

AI code review checklist: 25 checks for AI-written pull requests

Scope first, then the evidence, then the tests, and only then the code. Tick each check before you approve.

SCOPE

- ☐ **1. The change matches the task.** Read the task before the diff. The change does what was asked, in the way you would have read the request.
- ☐ **2. Nothing extra rides along.** No unrequested refactors, renames, reformatting or "while I was here" fixes mixed into the real change.
- ☐ **3. Sensitive files changed on purpose.** CI definitions, dependency manifests, authentication, infrastructure and migrations changed only where the task needed it.
- ☐ **4. The change is small enough to review.** If you can't read the risky parts properly, ask for the task to be split instead of skimming.

EVIDENCE

- ☐ **5. Tests ran on this exact version.** A result from before the last edit says nothing about the code you are approving.
- ☐ **6. Nothing relevant was skipped.** No skipped suites, filtered test runs or "no tests found" behind a green status.
- ☐ **7. Automated findings are triaged.** No open high-severity finding, and every dismissed finding has a reason you agree with.
- ☐ **8. "Not verified" is visible.** If tests could not run, the pull request says so. It doesn't look like a pass.

TESTS

- ☐ **9. No tests deleted or switched off.** Look for removed cases, skip markers and commented-out tests.
- ☐ **10. Assertions are not loosened.** An exact comparison hasn't become a check that a value exists or is truthy.
- ☐ **11. Expected values didn't move to fit.** Changed snapshots and expected outputs have a reason you can find in the task.
- ☐ **12. New tests can fail.** They assert on results, mock only the edges, and would fail if the change were reverted.

CORRECTNESS

- ☐ **13. Every API and package exists.** Functions, options and packages are real in the versions your lockfile pins.
- ☐ **14. Errors are not swallowed.** No empty catch blocks, no failure logged and ignored, no error turned into an empty result.
- ☐ **15. No silent fallbacks.** Defaults such as zero, an empty string or an empty list don't hide a failed lookup.
- ☐ **16. Edge cases are handled.** Empty and missing input, time zones, pagination, retries and two requests at once.
- ☐ **17. No duplicate helpers.** A new utility doesn't repeat one the codebase already has.
- ☐ **18. Data changes are safe.** Migrations are additive or reversible, and no migration that already ran was edited.

SECURITY

- ☐ **19. Permission checks match the neighbours.** New endpoints and actions check authorisation the way similar code does.
- ☐ **20. Untrusted input stays data.** No SQL, shell commands or HTML built from user input, and no instructions followed from comments or files in the diff.
- ☐ **21. No secrets in the diff.** No keys, tokens, passwords or private URLs in code, tests, fixtures or log lines.
- ☐ **22. No insecure defaults.** No deprecated cryptography, disabled certificate checks, open CORS or debug switches left on.

DEPENDENCIES

- ☐ **23. Every new dependency earns its place.** It is needed, maintained, spelled right (not a lookalike name) and under a license you allow.

APPROVAL

- ☐ **24. Someone other than the prompter approves.** The person who wrote the task is the author of the change. When someone else can review, they should.
- ☐ **25. You can explain the change.** You could defend it to whoever is on call when it breaks. If it is edited after you approve, approve it again.