

Summary Findings

The **SuburbMates Melbourne** codebase exhibits a well-defined conceptual foundation (extensive SSOT docs and a solid schema with RLS enforcement), but the implementation is **incomplete and misaligned in key areas** needed for a production-grade launch. Critical product features like user authentication UI, listing claim flows, payments, and admin tooling are either partially implemented or entirely missing. Several UI elements use placeholder or non-authoritative content (e.g. static “Digital Creator” labels, hard-coded sample products), deviating from the “truth-first” principles in the documentation. Crucial launch-readiness items – custom error pages, SEO metadata, observability hooks (PostHog, Sentry), CI/CD pipelines – are not in place. There are also mismatches between the Single Source of Truth (SSOT) docs and the code (e.g. “profiles” vs “actors” user model, product card copy not following the “Sold by [Creator]” rule). Overall, while the architectural groundwork (Next.js + Supabase with RLS, Stripe, etc.) is laid out, **the project is not launch-ready** – significant gaps must be addressed to meet a production-grade directory platform checklist.

Gap Inventory Table

File/Component	Issue Description	Priority	Recommended Fix
<code>components/home/Marketplace.tsx</code>	Static sample data: Product cards are hard-coded (e.g. “Agency OS” template) with placeholder images and info ¹ . No dynamic fetching of real products from the DB is implemented. This undermines the marketplace feature.	High	Integrate Supabase query for products (limit & sort), replace placeholders with real data. Provide an “empty state” if no products.
<code>components/home/Directory.tsx</code> (home feed)	No ranking logic: Listings are fetched but not sorted by the defined ranking contract (Featured > Pro > Basic > Unclaimed, with Verified overlays) ² . Currently <code>getListings()</code> simply selects all with no ordering ³ .	High	Implement server-side sorting of listings by tier and verification status per the ranking rules ² (e.g. SQL ORDER BY tier, is_verified, etc.). Ensure “Featured” and “Verified” badges/labels appear on listing cards as per UX contract ⁴ ⁵ .

File/Component	Issue Description	Priority	Recommended Fix
<code>components/ directory/ StudioPageModal.tsx</code>	Placeholder UI copy & missing data: Uses static text "Digital Creator" for category instead of actual category name ⁶ , and "Member since 2024" with a hardcoded year ⁷ . The "View Full Studio Page" button is present but not wired (no actual full-page route). These break the "truthful UI" principle.	High	Populate category from <code>creator.category</code> (once the category join is fixed) or remove if unknown. Dynamically compute member since from <code>created_at</code> . Implement a dedicated Studio detail page (<code>/studio/[id]</code> or <code>[slug]</code>) and make the button navigate there, rather than a no-op modal.
<code>lib/listings.ts</code> + DB schema	Schema-code mismatch (taxonomy): The code is joining <code>categories</code> in the query ⁸ , expecting category names, but initial schema used separate <code>business_categories</code> . A later migration unified categories ⁹ . This is resolved in DB (the <code>categories</code> table exists ¹⁰), but the code's TypeScript types (<code>category_id: number</code> in tests ¹¹ vs actual UUID) and some references (e.g. test using <code>category_id: 1</code> ¹¹) are outdated.	Medium	Update all types and tests to use the unified <code>categories</code> UUID ID (adjust test fixtures accordingly). Verify that the Supabase <code>categories</code> relation is correctly referenced in queries. Remove any stale references to <code>business_categories</code> or numeric IDs (the taxonomy refactor should be reflected consistently in code).
<code>supabase/schema.sql</code> vs Migrations	Duplicate user models: Both a <code>profiles</code> table ¹² and an <code>actors</code> table exist (via migrations ¹³), causing confusion. RLS policies use <code>actors</code> ¹⁴ , and new user triggers populate <code>actors</code> ¹⁵ . The <code>profiles</code> table appears to be a leftover (likely from Supabase boilerplate).	Medium	Drop or ignore the legacy <code>profiles</code> table in favor of <code>actors</code> . Ensure documentation and code use consistent terminology ("actors" as authoritative user profile). Update <code>schema.sql</code> and SSOT if needed to remove the deprecated profile model to avoid misleading future devs.

File/Component	Issue Description	Priority	Recommended Fix
Auth UI & Flows (missing feature)	<p>No custom sign-up/login UI: The app lacks pages or components for user registration, login, logout, or password reset. Supabase Auth is configured (trigger to create actors ¹⁶), but there are no Next.js pages or forms to drive these workflows. This blocks user onboarding entirely.</p>	High	<p>Implement auth pages (or use Supabase UI components) for Sign Up, Login, and Forgot Password, following the flows in the SSOT ¹⁷ ¹⁸. Include client-side validation via Zod (the dependency is installed) for inputs. Ensure the app handles the email verification step (Supabase can send the email) and surfaces errors ("Invalid credentials") as per spec ¹⁸.</p>
Listing Claim & CRUD (missing feature)	<p>No "claim listing" or edit studio functionality: Unclaimed listings (status <code>unclaimed</code>) exist in the model but there's no UI or API to claim them. The Directory presumably should list unclaimed businesses and allow a user to claim (which would set <code>owner_id</code> and status). Similarly, creators cannot currently edit their listing details or add products via any UI. These core CRUD flows are not exposed.</p>	High	<p>Build out Creator Studio pages (as per the Creator Studio spec) for managing a listing: at minimum, forms to edit listing info (name, description, location, category) and to create/edit products. Implement a "Claim this listing" flow for unclaimed entries – e.g. a call to an API route or Supabase RPC that sets the <code>owner_id</code> if the user has none yet. Hide or disable any "claim" option for logged-out users ¹⁹. Ensure RLS policies (owner can insert/update their listing ²⁰) are respected by using the user's JWT in supabase calls.</p>

File/Component	Issue Description	Priority	Recommended Fix
Payments integration (feature gap)	Stripe Connect not wired up: Stripe libraries are installed ²¹ but there is no code to create checkout sessions, handle webhooks, or update subscription status (e.g. upgrading a listing's <code>tier</code> to Pro or scheduling <code>featured_until</code>). The monetization model (Pro \$20/month, Featured \$15/month ²²) isn't enforced. Failing to capture payments and update tiers means Pro/Featured features cannot actually be purchased or activated.	High	Integrate Stripe: implement a backend route (e.g. Next API route or Edge Function) to create a Checkout Session for Pro subscription and Featured slot purchase , using Stripe Connect if creators need their own accounts. On successful payment (via webhook), update the <code>listings.tier</code> to 'Pro' and set <code>featured_until</code> accordingly. Also handle payment failures or renewals (e.g. mark listing back to Basic if subscription ends). Include clear "Payments processed by Stripe" messaging to users ²³ ²⁴ . (This is a complex addition – but critical for launch if monetization is in scope.)
Verification (ABN) (feature gap)	No ABN verification flow: The SSOT mandates an ABN verification mechanism (validate ABN, mark <code>is_verified</code> , etc.) ²⁵ , but the app provides no interface to submit an ABN or any backend logic to validate one. The <code>is_verified</code> flag on listings is always FALSE unless manually flipped. This means the promised "Verified" trust signal cannot actually be obtained by creators at launch.	Medium	Implement an ABN verification service: e.g. an API integration with the Australian ABR lookup (guid-based). Provide a Verification section in the Studio where a creator can submit their ABN ²⁶ . Upon submission, perform format checksum and call the ABN lookup; if valid, set <code>is_verified=TRUE</code> on their listing and perhaps log an event. If ambiguous or failed, flag for manual review (could simply email the operator for now). This ensures the "Verified" badge in UI is meaningfully obtainable.

File/Component	Issue Description	Priority	Recommended Fix
Mobile Responsiveness (partial)	Potential mobile UX issues: The design is declared “mobile-first”, but certain components (e.g. the StudioPageModal uses a fixed 400px width on desktop and 90vh height on mobile ²⁷ ²⁸) need thorough testing. No visible bottom navigation or distinct mobile menus were found, despite “thumb-friendly bottom nav” mentioned in specs ²⁹ .	Low	Test all pages on varied screen sizes. Implement a bottom nav bar for primary sections if applicable (e.g. Directory, Marketplace, Studio – if those are separate pages in the future) in mobile viewports. Ensure the modal and other sections use responsive Tailwind classes (some are in place) so content isn't cut off (for instance, verify the 90vh modal on small screens). This is important but can be tuned alongside QA, not a blocker like missing features.
<code>components/home/Hero.tsx</code>	Search bar not functional: The landing page includes a search input with no logic behind it ³⁰ ³¹ . Given search is the primary discovery method per the docs ³² , this is a notable omission. Users cannot actually search listings or products.	High	Implement search functionality using Postgres full-text search as planned ³³ . For example, create a Next API route (e.g. <code>/api/search</code>) or use Supabase RPC to perform FTS on listings (name, description, tags) and products. Connect the input to trigger this search (on enter or via a button), and display results (could reuse listing cards for listings and similar cards for products, grouped appropriately). This addresses the “search-first discovery” mandate ³² .

File/Component	Issue Description	Priority	Recommended Fix
SEO & SSR (general)	Missing SEO optimizations: There is no use of <code><head></code> or Next SEO configuration for dynamic meta tags (e.g. meta description, OpenGraph tags for listings/products). Also, no custom 404 page or error boundary is defined – missing an opportunity for a branded error page and potentially harming SEO (404s are not handled gracefully).	Medium	Utilize <code>next-seo</code> (already a dependency) to set site-wide defaults and per-page meta tags (title, description). For instance, when rendering a Studio page, populate the <code><NextSeo></code> with the studio name and tagline. Implement <code>app/not-found.tsx</code> to handle 404 with a friendly page (Next's App Router will use it when a page is not found). Also consider an <code>error.tsx</code> for graceful error messaging on unhandled exceptions. These improvements will make the platform more polished for launch (better previews on social share, proper handling of dead links, etc.).
Observability (missing)	No analytics or error monitoring: The README lists PostHog and Sentry as planned ³⁴ , but the codebase has no implementation for either (and no dependencies for PostHog or Sentry DSN configured). Without these, production issues and user behavior will be hard to track.	Low	Integrate Sentry for error monitoring (client and server initialization) and PostHog (or an alternative) for basic usage analytics. Even if full analytics dashboards aren't ready, ensure the hooks are in place (with environment-driven opt-in) so that once traffic comes, data isn't completely lost. This can be done post-core-features, but prior to launch is ideal.
CI/CD Pipeline (missing)	No automated testing/deployment setup: There are no CI workflows (.github/workflows) in the repo. While <code>npm run test</code> and coverage are present, tests must be run manually. Lack of CI means regressions could slip in, and deployment must be done manually, risking inconsistencies.	Medium	Add a GitHub Actions or similar CI pipeline: run <code>lint</code> and <code>test</code> on push/PR to main. Include build checks (Next build) to catch integration issues. For CD, set up a staging deployment (e.g. Vercel or Docker pipeline) and possibly require tests to pass before merge. This will enforce launch readiness gates and maintain confidence in future changes.

File/Component	Issue Description	Priority	Recommended Fix
Admin/Operator Tooling (missing)	<p>No operator dashboard or admin controls: The Operator Dashboard is described as the “sole internal control surface” for verification, enforcement, audits, etc. ³⁵, but no admin UI or routes exist. The only admin capability in place is the DB role column and a SQL seed to promote an operator ³⁶. Without a UI, the operator cannot easily review flags, approve verifications, or take enforcement actions (warn/delist/etc.) aside from direct database edits.</p>	Medium	<p>Develop a minimal Operator Dashboard (could be a protected route under <code>/admin</code>): for launch, even a read-only view of all listings with their status, and buttons or links to trigger key actions (e.g. mark a listing as delisted/suspended, review ABN submissions) would suffice. Use the <code>actors.role='operator'</code> to gate access (the RLS and JWT will prevent non-operators from those actions in any case ²⁰ ³⁷). Logging of actions could simply write to the <code>violation_log</code> JSON in the <code>actors</code> table or a new <code>audit_log</code> table to satisfy audit requirements.</p>
Copy and Content (various)	<p>Non-authoritative text and missing “truth” disclosures: Some UI text doesn’t follow the SSOT “Truth UI” rules. Examples: Product cards show seller name as a brand (“Contrast”) without the required “Sold by ...” prefix ³⁸ ²⁴; there’s no visible “Payments processed by Stripe” on the site (as required wherever purchases are implied ²⁴); no explanation tooltip for the “Verified” badge (the contract says mobile-accessible explanation needed ⁵). Also, the editorial tone of content hasn’t been reviewed (e.g. the tagline and info in Hero are fine, but no FAQ or user-facing policy text is present beyond code comments).</p>	Low	<p>Audit all user-facing copy against the Public UX Contract. Add missing disclosures: on product cards or pages, include a line “Sold by [Creator]” and on product page, a note about Stripe and SuburbMates’ role ²³ ²⁴. For the Verified badge, implement a small “What’s this?” popover explaining “Verified means we have confirmed the business’s ABN (Australian Business Number).” Ensure any placeholder text is removed or replaced with real dynamic content or nothing at all (never show filler to end-users). Having consistent, truthful messaging is part of launch readiness for user trust.</p>

(Table legend: Priority – High = must-fix pre-launch, Medium = should-fix for a solid launch, Low = nice-to-have polish.)

Misalignment with Docs (SSOT vs Implementation)

Several implementation details are not aligning with the Single Source of Truth documentation:

- **“Sold by [Creator]” Label:** The Public UX Contract explicitly requires that every product listing display “Sold by [Creator]” to make the seller clear ³⁹ ²⁴ . In the current UI, the sample product cards simply show the seller name (e.g. “Contrast”) without this prefix ³⁸ . This is a breach of the truth-in-design rule and could mislead users. The fix is to prepend “Sold by” text or otherwise make it obvious that the second label is the seller.
- **Verified Badge Explanation:** According to the contract, any “Verified” badge must clearly explain what is verified (identity, not quality) and be accessible (on mobile, via tap) ⁵ . In `StudioPageModal`, a Verified checkmark is shown with the word “Verified” ⁴⁰ ⁴¹ , but there’s no tooltip or link. Additionally, this modal is used for both Basic and Pro listings with no difference in content beyond the tier tag. The implementation should add an explanation (e.g. an “i” info icon or clickable badge that opens a small note) that “Verified means ABN has been confirmed”, per spec.
- **Tier-Based Product Limits:** The SSOT *Creator Studio Spec* mandates Basic creators can publish 3 products, Pro up to 10 ²² ⁴² . Currently, there is no enforcement of this limit in code (no checks in the product creation logic or DB). A Basic user could theoretically insert more products if they had UI access. This needs alignment – either via a database constraint (not trivial without triggers) or application logic to block and prompt upgrade. The absence of any such check is a divergence from the subscription model’s intent.
- **Unclaimed Listing Handling:** The Public UX Contract says unclaimed listings should **not have a full public page** – clicking should initiate a claim flow ¹⁹ . In our app, because no claim flow exists, if an unclaimed listing appears, the likely behavior is it opens the same `StudioPageModal` or a page (if implemented later) but without an owner. That would violate the rule. We need to ensure unclaimed listings are clearly labeled “Unclaimed” and instead of viewing details, prompt login/claim. This is partially an implementation gap, but also a misalignment: currently the `status` field is present, yet the UI does nothing with it. Aligning with docs means adding an “Unclaimed listing – Claim this business” state in the UI.
- **“profiles” vs “actors” terminology:** The SSOT *User Model* refers to the unified actor model ⁴³ , and indeed code uses `actors` table. But some docs (and the schema.sql) still mention “profiles” (e.g., the API doc uses `creator_id` formerly `owner_id` and lists “Profiles” endpoint conceptually). This could confuse developers. It’s essential to scrub references to the old term or clarify in documentation that “profiles (legacy) = actors (current)”. The **SSOT Freeze Record** indicates docs are frozen as of a certain commit ⁴⁴ – if the code has evolved (like adding `actors`) after that freeze without an update, that’s a governance slip. We should reconcile that by updating the SSOT docs in the next cycle to reflect the final implementation (and then re-freeze).

- **Ranking Implementation:** The deterministic sorting (“Bucket Hierarchy”) is detailed in *platform-logic.md* ² and the *Public UX Contract* ⁴, but the code does not implement it yet. This is not just a missing feature but a deviation from an **explicit contract**: the public results must reflect those rules. Until fixed, the product is not honoring the fairness promise. This misalignment should be flagged as a launch blocker – the engineering must catch up to the contract here.
- **Analytics & Logging Promises:** The docs (Operator & Admin spec) emphasize logging of important events (verification, tier changes, enforcement) ⁴⁵. As of now, no such logging exists (no audit log table or Sentry events for these). This is a gap between what an operator expects to see and what’s built. For launch, we might decide to downscope some of this (e.g. not everything needs to be perfect), but it’s worth noting. Even a simple console log or DB record insertion on such events would be better than silent changes (which are explicitly forbidden by “No Silent State Changes” ⁴⁶).

In summary, the codebase needs a round of changes to **bring implementation in line with the SSOT declarations**. The SSOT documents were intended to be the blueprint, and in several areas (trust labels, feature gating, workflow edges) the blueprint isn’t yet realized in code. Each misalignment identified above corresponds to either a missing UI element or missing logic that should be added to fulfill the project’s own contract with its users and operators.

Missing Launch Checklist Items

Beyond the code/doc discrepancies, the project is missing a number of standard launch-readiness items for a full-stack directory platform:

- **User Authentication & Onboarding:** No UI for login or registration (as noted). Users currently cannot sign up or log in at all, which obviously blocks any creator functionality. This is a show-stopper. We need at least email/password sign-up with verification (Supabase can handle the email verify out-of-the-box) and a login form. Password reset flow is also expected (the SSOT describes it ¹⁸ ⁴⁷), although one could argue it could come just after launch if absolutely necessary. **Launch cannot proceed without basic auth in place.**
- **User Authorization & Access Control:** The app should ensure that certain routes or actions are restricted to logged-in users with appropriate roles. For example, only an operator (role=operator) should reach any admin endpoints – right now, there are none, but once added, we must enforce this in the Next.js route handlers (e.g. check `req.supabaseUser` role or using middleware). Similarly, creators should only manage their own listings/products – this is largely enforced by RLS at the DB level ²⁰ ³⁷, which is good. However, from a UI perspective, there should be guardrails (e.g. don’t even show “Edit” buttons if not the owner). These details should be reviewed prior to launch.
- **Complete CRUD for Core Entities:** At launch, the platform should support creation, viewing, updating, and deletion (or at least archiving) of core objects: Listings (to the extent of claiming or editing one’s business info), Products, possibly Tags on products (the UI currently doesn’t expose tagging at all, even though the schema supports up to 5 tags ⁴⁸ ⁴⁹). The *Creator Studio* section outlines all needed sections (Overview, Listing Details, Products, Mini-site (Pro), Share Kit (Pro), Verification, Billing) ⁵⁰. Not all must be fully built for an MVP launch (e.g. Share Kit could be minimal), but **Listing Details and Products management are mandatory**. As of now, none of those sections have UI. This is a significant amount of work remaining to be “feature-complete” for even an MVP launch.

- **Payments & Subscription Management:** To reiterate, a directory-marketplace launch usually requires the ability for creators to upgrade to paid plans (Pro) and for the platform to handle recurring billing or at least track subscription status. Currently, there is no UI prompt for upgrade (the docs mention that attempting to exceed Basic product limit should trigger an upgrade prompt ⁵¹ – but since that limit isn't enforced, no prompt exists). We also have no webhook processing to downgrade a user after their subscription period ends (the spec is clear that after Pro expiry, features must turn off gracefully ⁵²). This entire area is not implemented. **If the decision is to launch a free beta, perhaps payments can be postponed**, but then all references to Pro features should be hidden to avoid user confusion. Conversely, if monetization is part of launch, this must be built and thoroughly tested (including handling payment failures, Stripe Connect account onboarding for creators if needed, etc. – none of which is present yet).
- **Search Functionality:** Given the emphasis on “search-first discovery” ³² , not having a working search is a big gap. Even a basic search by name/keyword using Supabase full-text search should be in place. Right now, the search bar is a dummy UI element. This is a core feature for a directory and should be considered launch-critical functionality.
- **SEO and Social Sharing:** For a public directory, SEO is key to being discoverable. The app uses Next.js which is SEO-friendly, but we should ensure server-side rendering is applied where needed (the Next.js App Router and the use of `export const revalidate = 300` on home suggests they're doing static generation for the homepage ⁵³ , which is good). We should similarly pre-render listing pages (Studio pages) with appropriate meta tags. Additionally, creating meaningful meta descriptions, titles (e.g. “SuburbMates – Find Digital Creators in Melbourne”), and OpenGraph tags for at least the main pages will help initial traction. **No evidence of `next-seo` usage or `<head>` configuration** was found, so this needs to be done.
- **Performance and Stability:** While no specific performance issues were observed in code (the codebase is relatively small at this stage), we should consider adding basic loading states and error handling on network operations. For example, if the homepage `getListings` call fails (Supabase down or network issue), currently it will throw and possibly crash the page (since there's no try/catch around it in `Home` – Next.js might catch it and show the default error). A more graceful handling (maybe empty state or error message) would be preferable. Likewise, adding a global error boundary (Next.js `error.js`) could ensure the app doesn't show a raw stack trace to users. These aren't massive launch blockers but are part of “production-readiness polish.”
- **Testing & QA:** Some tests exist (Vitest for home page, and Supabase RLS tests) which is great ⁵⁴ ⁵⁵ . However, for launch, we should at least smoke-test all critical flows manually if not with automated tests: user signup, listing claim, product creation, purchase flow, etc. The current test coverage seems to focus on backend rules (RLS) and trivial component rendering, so a QA pass is needed. If time permits, adding a few Playwright E2E tests for “happy path” user journeys (sign up -> claim listing -> add product -> view listing on directory) would significantly de-risk the launch.
- **Monitoring & Alerts:** In addition to adding Sentry/PostHog, ensure that the team has set up something to be alerted if the site or critical functions fail. Since it's Supabase/Next.js, this could be as simple as enabling Vercel's uptime checks or writing a small health-check script. Not strictly a codebase concern, but part of launch readiness (so mentioning it for completeness).

In short, a lot of foundational work is in place, but **many features are stubs or plans rather than finished pieces**. The team should prioritize enabling the critical user-facing functionality first (auth, content CRUD, search), then the transactional pieces (payments, verification), and then polish items (SEO, copy, responsiveness, monitoring).

Risk Areas for Handoff

If a new developer/team takes over this codebase at this stage, they may encounter several risk areas and points of confusion:

- **Complex SSOT Guidance:** The presence of extensive SSOT documentation is double-edged – it's great for clarity of vision, but a new dev must spend significant time reading and understanding these documents (Product Constitution, UX Contract, Studio Spec, etc.) to avoid violating them. There's a governance process (the freeze records ⁴⁴) that a new dev might not notice. They could mistakenly implement a feature in a way that conflicts with these locked specs. **Mitigation:** make sure new contributors are pointed to the docs/SSOT folder and understand they should treat it as the product bible. Perhaps even add comments in code linking to relevant spec sections.
- **Partial Implementations:** Many features are partway done – e.g., the backend has an `upsert_tag` RPC and tag tables ⁵⁶, but the front-end doesn't expose tagging UI. A new dev might see the tag system and not realize it's unused, potentially over-engineering or refactoring it without understanding the future intent. Similarly, the Stripe integration points (libs installed, pricing in docs) are there but not used – a new dev might wonder if it's dead code or pending work. **Mitigation:** clearly log or comment what is MVP vs nice-to-have. For example, if launching without tags or without Share Kit, comment those out or document it so it's not confusing.
- **Supabase RLS and Policies:** The project leans on Supabase's Row Level Security for core security (which is good). But a dev not experienced with RLS might attempt to bypass it inadvertently. For instance, if they try to run a Supabase query on the client without the proper user context, they might get empty results and not understand it's due to RLS. Or they might attempt to alter the RLS policies not knowing the rationale. The RLS tests indicate the importance (they ensure, e.g., an unauthorized update doesn't go through ⁵⁵ ⁵⁷). **Mitigation:** maintain those tests and perhaps add comments in the supabase SQL files explaining each policy's purpose. Also ensure new devs know to update tests if changing policies.
- **Database Migrations vs Schema:** There is a slight confusion possible with how the database schema is managed. We have individual migration files and a composite `schema.sql` (and a note in README to run `schema.sql` on a fresh DB ⁵⁸). A new dev could update the DB directly or via Supabase Studio, forgetting to update migration scripts, breaking the SSOT process. The presence of the **SSOT Freeze** suggests changes to docs require a process – presumably, changes to schema might too. **Mitigation:** establish a clear migration workflow (maybe use Supabase CLI migrations only, and generate a new `schema.sql` dump for reference). Make sure contributors know not to edit `schema.sql` by hand, but to rely on migrations.
- **Environment Configuration:** The `.env` setup in README is incomplete (no mention of `SUPABASE_SERVICE_ROLE_KEY` needed for admin client ⁵⁹, no mention of Stripe or Resend keys which will be needed). A new dev might struggle with setting up their env to run the project. They might get runtime errors like "Missing Supabase env var" ⁶⁰ on launch. **Mitigation:**

update the documentation (.env.example or README) with all required env vars and fallback behaviors. Also caution them never to expose the service role key in client-side code (currently it's safely only in server use, which is correct).

- **Admin Operations Manual:** At launch (and likely for a while), many “admin” tasks might be done manually (e.g., seeding initial listings, manually verifying an ABN by checking a government site, etc.). Future developers or operators need a clear playbook. For example, if a user writes to support asking “why am I not verified yet?”, the operator needs to know how to verify them – currently, that’s running an SQL update or building the feature. **Mitigation:** Document interim manual procedures – e.g., “How to promote a user to operator” (the `seed_operator.sql` exists ³⁶), “How to manually mark a listing verified or suspend a listing” (which tables/flags to flip). This can live in a `docs/OPERATIONS.md` or similar until an admin UI exists.
- **Testing and CI Discipline:** Without CI, a new dev might introduce a breaking change that isn’t caught until deploy. For instance, altering the database schema without updating the TypeScript types in `types/supabase.ts` (which likely contains the generated types) could cause type errors or runtime errors. The project should incorporate a re-generation of types when the DB changes (perhaps via `supabase` CLI). Right now, a dev might not know to do that, leading to drift. **Mitigation:** incorporate `types/supabase.ts` update step into migration PRs, or use a tool to generate types automatically. Also, setting up CI to run `npm run build` will catch any type mismatches early.
- **Understanding the Domain:** The project introduces custom domain terminology (“Studios”, “Mini-site mode”, “Featured placements”, “Truth UI”, etc.). A developer unfamiliar with these could misinterpret variable names or component purposes. For example, `StudioPageModal` is actually a public listing preview, not the actual Studio (which is private in spec). These nuances mean a dev could inadvertently swap concepts (e.g., treat “Studio” and “Listing” as separate when they’re linked). **Mitigation:** possibly add brief JSDoc comments in key components to clarify their role (e.g., “StudioPageModal – modal to display a Listing (Studio Page preview)”). Encourage new team members to read the platform-logic and terminology docs ⁶¹ ⁶² to get the context.

In essence, **the project is at an early but conceptually heavy stage** – any handoff should ensure the new devs understand the guiding principles first, because many implementation decisions flow from those. The risk is less in the code complexity (the code is straightforward) and more in the conceptual alignment. To reduce risk, the current maintainers should update documentation to be crystal clear on what is done vs. pending, and perhaps create a “Launch TODO” or use GitHub issues to track all the gaps identified (many of which are listed above). This way, a new contributor can immediately see what needs doing without needing to reverse-engineer the delta between docs and code.

Recommended Fix Order

Considering all the above, a pragmatic order to tackle fixes (assuming we aim for a viable launch ASAP) would be:

1. **User Auth & Studio Basics First:** Implement sign-up/login and the minimal Creator Studio pages (listing editing and product CRUD). Without these, we can’t onboard creators or content, so they come first. This includes ensuring RLS policies and forms work (test creating a listing, adding a product as a creator).

2. **Search & Discovery:** Next, get the search bar working and the directory listing properly sorted and displayed. This makes the platform usable for end-users to find creators once creators exist. Also implement the unclaimed vs claimed listing behaviors (even if claiming is just a stub that asks users to sign up or contact admin).
3. **Payments & Tier Enforcement:** If launching with the Pro tier and Featured placements from day one, integrate Stripe and enforce Basic vs Pro limits. If deciding to launch a free beta, hide or disable Pro/Featured features in the UI to avoid confusion, but still plan this soon. Verification (ABN) can be parallel or just after – it's important for trust but slightly less urgent than getting revenue or at least plan in place.
4. **Polish UI Copy & SEO:** Go through the UI text and add the required labels ("Sold by...", etc.) and make sure meta tags and 404 page are added. This is best done after core functionality, but before final launch so that the marketing/SEO folks can review it. Also ensure mobile responsiveness issues are addressed in this phase with some quick testing.
5. **Admin & Observability:** Just before or right after launch, get Sentry logging any errors, and have at least a rudimentary admin panel or even just direct DB access procedures for the operator to handle verifications and abuse reports. It's acceptable to launch with a "console and SQL" admin approach if user volume is low, but not having any plan for it would be dangerous. So either build a simple dashboard or prepare SQL scripts for common tasks and document them for the operator.
6. **Testing & CI:** As the fixes are implemented, set up the CI to protect these going forward. Write a couple of integration tests for the critical flows (they can run on CI with a local supabase or a stub). This can be done in parallel with later steps, but ensuring it's in place by code-freeze would help maintain quality as more changes come.

Each of these steps addresses a cluster of gaps identified. Following this order ensures that by the time you're polishing copy and adding monitoring, the fundamental product flows (sign up -> create listing -> find listing -> purchase product, etc.) are operational and aligned with the spec.

Finally, before declaring launch readiness, **perform a full audit against the "launch checklist"** (perhaps derived from the standard directory site checklist the question alludes to). For example: have we covered user auth (yes), listing CRUD (yes), category filters (part of search, ensure filtering by category ID works – the API spec had query params for categoryId ⁶³ though not implemented, maybe a to-do), tag functionality (maybe optional for v1), responsiveness (tested), SEO (meta tags done), payments (done or deferred), analytics (wired), error handling (in place). Any "No" on that list should either be justified as post-launch improvement or fixed before launch.

Integrity Footer

All findings above are derived from the repository's code and documentation as of the main branch audit (commit `6731a0a` as referenced in the SSOT freeze ⁴⁴). **Known facts** (explicitly seen in code or config) have been cited – e.g., missing features like the static placeholder content ⁶, absence of Sentry/Posthog in `package.json` ⁶⁴, or enforcement rules in SQL ¹⁴. **Inferred conclusions** (e.g. that search isn't implemented because no search handler exists, or that no 404 page is present) are based on thorough searches in the codebase and observing what is not there. These have been noted with reasoning. There is minimal **speculation** beyond launch planning – for instance, assuming why

something was omitted or how the team might address it (those are suggestions, not assertions of fact). All recommendations are made in the spirit of aligning the product with industry best practices and the project's own stated requirements.

1 38 **Marketplace.tsx**

<https://github.com/carlsburbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/components/home/Marketplace.tsx>

2 22 29 61 **platform-logic.md**

<https://github.com/carlsburbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/docs/platform-logic.md>

3 8 **listings.ts**

<https://github.com/carlsburbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/lib/listings.ts>

4 5 19 39 **PUBLIC_UX_CONTRACT.md**

https://github.com/carlsburbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/docs/SSOT/PUBLIC_UX_CONTRACT.md

6 7 27 28 40 41 **StudioPageModal.tsx**

<https://github.com/carlsburbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/components/directory/StudioPageModal.tsx>

9 **20260103000000_refactor_taxonomy.sql**

https://github.com/carlsburbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/supabase/migrations/20260103000000_refactor_taxonomy.sql

10 12 **schema.sql**

<https://github.com/carlsburbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/supabase/schema.sql>

11 **StudioPageModal.test.tsx**

<https://github.com/carlsburbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/components/directory/StudioPageModal.test.tsx>

13 **20260102000004_create_actors.sql**

https://github.com/carlsburbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/supabase/migrations/20260102000004_create_actors.sql

14 20 **20260102000006_listings_rls.sql**

https://github.com/carlsburbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/supabase/migrations/20260102000006_listings_rls.sql

15 16 **20260102000005_actor_trigger.sql**

https://github.com/carlsburbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/supabase/migrations/20260102000005_actor_trigger.sql

17 18 43 47 **USER_MODEL_AND_STATE_MACHINE.md**

https://github.com/carlsburbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/docs/SSOT/USER_MODEL_AND_STATE_MACHINE.md

21 64 **package.json**

<https://github.com/carlsburbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/package.json>

23 24 46 62 **PRODUCT_CONSTITUTION.md**

https://github.com/carlsurbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/docs/SSOT/PRODUCT_CONSTITUTION.md

25 26 35 45 **OPERATOR_AND_ADMIN_AUTOMATION.md**

https://github.com/carlsurbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/docs/SSOT/OPERATOR_AND_ADMIN_AUTOMATION.md

30 31 **Hero.tsx**

<https://github.com/carlsurbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/components/home/Hero.tsx>

32 33 **SEARCH_CONTRACT.md**

https://github.com/carlsurbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/docs/SSOT/SEARCH_CONTRACT.md

34 58 **README.md**

<https://github.com/carlsurbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/README.md>

36 **seed_operator.sql**

https://github.com/carlsurbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/supabase/seed_operator.sql

37 **20260102000007_products_rls.sql**

https://github.com/carlsurbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/supabase/migrations/20260102000007_products_rls.sql

42 50 51 52 **CREATOR_STUDIO_SPEC.md**

https://github.com/carlsurbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/docs/SSOT/CREATOR_STUDIO_SPEC.md

44 **SSOT_FREEZE_RECORD.md**

https://github.com/carlsurbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/docs/SSOT/SSOT_FREEZE_RECORD.md

48 49 56 **20260103000002_taxonomy_triggers_and_rpc.sql**

https://github.com/carlsurbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/supabase/migrations/20260103000002_taxonomy_triggers_and_rpc.sql

53 **page.tsx**

[https://github.com/carlsurbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/app/\(home\)/page.tsx](https://github.com/carlsurbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/app/(home)/page.tsx)

54 **page.test.tsx**

[https://github.com/carlsurbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/app/\(home\)/page.test.tsx](https://github.com/carlsurbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/app/(home)/page.test.tsx)

55 57 **rls_policies.test.ts**

https://github.com/carlsurbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/supabase/tests/rls_policies.test.ts

59 60 **supabase-admin.ts**

<https://github.com/carlsurbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/lib/supabase-admin.ts>

63 **api.md**

<https://github.com/carlsurbmates/suburbmatesmelbcloud/blob/4e68f6b27e2607aab02824c30de7bf5ab4570ece/docs/api.md>