Axieo Guard Security Audit Report

> **Sample report.** BookNest is a fictional app and company. Every name, address, number and finding below is made up to show what an Axieo Guard report looks like. It is not a client report.

Client	BookNest Studios Inc. (fictional)
App	BookNest, https://app.booknest.example
Platform	Lovable (React + Vite + Supabase), hosted on Vercel
Repository / commit	github.com/booknest-demo/booking @ `3f9c2e1`
Audit type	Security Audit Standard
Checklist	lovable-supabase.v1 (25 checks)
Testing dates	2026-11-02 to 2026-11-04
Report ID / date	AG-SAMPLE-001 / 2026-11-05
Prepared by	The Axieo team, Axieo Technologies Inc.

1. Summary

Risk score: 86 / 100 (Critical risk)

BookNest works well, and several important things are already done right: users' profiles are private, and payments are checked properly on the server. But two problems need fixing before you take on more studios. Right now, anyone who knows how can list every booking in the app, including customers' names and phone numbers, and your live Stripe secret key is visible in the website code. Section 4 explains each problem and recommends a fix your team can apply in a few hours in total.

Severity	Count
🔴 Critical	2
🟠 High	1
🟡 Medium	3
🔵 Low	0
✅ Passed	16
⚠️ Manual review	2
➖ Not applicable	1

Top 3 actions

1. Turn on Row Level Security on the `bookings` table and add "own rows only" rules (AG-SEC-004).

2. Move Stripe Checkout to an Edge Function, then roll (rotate) the Stripe secret key (AG-SEC-001).
3. Make the `waivers` storage bucket private and serve files with short-lived signed links (AG-SEC-007).

---

2. Scope and method

- **In scope:** https://app.booknest.example, the repository above (read-only access), and the Supabase project `bkn-prod` (project reference only), as listed in the signed authorization form.
- **Out of scope:** the marketing site, the staging environment, and any third-party service other than the public Stripe configuration visible in the code.
- **Method:** automated scan of the repository, read-only checks of the live app (headers, exposed files, and row counts with the public key; no data downloaded), and manual review of code and Supabase settings on a screen share with your team.
- **Limits:** this report reflects the app at the commit and dates above. It is not a guarantee that the app has no other vulnerabilities.

How the risk score works: each failed check adds points by severity (Critical 25, High 15, Medium 7, Low 3), capped at 100. Lower is better. Manual-review items add nothing until they're confirmed. BookNest: 2 × 25 + 1 × 15 + 3 × 7 = 86.

---

3. Results by check

ID	Check	Result	Severity
AG-SEC-001	No secret keys in front-end code	❌ Fail	Critical
AG-SEC-002	`.env` files not committed; no secrets in git history	✅ Pass	Critical
AG-SEC-003	Only public keys in the browser	✅ Pass	High
AG-SEC-004	Row Level Security enabled on every public table	❌ Fail	Critical
AG-SEC-005	No "allow everyone" policies on sensitive tables	✅ Pass	Critical
AG-SEC-006	Users can only read and change their own rows	✅ Pass	Critical
AG-SEC-007	Storage buckets private unless they must be public, with policies	❌ Fail	High
AG-SEC-008	Database functions and views don't expose private data	✅ Pass	High
AG-SEC-009	Auth settings are safe	❌ Fail	Medium
AG-SEC-010	Admin pages protected on the server, not just hidden	✅ Pass	Critical
AG-SEC-011	User roles checked in the database or Edge Functions	✅ Pass	High
AG-SEC-012	Redirect URLs allow-listed; logout and session expiry work	✅ Pass	Medium
AG-SEC-013	Edge Functions verify the user's login before sensitive work	✅ Pass	High
AG-SEC-014	User input validated; no SQL built from raw text	✅ Pass	High
AG-SEC-015	Rate limits on login, sign-up and costly endpoints	⚠️ Manual review	Medium
AG-SEC-016	CORS not set to `*` on sensitive endpoints	✅ Pass	Medium
AG-SEC-017	Stripe webhooks verify the signature; prices set on the server	✅ Pass	Critical

ID	Check	Result	Severity
AG-SEC-018	Paid features enforced on the server	➖ Not applicable	High
AG-SEC-019	No unsafe HTML rendering of user content	✅ Pass	High
AG-SEC-020	Security headers set on hosting	❌ Fail	Medium
AG-SEC-021	No known-vulnerable packages	❌ Fail	Medium
AG-SEC-022	Database backups enabled and a restore tested	⚠️ Manual review	High
AG-SEC-023	Error monitoring exists; logs have no secrets or personal data	✅ Pass	Medium
AG-SEC-024	Separate development and production environments	✅ Pass	Medium
AG-SEC-025	Privacy basics: policy, account deletion, minimal data	✅ Pass	Medium

Result key: ✅ Pass · ❌ Fail · ⚠️ Manual review · ➖ Not applicable (BookNest has no paid tiers inside the app; customers pay per booking).

---

4. Findings

AG-SEC-004 · Row Level Security is off on the `bookings` table

Severity	🔴 Critical
Where	Supabase table `public.bookings`; created in `supabase/migrations/20260210_init.sql`, line 22
Status	Open

What we found: the `bookings` table has Row Level Security (RLS) turned off. Your other tables (`profiles`, `studios`, `classes`) have it on, with good rules.

Why it matters: your app's public key is in every visitor's browser, by design. Without RLS, that key can read and change every row in `bookings`: every customer's name, phone number and class times, across all studios. Supabase's own guidance is to "enable RLS on every table in an exposed schema".

Evidence: an anonymous count request on `bookings` with the public key returned a row count of 1,284 (count only; no rows were downloaded or stored). The table is created on line 22 of the migration without `enable row level security`. Supabase's Security Advisor also lists "RLS disabled in public" for this table.

Recommended fix:

1. Turn on RLS for `public.bookings`.
2. Add rules so customers see and change only their own bookings, and studio staff see bookings for their own studio (example SQL below).
3. Add the same `enable row level security` line to the migration, so a new environment starts protected.

Example SQL to review before use (it assumes `bookings.customer_id` and `bookings.studio_id`, and the existing `studio_staff` table):

```
alter table public.bookings enable row level security;

create policy "Customers read own bookings" on public.bookings
  for select to authenticated using ((select auth.uid()) = customer_id);

create policy "Customers create own bookings" on public.bookings
  for insert to authenticated with check ((select auth.uid()) = customer_id);

create policy "Customers cancel own bookings" on public.bookings
  for update to authenticated using ((select auth.uid()) = customer_id)
  with check ((select auth.uid()) = customer_id);

create policy "Staff read their studio's bookings" on public.bookings
  for select to authenticated using (
    exists (select 1 from public.studio_staff s
            where s.studio_id = bookings.studio_id and s.user_id = (select auth.uid()))
  );
```

Suggested prompt for your builder (Lovable):

> Turn on Row Level Security for the `bookings` table in Supabase. Add policies so a signed-in customer can select, insert and update only rows where `customer_id = auth.uid()`, and studio staff (users in `studio_staff` for the same `studio_id`) can select bookings for their studio. Don't allow any access for anonymous users. Show me the SQL before running it.

Review what your builder proposes before you apply it. AI tools can give different results each time.

Effort: about 45 minutes · **Fix order:** 1 of 6

How to check your fix: sign in as one customer, then as another in a different browser: the second must not see the first's bookings. In Supabase, the Security Advisor should no longer list `bookings`. At re-test we run the anonymous count request again; it should return 0 rows.

AG-SEC-001 · Live Stripe secret key in the website code

Severity	🔴 Critical
Where	`src/lib/payments.ts`, line 3
Status	Open

What we found: the file creates Stripe Checkout sessions directly in the browser, using a live secret key (`sk_live_…`, redacted in our notes). Because it's in `src/`, the key is included in the JavaScript every visitor downloads.

Why it matters: the Stripe secret key can do almost anything in your Stripe account: create charges, issue refunds, read customer details. Anyone who opens the browser's developer tools can copy it. Because the site is live, assume it has already been copied.

Evidence: the key pattern `sk_live_` appears at `src/lib/payments.ts:3` and in the built file `assets/index-*.js` served by the live site. We did not use the key.

Recommended fix:

1. Create a Supabase Edge Function (e.g. `create-checkout`) that creates the Checkout session on the server, reading the key with `Deno.env.get("STRIPE_SECRET_KEY")`, and taking prices from your Stripe Price IDs, not from the browser.
2. Change the booking page to call that function instead of Stripe directly. Make the function check the signed-in user first (`supabase.auth.getUser()`).
3. **Roll the key:** Stripe Dashboard → Developers → API keys → roll the secret key, put the new key in the Edge Function secrets, then make sure the old one is revoked. Deleting the line from the code is not enough.
4. Search the repository and its history for any other copies of `sk_live_`.

Suggested prompt for your builder (Lovable):

> Move Stripe Checkout session creation out of `src/lib/payments.ts` into a new Supabase Edge Function called `create-checkout`. The function must check the signed-in user with `supabase.auth.getUser()`, read the Stripe key from the `STRIPE_SECRET_KEY` secret, and use our Stripe Price IDs on the server. Remove every Stripe secret key from the front-end code. Show me the changes before applying them.

Review what your builder proposes before you apply it. AI tools can give different results each time.

Effort: about 1.5 hours · **Fix order:** 2 of 6 (roll the key today, even before the code change)

How to check your fix: in the live site, open developer tools → Sources and search for `sk_live`: nothing should be found. A test booking should still reach Stripe Checkout. At re-test we scan the code and the built JavaScript again.

AG-SEC-007 · The `waivers` storage bucket is public

Severity	🟠 High
Where	Supabase Storage bucket `waivers`
Status	Open

What we found: the bucket that holds customers' signed waiver PDFs is marked "Public". File names follow a predictable pattern (`<booking id>.pdf`).

Why it matters: anyone with a file's address can open it without signing in, and the pattern makes addresses easy to guess. Waivers contain names, signatures and emergency contacts.

Evidence: the bucket is listed as Public in the Storage settings (viewed on the screen share). We confirmed that a public URL pattern exists; we did not open any customer file.

Recommended fix:

1. Make the bucket private (Storage → `waivers` → Edit bucket → turn off "Public bucket").
2. Store files under a folder per customer (e.g. `waivers/<customer id>/<file>`), and add a storage policy so customers read only their own folder and staff read their studio's files.
3. When the app shows a waiver, create a short-lived signed link with `createSignedUrl(path, 60)` instead of a public link.

Suggested prompt for your builder (Lovable):

> Make the Supabase Storage bucket `waivers` private. Save new waivers under `waivers/<user id>/`. Add storage policies so a signed-in user can read only files in their own folder, and studio staff can read files for

> bookings at their studio. Wherever the app shows a waiver, use a signed URL that expires after 60 seconds. Show me the changes before applying them.

Review what your builder proposes before you apply it. AI tools can give different results each time.

Effort: about 1 hour · **Fix order:** 3 of 6

How to check your fix: copy the address of a waiver while signed in, sign out, and open it in a private window: it should fail. The bucket should show as Private in Supabase.

AG-SEC-009 · Email confirmation is off and passwords can be 6 characters

Severity	🟡 Medium
Where	Supabase → Authentication settings
Status	Open

What we found: new accounts don't need to confirm their email address, the minimum password length is 6, and leaked-password protection is off.

Why it matters: anyone can sign up with someone else's email address, and short or leaked passwords are easy to guess.

Evidence: Authentication → Providers → Email ("Confirm email" off) and Authentication → Policies (minimum length 6), viewed on the screen share.

Recommended fix: turn on "Confirm email"; set the minimum password length to at least 8 (12 is better); turn on leaked-password protection if your Supabase plan includes it. Check that the sign-up page tells users to check their inbox.

Suggested prompt for your builder (Lovable):

> After sign-up, show a message asking the user to confirm their email, and handle the confirmation link. Require passwords of at least 12 characters on the sign-up form. (We'll turn on email confirmation in the Supabase dashboard ourselves.)

Effort: about 20 minutes · **Fix order:** 4 of 6

How to check your fix: sign up with a new address: you shouldn't be signed in until you click the confirmation email. A 6-character password should be refused.

AG-SEC-020 · Two security headers are missing

Severity	🟡 Medium
Where	https://app.booknest.example (Vercel)
Status	Open

What we found: HTTPS works and redirects correctly, but the site doesn't send `Strict-Transport-Security` or `Content-Security-Policy` headers, and nothing stops other sites from showing it in a frame.

Why it matters: these headers make it harder to downgrade a connection, inject scripts or trick users with a framed copy of your booking page.

Evidence: response headers of the home page and `/book`, read on 2026-11-03.

Recommended fix: add headers in `vercel.json`. Start the Content-Security-Policy in report-only mode for a week, then switch it on. Example to review before use:

```
{
  "headers": [{
    "source": "/(.*)",
    "headers": [
      { "key": "Strict-Transport-Security", "value": "max-age=63072000; includeSubDomains" },
      { "key": "X-Frame-Options", "value": "DENY" },
      { "key": "Content-Security-Policy-Report-Only", "value": "default-src 'self'; connect-src
'self' https://*.supabase.co; frame-src https://checkout.stripe.com; img-src 'self' data: https:" }
    ]
  }]
}
```

Effort: about 30 minutes, plus a check a week later · **Fix order:** 5 of 6

How to check your fix: open securityheaders.com (or the browser's Network tab) for your site and confirm the headers appear.

AG-SEC-021 · One package has a known high-severity issue

Severity	🟡 Medium
Where	`package-lock.json`
Status	Open

What we found: `npm audit` reports 1 high and 2 moderate issues, all in packages pulled in by the date-picker library, with fixed versions available.

Why it matters: known issues in packages are among the first things automated attacks look for.

Evidence: `npm audit` output, saved in our audit folder.

Recommended fix: run `npm audit`, update the date-picker library to its latest version, run `npm audit` again, and test the booking calendar.

Suggested prompt for your builder (Lovable):

> Update the date-picker library and its dependencies to the latest versions that fix the `npm audit` warnings. Don't change how the booking calendar looks or works. Show me what changes.

Effort: about 20 minutes · **Fix order:** 6 of 6

How to check your fix: `npm audit` shows no high issues; the calendar still works on desktop and mobile.

AG-SEC-015 · Rate limits: confirm on the review call (manual review)

We couldn't confirm from the code whether the `send-reminder` Edge Function (which sends SMS reminders) limits how often one user can call it. Supabase limits sign-ups and logins by default, but not your own functions. On the call, we'll look at it together; if there's no limit, we'll recommend one, because SMS costs money per message.

AG-SEC-022 · Backups: confirm on the review call (manual review)

Your Supabase plan's backup settings weren't shown on the screen share. On the call, we'll confirm which backups your plan includes and recommend a restore test into a separate project, so you know you can recover your bookings if something goes wrong.

---

5. What's working well

- `profiles`, `studios` and `classes` have RLS with "own rows only" rules (AG-SEC-005, 006).
- The Stripe webhook verifies Stripe's signature, and prices come from Price IDs on the server (AG-SEC-017).
- Admin pages and the staff dashboard check the user's role in the database, not just in the page (AG-SEC-010, 011).
- No `.env` files or secrets in the git history (AG-SEC-002).
- Separate development and production Supabase projects (AG-SEC-024), and a clear privacy policy with account deletion (AG-SEC-025).

---

6. Next steps

1. **Review call:** 30 minutes to go through the findings and the two manual items. Book at a time that suits you.
2. **Your team fixes** each finding, using the recommendations in section 4 in the order given. Roll the Stripe key today. You can use the suggested builder prompts, or hand section 4 to your developer. Email us with any questions about a finding.
3. **We re-test:** tell us when you're done. Your audit includes **one re-test within 30 days** of this report (by 2026-12-05). Extra re-tests are CA$350.
4. **Stay protected:** BookNest stores customers' personal details and takes payments, so we'd recommend the **Pro** Care Plan: we re-check the app every month as it changes.

Axieo Technologies Inc., Toronto, Canada · info@axieo.ca · axieo.ca

Sample report: fictional app and company, for illustration only. A real report is confidential and provided under the client's Service Agreement. No security review can guarantee an app is free of vulnerabilities.

Advisory only: the recommendations in this report are advice. Axieo has not changed and will not change your code, data, settings or environments. Suggested prompts, example code and commands are recommendations only; review them before use. The output of your AI builders and coding tools is outside Axieo's control. You decide whether and how to apply each recommendation, and you are responsible for the changes you, your staff or your contractors make. Re-tests confirm whether specific findings are still present at the time of the re-test.