Report and Triage an Issue

Anyone / Admin Two always-present entry points, one table behind them, and a lifecycle that never deletes.

Screens Report a problem `report-bug.html` · Share feedback `send-feedback.html` · Issue tracker `issue-tracker.html` · Issue detail `issue-detail.html`

Writes the report `issue`

Key fields which surface `kind: bug / feedback` · how bad `severity` · where it happened `context` · lifecycle `status: new / triaged / closed` · who decided `triaged_by · triaged_at · triage_note`

1 · Given / When / Then

Human wording first; the exact table/field in `mono`.

GIVEN	Any signed-in person can report. The header carries two entry points on every screen — a bug icon and a feedback icon — so nobody has to navigate anywhere to use them.
WHEN	A reporter opens the **Report-a-problem Screen** (`kind = bug` , with a `severity`) or the **Share-feedback Screen** (`kind = feedback` , which carries no severity). The app attaches a screenshot and the page context automatically (`context` : the screen, the build and the browser at that moment, which is what keeps a thinly described report diagnosable). Later an admin opens the **Issue-tracker Screen**, reads one on the **Issue-detail Screen**, and records what is being done about it.
THEN	One `issue` row is written with `status = new` . Triage sets `status` to `triaged` or `closed` and stamps `triaged_by` , `triaged_at` and a `triage_note` . One table backs both surfaces, and a database CHECK keeps the per-kind column honest: a bug carries a `severity` , feedback carries none. **writes** `issue`

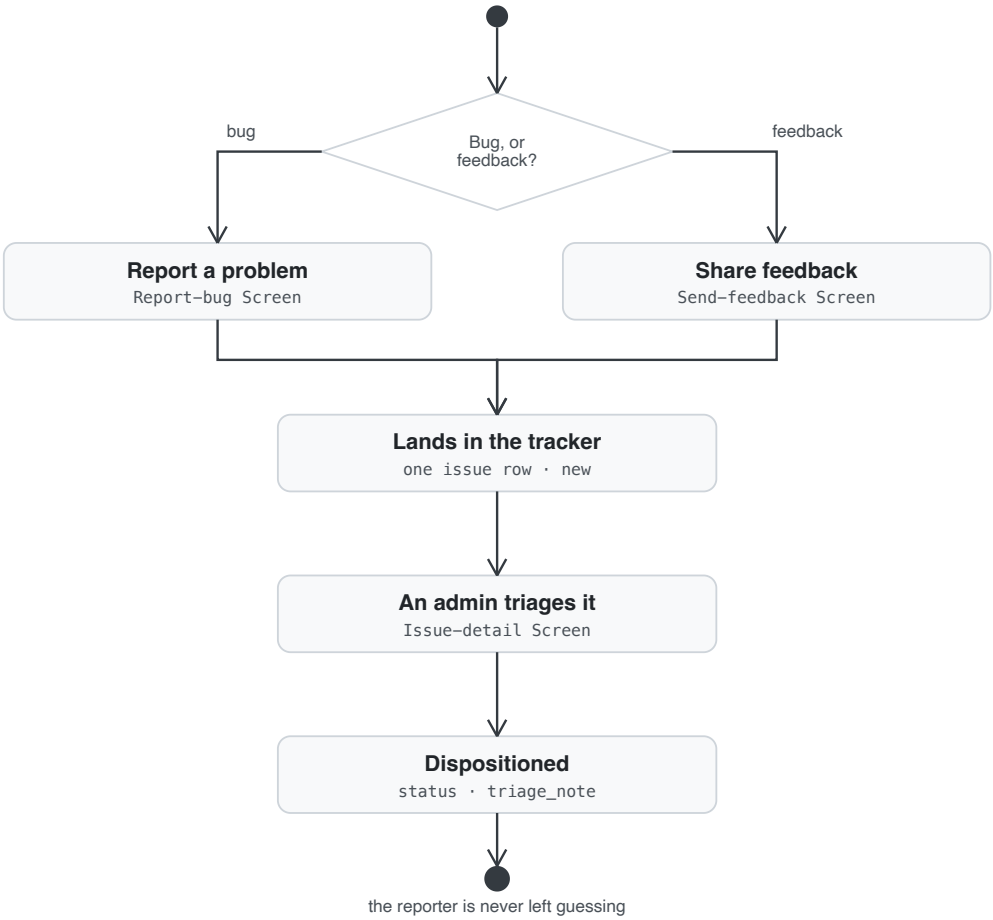
Variations and exceptions

IF…	THEN…
The reporter is an enrollee in the middle of a session	Allowed. What they write never enters their run: it produces no `chat_round` and is not scored, exactly like the operator message channel.
The reporter's profile is later hard-deleted	`reporter_id` goes null and `reporter_email` keeps a readable trace, so an old report stays legible without keeping an account alive for it.
The triager dismisses a report	It is `closed` with a note saying why. Nothing is removed, so the same complaint arriving three times is visible as a pattern rather than as three fresh surprises.

Acceptance test — *Given* a signed-in reporter, *When* they submit a bug with a severity and, separately, a piece of feedback, *Then* two `issue` rows exist with `kind` `bug` and `feedback` , both `status = new` , each carrying its captured `context` ; *and* when an admin triages one, its `status` , `triaged_by` , `triaged_at` and `triage_note` are set and no row is deleted.

2 · The Journey, Screen by Screen

Two ways in, one row, one decision.



3 · In Plain English

Two icons sit in the header of every screen: one for “something is broken,” one for “here is what I think.” They are always in reach, because a problem you have to go looking for a form to report is a problem that mostly goes unreported.

Both write to the same place. A bug carries a severity and feedback does not, and everything else about them is identical, so one table holds both and a single column says which it is.

The app fills in what people are bad at supplying: which screen they were on, which build they were running, what browser. That is usually the part that makes a report actionable, and it is the part a reporter is least likely to include.

Admins work the list, and the only thing they can do is decide and write down the decision. Reports are never deleted, including the ones that get dismissed, because the third time the same complaint arrives it should be visible as a pattern rather than land as a surprise.

Fits the schema cleanly. One `issue` table with `kind` discriminating the two surfaces — they share every field that matters and differ only in whether `severity` applies, which a CHECK holds to the bug kind — and `status` carrying the three-state lifecycle. It is distinct from `audit_log`: the trail records what the system did, `issue` what a human thought of it, and neither is derivable from the other.