PAPA Tech Solutions

FREE RESOURCE

Vibe Coding Security Checklist

30 AI audit prompts that check your vibe-coded app and website for security, quality, and launch gaps before real users do.

Papa Tech Solutions

Version 2.0 · Updated September 2026

papatechsolutions.com

Scan to open online

How to use this checklist

Paste one audit prompt at a time. Use a mode that answers without editing files. Read the report, choose which finding IDs to fix, then paste the fix prompt in a mode that can edit. Re-run the same audit prompt afterwards and read that report too.

Read this first: honest limits

- An AI review is a to-do list to verify, not a certificate. It can miss things and it can be wrong.
- Run these inside your coding tool on your own repo. Never paste production secrets or real customer data into a chat window.
- If a prompt finds a live key or credential, rotate it immediately — deleting it isn't enough.
- For apps that handle payments, health data, or sensitive information at scale, add a human security review. Legal pages produced by prompt 28 are drafts for a lawyer to check.

Which prompts first?

IF YOU ARE SHIPPING	RUN THESE, IN ORDER
Web app with logins and user data	`1 → 2 → 3 → 4 → 8 → 11`
Anything that takes payments	`9 → 4 → 3 → 1 → 11, then 24`
Android or iOS app	`20 → 1 → 14 → 15 → 24`
Marketing site or landing page	`25 → 26 → 27 → 28 → 29 → 30`
Already live and growing	`13 → 18 → 23 → 24 → 19`

How to run these prompts

Every prompt audits first and changes nothing. You choose what to fix.

1. Switch to a mode that does not edit files. In Cursor that is Ask or Plan. In Claude Code that is Plan mode.
2. Paste the audit prompt. Wait for the report. It must stop and ask which IDs to fix.
3. Read the report. Keep the IDs you agree with.
4. Switch to a mode that can edit. Paste the fix prompt and the IDs you chose.
5. Switch back to the read-only mode and paste the same audit prompt again. Confirm those IDs are gone.

Which mode to use

TOOL	AUDIT IN	FIX IN
Cursor	Ask mode or Plan mode	Agent mode
Claude Code	Plan mode (Shift+Tab cycles to it)	Normal mode, which can edit
Any other tool	Chat, Discuss, or Plan mode, whichever answers without editing files. If the tool has no such mode, the prompt itself forbids edits	The mode that is allowed to edit files

Fix prompt

Paste this after an audit, with the finding IDs you chose.

```
MODE: FIX. Use your tool's Agent (editing) mode. Fix only these finding IDs from your last audit report: <paste IDs>.
For each ID, one at a time:
  1. Restate the finding and the file(s) you will change.
  2. Make the smallest change that fixes it. Do not refactor, rename, or reformat unrelated code.
  3. If the fix needs a secret rotated, a dashboard setting changed, or data migrated, do not guess. Give me the exact manual step instead.
  4. Show what changed and how you verified it (test, build, or re-running that check).
Finish with: fixed, not fixed and why, and manual steps left for me.
```

Contents

SECTION A Security & Data Safety

SECTION B Quality, Speed & Resilience

SECTION C Website Launch Readiness

SECTION A

Security & Data Safety

Prompts 1–12

01 Secrets & Credential Leak Sweep

Why it matters A private key left in a repo or a shipped app can be copied and abused. Public client keys are a different thing and should not be treated as leaks.

Modeled on Secret scanners such as Gitleaks and TruffleHog.

AUDIT PROMPT

Paste in Ask or Plan mode

```
MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:
- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.
```

Secrets & Credential Leak Sweep: what to check

Look for secrets that can actually be abused. Do not flag values that are public by design: a Firebase web apiKey, a Supabase anon key, or a Stripe publishable key (pk_live_ or pk_test_). Those are meant to ship in client code. The dangerous values are private keys, service-account JSON, a Supabase service_role key, Stripe secret or restricted keys (sk_ or rk_), database passwords, and webhook secrets.

1. Search the whole repo, including config, comments, seed scripts, fixtures, CI files, and exported Postman or Insomnia collections, for hardcoded tokens, passwords, private keys, webhook URLs that embed a secret, and database connection strings.
2. For each hit, name the service, say whether it looks live or like a placeholder (your-api-key, changeme, example), and name the environment variable it should move to. Do not print the full secret. Show at most the first 4 characters.
3. In Next.js, Vite, and Create React App, flag a real secret stored in a client-exposed variable (NEXT_PUBLIC_, VITE_, REACT_APP_). A public Firebase web config in NEXT_PUBLIC_ is fine. A service_role key or an sk_ key there is Critical.
4. On Android, check Kotlin or Java source, git-tracked gradle.properties, and release BuildConfig fields. On iOS, check Info.plist and a committed GoogleService-Info.plist for anything beyond the public client config.
5. Confirm .env, .env.local, *.jks, *.keystore, service-account JSON, and a real google-services.json or GoogleService-Info.plist are gitignored. Then check whether any are already tracked with git ls-files.
6. If a secret was committed and later deleted, say so. Deleting the line does not remove it from history. The credential must be rotated. Name the commit if git log -p -S shows it.

7. Check CI workflow files for secrets printed in a run step, or secrets baked into a client build artifact.
8. Check that .env.example lists every required name with a placeholder, and that the example file itself has no live value.

Evidence rules:

- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P01, for example P01-1, P01-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

A Firebase service-account JSON committed, then deleted in a later commit, still recoverable with git log. The key still needs rotating.

Copy this prompt online →

SECTION A · SECURITY & DATA SAFETY

02 Test Data & Seed Cleanup

Why it matters A demo admin, a weak password, or an OTP bypass left in the login path can ship as a back door. Placeholder copy can ship as if it were real.

Modeled on The staging-to-production data checks release teams run before cutover.

AUDIT PROMPT Paste in Ask or Plan mode

MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:

- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.

Test Data & Seed Cleanup: what to check

Find everything that exists only for development, and say whether it can still run in production. Do not delete anything in this pass.

1. List every seed script, fixture, mock-data file, and "create default user" routine. Show the file and how it is invoked (npm script, app startup, migration, CI job). Flag any path that runs when the app boots in production.
2. Find demo accounts in code, seed files, and schema comments: admin@test.com, test@example.com, demo users, and passwords such as password, 123456, or admin. Note any with an elevated role.
3. Find placeholder copy shown in the UI: lorem ipsum, "Test User", fake prices, sample products, stock photos named as placeholders, TODO copy, and dummy phone numbers or emails.
4. Find test-mode payment keys, sandbox API hosts, and staging webhook URLs that are selected by a hardcoded branch instead of an environment check.
5. Find bypass switches: skip OTP, skip payment, skip auth, a magic code, or a feature flag that defaults to on. Say whether the default is off in production.
6. Find routes or pages named test, debug, seed, or preview that are still registered in the production router.
7. Say how you would confirm the production database is a separate instance and was not loaded from a dev dump. If you cannot see the host, mark it Needs manual check.
8. Return two lists: safe to remove, and needs my decision before anyone deletes it.

Evidence rules:

- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

```
Report:
  - Summary: count of findings by severity.
  - Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P02, for example P02-1, P02-2.)
  - Checked and fine: what you verified is already OK.
  - Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.
```

EXAMPLE FINDING

An OTP bypass of 123456 left in the login handler, still reachable when NODE_ENV is production.

Copy this prompt online →

SECTION A · SECURITY & DATA SAFETY

03 Database Rules & Row-Level Access

Why it matters If the database is called from the browser, the rules are the security of the app. A default or test-mode rule can leave every row readable.

Modeled on Supabase Row Level Security and Firebase Security Rules.

AUDIT PROMPT

Paste in Ask or Plan mode

```
MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:
  - Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
  - If a check below does not apply to this stack, write "Not applicable" and why.
  - If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.
```

Database Rules & Row-Level Access: what to check

```
Audit database access as if anyone can copy the public client keys and call the database directly. Rules in the repo can differ from rules deployed in the dashboard. Mark dashboard-only state as Needs manual check.

1. List every table or collection. For each, say who can read, insert, update, and delete, and whether that is enforced by database rules (RLS, Firestore or Storage rules, role grants) or only by application code.
2. Flag RLS that is off, a policy of USING (true) or WITH CHECK (true), and Firestore rules of the form allow read, write: if true. Also flag leftover test-mode rules.
3. For user-owned rows, confirm the rule compares the row's user id to the authenticated user id on insert, update, and delete, not only on read.
```

4. Confirm a client write cannot set role, is_admin, credits, price, or owner id. Those columns should be omitted from the policy's allowed columns, or set only by a trigger or the server.
5. Confirm the Supabase service_role key, a Firebase Admin SDK credential, or any connection that bypasses RLS is used only on the server. Searching the client bundle and NEXT_PUBLIC_ or VITE_ names is enough for the repo. A deployed dashboard setting is Needs manual check.
6. For storage buckets, say which are public, which are private, and whether a user can list or read another user's file by guessing the path.
7. Describe a read-only test you want me to run with a normal user's token against another user's id. Do not run a write against a shared or production database.
8. If SQL policies live in migrations, cite the migration file. If they are only in the hosted console, say the repo cannot prove what is deployed.

Evidence rules:

- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P03, for example P03-1, P03-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

A profiles table with RLS disabled, so a logged-out visitor can read every email if the anon key is used.

Copy this prompt online →

SECTION A · SECURITY & DATA SAFETY

04 Auth, Roles & Admin Route Lockdown

Why it matters Hiding an admin link is not security. If the route or API answers anyone who requests the URL, it is open.

Modeled on OWASP ASVS access-control and authentication requirements.

AUDIT PROMPT Paste in Ask or Plan mode

MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:

- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.

Auth, Roles & Admin Route Lockdown: what to check

Audit authentication, roles, and admin protection across the UI, the API, and the database. Do not change accounts or sessions.

1. List routes, pages, and API handlers. Mark each public, signed-in, or admin-only. Flag a sensitive handler whose only check is a client redirect or a hidden button.
2. For every admin or privileged handler, show the server-side role check and the file. A menu that is not rendered does not count.
3. Check IDOR: any handler that takes a user, order, or document id must compare it to the session user or an admin role before returning or changing the row.
4. Check whether a user can grant themselves a role through a profile update, a signup field, a JWT claim they can set, or a query parameter.
5. Check password hashing (bcrypt, argon2, or scrypt), session or JWT expiry, and whether logout invalidates the server session or only deletes a client cookie.
6. Check password-reset and email-verification tokens: they should be random, single-use, and short-lived (minutes to about an hour, not days), and bound to one user. Errors should not reveal whether the email is registered.
7. Check OAuth callback URLs for an open redirect and for a missing or unused state parameter.
8. Find default, seeded, or hardcoded admin accounts. Do not print passwords. Say where they are set and whether production startup would still create them.

Evidence rules:

- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

```
Report:
  - Summary: count of findings by severity.
  - Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P04, for example P04-1, P04-2.)
  - Checked and fine: what you verified is already OK.
  - Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.
```

EXAMPLE FINDING

GET /api/admin/users returns every user to any logged-in user because only the menu hides the link.

Copy this prompt online →

SECTION A · SECURITY & DATA SAFETY

05 Input Validation & Injection Defense

Why it matters Form fields, URL parameters, headers, and file names are attacker-controlled. Most injection bugs start with one that was never checked on the server.

Modeled on OWASP guidance on input validation and injection.

AUDIT PROMPT Paste in Ask or Plan mode

```
MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:
  - Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
  - If a check below does not apply to this stack, write "Not applicable" and why.
  - If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.
```

Input Validation & Injection Defense: what to check

```
Review untrusted input everywhere data enters from outside. Client-side checks are a convenience, not the control.

1. List input sources: form fields, JSON bodies, query and path parameters, headers, cookies, file names, webhook payloads, and fields copied from third-party API responses.
2. For each source that changes state or is used in a query, say whether the server checks type, length, format, allowed values, and required fields. Name the file. If the only check is in the browser, mark it High.
3. Find SQL or NoSQL queries built by string concatenation, template strings, or by passing a request object straight into a query operator. Parameterized queries and the ORM's bound API are the safe pattern.
```

4. Find user input rendered as HTML, passed to dangerouslySetInnerHTML or innerHTML, or placed in a URL or redirect. Say whether it is escaped or sanitized.
5. Check other injection paths: shell commands built from input, file paths that can contain .., template engines, and redirects to a user-supplied URL.
6. Check business numbers: negative quantities, huge numbers, decimals where an integer is required, and dates far in the past or future. Say whether the server rejects them.
7. Say whether validation is one shared schema (for example Zod or a similar library) or a different ad-hoc check in each handler.
8. Do not send exploit payloads at a live server. Describe the input that would prove the bug, and cite the line that would accept it.

Evidence rules:

- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P05, for example P05-1, P05-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

A search box whose text is concatenated into a SQL string instead of a bound parameter.

Copy this prompt online →

SECTION A · SECURITY & DATA SAFETY

06 File Upload Safety Test

Why it matters An upload can store malware, fill storage, or overwrite a file if the server trusts the browser's file name and type.

Modeled on The OWASP File Upload Cheat Sheet.

AUDIT PROMPT Paste in Ask or Plan mode

```
MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:
- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.

File Upload Safety Test: what to check

Review every upload path as if a hostile user will use it. Do not upload files to a production bucket in this pass.

1. List every upload endpoint or component, where bytes are stored (disk, object storage, database), and who can read them afterwards.
2. Say whether the server checks type from content or a signature, not only the extension or the browser-supplied MIME type, and whether the allowlist is explicit.
3. Say whether size, file count, and image dimensions are limited on the server, and whether the upload is streamed or buffered so one huge body can exhaust memory.
4. Say whether the stored name is a server-generated id. A client-supplied name can traverse paths or overwrite an existing object.
5. Say whether files are stored outside the web root or in a separate bucket, are never executed, and are served with a content type that does not sniff as HTML.
6. Say whether one user can fetch another's file by changing an id, and whether private files use a signed URL that expires.
7. Note SVG uploads (they can contain script), EXIF location data on photos, and image or PDF libraries pinned to an old version.
8. Say whether there is a per-user upload rate limit and a way to delete orphaned files. If you cannot see the limiter's store, mark it Needs manual check.
9. Write a test plan I can run on a staging bucket only: a renamed executable, an oversized file, an SVG with a script tag, and a name containing ../. Say what the code would do with each.

Evidence rules:
- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.
```

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P06, for example P06-1, P06-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

Uploads accepted whenever the name ends in .jpg, so a renamed script is stored and later served.

Copy this prompt online →

SECTION A · SECURITY & DATA SAFETY

07 Rate Limits & Abuse Protection

Why it matters Without limits, one client can guess passwords, spam users, or run up an API bill before anyone notices.

Modeled on OWASP API Security risks for unrestricted resource use and abuse of business flows.

AUDIT PROMPT

Paste in Ask or Plan mode

MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:

- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.

Rate Limits & Abuse Protection: what to check

Review rate limits and abuse controls. Do not send a flood of requests at a live host.

1. List handlers and classify them: authentication (login, signup, reset, OTP or SMS or email), expensive work (search, export, AI calls, uploads), and anything that spends money or sends a message.
2. For auth and message-sending handlers, say whether there is a small limit per IP and per account, with lockout or delay. Cite the file. If there is no limit, say so.
3. Say whether other API handlers return 429 with a Retry-After header, or fail with a generic 500, or have no limit at all.

4. Say where the limit is enforced (app process, shared store, or edge). If the limit trusts X-Forwarded-For from the client, mark it bypassable unless the platform overwrites that header and the app reads only the platform's value.
5. Check business abuse in the code: signup with no cap, promo or referral codes reused without a redemption record, free trials restarted by signing up again, coupon stacking, and list endpoints with no page size.
6. Say whether CAPTCHA or a similar challenge exists on signup or reset, and whether it is required only where abuse is plausible.
7. Say whether body size and request timeouts are set, so one slow or huge request cannot hold a worker.
8. Say whether limiter state is in a shared store (Redis, database, or the platform) or only in memory. In-memory state does not survive a second server process.
9. Do not recommend a specific third-party product. Describe the limit as numbers I can change: which route, which key (IP and account), and which store.

Evidence rules:

- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P07, for example P07-1, P07-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

A password-reset handler that sends an email on every request, with no limit.

Copy this prompt online →

SECTION A · SECURITY & DATA SAFETY

08 Personal Data Flow & Storage Audit

Why it matters A console.log of a user object can land in a host's logs, where anyone with dashboard access can read it.

Modeled on Data-flow reviews that trace personal data from collection to storage to third parties.

AUDIT PROMPT

Paste in Ask or Plan mode

```
MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:
- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.

Personal Data Flow & Storage Audit: what to check

Map how personal data moves through this app. Do not print real user values you find in fixtures. Describe the field and the path.

1. List where the app collects data: signup, profile, payment forms, device or location fields, and uploads.
2. For each field, trace the next hop you can see in code: which table, which third-party request, which log line.
3. Search console.log, logger calls, and error handlers for emails, phone numbers, tokens, passwords, or a whole user object. Quote the file and line, not the value.
4. List third-party SDKs (analytics, crash reports, email, AI) and which user fields are passed in. Flag a payload that sends more than that call needs.
5. Confirm passwords are hashed with bcrypt, argon2, or scrypt before storage, and are not returned by an API or written to a log. If you only see a comment that says "hashed" and not the call, mark it Needs manual check.
6. Check cookies for HttpOnly, Secure, and SameSite. Flag personal data written to localStorage, which any script on the page can read.
7. Check API responses for over-fetching: a handler that returns columns the screen does not use, especially email, phone, or tokens.
8. Say whether the code has a path for a user to request account or data deletion. If it does not, list it as a gap even if you are not giving legal advice.
9. Finish with a short map: collected, stored, sent externally, and what a later fix pass should change. Do not change it now.

Evidence rules:
- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
```

- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P08, for example P08-1, P08-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

A support-ticket call that sends the full profile when the ticket only needed a name and a message.

Copy this prompt online →

SECTION A · SECURITY & DATA SAFETY

09 Payments & Webhooks Deep Check

Why it matters Payment bugs spend real money: access without paying, a double charge, or a paid order that never unlocks.

Modeled on Stripe and Razorpay webhook guidance: signature checks, idempotency, and retries.

AUDIT PROMPT

Paste in Ask or Plan mode

MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:

- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.

Payments & Webhooks Deep Check: what to check

Trace payment from checkout to fulfillment to refund. Do not send test charges unless I give you a sandbox key and ask you to.

1. Trace the flow in code: what the client sends, what the server computes, what the provider is asked to charge, and what grants access.

2. Confirm the server sets price, currency, quantity limits, tax, and discounts. The client should send product ids, not amounts. Cite the handler.
3. Confirm fulfillment runs from a verified webhook or a server-side status check. Flag fulfillment that runs because the browser hit a success URL or posted payment succeeded.
4. Webhook signatures must be checked on the raw body, before JSON parsing changes bytes. Stripe sends Stripe-Signature and a timestamp tolerance. Razorpay sends X-Razorpay-Signature as HMAC-SHA256 of the raw body. Say which provider this repo uses and whether the raw body is preserved.
5. Say whether the handler is idempotent (the same event id must not grant twice), responds quickly, and stores processed event ids. Out-of-order events should not grant access early.
6. Say what the code does on a declined card, an abandoned checkout, a 3-D Secure challenge, a timeout, a charge that succeeded but fulfillment failed, a refund, and a chargeback. If a branch is missing, say so.
7. Confirm test and live secrets are different env vars, and live secrets are not in the client bundle.
8. Say whether there is a way to compare provider payments to your orders and find paid-but-not-fulfilled or fulfilled-but-not-paid rows. If it is only a dashboard report, mark Needs manual check.
9. Confirm card numbers do not touch your server or logs. Hosted fields or a hosted checkout should collect them. Search logs and request types for pan, cvc, or card number fields.

Evidence rules:

- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P09, for example P09-1, P09-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

An order marked paid when the browser reaches /success, even if the payment failed.

Copy this prompt online →

SECTION A · SECURITY & DATA SAFETY

10 Debug Logs & Error Leakage Cleanup

Why it matters A stack trace or a logged token shown to the wrong person reveals paths, queries, and sometimes keys.

Modeled on OWASP logging and error-handling guidance.

AUDIT PROMPT Paste in Ask or Plan mode

```
MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:
- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.

Debug Logs & Error Leakage Cleanup: what to check

Find what this app leaks through logs and error responses. Do not print secret values. Cite the file.

1. List console.log, print, and debug calls that would still run in production. Note which ones log objects rather than a short message.
2. Flag any log line that can include a password, token, API key, full payment or personal data, or an entire request or response body.
3. Check error responses sent to the client for stack traces, raw database errors, file paths, framework versions, or internal ids. A generic message plus a correlation id is the safe shape.
4. Confirm the detailed stack is logged only on the server and tied to that correlation id. If there is no correlation id, say so.
5. Find debug switches that default to on: debug=true, verbose error pages, source maps served to browsers, GraphQL introspection, and an open Swagger or playground route.
6. Check client error boundaries. They should show a friendly message, not the raw error object.
7. Say where logs are stored and who can read them, if the code or config shows it. Retention is Needs manual check if it is only a host setting.
8. Separate findings into "shown to users" and "only in server logs". Both matter. Server logs that contain secrets are still a leak.

Evidence rules:
- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.
```

Report:

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P10, for example P10-1, P10-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

A 500 response that includes the SQL text and table names.

Copy this prompt online →

SECTION A · SECURITY & DATA SAFETY

11 Pre-Deploy Production Readiness Check

Why it matters Many launch incidents are missing safeguards a short checklist would have caught before the first real user.

Modeled on Production-readiness checks engineering teams run before a release.

AUDIT PROMPT

Paste in Ask or Plan mode

MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:

- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.

Pre-Deploy Production Readiness Check: what to check

Walk these checks against the repo and say pass, fail, or needs manual check. Do not change config in this pass.

1. List required environment variables and whether startup fails clearly when one is missing, or continues and breaks later.
2. Confirm debug flags default to off, and that routes named test, debug, or seed are not registered in the production build. Show the router file.
3. Look for security headers: X-Content-Type-Options, X-Frame-Options or CSP frame-ancestors, Strict-Transport-Security, and a Content-Security-Policy. Say where they are set (app or host config).
4. Check CORS. Flag Access-Control-Allow-Origin: * combined with credentials. A public API with no cookies can allow a wildcard. An API that uses cookies should not.

5. Say whether HTTP redirects to HTTPS and whether cookies are marked Secure. Host-level redirects may be Needs manual check.
6. Say whether the database connection string uses TLS and is not a default password. Reachability of the port is Needs manual check if you cannot see the network rules.
7. Say whether the repo or docs mention backups and a tested restore. If they do not, mark it as a gap I must confirm in the host console. Do not claim a backup exists because a vendor offers one.
8. Confirm a lockfile is present so installs are reproducible. Name the file.
9. Say whether there is a health-check route and where uptime would be monitored. Monitoring outside the repo is Needs manual check.
10. Say whether a previous version can be redeployed, based on the host config in the repo (for example Firebase Hosting releases or a platform rollback command). If it is only a console button, mark Needs manual check.

Evidence rules:

- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P11, for example P11-1, P11-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

A /debug/reset-db route still registered in the production build.

Copy this prompt online →

SECTION A · SECURITY & DATA SAFETY

12 Try to Break Your App: Attacker's-Eye Review

Why it matters Many bugs are not obvious when you read code top to bottom. They show up when you follow an attack path or a clumsy user path.

Modeled on Adversarial reviews that follow attack paths and broken flows, not only style.

AUDIT PROMPT Paste in Ask or Plan mode

```
MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:
- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.

Try to Break Your App: Attacker's-Eye Review: what to check

Think like an attacker and like a clumsy user. Review only this repo, or a staging app I own and have told you to test. Do not probe a system you cannot see permission for. Do not exploit a live production host.

1. ID tampering: for handlers that take a user, order, or document id, say whether the code checks ownership. Cite the check or its absence.
2. Auth bypass: which handlers have no auth check, how expired or malformed tokens are rejected, and whether a seeded admin account is still created.
3. Privilege: if roles exist, say whether a normal user can hit an admin path by URL or by editing their role in the client. The check must be on the server.
4. Abuse: signup, messaging, uploads, and promo or referral code reuse. Point at the missing limit or the missing redemption record.
5. Injection: where script or SQL metacharacters would be stored or concatenated. Cite the line. Do not fire those payloads at production.
6. Accidental exposure: an admin page with no auth, an error page that prints env values, a public .env or .git path, or API docs left open.
7. Money or credits: can the client send a negative amount, stack a discount without a cap, or restart a trial by signing up again? Cite the handler.
8. Chaining: name two lower-severity issues that together become serious, if you see a pair. If you do not, say so.
9. Broken flows in the UI code: double submit, refresh during payment, two tabs, session expiry mid-action, and a save that fails offline. Say what the code does, not what you hope it does.
10. Order findings with data theft and account takeover first, then abuse and logic bugs. For each: what the person does, the realistic damage, and the fix. Then stop.

Evidence rules:
- Every finding cites a file path and line number, or the exact command output used.
```

- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P12, for example P12-1, P12-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

A delete-account handler that trusts a user id in the body instead of the session.

Copy this prompt online →

SECTION B

Quality, Speed & Resilience

Prompts 13–24

13 Performance & Load Simulation

Why it matters An app that feels instant in one browser tab can stall when many people use it at once.

Modeled on Grafana k6, a load-testing tool.

AUDIT PROMPT

Paste in Ask or Plan mode

```
MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:
- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.
```

Performance & Load Simulation: what to check

```
Review this code as if it will be hit by many concurrent users. Do not run a load test against production. You may describe a k6 script. Do not start one unless I give you a staging URL.

1. Find handlers that read or write a table with no pagination and no limit. Cite the query.
2. Find blocking work inside a request: a slow external call, a large file read, or heavy CPU, with no timeout and no queue.
3. Find database access that opens a new connection per request instead of using a pool. Say what the driver config shows.
4. Find handlers called often with the same inputs and no cache. Do not invent a cache product. Say what could be cached and for how long.
5. Find loops that query or call an API once per item (N+1) where one batched query would do.
6. Find request-specific state stored in a module-level variable or memory. That breaks when a second instance serves the next request.
7. For each issue, name the file, why it gets worse as concurrency grows, and a rough guess of when it would hurt. Label the guess as a guess.
8. List the read-only queries or EXPLAIN plans that would confirm the worst items. Do not run EXPLAIN ANALYZE on a statement that writes.
```

```
Evidence rules:
- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
```

- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P13, for example P13-1, P13-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

An activity feed that runs one database query per row instead of one query for the page.

Copy this prompt online →

SECTION B · QUALITY, SPEED & RESILIENCE

14 Slow Network, Offline & API Failure Handling

Why it matters Apps are often tried on fast Wi-Fi. Users are on crowded mobile networks, and third-party APIs fail more often than a demo suggests.

Modeled on Browser network throttling and Lighthouse's throttled mobile audits.

AUDIT PROMPT

Paste in Ask or Plan mode

MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:

- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.

Slow Network, Offline & API Failure Handling: what to check

Review behavior when the network is slow, drops, or an API fails. Do not need a device. Read the client and server code.

1. List network calls and say whether each has a timeout, an error path, and a user-visible failure. A call with no timeout can hang.
2. Check retries. They should apply to safe, idempotent calls, use backoff, and stop after a maximum. Retrying a payment or a create on every failure can duplicate it.

3. Check screens for loading, empty, error, and retry. Flag a spinner with no timeout and a blank region when the request fails.
4. Check submit, upload, payment, and save when the connection drops: is data lost, can the action run twice, and is the user told?
5. If a payment, maps, email, or AI provider errors, does the app show a degraded path or does the whole page throw?
6. Note large bundles, unoptimized images, and request waterfalls that would dominate on a slow mobile link. Cite the component, not a guessed millisecond score.
7. For a PWA or mobile client, say what is cached offline and what the UI says when offline. If there is no service worker, say so.
8. Write a manual test plan using the browser's Slow 4G and Offline modes, and by blocking one API host. Say what I should see at each step. Do not claim you ran it unless you did.

Evidence rules:

- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P14, for example P14-1, P14-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

A Pay button with no loading state, so a slow tap fires three times.

Copy this prompt online →

SECTION B · QUALITY, SPEED & RESILIENCE

15 Mobile Layout & Responsive Testing

Why it matters A layout that looks finished on a laptop can be unusable on a phone, which is where many first visits happen.

Modeled on Browser device emulation and Lighthouse mobile audits.

AUDIT PROMPT Paste in Ask or Plan mode

MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:

- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.

Mobile Layout & Responsive Testing: what to check

Review layouts for phones and tablets from the code and styles. If you can open the app, also look at 320, 360, 390, 768, 1024, and 1440. If you cannot, say which screens I should check.

1. Confirm a viewport meta tag exists and that layout uses flexible widths, not a fixed page width that forces sideways scrolling.
2. Flag horizontal overflow, overlapping elements, cut-off text, and images wider than their container. Name the component.
3. Check tap targets. WCAG 2.2 AA asks for at least 24 by 24 CSS pixels. 44 by 44 is a stricter AAA target and Apple's Human Interface Guidelines recommendation. Flag controls smaller than 24px, and hover-only actions with no tap equivalent.
4. Check mobile forms: input types (email, tel, number), autocomplete attributes, labels that stay visible, and whether a sticky bar can cover the submit button when the keyboard is open.
5. Check fixed headers and bottom bars against notches and safe-area insets. A bar that covers the last button is a bug.
6. Check base font size and line length. Text should be readable without pinch zoom, and not only inside an image.
7. Check tables, code blocks, and wide content. They should scroll inside their own box, not stretch the page.
8. Say what to recheck in landscape and at 200% browser zoom. If you did not render the page, mark those Needs manual check.

Evidence rules:

- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P15, for example P15-1, P15-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

A pricing table that scrolls sideways on a 360px screen and hides the buy button.

Copy this prompt online →

SECTION B · QUALITY, SPEED & RESILIENCE

16 Accessibility Pass

Why it matters Accessibility failures exclude people. Some laws and app stores also treat them as compliance issues.

Modeled on axe-core, Deque's open-source accessibility engine.

AUDIT PROMPT Paste in Ask or Plan mode

MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:

- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.

Accessibility Pass: what to check

Audit the frontend for issues a keyboard-only or screen-reader user would hit. Do not claim WCAG conformance. List concrete issues.

1. Check images. Meaningful images need a useful alt. Decorative images need an empty alt. Flag alt text that repeats the file name or says "image".
2. Check controls built from div or span that are clickable but have no button or link role, no keyboard handler, and no accessible name.
3. Check contrast. WCAG 2.2 AA is 4.5:1 for normal text and 3:1 for large text (18pt or 14pt bold) and for UI component boundaries. Name the colors if you can see them in CSS. If you cannot compute the ratio, say so.

4. Check that every control is reachable by keyboard and has a visible focus style. Flag outline: none with no replacement.
5. Check inputs for a label tied with for and id, or an aria-label. Placeholder text is not a label.
6. Check toasts, live counts, and loading states for an aria-live region. A message that only appears visually is easy to miss.
7. Check heading order and that the page has one h1. Flag skipped levels only when they make the page harder to follow.
8. If you can run a local axe scan in a read-only way, summarize it and still list the issues above in your own words. If you cannot, say the scan is Needs manual check.

Evidence rules:

- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P16, for example P16-1, P16-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

A success toast that is only visual, so a screen reader never hears it.

Copy this prompt online →

SECTION B · QUALITY, SPEED & RESILIENCE

17 Code Quality & Maintainability

Why it matters The cost of messy generated code shows up when someone has to change it safely later.

Modeled on Maintainability reviews of the kind static-analysis tools such as SonarQube score.

AUDIT PROMPT Paste in Ask or Plan mode

MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:

- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.

Code Quality & Maintainability: what to check

Do a maintainability review. Do not refactor in this pass. Rank issues by how much they would slow a new contributor.

1. Flag functions that are very long (roughly over 50 lines) or nest conditionals more than about 4 levels. Cite the file. The number is a heuristic, not a law.
2. Find logic copied in two or more places that should be one function. Quote a short identifying line from each copy.
3. Find unused exports, imports, and variables you can confirm. If a symbol is used from outside the repo, say you are not sure.
4. Flag magic numbers and repeated hardcoded strings that should be named constants. Ignore one-off layout values unless they encode a business rule.
5. Find inconsistent error handling: some paths log and recover, others swallow the error with an empty catch.
6. Flag functions that mix layers, such as raw SQL next to JSX in the same function.
7. Rank the top 10 by drag on a new contributor. Recommend only the top 3 as the first fix-prompt IDs. Do not apply them.
8. Do not restyle the project or rename files for taste. Stick to changes that reduce risk.

Evidence rules:

- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.

- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P17, for example P17-1, P17-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

Three files each with a slightly different copy of the same currency formatter.

Copy this prompt online →

SECTION B · QUALITY, SPEED & RESILIENCE

18 Database & Query Health

Why it matters A query can look fine on a small test database and time out after real data arrives.

Modeled on PostgreSQL EXPLAIN and pg_stat_statements, or the equivalent for your database.

AUDIT PROMPT Paste in Ask or Plan mode

MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:

- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.

Database & Query Health: what to check

Review queries for correctness and for how they will behave as tables grow. Do not run EXPLAIN ANALYZE: it executes the statement. Use plain EXPLAIN only on reads, and only if I have said this database is safe to touch. Never EXPLAIN a write against production.

1. Flag queries that filter a growing table on a column with no index in migrations. Cite the query and the migration.
2. Flag a query shape that will scan the whole table once the table is large. Say why. Do not invent a row count you did not measure.
3. Flag missing foreign keys where a child row can outlive a deleted parent, if the schema shows that relationship.
4. Flag code that loads a whole table and filters in application memory.
5. Flag multi-step writes that must succeed or fail together (for example debit one balance and credit another) with no transaction.
6. Flag list queries with no LIMIT or page size.

7. For each issue, show the query or the ORM call, the problem, and a corrected version as a suggestion. Do not apply it.
8. If the database is SQLite, Firestore, or MongoDB, use that engine's terms instead of pretending it is Postgres. Say which engine you detected.

Evidence rules:

- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P18, for example P18-1, P18-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

An orders query with no LIMIT that is fine on 200 rows and too slow on millions.

Copy this prompt online →

SECTION B · QUALITY, SPEED & RESILIENCE

19 Dependency & Supply Chain Risk

Why it matters You can review every line you wrote and still ship a bug that lives in a package you did not write.

Modeled on OWASP Dependency-Check and npm audit, which report known vulnerabilities in dependencies.

AUDIT PROMPT Paste in Ask or Plan mode

MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:

- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.

Dependency & Supply Chain Risk: what to check

Audit third-party dependencies. You may run read-only commands such as npm audit --omit=dev, npm ls, or the equivalent for this package manager. Do not run npm audit fix, yarn upgrade, or any command that edits the lockfile.

1. List dependencies with a known high or critical advisory at the installed version, from the audit command output. If the command fails, paste the error and stop that check.
2. Flag packages that look abandoned: no release for about two years, or a repository that says it is unmaintained. Say how you checked. If you only guessed from the name, mark it unconfirmed.
3. Flag install scripts (postinstall, preinstall) and packages that ask for broad permissions. Cite package.json.
4. Flag two libraries that do the same job (for example two date libraries).
5. Flag git URLs, file: paths, or tags that are not a registry version pinned in the lockfile.
6. Flag a license that may conflict with selling or closing the source (for example GPL) if the license field says so. Do not give legal advice. Say "confirm with a lawyer" when a license is unusual.
7. For each finding, give severity, the installed version, and either a safer version from the advisory or "no fixed version listed".
8. Do not upgrade anything in this pass.

Evidence rules:

- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P19, for example P19-1, P19-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

A tiny utility added for one function, last published years ago, with a known advisory.

Copy this prompt online →

SECTION B · QUALITY, SPEED & RESILIENCE

20 Mobile App Security (Android/iOS)

Why it matters Anything in an APK or IPA can be unpacked. A secret in the client is not a secret.

Modeled on OWASP MASVS and the Mobile Security Framework (MobSF).

AUDIT PROMPT Paste in Ask or Plan mode

MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:

- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.

Mobile App Security (Android/iOS): what to check

Review this mobile app from the project files. If there is no Android or iOS project, say Not applicable and stop. Do not publish a build.

1. Compare permissions in the manifest or Info.plist with features you can see in code. Flag a permission with no matching feature.
2. Search for API keys and private secrets inside the app module, Gradle files, plist files, and generated config. A Firebase web apiKey or a Google API key restricted by package name is not the same as a service-account JSON. Say which you found.
3. Check whether tokens or personal data are stored in plaintext SharedPreferences, UserDefaults, or a local database instead of the Android Keystore or the iOS Keychain.

4. Say whether TLS certificate pinning is implemented. Also say that pinning and root checks can be bypassed by a determined attacker. Do not describe them as complete protection.
5. Search for debug logs that print tokens or personal data and would remain in a release build.
6. On Android, list exported activities, services, and providers. Flag one that is exported without a permission if it exposes data or actions.
7. If the app handles payments or sensitive data, say whether it uses Play Integrity (Android) or App Attest (iOS), or only a homemade root or jailbreak check. A homemade file check is weaker and easier to bypass.
8. List issues with severity and the exact fix. Do not modify the manifest in this pass.

Evidence rules:

- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P20, for example P20-1, P20-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

A payment app with no Play Integrity or App Attest check, so a modified device can intercept traffic more easily. Those checks can still be bypassed. They are a signal, not a guarantee.

Copy this prompt online →

SECTION B · QUALITY, SPEED & RESILIENCE

21 API Design & Contract Quality

Why it matters An inconsistent API often fails later, when a second client expects a different response shape.

Modeled on Spectral, Stoplight's OpenAPI linter.

AUDIT PROMPT Paste in Ask or Plan mode

MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:

- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.

API Design & Contract Quality: what to check

Review the HTTP API for consistency. Do not rename routes in this pass.

1. List routes and say whether names, plural nouns, and casing are consistent.
2. Check status codes. A create should not pretend success with 200 if the project standard is 201. An error should not be HTTP 200 with an error field, unless you find that this API deliberately does that and every handler matches.
3. Flag fields that a mobile client might already depend on. Say whether there is a version prefix or another way to change the contract without breaking old clients. If there is no versioning, say the API is risky to rename.
4. Compare error JSON shapes across handlers. Quote two different shapes if they exist.
5. Flag list endpoints with no pagination fields (cursor, page, or limit).
6. Say whether an OpenAPI file, README, or comment block is enough for a second engineer to call the API. If docs and code disagree, trust the code and say the docs are stale.
7. End with a yes or no: is this safe to call version 1, or does it need a cleanup pass first? Base that only on the inconsistencies you cited.
8. Suggest the fixes. Do not apply them.

Evidence rules:

- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.

- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P21, for example P21-1, P21-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

One handler returns {"error": "message"} and another returns {"success": false, "msg": "message"}.

Copy this prompt online →

SECTION B · QUALITY, SPEED & RESILIENCE

22 Test Coverage & Quality

Why it matters A test suite that does not check the real outcome can create false confidence.

Modeled on Coverage tools such as Istanbul, and mutation testing such as Stryker.

AUDIT PROMPT Paste in Ask or Plan mode

MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:

- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.

Test Coverage & Quality: what to check

Review tests for usefulness, not only for whether files exist. You may run the test command if it is read-only and does not touch production. Do not add tests in this pass.

1. Say whether payments, auth, and data changes have any tests. Name the files or say none.
2. Flag tests that only check "it did not throw" or that a result is the right type, without checking the value.
3. Flag missing cases you can see from the code: empty input, null, a failed network call, or two writes that must not double-apply. Do not invent cases that the function cannot reach.
4. Flag tests that assert private function details or snapshot large markup that will break on a harmless edit.
5. Flag tests that are skipped, commented out, or marked TODO.
6. Say whether reading the tests would teach a new person how the feature should behave.
7. Name the 5 missing tests most likely to catch a real bug, in order, with the input and the expected result.

8. If there is no test runner, say so in the first line and still list the 5 tests you would write first.

Evidence rules:

- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P22, for example P22-1, P22-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

A discount function whose test only asserts that the result is a number.

Copy this prompt online →

SECTION B · QUALITY, SPEED & RESILIENCE

23 Cost & Resource Efficiency

Why it matters The bill that ends a side project is often one loop calling a paid API, not the base hosting fee.

Modeled on FinOps checks of the kind tools such as Infracost popularized for infrastructure cost.

AUDIT PROMPT

Paste in Ask or Plan mode

MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:

- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.

Cost & Resource Efficiency: what to check

Look for code that can make a cloud or API bill jump. Do not change quotas. Do not guess a dollar amount you cannot compute from the repo.

1. Find loops or retries that call a paid API (AI, SMS, email, maps) with no maximum attempts and no per-user cap.
2. Find an expensive call that repeats for the same input with no cache and no memoization.
3. Find uploads with no size limit. Say which route.
4. Find cron jobs or queues that can run more often than intended, or a worker with no stop condition.
5. Find a path where one user can trigger unlimited paid work on your account.
6. Note infrastructure config that is always on for work that looks occasional. If the host config is not in the repo, mark Needs manual check. Do not invent the monthly cost.
7. Order the list by what could spike first. For each, name the guardrail (max attempts, max bytes, max calls per user per day) as a suggestion.
8. Do not apply the guardrails in this pass.

Evidence rules:

- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P23, for example P23-1, P23-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

An AI feature that retries forever, so one user action can call a paid API hundreds of times.

Copy this prompt online →

SECTION B · QUALITY, SPEED & RESILIENCE

24 Error Handling & Observability

Why it matters The failures that hurt are often quiet: the user leaves, and nobody is alerted.

Modeled on Structured logging and error tracking of the kind Sentry's open-source core is built around.

AUDIT PROMPT Paste in Ask or Plan mode

MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:

- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.

Error Handling & Observability: what to check

Review error handling. Do not add a vendor SDK in this pass.

1. Find async calls and request handlers with no try/catch and no .catch, where a failure would be swallowed or crash the process. Cite the file.
2. Say whether errors are logged with a request id and what was attempted. A log of the word "error" with no id is weak.
3. Say whether the user sees a message and a retry, or a blank screen or a spinner that never ends.
4. Say whether logging is one module or scattered console.log calls.
5. Flag a payment or data-loss failure that is only logged, with no alert path in code (email, webhook, or error-tracker capture).
6. If alerting is configured only in a dashboard, mark Needs manual check and say what I should look for.
7. List the 5 most dangerous silent failures, ordered by damage, with the file and the missing signal.
8. Do not print personal data from sample logs. Describe the fields.

Evidence rules:

- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.

```
  - Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P24, for example P24-1, P24-2.)
  - Checked and fine: what you verified is already OK.
  - Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.
```

EXAMPLE FINDING

A payment webhook that fails and is only written to a console nobody reads.

Copy this prompt online →

SECTION C

Website Launch Readiness

Prompts 25–30

25 SEO & Shareability Foundations

Why it matters If a crawler or a chat app cannot read a page, the site is hard to find and shared links look empty.

Modeled on Google Search Central guidance, Lighthouse SEO checks, and the Open Graph protocol.

AUDIT PROMPT

Paste in Ask or Plan mode

```
MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:
- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.
```

SEO & Shareability Foundations: what to check

```
Audit SEO and link previews from the code and the built HTML if a build exists. Do not submit the site to a search engine.

1. List pages and their title and meta description. Titles do not have a fixed length. They are often cut off in results around 50 to 60 characters. Meta descriptions do not change ranking. They can be cut off around 150 to 160 characters. Flag duplicates and framework default titles such as "Vite + React".
2. Check each page for one h1, a sensible heading order, and link text that says where it goes.
3. Check Open Graph and Twitter tags: title, description, image, and url. The image should be 1200 by 630. Say if any page is missing them.
4. Check the favicon, apple touch icon, and web manifest. Flag the framework default icon if it is still there.
5. Read robots.txt. It should allow public pages and must not disallow the whole site by accident. Note noindex on pages that should be public.
6. Check sitemap.xml includes the public pages and is linked from robots.txt. Thank-you or private URLs should be absent.
7. Check canonical URLs and whether www and non-www are consistent in config. A host-level redirect you cannot see is Needs manual check.
8. Confirm the main text is in the server HTML, not only inserted after JavaScript runs. View the built file or the response, not only the component.
```

9. Add structured data only where it matches the page (for example Article or Organization). Do not invent reviews or ratings.
10. Check the 404 page returns a real 404 status and links to useful pages.
11. Return a table: URL | title | description | OG image | issues. Do not edit files.

Evidence rules:

- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P25, for example P25-1, P25-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

Every page shares one title, so search results cannot tell them apart.

Copy this prompt online →

SECTION C · WEBSITE LAUNCH READINESS

26 Conversion & CTA Audit

Why it matters A site can look finished and still give a visitor no clear next step.

Modeled on Conversion checks: one primary action, a clear message, and a short form.

AUDIT PROMPT Paste in Ask or Plan mode

MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:

- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.

Conversion & CTA Audit: what to check

Review clarity and the primary action, page by page. Do not rewrite the site in this pass. Suggest copy. Do not invent testimonials or numbers.

1. For each page, state the one primary action in a sentence, then say whether a clear control for it is visible without scrolling, on a wide window and on a 320px window.
2. Say whether the headline and the line under it tell a new visitor what this is, who it is for, and what they get.
3. If there is a sticky mobile bar, say whether it covers content or the keyboard. If a long page has no sticky action, say whether one would help.
4. Check button labels. They should say the outcome. Flag Submit, Click here, and Learn more when a specific label would fit.
5. Check forms: only the fields that are needed, labels, inline errors, and a privacy note near the email field.
6. Check trust: real contact details, and no fake countdown or invented customer count. If a number is in the code with no source, flag it.
7. Confirm a successful form or download has a thank-you state and that the success is tracked without putting the email in the analytics event.
8. On landing pages, flag nav links that pull people away from the primary action before they can take it.
9. Return suggested copy and layout changes, ordered by how likely they are to help. Mark that order as your judgment, not a measured result.

Evidence rules:

- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P26, for example P26-1, P26-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

A landing page whose only button says Learn more, below several screens of text.

Copy this prompt online →

SECTION C · WEBSITE LAUNCH READINESS

27 UX States: 404, Loading, Errors & Thank-You

Why it matters The happy path is the easy part. Loading, empty, and error states are what make a product feel finished.

Modeled on Nielsen Norman Group heuristics: show system status, and help people recover from errors.

AUDIT PROMPT

Paste in Ask or Plan mode

```
MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:
- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.
```

UX States: 404, Loading, Errors & Thank-You: what to check

```
Find missing UI states. Do not implement them in this pass. Say where each one belongs.
1. 404: is there a custom page, does it return HTTP 404, and does it link home and to a useful section?
2. Loading: do slow actions disable the button and show a spinner or skeleton? Flag a button that can be clicked twice.
3. Empty: do lists and search results with no rows show a message and a next step, or a blank box?
4. Form errors: are messages next to the field, is the typed value kept, is focus moved to the first error, and is the error announced to a screen reader?
5. Success: after submit or purchase, is there a confirmation, and for the main conversion a thank-you page with a next step?
6. Errors: is there a retry on a failed request, and an error boundary so one crash does not blank the whole app? The boundary should not show a raw stack to the user.
7. Say what a visitor sees if JavaScript is slow, and whether session-expired or permission-denied pages exist. If this site has no accounts, say Not applicable for session expiry.
8. List each gap with the file or route and the state to add. Do not add it yet.
```

```
Evidence rules:
- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:
```

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P27, for example P27-1, P27-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

A contact form that clears and shows nothing, so the visitor cannot tell if it was sent.

Copy this prompt online →

SECTION C · WEBSITE LAUNCH READINESS

28 Legal & Trust Pages

Why it matters A missing age check, a font loaded from a third party, or a marketing email with no unsubscribe can create legal exposure before the first sale. This is a checklist for a lawyer, not legal advice.

Modeled on Privacy-by-design practice, the US rules named below, and India's Digital Personal Data Protection Act, 2023. This is a technical checklist, not legal advice.

AUDIT PROMPT Paste in Ask or Plan mode

MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:

- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.

Legal & Trust Pages: what to check

Prepare notes for legal pages and pre-launch legal gaps based on what this repo actually does. I will have a lawyer review them. Do not present the result as legal advice. Do not invent a company name, address, refund policy, or a fine amount. Flag the gap. Do not predict a lawsuit.

1. List what the site collects and where it goes: forms, accounts, payments, cookies, analytics, third-party scripts, and embedded videos. Base every later point on that list.
2. Draft a privacy policy outline from that list: what is collected, why, who receives it, how long it is kept if the code or docs say so, how a person can ask for access or deletion, and how to contact the owner. If retention is not in the repo, write "ask the owner" instead of a number.
3. Draft terms headings that match the product: accounts, purchases, refunds, acceptable use, liability, and a governing-law placeholder. Mark every placeholder.

4. Say whether a cookie banner is needed. GDPR in the EU and UK, India's DPDP Act 2023, and the California CCPA/CPRA are examples that depend on where users are. Do not decide that a banner is required. If analytics or non-essential cookies load before consent, say that a banner which does not block those scripts would be misleading.
5. Check the contact block uses a real email and address from the project config. Flag placeholders such as example.com.
6. Check the footer and every data-collection form link to the privacy page.
7. End with questions for a lawyer: jurisdiction, retention, and whether a banner is required for this audience.
8. Do not publish the drafts as final policy text.
9. Age on signup. If the product is directed at children, or the app can know a user is under 13, look for an age gate or date of birth before the account is created. The US COPPA rule can apply in those cases. The FTC publishes an inflation-adjusted civil penalty per violation. Do not say that every signup without an age field is a violation.
10. Fonts loaded from Google. Search for fonts.googleapis.com and fonts.gstatic.com. A Munich court (LG München I, 20 January 2022) awarded a small sum in one case where a visitor's IP was sent to Google. The check is whether fonts are self-hosted. If fonts are bundled by the framework, write "Not applicable" and name the file.
11. Session replay. Look for Hotjar, FullStory, Microsoft Clarity, LogRocket, or PostHog session recording that starts without consent or records keystrokes and form fields. Plaintiffs use California's Invasion of Privacy Act, which has a statutory amount per violation. Consent plus masking of inputs is the mitigation. Do not call every analytics script wiretapping.
12. Marketing email. In templates for a launch or promo email, require an unsubscribe link and a real postal address. CAN-SPAM applies to commercial email. A receipt or a password reset is a different case. The FTC's per-email ceiling is inflation-adjusted, so name the rule and say to look up the current figure. Do not invent the dollar amount.
13. Subscription checkout. If the app sells a renewal, the price, the renewal interval, and how to cancel must sit next to the subscribe button, not only in a terms page. California's automatic-renewal statute can treat non-compliant charges as refundable. If there is no subscription, write "Not applicable".
14. User uploads and a DMCA agent. If users can upload images or other content, look for a copyright policy and a designated agent registered with the US Copyright Office. The registration fee is 6 US dollars. Missing an agent can remove the hosting safe harbor. Statutory damages for infringement can be large. They are not an automatic fine for skipping that filing. If the app has no user uploads, write "Not applicable".
15. DPDP notice and consent. India's Digital Personal Data Protection Act, 2023 can apply when personal data is processed in India, or outside India while offering goods or services to people in India. Do not mix this with COPPA or GDPR. Before or at collection, look for a notice that says what personal data is collected and why, and for consent that is an explicit opt-in. A pre-ticked box, or "by using this site you agree", is not that consent. Withdrawal must be as easy as giving consent.
16. DPDP rights and grievance contact. Look for a way to access, correct, and erase personal data, and a real contact for grievances. Do not require a Data Protection Officer unless the app is, or might be, a Significant Data Fiduciary. The government notifies who those are. Do not guess that this app is one.
17. DPDP and children. COPPA's age is 13. DPDP's age is 18. If a child can use the product, look for verifiable consent of a parent or guardian, and for tracking, behavioural monitoring, or targeted advertising aimed at children. Flag those as gaps.
18. DPDP security and breaches. Look for a stated security safeguard and a path to tell the Data Protection Board and the affected person about a personal-data breach. The Act's Schedule sets ceilings, including up to 250 crore rupees for failing reasonable security safeguards and up to 200 crore rupees for children's-data or breach-notice failures. Those are ceilings, not automatic fines. Do not calculate a penalty.

19. DPDP cross-border transfer. Transfer is allowed unless the government has restricted the destination country. Do not apply GDPR's adequacy test to an Indian user base.
20. DPDP rules and dates. Check whether the DPDP Rules are in force for that duty. Do not assume every section is already enforceable. Still mark a missing notice or consent as a gap to fix.

Evidence rules:

- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P28, for example P28-1, P28-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

A signup with no age check on a product children can use, and analytics that start before any notice or consent.

Copy this prompt online →

SECTION C · WEBSITE LAUNCH READINESS

29 Analytics & Measurement Setup

Why it matters If you cannot see what visitors do, you find out a launch failed only after the chance to fix it has passed.

Modeled on Privacy-conscious analytics: measure outcomes, keep personal data out of events, and respect consent.

AUDIT PROMPT Paste in Ask or Plan mode

MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:

- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.

Analytics & Measurement Setup: what to check

Review analytics. Do not add a new vendor. Do not send a test event that includes an email address.

1. Say which analytics tool is installed, whether it is on every page or only some layouts, and whether it loads before consent. If consent is not implemented, say so.
2. List custom events and where they fire. Flag an event property that includes an email, name, or other personal data.
3. Say which event is the conversion for the main action, and whether a double click could record it twice.
4. Say whether utm_source, utm_medium, and utm_campaign are read on landing and still available when a form is submitted. Cite the storage (cookie, sessionStorage, or none).
5. Say whether local and staging hosts are excluded from production analytics.
6. Say whether frontend errors are reported anywhere, and whether that report includes the error message (which might contain personal data).
7. Describe a weekly check I can do in the vendor's UI: visitors, sources, top pages, conversion rate, and top calls to action. Do not invent a dashboard that is not in the repo.
8. Return the event list, the file that fires each event, and how I can verify it once without putting personal data in the payload.

Evidence rules:

- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P29, for example P29-1, P29-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

Analytics on the home page only, so a visit that lands on a guide is invisible.

Copy this prompt online →

SECTION C · WEBSITE LAUNCH READINESS

30 Images & Page Weight

Why it matters Oversized images are a common reason a good-looking site feels slow on a phone, and speed is part of the page experience and of search quality signals.

Modeled on Lighthouse and web.dev guidance on images and page weight.

AUDIT PROMPT Paste in Ask or Plan mode

MODE: AUDIT ONLY. Do not create, edit, or delete any file. Do not run commands that change anything: no installs, migrations, git commits, deploys, or "--fix" flags. If your tool has an Ask, Plan, Chat, or Discuss mode, use it for this prompt.

Before you start:

- Tell me the stack you detect (framework, language, database, auth, hosting, payment provider) and which folders you will review.
- If a check below does not apply to this stack, write "Not applicable" and why.
- If you can run read-only commands, run the ones listed. If you cannot, list them so I can run them and paste the output.

Images & Page Weight: what to check

Audit images and page weight from the repo and, if a build exists, from the built files. Do not invent a Lighthouse score you did not run.

1. List images with file size and the size they are displayed at, if the layout sets it. Flag a file much larger than its display size, or over about 200 KB without a stated reason.
2. Say whether modern formats (WebP or AVIF) are used, whether width and height are set to limit layout shift, and whether phones download the desktop file.
3. Check that below-the-fold images can lazy-load, and that the hero image is not lazy-loaded.
4. Check alt text: descriptive for meaningful images, empty for decorative ones. Flag names like IMG_2043.png.

5. Check font loading: how many families and weights, and whether font-display is swap or optional so text is not invisible.
6. Note third-party scripts and unused-looking CSS or JS only when you can point at a file that nothing imports.
7. Core Web Vitals are LCP, CLS, and INP. INP replaced FID as a Core Web Vital in March 2024. If you did not run Lighthouse, say which pages I should run and which numbers to record. Do not fabricate a score.
8. Return a table of image findings and the changes you would make in a later fix pass. Do not convert files now.

Evidence rules:

- Every finding cites a file path and line number, or the exact command output used.
- Mark each finding Confirmed (seen in the code) or Needs manual check (depends on something outside the repo, such as a dashboard setting or production data).
- Never print a full secret. Show the first 4 characters and the location only.
- If you are not sure, say so. Do not invent files, settings, or results.

Severity: Critical = exploitable now, or leaks real data or money. High = serious with little effort. Medium = weakens defenses or needs a second bug. Low = hygiene.

Report:

- Summary: count of findings by severity.
- Table: ID | Severity | Confirmed? | Finding | Evidence | Why it matters | Suggested fix | Effort (IDs for this prompt use the prefix P30, for example P30-1, P30-2.)
- Checked and fine: what you verified is already OK.
- Could not check: what I need to look at myself, and where.

Then stop. Do not fix anything. Ask me which IDs I want fixed.

EXAMPLE FINDING

A multi-megabyte hero PNG sent to every phone.

Copy this prompt online →

PAPA Tech Solutions

About Papa Tech Solutions

Papa Tech Solutions is based in Ghaziabad, India. Our team brings 14+ years of hands-on work across mobile, web, desktop, and games — the kind of experience that shows up in estimates, architecture, and how we talk about trade-offs.

Day to day we ship Android and iOS apps, web products, desktop tools, and games on stacks like Flutter, React Native, Kotlin Multiplatform, Next.js, Angular, and Meteor. The same delivery approach applies when a client needs a business system — booking, gym ops, SaaS, or ERP-style modules — planned in JIRA or Trello and released through CI/CD. AI assistants help with speed; they never replace security review.

We default to Agile. If you already have a process that works, we adapt to it. Either way you get clear updates and software that belongs in production.

Want a human review before launch?

Email papatechsolution@gmail.com or write to us at papatechsolutions.com/#contact.

Open the checklist online

Version 2.0 · Updated September 2026

Changelog: renamed to Vibe Coding Security Checklist. All 30 prompts rewritten as read-only audits with a separate fix prompt.

Scan to open online