SovereignBoard AI — Security and Quality Audit Report

1. Executive Summary

SovereignBoard AI is a multi-tenant Next.js application providing an AI executive dashboard with 15 specialized agents, role-based access control, report generation, and a super-admin console. The app uses Supabase as its data layer with an anon-key client in the browser and a service-role key in API routes. The codebase is well-structured with clear separation between the store, API routes, and UI components, and the visual design is cohesive and polished.

However, the application has a critical security architecture flaw: every Row Level Security policy across all tables uses `USING (true)`, meaning the anon key — which is embedded in the client bundle — can read, insert, update, and delete every row in every table with no tenant isolation whatsoever. The RBAC system just added compounds this: its five tables are also wide open, meaning any visitor can create roles, assign themselves Admin, and grant themselves any permission. Authentication is bypassed entirely by demo login buttons that synthesize profiles client-side. API routes accept org IDs from the client with no server-side auth check, enabling cross-tenant data access. The type checker reports 3 errors and lint reports 12 errors, both of which would block a clean CI pipeline. The verdict is DO NOT SHIP.

2. Verification Status

Check	Status
Build (`npm run build`)	MACHINE-VERIFIED — PASS
Type check (`npm run typecheck`)	MACHINE-VERIFIED — FAIL (3 errors)
Lint (`npm run lint`)	MACHINE-VERIFIED — FAIL (12 errors)
Tests	NOT RUN — no test script defined in package.json
RLS policy inspection	MACHINE-VERIFIED via Supabase MCP
Security advisor	MACHINE-VERIFIED — 0 lints (advisor does not flag `USING(true)` policies)
Runtime / visual / responsive	REQUIRES HUMAN VERIFICATION

3. Baseline Metrics

Metric	Value
Build	PASS
Type errors	3
Lint errors	12
Tests	Not configured
Bundle size	NOT MEASURED (no bundle analyzer configured)
Dependency count	12 production, 12 dev (from package.json)
Source files under audit	~15
Source lines (estimated)	~3,500

4. Known Symptoms — Root Causes

No specific bugs were reported (known_symptoms: NONE).

5. Findings

P0 — Critical

ID	Confidence	File:line	Problem	Evidence	Proposed fix	Blast radius
F01	CONFIRMED	supabase/migrations/20260630190641_sovereignboard_core.sql:58-98	All RLS policies use `USING(true)` — no tenant isolation. Any client with the anon key (embedded in the browser bundle) can read, insert, update, and delete every row in organizations, profiles, agent_logs, and redline_events across all tenants.	`CREATE POLICY "profiles_select" ON profiles FOR SELECT TO anon, authenticated USING (true);` — same pattern for all 16 policies across 4 core tables. Supabase MCP confirms all policies are `USING(true)`.	Replace all `USING(true)` with org-scoped predicates: `USING (org_id = (SELECT org_id FROM profiles WHERE auth_id = auth.uid()) OR auth.uid() IS NULL AND EXISTS (SELECT 1 FROM profiles WHERE auth_id = auth.uid() AND role = 'super_admin'))`. Use `auth.uid()` for ownership checks. Enable per-verb policies.	Entire database — all tenants, all tables.
F02	CONFIRMED	supabase/migrations/20260811191327_add_rbac_tables.sql:52-109	All 5 RBAC tables use `USING(true)` for all CRUD operations. Any anonymous visitor can create roles, assign themselves the Admin role, grant themselves any permission, create sessions, and revoke other users' sessions.	`CREATE POLICY "anon_crud_rbac_roles" ON rbac_roles FOR ALL TO anon, authenticated USING (true) WITH CHECK (true);` — identical for permissions, role_permissions, user_roles, and sessions.	Restrict RBAC table mutations to authenticated users with an Admin role. Use a subquery check: `USING (EXISTS (SELECT 1 FROM rbac_user_roles ur JOIN rbac_roles r ON ur.role_id = r.id WHERE ur.profile_id = (SELECT id FROM profiles WHERE auth_id = auth.uid()) AND r.name = 'Admin'))`. Read access can remain broader.	Entire RBAC system — all roles, permissions, sessions.
F03	CONFIRMED	app/page.tsx:79-119	Demo login buttons bypass Supabase Auth entirely, creating synthetic Profile	`const superAdminProfile: Profile = { id: 'super-`	Remove demo login buttons for production. If demo access is needed for	All authenticated routes — dashboard, super-admin,

ID	Confidence	File:line	Problem	Evidence	Proposed fix	Blast radius
			objects client-side and setting them in the Zustand store. This grants full access to anyone who clicks a button, with no server-side verification.	`admin', ... role: 'super_admin', is_approved: true, ... }; setCurrentUser(superAdminProfile);` — no DB lookup, no token, no server check.	development, gate it behind `process.env.NODE_ENV === 'development'` and never ship it to production.	RBAC, user management.
F04	CONFIRMED	app/api/generate-report/route.ts:212-299	API route uses the service-role key and accepts `orgId` from the request body with no authentication check. Any caller can generate a report for any organization, including ones they do not belong to.	`const { orgId, reportType, createdBy } = body; const supabase = createClient(supabaseUrl, supabaseServiceKey);` — no auth header check, no user verification, orgId is trusted from client.	Validate the caller's identity via the Authorization header (JWT from Supabase Auth). Verify the user belongs to the requested org before generating the report. Do not trust `orgId` from the client without server-side ownership verification.	Report generation for all organizations.
F05	CONFIRMED	app/api/download-report/route.ts:9-45	API route uses the service-role key and accepts `reportId` from the query string with no authentication check. Any caller can download any report from any organization.	`const reportId = req.nextUrl.searchParams.get('id'); const supabase = createClient(supabaseUrl, supabaseServiceKey);` — no auth check, reportId trusted from client.	Validate the caller's JWT and verify the report belongs to their organization before serving it.	All reports across all tenants.
F06	CONFIRMED	app/api/orchestrator/route.ts:232-296	API route has no authentication check. Any caller can trigger agent runs and consume API credits (Anthropic, Gemini, Perplexity).	`export async function POST(req: Request) { const { prompt, targetingAgent, orgContext } = await req.json();` — no auth header check before calling paid external APIs.	Require a valid JWT in the Authorization header. Verify the user is authenticated and approved before calling external API providers.	All agent executions — external API cost and data exposure.

P1 — High

ID	Confidence	File:line	Problem	Evidence	Proposed fix	Blast radius
F07	CONFIRMED	app/dashboard/page.tsx:56-59, app/super-admin/page.tsx:69-74, app/admin/users/page.tsx:43-46, app/admin/rbac/page.tsx:73-78	All route guards are client-side only. `isAuthenticated` and `isSuperAdmin` are stored in localStorage (Zustand persist) and checked in useEffect. A user can manually set `sovereign-state` in localStorage to bypass all guards.	`if (!isAuthenticated) router.push('/')` — client-side redirect only. The page content renders before the redirect fires. Store state is in `localStorage` under key `sovereign-state`.	Add server-side middleware (Next.js `middleware.ts`) that validates the Supabase session JWT before allowing access to `/dashboard`, `/super-admin`, `/admin/*`. Client-side guards can remain as UX, but the server must be the authority.	All protected routes.
F08	CONFIRMED	app/admin/rbac/page.tsx:88, app/admin/users/page.tsx:41	Admin authorization check is client-side only. `isAdmin` is computed from the Zustand store which reads from localStorage. A user can set `isSuperAdmin:true` in localStorage to unlock all admin UI.	`const isAdmin = isSuperAdmin || currentUser?.role === 'tenant_admin';` — computed from client state, no server verification.	Server-side middleware or API route must verify the user's role before serving admin pages or accepting admin mutations. The client check is decorative only.	All admin functionality — RBAC, user management, org configuration.
F09	CONFIRMED	app/api/generate-report/route.ts:20-210	Report HTML is built by interpolating org data, agent logs, and redline events directly into template strings without HTML escaping. If any field (org name, agent payload, dissent memo) contains HTML or script tags, they will be rendered in the report iframe.	`const orgName = org.business_name;` then `<h1>${config.title} — ${orgName}</h1>` — no escaping. `r.dissent_memo?.slice(0,150)` injected into `<td>`.	HTML-escape all interpolated values before inserting them into the template string. Use a helper like `escapeHtml(str)` that replaces `<`, `>`, `&`, `"`, `'`.	Report content — XSS in the report viewer iframe.
F10	CONFIRMED	app/page.tsx:48-49	Login flow calls `supabase.auth.signInWithPassword` but then ignores the session and queries the profiles table with the anon key using `.eq('email',`	`const { error: authError } = await supabase.auth.signInWithPassword(...); if (authError) { ... } const { data:`	After successful auth, use the authenticated Supabase client (with the user's access token) to query the profile. Verify `auth.uid()` matches the profile's	Login flow — any valid email returns the profile even without a valid password if the auth call is bypassed.

ID	Confidence	File:line	Problem	Evidence	Proposed fix	Blast radius
			email) . The profile is returned regardless of whether the auth session is valid, because RLS is USING(true) .	profile } = await supabase.from('profiles'). select('*').eq('email', email).single(); — the profile query uses the anon client, not the authenticated session.	auth_id on the server side.	
F11	CONFIRMED	app/api/download-report/route.ts:38-44	The format=pdf query parameter is accepted but ignored — both the html and pdf branches return the same HTML content with the same headers. The download filename always ends in .html even when the client requests PDF.	if (format === 'html') { return new NextResponse(report.content_html, ...) } return new NextResponse(report.content_html, ...) — identical response for both branches.	Either implement actual PDF conversion (e.g., via a headless browser or a library like Puppeteer) or remove the format=pdf option and only offer HTML download. Document the limitation.	Report download — users requesting PDF get HTML with a .html extension.

P2 — Maintainability / Polish

ID	Confidence	File:line	Problem	Evidence	Proposed fix	Blast radius
F12	CONFIRMED	app/admin/rbac/page.tsx:191	TypeScript error: Set<string> iteration requires --downlevelIteration or es2015 target. The Array.from(editPermIds) call fails type checking.	npm run typecheck output: app/admin/rbac/page.tsx(191,30): error TS2802: Type 'Set<string>' can only be iterated through when using the '--downlevelIteration' flag	Set target to es2015 or higher in tsconfig.json, or use Array.from(editPermIds) which is already the correct call — verify the tsconfig target setting.	RBAC page — type check failure blocks CI.
F13	CONFIRMED	app/dashboard/page.tsx:16, app/super-admin/page.tsx:4	TypeScript errors: missing properties on type — active_agents does not exist on Organization in some code paths, and constitution_articles column is referenced but not in the type.	npm run typecheck output: app/dashboard/page.tsx(16,3): error and app/super-admin/page.tsx errors	Add missing fields to the Organization type or use optional chaining. Add constitution_articles to the type if the column exists, or remove the feature if it does not.	Dashboard and super-admin pages — type check failures.

ID	Confidence	File:line	Problem	Evidence	Proposed fix	Blast radius
F14	CONFIRMED	app/page.tsx:783, app/dashboard/page.tsx:258, app/super-admin/page.tsx:541	ESLint: unescaped entities (`"` in JSX text) across multiple files.	`npm run lint` output: 12 `react/no-unescaped-entities` errors	Replace `"` with `&quot;` or use curly brace string literals `{'"'}` in JSX text content.	Lint — CI pipeline blocked.
F15	CONFIRMED	app/super-admin/page.tsx:172-177	`saveConstitution` calls `supabase.from('organizations').update({ constitution_articles: ... })` but the `constitution_articles` column does not appear in the `organizations` table schema or any migration. This update will silently fail or throw a Postgres error.	No migration adds `constitution_articles` to `organizations`. The `Organization` type in client.ts does not include the field.	Add a migration to create the `constitution_articles` column (JSONB) on `organizations`, or remove the constitution feature if it is not ready.	Constitution feature — data is not persisted.
F16	CONFIRMED	src/store/sovereignState.ts:559	`isSuperAdmin` is derived from `user?.role === 'super_admin' || user?.access_privileges?.is_super_admin === true`. The `access_privileges` JSONB field is user-editable via the user management page (F08). A user can set `is_super_admin: true` on their own profile.	`setCurrentUser: (user) => set({ ... isSuperAdmin: user?.role === 'super_admin' || user?.access_privileges?.is_super_admin === true })` — trusts a user-editable field for admin determination.	Use only the `role` column (which is also editable but at least is a single source of truth) or better, derive admin status from server-side session claims. Never trust a user-editable JSONB field for authorization.	Super admin access — privilege escalation.

P3 — Cosmetic

ID	Confidence	File:line	Problem	Evidence	Proposed fix	Blast radius
F17	CONFIRMED	app/api/download-manual/route.ts:7, app/api/download-simple-guide/route.ts:7	Both routes use `readFileSync` synchronously, blocking the event loop. For small static files this is negligible, but it is not idiomatic for Next.js route handlers.	`const fileContent = readFileSync(filePath, 'utf-8');` — synchronous file read in an async function.	Use `fs.promises.readFile` (async) or better, serve the files directly from the `public/` directory via static links instead of API routes.	Download routes — minor performance.

ID	Confidence	File:line	Problem	Evidence	Proposed fix	Blast radius
F18	CONFIRMED	app/reports/page.tsx:118-122	Auto-refresh interval fires every 15 seconds and runs 4 Supabase queries each time, regardless of whether the tab is visible. This creates unnecessary load and Supabase quota consumption.	`const interval = setInterval(() => loadReportData(selectedOrgId), 15000);` — no visibility check.	Use the Page Visibility API to pause polling when the tab is hidden, or switch to Supabase realtime subscriptions instead of polling.	Reports page — unnecessary network traffic.

6. Rejected Findings

1. **Considered flagging: API keys (ANTHROPIC_API_KEY, GEMINI_API_KEY, PERPLEXITY_API_KEY) used in the orchestrator route.** Rejected because these are server-side environment variables accessed via `process.env`, not exposed to the client bundle. The route is a Next.js API route (server-side). The real issue is the missing auth check (F06), not key exposure.
2. **Considered flagging: `crypto.randomUUID()` used for session tokens in the RBAC page.** Rejected because `crypto.randomUUID()` produces cryptographically random UUIDs suitable for session tokens in a pre-launch context. For production, a JWT or signed token would be more appropriate, but this is not a vulnerability at the pre-launch stage.
3. **Considered flagging: the `persist` middleware in Zustand stores auth state in localStorage.** Rejected as a standalone finding because it is already covered by F07 (client-side guards). The localStorage storage itself is not the vulnerability — the lack of server-side validation is. localStorage is a valid pattern for UX state as long as the server does not trust it.
4. **Considered flagging: the `buildLocalResponse` function in the orchestrator returns hardcoded strings that look like real analysis.** Rejected because this is clearly a fallback for when API keys are not configured, and the `live: false` flag in the response distinguishes it from real API output. It is a feature, not a bug.

7. Security Posture

Secrets: All API keys (Anthropic, Gemini, Perplexity) and Supabase keys (URL, anon key, service role key) are stored in environment variables and accessed via `process.env`. No secrets were found hardcoded in source files. The Supabase anon key is intentionally exposed to the client (this is the Supabase design), but the service role key must never reach the client bundle — confirmed it is only used in API route handlers.

RLS: All 20 tables have RLS enabled, but every policy uses `USING(true)`. This is the single most critical finding. RLS is effectively disabled — the anon key can access all data in all tables.

Auth: Supabase Auth is used for the real login flow, but demo login buttons bypass it entirely. The profile query after login uses the anon client, not the authenticated session, so RLS (if it were properly configured) would not scope the query to the user.

API routes: Three of four API routes use the service-role key with no authentication check. Any caller can trigger report generation, download any report, and run agents.

8. Recommended Repair Order

Batch 1 — Critical security (fix before any deployment) - F01: Replace all `USING(true)` RLS policies with org-scoped predicates - F02: Lock down RBAC tables to authenticated admin users only - F03: Remove or gate demo login buttons

behind development mode - F04: Add auth check to generate-report API route - F05: Add auth check to download-report API route

Batch 2 — High security and correctness - F06: Add auth check to orchestrator API route - F07: Add Next.js middleware for server-side route guards - F08: Move admin authorization to server-side - F09: HTML-escape all interpolated values in report template - F10: Use authenticated Supabase client for profile query after login

Batch 3 — Type safety and data integrity - F11: Fix or remove the PDF download option - F12: Fix tsconfig target for Set iteration - F13: Add missing fields to Organization type - F15: Add constitution_articles column or remove the feature - F16: Stop trusting access_privileges.is_super_admin for admin determination

Batch 4 — Polish - F14: Fix ESLint unescaped entities - F17: Use async file reads in download routes - F18: Add visibility check to reports auto-refresh

9. Readiness

DO NOT SHIP

Top five blockers: 1. F01 — All RLS policies are `USING(true)` : no tenant isolation, all data exposed to the anon key 2. F02 — RBAC tables are wide open: anyone can grant themselves Admin 3. F03 — Demo login bypasses authentication entirely 4. F04/F05/F06 — API routes have no authentication: anyone can generate reports, download any report, and trigger paid API calls 5. F07/F08 — All route guards and admin checks are client-side only, trivially bypassed via localStorage