

Check an AI-built app before release

<https://prompt-to-prototype.com/lifecycle/verify/> · as of 2026-10-22

Stage 4 of 8: verify

To verify an AI-built app before release, run the six-step V.E.R.I.F.Y. self-check and make the app's own test cases pass. How much more you need depends on the app's [risk tier](#). The self-check and passing tests are required on every release from Team up. Important apps add negative access tests, a business review by a colleague who knows the process, and a technical check by a named technical reviewer. Critical apps add independent tests and a security review.

Who this is for

Builders (non-developers and developers), the colleagues who do business reviews, technical reviewers and app owners. You get the self-check, how to run each review, who signs what, and a section for professional developers. About 9 minutes to read; the self-check takes 30 to 60 minutes the first time.

This page covers how to check a release. *What* to check for security lives on the [vibe coding security checklist](#), and what must be in place before other people rely on the app is the [go-live checklist](#). Which checks each tier requires is set only in the [gate matrix](#).

Why does verification have to be a system?

AI tools produce apps that run and look finished, and they often choose insecure options. An early study found about 40% of one code assistant's programs in security-relevant scenarios were vulnerable,² and a 2025 vendor benchmark of more than 100 models found a flaw in 45% of tasks that offered a secure and an insecure choice, with no gain from newer models³ (more on [what studies of AI-generated code show](#)). In a 2022 randomized study of security tasks, people with an AI assistant wrote less secure code *and* rated it as more secure.¹ So "it looks right" isn't evidence.

The UK National Cyber Security Centre calls for light checks on a prototype with no sensitive data, and much more for authentication or authorization logic or sensitive personal data.⁴ Here, that calibration is the risk tier, and verification rests on things that don't depend on anyone's confidence: tests written before the run, a second person at Important and above, and machine checks.

Don't: accept fluent but wrong output as reviewed (AP10)

Generated code and configuration read as plausible whether or not they're correct.²³ "I read it and it looked fine" isn't a review. A test with an expected answer, a negative access test or a scan result is ([failure patterns](#)).

How do you run the V.E.R.I.F.Y. self-check?

The builder runs these six checks before every release, and records the result. It's recommended in Sandbox and a recorded gate from Team up ([gate matrix](#)). Each letter links to the page that holds the detail.

Letter	Canonical wording	The question the builder answers	Detail lives at
V	Validate the purpose	Does it do the job in the spec, for the people in the spec, and nothing extra?	Write a build spec
E	Exercise it with real examples	Did your 10–20 test cases, including the awkward ones, give the expected answers on this release?	Test your app
R	Restrict access	Can only the right people reach the app and its data, and did a negative test (someone without access) fail to see the data?	Access
I	Inspect secrets and settings	Are there no keys or passwords in the app, its front end or its prompts, and are sharing, storage and database settings private by default?	Secrets
F	Fact-check the output	Do totals reconcile with the source system, and do the AI's claims (what it built, what it changed, figures, sources) match what the data and logs show?	Reconcile totals
Y	Yield a recorded decision	Did you record yes or no in the app record, with the date and who checked? A no means fix it and check again	App record

Download the self-check as [verify-self-check.csv](#), with columns for how to check each step, the evidence, the result, the date and who checked (Prompt to Prototype, Check an AI-built app before release, <https://prompt-to-prototype.com/lifecycle/verify/>, as of 2026-10-22. Licensed CC BY-SA 4.0). Using or adapting it inside your business isn't distribution under the licence.

How to run each step

You need the spec, the test table, a second test account and the app record. No code is required.

1. **V: validate the purpose.** For each rule in the spec, find where the app does it. Then look for anything the spec doesn't ask for: an extra screen, data field, email or connection. An extra is a change you haven't tested. Remove it or add it to the spec.
2. **E: exercise it with real examples.** Run every row of your [test table](#) on this release, not the last one, and write the result and date. A row without a result counts as a fail.
3. **R: restrict access.** Open the app signed out, in a private browser window: you should see a sign-in page and nothing else. Sign in as a second test account that shouldn't see the data and try to open it. If you can't create one, ask IT for a test account, or ask a colleague outside the app's audience to try it while you watch. Check that sharing isn't "anyone with the link" ([how to test your app without a developer](#)).
4. **I: inspect secrets and settings.** Search the app's files and prompts for keys (step 4 of the same tests). Confirm the sharing, storage and database settings are private. If the AI added a connection, check where its key lives ([secrets](#)).
5. **F: fact-check the output.** Count rows and total one number in the source system and in the app ([reconcile totals](#)). Then test what the AI told you: if it said "I added validation to the amount field", type a bad amount; if it said "the tests pass", run them; if it quoted a figure or source, find it.

6. **Y: yield a recorded decision.** In the app record, under **Tests and checks**, write yes or no, the date, the release and who checked. A no means fix and check again. Don't release on a no.

We estimate 30 to 60 minutes the first time and much less later, since the test table is reused. Time it in your pilot.

Don't: believe the agent's own report (AP5)

In July 2025 a coding agent deleted a company's production database during a code freeze, generated fake data and test results, and wrongly said rollback was impossible.⁵ An AI tool's summary is a claim; step F checks it against the data, the logs and your own test run.

Don't: ship a Team-or-above app whose test cases haven't passed

From Team up, passing test cases on this release is a gate ([gate matrix](#)). If a test fails and you can't fix it in time, the release waits. If the expected result was wrong, fix the test table and record why.

A recorded self-check (*illustrative example*): "Release 4, 2026-10-19. V yes (removed an export button the AI added). E 18/18 pass. R signed-out and second-account tests pass. I key search clean; storage private. F 2,140 rows = 2,131 loaded + 9 rejected; total matches the ERP. Y yes. Checked by the builder."

Business review and technical check

At Important and above, two more people review the app, on top of the builder's self-check and the negative access tests. The roles are on [roles and decision rights](#). The tests and the automated part of the technical check run in full on every release. The business review and the reading of the diff or settings may focus on what changed, until a full re-review trigger fires ([how much review each release needs](#)).

The **business review** is done by any colleague who knows the process. It covers the logic, the data used and whether the access matches the intent.

The **technical check** is an automated exposure scan and the data-rule negative tests, plus a 30–45 minute check against a fixed list (data rules or row-level security, no secrets in the client, sharing settings, sign-in). It's done by a **technical reviewer**: a platform or IT engineer, or a champion who has been trained and listed as a technical reviewer. The target wait is 5 working days (A-042). If no technical reviewer is available, the app stays at Team scope.

If people outside the business use the app, or receive what it sends or drafts, the release also needs the accessibility (WCAG 2.2 AA) and AI-disclosure check ([accessibility](#); [AI disclosure](#)).

The builder can't be the business reviewer or the technical reviewer of their own app ([separation of duties](#)).

The business review (about 30 minutes)

The builder walks the reviewer through the app. The reviewer checks:

- ☐ **Logic.** For three or more real cases, including one awkward case the reviewer picks, the app gives the answer the reviewer would have given.
- ☐ **Data.** The app reads and stores what the process needs, no more.
- ☐ **Access matches intent.** Who can open the app, and what each role can do, match who should do that part of the process. (The reviewer judges who *should*; the technical reviewer confirms the rules enforce it.)
- ☐ **Failure.** The reviewer knows what users do if the app is down or wrong for a day, and the runbook says so.
- ☐ **Recorded.** The reviewer's name, the date, the release and yes or no are in the app record.

Don't: have a non-developer business reviewer sign off the data rules

Whether row-level security or table permissions actually stop someone reading a record is a technical question the business reviewer can't test. That's the technical check's job.

The technical check: the fixed list

The technical reviewer works from the reviewer's column of the [security checklist](#), always including:

1. **Exposure scan** with the [active checks](#): secrets in front-end bundles, data endpoints queried signed out and with the public key, one user's record requested by another.
2. **Data rules** deny by default on every table and bucket, with negative tests for reads, inserts, updates and deletes ([data rules](#)).
3. **No secrets in the client** or repository history; each is held server-side and recorded ([secrets](#)).
4. **Sharing settings**: no public links, guest profiles or anonymous access ([access](#)).
5. **Sign-in** on every page and API route, not only the home page.
6. **Agents and AI features**: grants recorded and scoped, trifecta check recorded ([AI features](#)).

Record `technical_check_on`, the result and any findings in the app record. A finding is fixed and re-checked before release.

Code in a repository

The reviewer also reads the pull request or the diff between releases. Does it link the spec, and do only what it says, with no "while I was here" edits? Are there tests for empty, wrong-format, very large and concurrent input? Will the next maintainer follow it without the chat history? Then ask the author to explain one changed line in their own words.

No-code or configuration app

The risk is in the settings. The reviewer walks through the app as the lowest role and as an admin, reads the platform's permission and sharing settings (row-level security, guest or anonymous profiles, link sharing, connectors), checks the dated test evidence and scan output, and, where the platform allows it, exports the configuration to the app record as the reviewed baseline.

Don't: review only the code when the risk is in the configuration

The documented exposures of AI-built and low-code apps mostly came from settings such as public-by-default tables, guest profiles and missing row-level security ([documented exposures](#)).

Don't: rubber-stamp approvals

A review that's always "yes" within minutes isn't a review. If a reviewer has found nothing in ten reviews, have someone else repeat one (a rule of thumb; adjust it in your pilot).

If reviews queue up. Vendor telemetry found review time up 91% with AI assistants (Faros, a vendor, 2025).⁶ Keep releases small, publish a list of trained technical reviewers ([enablement and training](#)), and measure the wait. If the median goes over 5 working days, add reviewers or ask why so many apps reach Important; don't skip the check.

What do Critical apps add?

Critical apps are co-owned with IT or engineering ([graduate to IT](#)). On top of everything above:

- **Independent tests**, run by someone other than the builder (usually IT or engineering), written from the spec rather than copied from the builder's test table.
- **A security review** by the named security reviewer (security staff or an outside provider), covering the threat model, the agent and AI-feature controls, and anything the technical check marked for follow-up.

Releases go through IT's change process ([change gates](#); [how to change a live app safely](#)).

What runs automatically?

Machines should do every check they can, because human review alone doesn't catch AI-generated flaws.¹

- **With no pipeline** (most worker-built apps): turn on every check the platform has built in, such as a security advisor, sharing report or connector audit, and run them before each release. IT runs the scheduled exposure scan, monthly by default, for every app at Important and above. Manual equivalents are in the [security checklist](#).
- **If you already run build pipelines** (circumstance C6): secret, static-analysis and dependency scans run on every change and block on high-severity findings ([guardrails, C6](#)).

Scans find known mistakes; only your test table finds the app doing the wrong job.

Who signs off what?

Who	Signs	Doesn't sign
Builder	The self-check and test results for each release	Their own business review, technical check or downgrade
Business reviewer (Important and above)	Logic, data used, access matches intent	Data rules or security settings
Technical reviewer (Important and above)	The exposure scan, data-rule negative tests and fixed list	Whether the logic suits the business
App owner (Important and above, if outside people use the app or receive what it sends or drafts)	The accessibility (WCAG 2.2 AA) and AI-disclosure check, with counsel on which law applies	A check that hasn't been run and recorded
Security reviewer (Critical)	The security review	The business logic
App owner	Go-live, once the reviews above are recorded	Any check they haven't seen recorded

The full decision table is on [roles and decision rights](#), and go-live sign-off is on [make a vibe-coded app production ready](#).

If you're a professional developer

The same gates apply, with stricter habits because you can go faster.

- **Keep pull requests small**, for example a few hundred changed lines; the same telemetry found pull requests 154% larger, and AI makes large changes no cheaper to review.⁶
- **Agent-authored pull requests** name the agent and the prompt or spec, come from a branch the agent can't merge itself, and are owned by the person who opens them.
- **Write the tests first, from the spec.** Tests an agent writes after the code often restate it. Break a rule on purpose and confirm a test fails.
- **Check the agent's claims** against the diff, the test output and the logs ([safe permissions for AI coding agents](#)), and don't approve a diff you couldn't explain.

Assumptions and evidence

- **A-021** (Medium for developers, Low for non-developers): *Hypothesis.* We expect builders to review AI output only when the process makes review visible and expected; test this by checking whether each pilot release has a dated self-check in the app record.
- **A-012** (Medium): we expect review to become the bottleneck as AI-assisted change volume rises; test this by tracking the wait for technical checks and the size of each release.
- **A-042** (Low): *Hypothesis.* We expect Important-tier review capacity to keep up once the technical check is a fixed 30–45 minute list; test this by measuring, for the pilot's first three months, the median wait (target 5 working days) and how long each check takes.

Evidence: two research studies (older tools, durable findings), vendor benchmarks and telemetry, regulator guidance and an incident report. No study measures how well non-developer builders verify their own apps. Facts on this page are as of October 2026.

Related

- [Secure vibe coding: a review checklist](#): what the technical reviewer checks.
- [Change a worker-built app safely](#): how much review each release needs.
- [Make a vibe-coded app production ready](#): the go-live checklist that follows a passed review.
- [Risk tiers for worker-built apps](#): which checks each tier requires.

-
1. Perry, Srivastava, Kumar and Boneh, "Do Users Write More Insecure Code with AI Assistants?", ACM CCS 2023 (study run in 2022; a durable finding, the tools studied are older). <https://arxiv.org/abs/2211.03622> [research], accessed 2026-10-22.
 2. Pearce et al., "Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions", IEEE S&P 2022 (an early tool; cited for the durable principle only). <https://arxiv.org/abs/2108.09293> [research], accessed 2026-10-22.
 3. Veracode, "2025 GenAI Code Security Report", Jul 2025 (October 2025 update). <https://www.veracode.com/resources/analyst-reports/2025-genai-code-security-report/> [vendor], accessed 2026-10-22.
 4. UK National Cyber Security Centre, blog post on the "vibe coding spectrum" approach to AI-assisted software development, 18 Jun 2026. <https://www.ncsc.gov.uk/blogs/the-vibe-coding-spectrum-approach-to-ai-assisted-software-development> [regulator], accessed 2026-10-22.
 5. The Register, "Vibe coding service Replit deleted production database", 21 Jul 2025 (draws on the affected user's own posts). https://www.theregister.com/2025/07/21/replit_saastr_vibe_coding_incident/ [news], accessed 2026-10-22.
 6. Faros AI, research on AI in software engineering (telemetry from about 10,000 developers; no company-level productivity gain), Jul 2025. <https://www.faros.ai/blog/ai-software-engineering> [vendor], accessed 2026-10-22.

Prompt to Prototype, Check an AI-built app before release, <https://prompt-to-prototype.com/lifecycle/verify/>, as of 2026-10-22.
Licensed [CC BY-SA 4.0](#).