

Audit log

Who changed what, when and from where – across tasks, projects, people, settings, money and time. Entries cannot be edited or deleted, and a hash chain makes any later change detectable.

Overview

The audit log is for the questions a security review, an internal audit or a data protection officer asks: *Who gave this user the Finance role? Who changed the due date of that task, and from where? Who downloaded the invoice? When was this setting switched off?* It is written by the application itself, at the point where the data is changed, so a new screen or an API call is covered without anyone having to remember it.

It lives on **one page of its own (Admin > Audit Log)** and is not shown on task or project pages. Only the **RC Admin** and the **Auditor** role can open it.

1. What is recorded

Recorded	Never recorded
----------	----------------

Sign-in: sign-in (password, LDAP, passkey, SSO), sign-out, failed attempts

People and access: users created, changed, activated, deactivated, deleted; password changes; security roles; team membership (added, moved, dedication, lead flag)

Settings: every change of the global settings, teams, API tokens, webhooks

Tasks: created, changed, finished, reopened, deleted, restored - with the old and new value of each field

Projects and milestones: status changes, settings, dates, costs

Money and time: invoices (status, amounts), changed or deleted time bookings, vacation requests and decisions, worktime corrections

Master data: customers, companies, custom field definitions, project templates, baselines

Data leaving the system: invoice downloads, the one-time passwords of a bulk import, exports of the audit log itself

Imports: one summary entry per import run, started, finished, undone

Passwords, tokens and other secrets (shown as *******)

The text of comments and descriptions - only that a description changed, and its length

A person's name, e-mail and phone number inside a change - only their length

What a person only *read*

The planner's own moves of dates, importance and the like

Making a time booking (every comment books minutes); changing or deleting one is recorded

Every entry says **who** (a person, an API token, the MCP endpoint, the scheduler or the system), **through what** (web, REST API, MCP, mail, import, scheduler), **from which address**, **what** happened, **to which object** (its name at the time), the **team and project** it belongs to, and what changed as *old* → *new*.

A **private task** leaves only the fact that something happened, not its title or content, and the entry is visible only to the people on the task - not even to an RC Admin who is not one of them.

2. The audit log page

- **Filter** by period, who, what happened (an action such as **task.done** or a prefix such as **user.**), a search text, and the number of entries per page (25 to 200). **Reset** clears the filters.

- **Page through** with **Older entries** and **Newer entries**. The page stays fast however large the log gets.
 - **Export CSV** or **JSON** exports what the current filters show - all of it, not only the page on screen. The export is itself an entry in the log. In the CSV, cells that start like a spreadsheet formula are made harmless.
 - The **Details** column shows what changed as *field: old → new*.
-

3. Who may see it

Only the **RC Admin** and the **Auditor**. Team leads and project managers do not see the log: it is not a management report. To give someone the Auditor role, open the user in **Admin > Org Management** and add **Auditor** to the secondary roles. The Auditor can read, filter, export and check the integrity; they cannot change anything, including the retention.

4. Integrity: seal chain and anchor copies

Every entry is written inside the same database transaction as the change itself: if the change is rolled back, so is its entry. About half a minute later a background job **seals** new entries: it gives each a position and computes a **keyed hash** (HMAC-SHA256) over the entry and the hash of the one before it. Change, remove or insert an entry later and the chain no longer fits from that point on. The page header shows how many entries are sealed and how many are still waiting.

- **Verify integrity** recomputes the whole chain and says which entry is the first that does not fit, and why: changed, missing, wrong link, missing at the end, or sealed with another key. Because it reads the whole log, use it when it matters, not every minute.
- The database also **refuses UPDATE and DELETE** on sealed entries (where the database allows installing the protection; the page tells you if not).
- **Anchor copies**: once a day the application writes a small signed statement - "at position N the chain stood at hash H" - to the **file storage bucket (S3)**, or, if no S3 is configured, to a folder on the server. *Verify integrity* also compares these copies with the chain. That is what catches someone who has both the database and the key and rewrites the whole chain: they cannot reach the copies in the bucket. Use a bucket the database administrator cannot change, ideally with object lock.

The audit key. The chain is verified with a key that is not in the database. The application takes it from, in this order: the environment variable `RC_AUDIT_KEY`, the config value `audit_key`, or - created automatically on first use - an object in the **S3 bucket** of the file storage (`<prefix>/audit/audit.key`). Only when no S3 is configured does it fall back to the file `userdata/audit.key`; the page then shows a warning, because on a disk that does not persist (containers, several replicas) every redeploy would start a new key and the entries sealed before could no longer be verified. With S3 as the file storage the key survives redeployments and is shared by every replica and the scheduler. For stricter separation, set `RC_AUDIT_KEY` in your deployment secrets instead. **Without the key the chain cannot be verified.** So the page asks the RC Admin to **download the key** (the Auditor may too; every download is an entry) and to confirm a copy is stored. An older key can be **added** (kept in the bucket under `audit/keys/`), and a lost one can be **accepted as lost**: its entries are then still checked for order and links, not for content, and the check reports how many (`unverified`). Verification picks the key by the entry's `key_id`.

What this achieves, honestly: any change to the log by a person without the key is **detectable**; with the anchor copies, so is a rewrite by someone who has everything but the bucket. That is *tamper-evident*, not *tamper-proof*.

5. Retention and privacy

An RC Admin sees the card **Retention and privacy**.

- **Keep entries for (years)** - the default is 10, the usual horizon for business records; **0 keeps them forever**. Once a day, entries older than that are removed from the oldest end of the chain, in whole ranges; the rest still verifies, and the removal is an entry of its own. Changing the setting is also an entry.
- **Pseudonymise a person** - for an erasure request. Enter the user name; after a confirmation, every entry by or about that person shows a pseudonym (`pseudonym-ab12cd34`) instead of the name. The same person gets the same pseudonym (as long as the key stays the same), so their entries still belong together, but the name cannot be read back. It **cannot be undone**, it works for deleted accounts too, and the chain stays valid. Entries do not hold a person's name, e-mail or phone number inside a change in the first place.

For a works council: the log records **changes** made by people to data and settings, not what they looked at, and not their working time. It does contain who did what. Tell your works council and your data protection officer that it exists, who can open it, and how long it is kept; the retention setting and the pseudonymising are there for that conversation.

6. REST API and MCP

Read-only, same access rule as the page (RC Admin or Auditor; everyone else gets 403). No API can write to the log.

Call	What it does
<code>GET /api/v1/audit-log</code> · MCP <code>list_audit_log</code>	Entries, newest first. Filters: <code>from</code> , <code>to</code> , <code>actor</code> , <code>action</code> , <code>object_type</code> , <code>object_id</code> , <code>team_id</code> , <code>project_id</code> , <code>q</code> ; <code>limit</code> (at most 200). The answer has <code>next</code> (pass as <code>before</code> for older entries) and <code>prev</code> (pass as <code>after</code> for newer ones).
<code>GET /api/v1/audit-log/verify</code> · MCP <code>verify_audit_log</code>	Counts, retention, anchor status and the result of recomputing the chain and comparing the anchor copies. Reads the whole log.

A read-only token is enough. The full description is in `api/openapi.yaml`.

7. Checking the chain yourself

An auditor does not have to trust the Verify button. For an entry at position n with the hash of the previous entry p (64 zeros for the first):

`hash = HMAC-SHA256(key, p + "|" + canonical)`, where `canonical` is the JSON array of these fields in this order: `id`, `seq`, `at`, `actor_id`, `actor_type`, `via`, `ip`, `request_id`, `action`, `object_type`, `object_id`, `team_id`, `project_id`, `changes`, `meta`, `restricted_to` (each as text, `null` stays `null`; JSON without escaping of slashes or non-ASCII characters). The actor's name and the object's name are **not** part of the hash - that is what lets a person be pseudonymised without breaking the chain. Each sealed entry in the JSON export and in the API

answer carries a `chain` block with the raw values that were hashed (`at` with microseconds, `changes` and `meta` as the stored text, `seq` , `prev_hash` , `row_hash`); the CSV has `seq` and `row_hash` . The function `rc_audit_row_hash()` in `app_libs/audit.func.php` is the reference.

Rules and limits

- Entries from the last half minute are not sealed yet; the page says how many wait.
- Exports that a browser makes on its own (the table export buttons on report pages) never reach the server and are not recorded; downloads the server makes (invoices, bulk-import passwords, the audit log) are.
- Changes made directly in the database, past the application, are not recorded - but a missing or altered entry in the chain, and the anchor copies, show that the log itself was touched.
- The log starts when the feature was installed; older history stays in the older per-area logs (project log, story log, invoice events).
- Entries age out after the retention period. Anchor copies of the removed part are ignored by the check.
- The dev and test databases fill with test entries; the log is append-only, so they stay.