AI-generated code audit checklist for regulated products

Lovable, Cursor, Bolt and Claude Code produce code that runs. In a product that holds patient data, controlled-substance records or client files, "runs" is the easy half. This is the list we go through on a repo before we quote a rescue. The right column holds what we actually found in one of them: **a DEA-compliance platform delivered to its owner as "MVP complete"**, which no new user could log into. That rescue took eight weeks and about 120 hours.

HOW TO USE IT

Tick a box only when you have seen the evidence yourself: the config line, the log entry, the failing test. "The developer said so" is not a tick.

WHAT A BLANK RIGHT CELL MEANS

The point matters and we check it on every audit, but that particular rescue didn't produce a receipt for it. We don't invent findings to fill a column.

SCORING

Fewer than 14 ticks on a product that already has real users is a rescue, and the audit should come before anyone quotes a build price.

CHECK, AND WHAT TO LOOK FOR	WHAT WE FOUND IN A REAL RESCUE
A AUTH & SESSIONS	
☐ **1. A brand-new user can sign up and log in, end to end, on a clean database.** LOOK FOR a registration step that needs something only an existing session has. Test it with an empty users table, not the developer's account.	Passkey registration required an authenticated session. Getting a session required a passkey. **No new user could ever log in.** The fix was a three-scope token (invite, partial, full) so a person can exist half-onboarded.
☐ **2. Tokens expire, carry a scope, and can be revoked server-side.** LOOK FOR one long-lived token used for every state of a user, no logout that invalidates anything, secrets for signing hardcoded in the repo.	
☐ **3. Role checks live in server middleware, and there is a second factor.** LOOK FOR roles enforced only by hiding buttons in the UI. Call an admin endpoint with a regular user's token and see what comes back.	
B DATA HANDLING & PHI/PII	
☐ **4. Every store is encrypted at rest: database, object storage, backups.** LOOK FOR a bucket without server-side encryption, a database instance without the encryption flag, a backup copied somewhere unencrypted.	There was no store to check. The "complete" build had no database, no file storage and no hosting. Infrastructure was set up from zero: encrypted RDS, S3 with AES-256 at rest.
☐ **5. No PHI or PII in email subjects, bodies, push notifications or SMS.** LOOK FOR templates that print a name, a diagnosis, a case number or a drug name. A safe notification says "you have an update, log in".	Email was never configured. The notification code existed, nothing sent it. The rebuild routes everything through SES with no PHI in any body.
☐ **6. Uploads are size-limited, type-checked and served through signed URLs.** LOOK FOR files proxied through the app server, public bucket ACLs, no content-type validation, no limit on a 2 GB upload.	
C AUDIT TRAIL	
☐ **7. Every mutation writes an append-only log entry: who, what, which record, when, from where.** LOOK FOR logging done from the UI instead of the server, or only on some endpoints. Change a record and find the entry.	
☐ **8. The log is readable by an admin without a developer, and exportable.** LOOK FOR a table nobody can query from the product. An inspector will ask for it in a format they can read.	
☐ **9. Nothing in the application can update or delete a log row.** LOOK FOR an ORM model with a normal delete method on the audit table, or the app's database user holding UPDATE/DELETE on it.	
D ERROR HANDLING	
☐ **10. Integrations fail loud. No stub returns success.** LOOK FOR functions named verify, validate or check that return true without calling anything. Grep for "return true" and "TODO".	Voice biometric verification was a stub that **always returned true**. The screen showed a passed identity check for anyone who spoke.
☐ **11. The project compiles with zero type errors, and the build is what gets deployed.** LOOK FOR type checking switched off in config, or a build that "works" only with errors suppressed.	**61 TypeScript errors** in the delivered build. 8 to 10 of them would crash a core feature at runtime, the rest would rot into that.
☐ **12. Every screen's happy path completes against a real backend.** LOOK FOR UI that calls service methods that don't exist yet. Click through each flow to its final state, not its first screen.	Service methods the witness flow called didn't exist. A witness session could start and never complete, and the UI gave no sign.

CHECK, AND WHAT TO LOOK FOR	WHAT WE FOUND IN A REAL RESCUE
E DEPENDENCIES & SECRETS	
☐ **13. No secret in the repo, its history, or a client bundle.** LOOK FOR .env files committed, API keys in frontend code, keys in a Docker image layer. Check git history, not just the working tree.	
☐ **14. Dependencies are pinned, inventoried and free of known CVEs.** LOOK FOR a lockfile that doesn't match the manifest, packages the AI invented that exist under a similar name, an audit command nobody has run.	
F TESTS & CI	
☐ **15. A test suite exists, runs in CI on every push, and blocks a failing deploy.** LOOK FOR a tests folder with one example file, or CI config that only lints. Break something on purpose and see if anything goes red.	**No CI/CD pipeline at all.** Nothing ran the code before a human clicked through it.
☐ **16. No source file mixes markup, business logic and state at scale.** LOOK FOR page files above 500 lines. AI tools grow one file until it works and never split it.	Six page files **averaging 700+ lines**, the largest 1,339, everything inline. Auth was one 762-line route file; it became six modules.
☐ **17. Business logic is separated from route handlers, so it can be tested alone.** LOOK FOR database queries and rule checks written directly inside HTTP handlers. Ask how you would unit-test one rule without spinning up the server.	Logic lived inside the route handlers. A service layer had to be extracted before anything could be tested in isolation.
G DEPLOY & INFRA	
☐ **18. Hosting, database, storage, email and DNS exist, are owned by the client, and are written down.** LOOK FOR a product that has only ever run on the developer's laptop, or cloud accounts in the developer's name.	**None of it existed.** No hosting, no database, no file storage, no email, no CI/CD. "MVP complete" with nowhere to run.
☐ **19. TLS everywhere, including whatever the auth method needs to work at all.** LOOK FOR plain HTTP between services, a self-signed cert in production, auth features that only behave on localhost.	Passkeys (WebAuthn) refuse to run without HTTPS on a real domain, so the login flow couldn't even be tested until a load balancer and a certificate existed.
☐ **20. Backups run on a schedule, errors are tracked, and someone is paged.** LOOK FOR no restore ever attempted, exceptions swallowed with an empty catch, no error tracker wired in.	Nothing to back up and nothing watching. The rebuild added automated RDS backups, Sentry and CloudWatch before the first user was invited.

$2,420
FIXED-PRICE AUDIT

How we use this list. Before we quote a rescue, we run this audit on the repo and hand back every finding with a phased plan. The audit is a fixed $2,420 and is credited in full toward the build if the build goes ahead. On the DEA-compliance platform above, that audit was three days of work, and the rebuild that followed took eight weeks and about 120 hours: auth redesigned around three token scopes, six page files decomposed, a service layer extracted, every integration wired for real, and the whole AWS stack built from nothing. The product now generates its own DEA Form 41 certificates with an append-only audit trail behind each one.

Where the numbers come from. The right column is one engagement, identifying details withheld. The blog post this PDF belongs to adds a second build, a HIPAA case-management platform with three role-separated portals, and the cost of building these controls in versus retrofitting them. Questions: office@oktopeak.com.