Tallyfern

Version 1.0
6 Nov 2026
Owner Tallyfern Engineering
English · Original

Architecture notes for new developers

A developer who joins the Tallyfern team and wants to know why the system is built the way it is

Tallyfern is bookkeeping software for small firms. It has two parts: the web app with its API, and the nightly jobs that run while the team sleeps. These notes give the reasons behind the design, so you can change the code without breaking the ideas it stands on.

One idea. The ledger only grows. An entry is never edited or deleted.

Money. It is stored as a whole number of sen, never as a decimal.

Nightly jobs. They run one at a time. A company never has two jobs writing together.

Contents

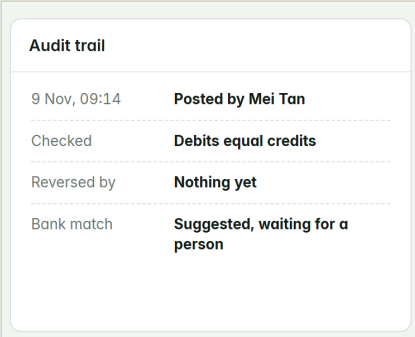
1. The ledger only grows
2. Entries and money
3. Closing a period
4. The nightly jobs
5. Where to start

1. *The ledger only grows*

An entry is never edited and never deleted. To fix a mistake you post a reversing entry, with the same lines and debit and credit swapped. Then you post the correct entry.

Every entry has two or more lines, and the total of the debits always equals the total of the credits. The API checks this before it saves. The database checks it again when the entry is committed.

1 Auditors want to see what happened and when. An append-only ledger also makes a bug easy to find: the history is all still there.



An entry's audit trail: debits equal credits, reversed by nothing yet

1 We once trusted the API check alone, and it burned us. Never again: the second check lives in the database.

2. *Entries and money*

2.1 Money is a whole number

Money is stored as an integer in sen, plus a currency code such as `MYR`. RM 18.50 is `1850`. Never use a float for money. Rounding happens in one place only: the tax function.

1 Do not change the tax rounding without asking Sam. Every other part of the code relies on it being the only place.

2.2 Post an entry

Send the entry to the API. Each request carries an `idempotency_key`. If the same key arrives twice, you get the first entry back and no duplicate is made.

1 The web app makes a new key for every click on Post, so each click is meant to be a new entry. Your own code must reuse a key only to retry.

```
curl -X POST https://api.example.com/v1/entries \
  -H "Authorization: Bearer <api_key>" \
  -H "Content-Type: application/json" \
  -d '{
    "company_id": "co_tf01",
    "date": "2026-11-09",
    "memo": "Paper stock",
    "idempotency_key": "tf-2026-11-09-001",
    "lines": [
      { "account": "5100", "debit": 185000 },
      { "account": "2100", "credit": 185000 }
    ]
  }'
```

The API answers `201` with the entry id, for example `ent_0007421`. These are the errors:

Code	*Meaning*
`422`	The lines do not balance
`409`	The month is closed: `period_closed`
`404`	An account was not found

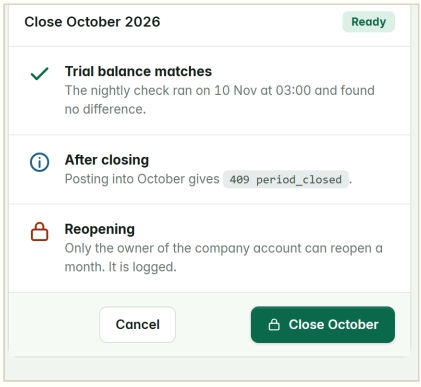
3. *Closing a period*

A month is open or closed. A person closes it from **Accounting**, then **Close period**. After that, posting into the month gives `409 period_closed`. Only the owner of the company account can reopen a month, and the reopening is logged.

Closing is blocked when the nightly check says the trial balance does not match. Every night we add up each account again from the entries and compare the result with the saved balance snapshot. This is the query, simplified:

```
SELECT account_id, SUM(debit) - SUM(credit)
FROM entry_lines
WHERE company_id = :co
GROUP BY account_id;
```

A difference makes the job fail. Nobody can close the period, and the person on call gets a message.



The Close period page: the check passed, October is ready

4. *The nightly jobs*

4.1 What runs, and when

Times are Asia/Kuala_Lumpur. Each job takes a lock on the company, so two jobs never write the same company together.

1 In 2025 import-bank and match-bank ran together and matched one bank line twice. The lock exists because of that.

Time	*Job*	*What it does*
02:00	`import-bank`	Brings in the bank lines from the feed
02:20	`match-bank`	Suggests which bank line belongs to which entry
03:00	`snapshot-balances`	The trial balance check above
03:30	`backup`	Copies the data. Kept 35 days
07:00	`email-digest`	Tells the owners what needs attention

A job that fails is tried 3 times, after 5, 15 and 45 minutes. After that its status is Failed and the person on call is paged.

4.2 Matching never posts

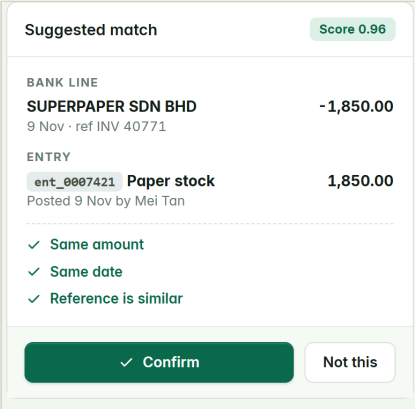
`match-bank` gives each bank line a score from 0 to 1. It looks for the same amount, a date within 3 days and a similar reference.

A score of 0.9 or more becomes a suggested match. A lower score is left for a person. It never posts anything by itself.



Job runs: the running job holds the company lock

1 A person always clicks **Confirm**. A wrong match that posts itself would put a mistake in a ledger that cannot be edited.



A suggested match with its score and the Confirm button

5. *Where to start*

Every row has a `company_id`. The data layer adds the filter by itself, and a test fails the build when a query has no `company_id`.

Read `entry.ts` and `period.ts` first. Then read the job runner. The screens to know are the journal entry under **Accounting**, the reconciliation under **Banking**, and **Admin**, then **Jobs**, which only staff can open.

1 So you never write the company filter by hand, and you cannot forget it.

Change log

Version	*Date*	*Change*	*By*
1.0	2026-11-06	First version	Tallyfern Engineering