Rescue Audit — Sample Report

Prepared by: Paul Caruana, CTO · Dargavel AI **Subject:** "Acme Bookings" — booking platform (built with Lovable + Cursor, ~4 months live) **Audit type:** Tier 1 Rescue Audit · **Turnaround:** 72 hours

> **This is a real audit of a real (demonstration) codebase, redacted and anonymised.** It's here so you can see exactly what you receive before you pay. Your report follows this structure, on your code.

1. Executive summary — in plain English

Your app works, and from the outside it looks fine. Underneath, it has **four issues that could end the business**, three that will cost you money or customers soon, and a handful of smaller things worth tidying. None of this means the app was built badly for what it was — it means it was built to demo, not to run in production with real customers and real card payments. That's normal for AI-assisted builds. It's also fixable.

The single most urgent finding: **your live Stripe secret key is exposed in code that ships to every visitor's browser.** Anyone who opens their browser's developer tools can read it and charge cards or issue refunds against your account. This needs to be rotated **today**, before anything else in this report.

Here's the honest bottom line: **you do not need a rebuild.** You need roughly 2–4 days of focused remediation on the critical items, and then a light ongoing discipline. I've laid out exactly what to do first, what can wait, and what to ignore, so you're not paying to fix things that don't matter.

Risk rating: HIGH — driven by live financial and customer-data exposure, not by code volume.

2. Critical — fix now (business-ending or legal exposure)

C1 · Live Stripe secret key exposed to the public. Your `sk_live_…` secret key appears in front-end config that is bundled and served to every browser, and again hard-coded in the payments route. A secret key can create charges, issue refunds, and read your customers' payment history. Right now, anyone technical who visits your site can extract it. *Impact:* fraudulent charges, drained balance, chargebacks, Stripe account termination. *Fix:* rotate the key in Stripe immediately; move the new key to a server-side environment variable never sent to the browser; the browser should only ever hold the *publishable* (`pk_`) key. **Do this first, today.** (~1–2 hours)

C2 · SQL injection on login and search. Your login and booking-search queries paste user input straight into the SQL string. The classic consequence: someone types a crafted value into the login box and is admitted **without a password**, or dumps your entire database (including the customer table) through the search box. *Impact:* full account takeover; total data breach; under UK GDPR, a reportable incident with potential ICO penalties. *Fix:* switch every query to parameterised statements (placeholders, not string-building). It's a mechanical change across a handful of queries. (~half a day)

C3 · Admin customer data has no server-side protection. The endpoint returning all customer records — names, contact details, payment info — checks nothing on the server. The code comment says access is "handled on the frontend (admin page is hidden)." A hidden page is not security: the endpoint is directly reachable by URL, so the entire customer list is one request away for anyone who guesses it. *Impact:* complete customer-data breach; GDPR-reportable. *Fix:* enforce authentication and an admin-role check on the server for every admin endpoint. (~half a day)

C4 · Payment amount is set by the customer's browser. Your charge endpoint takes the amount to bill from data sent by the browser. That means the price can be edited before it's sent — a £200 booking can be paid as £2. *Impact:* direct revenue loss; trivially exploitable once known. *Fix:* the server must calculate the price from the booking record, never trust an amount sent by the client. (~2–3 hours)

3. High — fix soon (weeks, not months)

H1 · Stripe webhook accepts unverified events. Your webhook trusts any incoming request without verifying Stripe's signature, so anyone can POST a fake "payment succeeded" and mark bookings paid for free. Add signature verification. (~1–2 hours)

H2 · JWT signing secret is `secret123`, hard-coded. A weak, public secret means tokens can be forged — including admin tokens. Replace with a long random secret in server-side env, and rotate. (~1 hour)

H3 · CORS is open to every origin (`*`). Combined with token-based auth, this widens the blast radius of other issues and enables abuse from any website. Restrict to your own domain(s). (~30 mins)

4. Medium — worth doing (housekeeping & resilience)

M1 · No database backups. A single SQLite file with no backup routine = one bad deploy or disk failure from losing every booking and customer. Add automated daily backups. (~half a day) **M2 · Errors leak stack traces to users.** Your catch blocks return internal error details (and stack traces) to the client, handing attackers a map of your system. Return generic messages; log the detail server-side. (~1–2 hours) **M3 · Dependencies are years out of date with known vulnerabilities.** Express, jsonwebtoken, sqlite3 and Stripe are all pinned to old versions carrying published CVEs. Plan a staged upgrade. (~half a day + testing)

5. Ignore for now (deliberately)

You do **not** need to: rewrite in a "proper" framework, migrate off SQLite (fine at your volume — revisit past ~50k bookings), add a microservices architecture, or hire a full-time developer. None of these move your risk needle right now, and all of them cost money you don't need to spend. Anyone telling you otherwise is selling a rebuild.

6. Prioritised roadmap

Priority	Item	Effort	When
□Do today	C1 rotate exposed Stripe key	1–2 h	Today

Priority	Item	Effort	When
□Do now	C2 SQL injection, C3 admin auth, C4 server-side pricing	~1.5 days	This week
□Do soon	H1 webhook sig, H2 JWT secret, H3 CORS	~half day	Next 2 weeks
□Housekeeping	M1 backups, M2 error handling, M3 dependencies	~1.5 days	This month

Total to get from HIGH risk to comfortable: roughly 3–4 focused days. The Tier-3 Rescue Sprint covers all Critical + High items plus backups and monitoring, done for you, with handover docs.

7. What happens next

We'll spend 30 minutes on a call walking through this together — I'll answer questions in plain English and help you decide what to do yourself versus have done. There is no obligation to buy anything further. If you'd like the fixes implemented, the Audit + Stabilise and Rescue Sprint tiers exist for exactly that; if you'd rather hand this to your own developer, the roadmap above is written so they can act on it directly.

Report signed: Paul Caruana, CTO, Dargavel AI. This audit reflects a point-in-time review of the codebase and configuration provided. It prioritises material risk to your business over exhaustive code critique — by design.