

- Updated Admin Create/Edit Client to support Client Profile 2.0: kept required {name, email, phone, location}; wired optional fields (is\_whatsapp, preferred\_channel, emergency contacts, preferred\_staff\_profile\_id, notes, marketing, birth\_date, address).

- Fixed staff lookup: removed non-existent columns (full\_name/role), now select {id, name, email, phone, location\_id}; load when modal opens and filter by location; added safe label + empty states.

- Resolved Radix Select crash: replaced `SelectItem value=""` placeholders with non-empty sentinels and DB null-mapping.

- Restored create flow: admin markers set on insert {admin\_created=true, created\_by, needs\_registration=true, user\_id=NULL}; create/edit now persists all fields.

- Corrected delete regression: removed boolean-vs-integer comparisons in admin delete path.

- Minor UX: WhatsApp toggle placed under phone; non-blocking toasts + minimal debug logging.

rationale:

- Admin client creation broke after Profile 2.0 schema changes; bring admin up to parity without breaking invite/claim and prevent UI hard crashes on older clinic machines.

touch\_points:

- code: `AdminClients.tsx` (form, modal, staff fetch, selects, null-mapping, error handling), staff management page (added location\_id support to create/edit/list to enable preferred staff), small fixes in delete helper/SQL, constants for select sentinels.

- db: no schema changes; validated columns in `public.clients`; ensured queries only target existing columns; cleaned delete logic.

tests:

- Local/Staging manual:

- Open “+ Novo Cliente” → modal stable (no gray screen/chunk error).

- Staff dropdown loads; filters by location; shows fallback text when empty.

- Create client with new email → row created with admin markers; fields persist.

- Edit client → changes persist.

- Delete test client → succeeds; no “boolean > integer” error.

status: staging outcome: pass

follow\_ups:

- Cosmetics pass later (labels, spacing, helper copy).

- Decide final required vs optional field set for admin create.

- Add lightweight e2e smoke for modal open, create, edit, delete.

[/LOG\_UPDATE]

[LOG\_UPDATE]

date: 2025-08-30

by: GPT-5 Thinking

area: Admin routing hardening + Availability UI + Pet form validation

change\_summary:

- Audited all Admin links; fixed broken targets and ensured every nav item resolves (no 404s). Added a generic `AdminComingSoon` fallback page.

- Replaced `/admin/staff-availability` page with a clean “Em Breve” component that preserves `AdminLayout` so the left menu never disappears.

- Fixed Radix Select crash by removing `SelectItem value=""` placeholders; replaced with non-empty sentinels and mapped to `null` on save.

- Corrected `staff\_profiles` fetch: removed non-existent columns (`full\_name`, `role`), aligned to {id, name, email, phone, location\_id}; added guarded fetch & empty states.

- Fixed JSX structure error in `AdminAvailabilityManager` so `AdminLayout` wraps content and page renders reliably.

- Made pet `birth\_date` \*\*required\*\* in create/edit flows; added validation + asterisk on labels.

rationale:

- Stabilize admin navigation and forms before deeper booking work; eliminate crashes from invalid Select values and schema drift.

touch\_points:

- code: `AdminLayout.tsx`, `App.tsx`, `AdminComingSoon.tsx` (new), `AdminStaffAvailability.tsx` (replaced with Coming Soon), `AdminAvailabilityManager.tsx` (JSX fix), `AdminPets.tsx` (DOB required), any Select components using placeholder items, staff fetch helper in admin screens.

- db: none (query column selection corrected to existing columns).

tests:

- Manual: All admin nav items open without 404; Staff Availability shows Coming Soon and keeps menu; creating/editing pets enforces DOB; no Select crash; console clean for staff fetch.

status: staging outcome: pass

follow\_ups:

- Cosmetic pass later (copy/spacing).

- Keep `AdminComingSoon` as the default fallback for any future, not-yet-implemented admin routes**.**

[/LOG\_UPDATE]

[LOG\_UPDATE]

date: 2025-08-31

by: GPT-5 Thinking

area: Admin Booking (Dual-Service sequencing, Availability, Calendar & Summary)

change\_summary:

Locked booking architecture to order-based mapping: service\_order=1 → primary, service\_order=2 → secondary across appointment\_services and appointment\_staff.

RPC now derives authoritative durations from services.default\_duration and applies sequential availability (primary from t, secondary from t + dur(primary)) on staff\_availability 10-min grid with set-based updates and row-count checks.

Implemented slot search RPC for Step 3 (sequential membership): returns only valid start times for the selected primary/secondary staff+services; optional validator RPC for override reasoning.

Unified UI state to explicit keys: primaryServiceId, secondaryServiceId|null, primaryStaffId, secondaryStaffId|null, selectedDateISO (YYYY-MM-DD), selectedTimeHHMM.

Fixed PGRST202 by consolidating a single payload builder; RPC always called with a complete payload.

Fixed calendar off-by-one: removed UTC conversions; all dates stored/sent in local America/Sao\_Paulo (YYYY-MM-DD). Added 2-column Step-3 layout (calendar left, slots right).

Summary (“Resumo”/success) shows staff per service by order (pre-submit from UI state; post-submit via join by service\_order).

rationale:

Aligns with LOG non-negotiables: availability is atomic & per staff; updates occur over the generated minute range and are verified. Mapping by order removes ambiguity and avoids role/provider naming drift. Durations/pricing are DB-sourced, not UI-computed, preserving integrity.

touch\_points:

code:

src/pages/AdminManualBooking.tsx (state model, validation, payload builder, Step-3 layout & calendar handling, summary mapping)

src/components/BookingSuccess.tsx (post-submit mapping by service\_order)

src/hooks/useAdminAvailability.ts (calls slot RPC; transforms chips)

src/utils/time.ts (toHHMMSS, local ISO helpers)

db:

RPC (updated): create\_admin\_booking\_with\_dual\_services(...) — sequential availability updates; row-count validation.

RPC (new): find\_dual\_service\_slots(\_date, \_primary\_staff\_id, \_primary\_service\_id, \_secondary\_staff\_id, \_secondary\_service\_id) — sequential slot finder.

RPC (new, optional): validate\_dual\_service\_slot(...) — conflict reason on grey/red selections.

tables used unchanged: appointments, appointment\_services(service\_order), appointment\_staff(service\_order, staff\_profile\_id), staff\_availability, services.

tests:

Dual service (Banho 60 + Tosa 30), Amanda→Rogério, start 11:00:

staff\_availability: Amanda 11:00–11:50 → FALSE; Rogério 12:00–12:20 → FALSE; surrounding slots remain TRUE.

Step-3: only valid starts green; others grey/red; override dialog shows validator reason.

Summary before submit: Serviço 1 shows Amanda, Serviço 2 shows Rogério; after submit, success view matches via DB join on service\_order.

Single service: only primary rows created/blocked; secondary omitted.

Calendar: click 2025-09-19 → \_booking\_date='2025-09-19' (no off-by-one); Step-3 layout renders calendar left / slots right on desktop.

status: staging outcome: pass

[LOG\_UPDATE]  
date: 2025-08-31  
by: GPT-5 Thinking  
area: Website Frontend / Home & Services Copy / Packages

change\_summary:

* Home → “Nossos Serviços Principais”: definidos os 3 cards como **Vacinação**, **Consulta Veterinária** (badge **Mais Agendado**) e **Banho & Tosa Higiênica** (texto ajustado).
* Home → “Banho & Tosa Especializada”: removida duplicidade; agora **Banho Ionizado**, **Primeira Tosa do Filhote** e **Pacote Spa Luxo** com badge **Mais Popular**; descrição do Spa Luxo revisada; removido “tratamento de patas”.
* /services → “Serviços Veterinários”: mantida grade expandida; aplicada variação de cores para reduzir áreas brancas; sem mudanças funcionais.
* /services → “Banho & Tosa” (serviços individuais): convertida para **8 cards** — **Banho Simples**, **Banho Ionizado**, **Tosa Completa** (badge **Mais Popular**), **Tosa na Tesoura**, **Tosa Higiênica**, **Corte de Unhas**, **Limpeza de Dentes**, **Primeira Tosa do Filhote**; removidos combos desta grade.
* /services → **Pacotes**: padronizados **Essencial**, **Clássico**, **Spa Luxo** (badge **Mais Popular**) e **Master Ozônio**; títulos **sem parênteses**; composição movida às descrições; duração padrão **3 meses**; alturas equalizadas dos cards; paleta consistente.
* Micro-animações melhoradas (hover/entrada) para dar “pop” nos cards; apenas CSS/estilo.
* Nenhuma alteração em DB/APIs/rotas; atualizações restritas a textos, badges, ordem e classes visuais.

rationale:

* Evitar repetição entre sessões da Home e destacar diferenciais (**Banho Ionizado**, **Primeira Tosa do Filhote**, **Pacotes**).
* Alinhar linguagem ao Manual de Tom de Voz: **segunda pessoa**, frases curtas, **voz ativa**, foco no presente e sem promessas irreais.
* Manter consistência visual e legibilidade em mobile (grids com variação de cor e prevenção de overflow conforme diretriz já mapeada no LOG).

touch\_points:

* code: componentes/trechos de conteúdo dos cards da Home (serviços principais e “Banho & Tosa Especializada”); página **/services** (seções “Serviços Veterinários”, “Banho & Tosa” e “Pacotes”); ajustes CSS utilitários para alturas iguais e micro-animações.
* db: none.

tests:

* Verificação visual: textos/badges corretos; **Spa Luxo** marcado como **Mais Popular**; **Consulta** como **Mais Agendado** na Home; remoção de duplicidades.
* Mobile: grid 1-col sem overflow; min-w-0 + break-words quando necessário.
* Acessibilidade/UX: CTA “Agendar Agora” preservado; foco/hover nos cards funcionando; LCP estável.
* Smoke: navegação Home → Services sem erros.

status: production outcome: pass

follow\_ups:

* Centralizar a cópia em um JSON/MDX único para evitar divergências entre seções.
* Confirmar comercialmente a duração “3 meses” dos pacotes antes de integrar pagamento/DB.
* Instrumentar cliques de CTA (GA4) para medir desempenho de ordem/badges.
* Avaliar nomes alternativos de pacotes (ex.: **Essencial Cuidado**, **Clássico Estilo**, **Luxo Brilho**, **Ozônio Premier**) em A/B.  
  [/LOG\_UPDATE]

Staff invite log:  
# Staff Invite & Claim Flow - Complete Build Log

\*\*Date:\*\* October 8, 2025

\*\*Status:\*\* ✅ COMPLETED & WORKING

\*\*Project:\*\* Vettale Pet Grooming System

---

## 🎯 \*\*OVERVIEW\*\*

This document provides a comprehensive log of the \*\*Staff Invite & Claim Flow\*\* implementation, including all changes, fixes, and the final working mechanism. This flow allows administrators to create staff profiles and send invite emails, enabling staff members to claim their accounts by setting passwords.

---

## 🏗️ \*\*SYSTEM ARCHITECTURE\*\*

### \*\*Flow Overview:\*\*

1. \*\*Admin creates staff profile\*\* → Database record created

2. \*\*Admin clicks "Enviar Setup"\*\* → Edge Function sends invite email

3. \*\*Staff receives email\*\* → Clicks claim link

4. \*\*Staff sets password\*\* → Account claimed and linked

5. \*\*Staff can access profile\*\* → Full system access

### \*\*Key Components:\*\*

- \*\*Frontend:\*\* React/TypeScript with Supabase client

- \*\*Backend:\*\* Supabase Edge Functions with service role

- \*\*Database:\*\* PostgreSQL with triggers and citext extension

- \*\*Auth:\*\* Supabase Auth with custom email templates

- \*\*Email:\*\* Supabase native email system

---

## 📁 \*\*FILES MODIFIED\*\*

### \*\*1. Edge Functions\*\*

- `supabase/functions/send-staff-invite/index.ts` (Version 49 - Final)

### \*\*2. Frontend Components\*\*

- `src/pages/AdminSettings.tsx` (Staff management UI)

- `src/pages/StaffClaim.tsx` (Password setup page)

### \*\*3. Database Migrations\*\*

- `fix\_staff\_email\_confirmation\_trigger`

- `convert\_staff\_email\_to\_citext`

- `update\_staff\_trigger\_for\_citext`

### \*\*4. Configuration\*\*

- Supabase Auth URL Configuration

- Email Templates (Reset Password)

---

## 🔧 \*\*DETAILED IMPLEMENTATION\*\*

### \*\*1. Edge Function: `send-staff-invite`\*\*

\*\*Purpose:\*\* Sends staff invite emails using Supabase's native auth system

\*\*Key Features:\*\*

- ✅ Service role authentication

- ✅ CORS-compliant headers

- ✅ Email normalization (lowercase, trimmed)

- ✅ Existing user cleanup

- ✅ Production URL handling

- ✅ Error handling with detailed logging

\*\*Final Implementation (Version 49):\*\*

```typescript

import { serve } from "https://deno.land/std@0.224.0/http/server.ts";

import { createClient } from "https://esm.sh/@supabase/supabase-js@2";

const corsHeaders = {

"Access-Control-Allow-Origin": "\*",

"Access-Control-Allow-Headers": "authorization, x-client-info, apikey, content-type",

"Access-Control-Allow-Methods": "POST, OPTIONS",

};

serve(async (req) => {

if (req.method === "OPTIONS") {

return new Response(null, { status: 204, headers: corsHeaders });

}

try {

const admin = createClient(

Deno.env.get("SUPABASE\_URL")!,

Deno.env.get("SUPABASE\_SERVICE\_ROLE\_KEY")!

);

const { email, staff\_profile\_id } = await req.json();

console.log(`[STAFF\_INVITE\_PROD] Processing ${email} with ID ${staff\_profile\_id}`);

if (!email || !staff\_profile\_id) {

return new Response(JSON.stringify({

error: "Email and staff\_profile\_id are required"

}), {

status: 400,

headers: { ...corsHeaders, "content-type": "application/json" },

});

}

const normalizedEmail = email.trim().toLowerCase();

console.log(`[STAFF\_INVITE\_PROD] Normalized email: ${normalizedEmail}`);

// Verify staff profile exists

const { data: staff, error: staffErr } = await admin

.from("staff\_profiles")

.select("id, email")

.eq("id", staff\_profile\_id)

.single();

if (staffErr || !staff) {

console.error(`[STAFF\_INVITE\_PROD] Staff not found:`, staffErr);

return new Response(JSON.stringify({

error: "Staff profile not found"

}), {

status: 404,

headers: { ...corsHeaders, "content-type": "application/json" },

});

}

console.log(`[STAFF\_INVITE\_PROD] Staff found, cleaning up any existing auth users...`);

// Clean up: Delete any existing auth users with this email

try {

const { data: existingUsers } = await admin.auth.admin.listUsers();

const userToDelete = existingUsers?.users?.find(u =>

u.email?.toLowerCase() === normalizedEmail

);

if (userToDelete) {

console.log(`[STAFF\_INVITE\_PROD] Deleting existing user: ${userToDelete.id}`);

await admin.auth.admin.deleteUser(userToDelete.id);

console.log(`[STAFF\_INVITE\_PROD] Existing user deleted`);

}

} catch (e) {

console.log(`[STAFF\_INVITE\_PROD] No existing user to delete`);

}

// Use production URL for staff claim

const redirectUrl = "https://vettale.shop/staff/claim";

console.log(`[STAFF\_INVITE\_PROD] Using redirect URL: ${redirectUrl}`);

// Use Supabase's native invite system

console.log(`[STAFF\_INVITE\_PROD] Sending native Supabase invite...`);

const { data: inviteData, error: inviteErr } = await admin.auth.admin.inviteUserByEmail(

normalizedEmail,

{

redirectTo: redirectUrl,

data: { staff\_profile\_id, type: 'staff\_setup' }

}

);

if (inviteErr) {

console.error(`[STAFF\_INVITE\_PROD] Native invite failed:`, inviteErr);

return new Response(JSON.stringify({

error: `Failed to send invite: ${inviteErr.message}`

}), {

status: 400,

headers: { ...corsHeaders, "content-type": "application/json" },

});

}

console.log(`[STAFF\_INVITE\_PROD] Native invite sent successfully:`, inviteData?.user?.id);

// Update staff profile

const { error: updateErr } = await admin

.from("staff\_profiles")

.update({

claim\_invited\_at: new Date().toISOString()

})

.eq("id", staff\_profile\_id);

if (updateErr) {

console.error(`[STAFF\_INVITE\_PROD] Update failed:`, updateErr);

} else {

console.log(`[STAFF\_INVITE\_PROD] Staff profile updated`);

}

return new Response(JSON.stringify({

status: "success",

message: "Invite sent successfully",

user\_id: inviteData?.user?.id,

redirect\_url: redirectUrl

}), {

status: 200,

headers: { ...corsHeaders, "content-type": "application/json" },

});

} catch (e) {

console.error(`[STAFF\_INVITE\_PROD] Unexpected error:`, e);

return new Response(JSON.stringify({

error: `Unexpected error: ${e?.message || 'Unknown error'}`

}), {

status: 500,

headers: { ...corsHeaders, "content-type": "application/json" },

});

}

});

```

### \*\*2. Frontend: AdminSettings.tsx\*\*

\*\*Purpose:\*\* Admin interface for creating staff and sending invites

\*\*Key Changes:\*\*

- ✅ Automatic invite sending after staff creation

- ✅ Email normalization (lowercase)

- ✅ Case-insensitive duplicate checking

- ✅ Direct fetch() calls for better error handling

- ✅ Detailed error logging and user feedback

\*\*Critical Functions:\*\*

```typescript

const sendStaffInvite = async (staffProfile: StaffProfile) => {

const toastId = `send-setup-${staffProfile.id}`;

try {

setResendingSetupFor(staffProfile.id);

// Use fetch directly to get better error details

const response = await fetch(`https://ieotixprkfglummoobkb.supabase.co/functions/v1/send-staff-invite`, {

method: 'POST',

headers: {

'Authorization': `Bearer ${supabaseKey}`,

'Content-Type': 'application/json',

},

body: JSON.stringify({

email: staffProfile.email.trim().toLowerCase(),

staff\_profile\_id: staffProfile.id,

}),

});

const responseData = await response.json();

if (!response.ok) {

throw new Error(responseData?.error || `HTTP ${response.status}: ${response.statusText}`);

}

toast.success("Setup enviado 🎉", { id: toastId });

fetchStaff(); // Refresh to show updated status

} catch (e: any) {

console.error("[SEND\_STAFF\_SETUP] error", e);

toast.error(e?.message ?? "Falha ao enviar setup", { id: toastId });

} finally {

setResendingSetupFor(null);

}

};

const handleCreateStaff = async () => {

// ... staff creation logic with email normalization ...

// Send invitation email automatically

try {

const response = await fetch(`${supabaseUrl}/functions/v1/send-staff-invite`, {

method: 'POST',

headers: {

'Authorization': `Bearer ${supabaseKey}`,

'Content-Type': 'application/json',

},

body: JSON.stringify({

email: staffData.email.trim().toLowerCase(),

staff\_profile\_id: staffData.id,

}),

});

const responseData = await response.json();

if (!response.ok || responseData?.error) {

toast.success('Staff criado com sucesso!', {

description: `Falha ao enviar convite: ${responseData?.error || 'Erro desconhecido'}. Use o botão "Enviar Setup" para reenviar.`,

});

} else {

toast.success('Staff criado com sucesso! 🎉', {

description: `Convite enviado para ${staffData.email}`,

});

}

} catch (inviteErr) {

toast.success('Staff criado com sucesso!', {

description: 'Falha ao enviar convite automaticamente. Use o botão "Enviar Setup" para reenviar.',

});

}

};

```

### \*\*3. Frontend: StaffClaim.tsx\*\*

\*\*Purpose:\*\* Staff password setup and account claiming

\*\*Key Features:\*\*

- ✅ Robust session processing from URL

- ✅ Multiple fallback methods for session retrieval

- ✅ Detailed error messages

- ✅ Password strength validation

\*\*Session Processing Logic:\*\*

```typescript

const processSession = async () => {

try {

setIsProcessing(true);

console.log('[STAFF\_CLAIM] Processing session from URL...');

const hashParams = new URLSearchParams(window.location.hash.substring(1));

const searchParams = new URLSearchParams(window.location.search);

console.log('[STAFF\_CLAIM] Hash params:', Object.fromEntries(hashParams));

console.log('[STAFF\_CLAIM] Search params:', Object.fromEntries(searchParams));

// Method 1: Try setSession with tokens from hash

const accessToken = hashParams.get('access\_token');

const refreshToken = hashParams.get('refresh\_token');

if (accessToken && refreshToken) {

console.log('[STAFF\_CLAIM] Found tokens in hash, attempting setSession...');

const { data: sessionData, error: sessionError } = await supabase.auth.setSession({

access\_token: accessToken,

refresh\_token: refreshToken,

});

if (!sessionError && sessionData?.session) {

console.log('[STAFF\_CLAIM] Session set successfully via setSession');

setSession(sessionData.session);

setIsProcessing(false);

return;

}

}

// Method 2: Try getSessionFromUrl

console.log('[STAFF\_CLAIM] Trying getSessionFromUrl...');

const { data: urlSessionData, error: urlError } = await supabase.auth.getSessionFromUrl();

if (!urlError && urlSessionData?.session) {

console.log('[STAFF\_CLAIM] Session retrieved via getSessionFromUrl');

setSession(urlSessionData.session);

setIsProcessing(false);

return;

}

// Method 3: Check existing session

console.log('[STAFF\_CLAIM] Checking existing session...');

const { data: existingSessionData } = await supabase.auth.getSession();

if (existingSessionData?.session) {

console.log('[STAFF\_CLAIM] Found existing session');

setSession(existingSessionData.session);

setIsProcessing(false);

return;

}

console.log('[STAFF\_CLAIM] No valid session found, but allowing password setup attempt');

setIsProcessing(false);

} catch (error) {

console.error('[STAFF\_CLAIM] Session processing error:', error);

setIsProcessing(false);

}

};

```

### \*\*4. Database Migrations\*\*

\*\*Migration 1: `fix\_staff\_email\_confirmation\_trigger`\*\*

```sql

-- Fix the trigger function to properly handle email comparison without citext casting issues

CREATE OR REPLACE FUNCTION link\_staff\_when\_email\_confirmed()

RETURNS TRIGGER AS $$

DECLARE

v\_email text;

v\_user\_id uuid;

BEGIN

-- act only when confirmation is set

IF NEW.email\_confirmed\_at IS NULL THEN

RETURN NEW;

END IF;

v\_email := LOWER(TRIM(NEW.email));

v\_user\_id := NEW.id;

-- Link staff profile(s) with matching email that are not yet linked

-- Use LOWER() for case-insensitive comparison instead of citext casting

UPDATE public.staff\_profiles

SET user\_id = v\_user\_id,

claimed\_at = now()

WHERE LOWER(TRIM(email)) = v\_email

AND (user\_id IS NULL OR user\_id <> v\_user\_id);

RETURN NEW;

END

$$ LANGUAGE plpgsql SECURITY DEFINER;

```

\*\*Migration 2: `convert\_staff\_email\_to\_citext`\*\*

```sql

-- Convert staff\_profiles.email to citext for proper case-insensitive handling

ALTER TABLE public.staff\_profiles

ALTER COLUMN email TYPE public.citext;

```

\*\*Migration 3: `update\_staff\_trigger\_for\_citext`\*\*

```sql

-- Update the trigger function to use citext properly now that the column is citext

CREATE OR REPLACE FUNCTION link\_staff\_when\_email\_confirmed()

RETURNS TRIGGER AS $$

DECLARE

v\_email public.citext;

v\_user\_id uuid;

BEGIN

-- act only when confirmation is set

IF NEW.email\_confirmed\_at IS NULL THEN

RETURN NEW;

END IF;

v\_email := NEW.email::public.citext;

v\_user\_id := NEW.id;

-- Link staff profile(s) with matching email that are not yet linked

UPDATE public.staff\_profiles

SET user\_id = v\_user\_id,

claimed\_at = now()

WHERE email = v\_email

AND (user\_id IS NULL OR user\_id <> v\_user\_id);

RETURN NEW;

END

$$ LANGUAGE plpgsql SECURITY DEFINER;

```

---

## 🐛 \*\*CRITICAL ISSUES RESOLVED\*\*

### \*\*Issue 1: "Error confirming user" Message\*\*

\*\*Root Cause:\*\* Database trigger function `link\_staff\_when\_email\_confirmed()` was failing due to `citext` type casting issues.

\*\*Symptoms:\*\*

- ❌ Email confirmation failed with database error

- ❌ Users saw "Error confirming user" message

- ❌ Staff profiles weren't linked to auth users

\*\*Solution:\*\*

1. Fixed trigger function to handle `citext` properly

2. Converted `staff\_profiles.email` column to `citext` type

3. Updated function to use `public.citext` schema reference

### \*\*Issue 2: Case Sensitivity in Email Handling\*\*

\*\*Root Cause:\*\* Email comparisons were case-sensitive, causing mismatches.

\*\*Solution:\*\*

1. Normalized all emails to lowercase in Edge Function

2. Used `citext` type for case-insensitive database storage

3. Updated UI to use case-insensitive queries (`ilike`)

### \*\*Issue 3: Edge Function CORS and Error Handling\*\*

\*\*Root Cause:\*\* Poor error handling and CORS issues prevented proper debugging.

\*\*Solution:\*\*

1. Implemented comprehensive CORS headers

2. Added detailed logging throughout the flow

3. Used direct `fetch()` calls for better error visibility

4. Added proper error responses with status codes

---

## 🔧 \*\*CONFIGURATION REQUIREMENTS\*\*

### \*\*Supabase Dashboard Settings\*\*

1. \*\*Authentication → URL Configuration:\*\*

- \*\*Site URL:\*\* `https://vettale.shop`

- \*\*Additional Redirect URLs:\*\* `https://vettale.shop/staff/claim`

2. \*\*Email Templates:\*\*

- \*\*Template Used:\*\* "Reset Password"

- \*\*Custom HTML:\*\* Provided for branded email experience

3. \*\*Extensions:\*\*

- \*\*Required:\*\* `citext` extension (already installed)

### \*\*Environment Variables\*\*

- `SUPABASE\_URL`: Project URL

- `SUPABASE\_SERVICE\_ROLE\_KEY`: Service role key for Edge Functions

- `VITE\_SUPABASE\_URL`: Client-side project URL

- `VITE\_SUPABASE\_ANON\_KEY`: Client-side anonymous key

---

## 📊 \*\*TESTING CHECKLIST\*\*

### \*\*Complete Flow Test:\*\*

- ✅ Admin creates staff profile

- ✅ Staff profile appears in admin list

- ✅ "Enviar Setup" button sends email

- ✅ Email received with correct link

- ✅ Link redirects to `/staff/claim` page

- ✅ Password setup form works

- ✅ Account claimed successfully

- ✅ Staff can access their profile

- ✅ No "Error confirming user" message

### \*\*Edge Cases Tested:\*\*

- ✅ Duplicate email handling

- ✅ Case-insensitive email matching

- ✅ Existing auth user cleanup

- ✅ Network error handling

- ✅ Invalid staff profile IDs

- ✅ Malformed email addresses

---

## 🚀 \*\*DEPLOYMENT NOTES\*\*

### \*\*Production Deployment:\*\*

1. \*\*Edge Function:\*\* Deployed as version 49 to Supabase

2. \*\*Database Migrations:\*\* Applied successfully

3. \*\*Frontend:\*\* Deployed to Vercel with latest changes

4. \*\*Configuration:\*\* Updated in Supabase Dashboard

### \*\*Monitoring:\*\*

- \*\*Logs:\*\* Available in Supabase Dashboard → Edge Functions

- \*\*Auth Logs:\*\* Available in Supabase Dashboard → Authentication → Logs

- \*\*Database Logs:\*\* Available in Supabase Dashboard → Database → Logs

---

## 📚 \*\*TECHNICAL DEPENDENCIES\*\*

### \*\*Frontend:\*\*

- React 18+

- TypeScript

- Supabase JS Client v2

- React Router DOM

- Sonner (for toasts)

### \*\*Backend:\*\*

- Supabase Edge Functions (Deno runtime)

- PostgreSQL with citext extension

- Supabase Auth system

### \*\*Email System:\*\*

- Supabase native email service

- Custom "Reset Password" template

- SMTP configuration handled by Supabase

---

## 🔄 \*\*MAINTENANCE GUIDELINES\*\*

### \*\*Regular Checks:\*\*

1. \*\*Monitor auth logs\*\* for any new citext-related errors

2. \*\*Test email delivery\*\* periodically

3. \*\*Verify redirect URLs\*\* remain correct

4. \*\*Check Edge Function logs\*\* for performance issues

### \*\*Future Enhancements:\*\*

1. \*\*Email template customization\*\* for better branding

2. \*\*Bulk staff invite\*\* functionality

3. \*\*Staff role management\*\* integration

4. \*\*Email delivery status\*\* tracking

---

## 📝 \*\*LESSONS LEARNED\*\*

### \*\*Key Insights:\*\*

1. \*\*Database triggers\*\* require careful type handling, especially with extensions

2. \*\*Supabase Auth logs\*\* are invaluable for debugging auth flows

3. \*\*citext extension\*\* must be properly referenced with schema (`public.citext`)

4. \*\*Direct fetch() calls\*\* provide better error visibility than Supabase client functions

5. \*\*Email normalization\*\* is critical for reliable matching

### \*\*Best Practices Established:\*\*

1. Always use `citext` for email columns

2. Normalize emails to lowercase in all operations

3. Implement comprehensive error handling in Edge Functions

4. Use detailed logging for debugging complex flows

5. Test database triggers thoroughly before deployment

---

## ✅ \*\*FINAL STATUS\*\*

\*\*✅ FULLY FUNCTIONAL AND DEPLOYED\*\*

The Staff Invite & Claim Flow is now completely working in production. Staff members can successfully:

1. Receive invite emails from administrators

2. Click email links to access the claim page

3. Set their passwords to claim accounts

4. Access their staff profiles and system features

\*\*No more "Error confirming user" messages!\*\* 🎉

---

\*\*End of Build Log\*\*

\*\*Total Development Time:\*\* ~8 hours of debugging and implementation

\*\*Final Result:\*\* Production-ready staff onboarding system

-

End of document.