Production-readiness checklist

A 15-point audit to run before you open an app written with AI tools to real users

C.E.O. Bee

App: _____ Date: _____

RED If it failed, do not go live. NO CODE NEEDED Can be screened by asking for answers and evidence.

Fix effort: S is a single setting or a small change, M touches several files or screens and needs testing, L means restructuring something fundamental. Not a duration or a price.

	ITEM	HOW TO CHECK	PASS CRITERION
☐	**1 Secrets: API keys, .env and leaks into the client bundle** RED NO CODE NEEDED Fix effort: S	Search the repository history and the built bundle for live keys (for example sk_live_). Check whether a string starting with eyJ decodes to "service_role". In the browser, read the requests in the Network tab. **Without reading code:** Was a key ever shared, and if so, was it changed? Evidence: the last rotation date of the keys.	No live key in the history or the bundle. Every key that was ever exposed has been revoked and replaced; deleting it from the file is not enough, the key stays in the repository history.
☐	**2 Authentication and sessions** RED Fix effort: S–M	Is a ready-made identity service or a mature library used? If written by hand, are passwords stored with a slow hashing algorithm (argon2, bcrypt)? Is the session token in a cookie (HttpOnly, Secure, SameSite) or in localStorage? Try to use a reset link twice and send repeated wrong passwords.	Passwords are hashed; the session token is in an HttpOnly, Secure cookie (if it is in localStorage, the reason is written down and item 4 has passed); reset links are single-use and expire; repeated failures are throttled.
☐	**3 Authorization: object level** RED Fix effort: M; L if there is no owner of the record	Use two accounts: open a record as A, replace its ID with the ID of a record that belongs to B and resend the request. Call admin endpoints with a normal user's session. If the app talks to the database straight from the browser (as with Supabase), check that row level security (RLS) is on.	B's data never comes back to A by any route (403 or 404); admin endpoints check the role on the server, on every request; every table has RLS enabled and a policy defined, and no view exposes a table (or it is defined with security_invoker).
☐	**4 Input validation and injection** RED Fix effort: S–M	Search for queries built by concatenating strings and for places that insert raw HTML. Type ' and <img src=x onerror=alert(1)> into forms and watch what happens. Confirm that the type, size and destination of uploaded files are checked on the server.	All database access is parameterized or goes through a trusted ORM; user text is displayed as plain text, not as HTML; uploads have a type and size limit enforced on the server.
☐	**5 Database: schema, constraints, indexes, migrations** Fix effort: M; L if the data model is wrong	Can an empty database be built from the migration files in the repository alone? Check foreign keys, uniqueness and NOT NULL constraints on the important tables, and indexes on the columns you filter on. Compare the production schema with the repository's; are sample data or test users still in production?	An empty database is built from the repository's migrations and the app starts; related records are protected by constraints; the production schema matches the repository.
☐	**6 Backups and a restore test** RED NO CODE NEEDED Fix effort: S–M	Where is the backup, how often is it taken, and in whose account? Restore the latest backup into a new, empty database; check the row count and the most recent record; note how long it took. Are uploaded files backed up too? **Without reading code:** When was the last backup restored? Evidence: the date and duration of the restore.	The date and duration of the last restore test are written down; the restored data is complete and readable; the backup lives somewhere separate from the account the app runs in.
☐	**7 Environments and deployment** NO CODE NEEDED Fix effort: M	Are there separate databases and keys for development and production? Are preview builds connected to production data? Can you return to the previous version in one step (try it once)? Whose account holds the domain, the DNS records and the TLS certificate? **Without reading code:** Can a test change touch live data? Evidence: a separate test environment and a tried rollback.	A test change cannot touch production data; a rollback to the previous release has been tried and worked; configuration lives in environment variables.
☐	**8 Error handling, logging and monitoring** Fix effort: S–M	Break something on purpose: cut the database connection, send invalid input. What does the user see, and how do you find out? Is there error tracking and an uptime check, and does the notification reach a person? Logs should not contain passwords, tokens, keys, connection strings or card data.	The user gets a clear message and you get the details; a critical error reaches a person as a notification; there are no secrets in the logs.

	ITEM	HOW TO CHECK	PASS CRITERION
☐	**9 Dependencies: unused, invented, vulnerable, licensed** Fix effort: S–M	Scan production dependencies for known vulnerabilities (npm audit --omit=dev in npm projects). Check that each package really exists and that its publisher and age look reasonable; remove what is unused. Is the lock file in the repository? Are the licenses fine for commercial use?	No high or critical vulnerability in production dependencies, or a written, reasoned exception; every package verified; the lock file is committed.
☐	**10 Tests: critical flows and meaningful tests** Fix effort: M	List the critical flows (sign-up, login, payment, writing data): does each have a test? Break a rule on purpose; if no test fails, the tests are not doing their job. Do the tests run automatically on every change?	The critical flows have tests that fail when the rule is broken, and they run automatically on every change.
☐	**11 Performance and scale** Fix effort: M	Open the list and search pages with hundreds and thousands of test records and measure them (the browser's Network tab is enough). Look at how many queries one request runs. Do list endpoints paginate, are images resized, do calls to outside services have timeouts?	With a realistic data volume the main screens open within a target time you wrote down beforehand; lists are paginated; outbound calls have timeouts.
☐	**12 Payments and sensitive flows** RED Fix effort: M; L if built wrongly from the start	Send an unsigned, forged request to the webhook endpoint (it should be rejected). Send the same event twice (only one order should result). Change the amount in a request body (the server should calculate the amount itself). Does an order become "paid" because of a verified notification, or because the user landed on a redirect page?	A forged, repeated or altered request creates no record; the amount is computed on the server; status changes only on a verified notification.
☐	**13 Personal data and privacy law** RED NO CODE NEEDED Fix effort: M–L	Build a data inventory: which personal data, where is it stored, who can read it, which third parties receive it, how long is it kept, how is it deleted? Does user data go into prompts? Delete a test user and see what is left in the database, the logs and the backups. **Without reading code:** Which personal data, where, with whom? Evidence: the data inventory table.	For every piece of personal data, its location, access, third parties and deletion route are written down; a deletion request can be fulfilled; you know what data goes to the AI provider. Not legal advice; ask a lawyer for a legal assessment.
☐	**14 Cost and abuse** NO CODE NEEDED Fix effort: S–M	Confirm that paid-service keys never reach the browser (item 1). Send repeated requests to one endpoint of your own app; at some point you should see 429 (if the endpoint calls a paid service, try a few requests first or use a test environment). Is a monthly spending cap and budget alert set in the provider console? Is there bot protection on the sign-up form? **Without reading code:** Is there a spending cap? Evidence: a screenshot from the provider's dashboard.	Repeated requests are throttled; a spending cap and alert are in place; paid keys exist only on the server.
☐	**15 Ownership and maintainability** NO CODE NEEDED Fix effort: S–L	Fill in this list: code repository, hosting, domain and DNS, database, email and SMS, payment account, store accounts, AI provider account. Who owns each and holds the admin account, is it tied to a company email? Can the code be exported to a repository, and does it run on a clean computer by following the README? **Without reading code:** Who is the admin of every account? Evidence: the account list and the admin emails.	All accounts are in the company's or the product owner's name; the code is exported and in a repository; an engineer can run the project on day one without the AI tool.

Verdict: red, yellow, green

Red	**One of items 1, 2, 3, 4, 6, 12 or 13 has failed**	Do not go live. User data or money is involved; fix that item first. A red item cannot be waived with a reasoned exception (apart from items that are "not applicable").
Yellow	**The red items passed; one or more of the others failed**	Launch to a limited, friendly group of users; write a fix plan with an owner and a date for every remaining item.
Green	**All items passed (a reasoned exception may be written down on items that are not red)**	Go live; repeat the audit after every large change.

The thresholds are the article's recommendation, not a standard. If your app has no such feature (for example, payments), that item counts as "not applicable."

Describe your project: ceobee.dev

C.E.O. Bee · ceobee.dev

Source article: Is your vibe-coded app production-ready? A 15-point audit · ceobee.dev/engineering/is-your-vibe-coded-app-production-ready/ (commands and reasoning are in the article)