GAAPCORE · BUILDER'S GUIDE

Building Your Own Console on GAAPcore

How to give your business its own accounting screens, workflow and controls, with GAAPcore keeping the books correct underneath.

Who it is for: a business owner who wants their own console, and the developer or coding agent who builds it.

Written against: GAAPcore API 0.1.0, database schema v107 (November 2026). The reference console is `accountingkit-console`, pinned to the same version.

How to read it: sections 1–4 are for everyone. Sections 5–8 are the technical setup. Sections 9–13 describe the screens and controls. Sections 14–17 cover the build plan, testing and checklists.

Contents

1 Who this guide is for 4

What you will end up with 4

2 What GAAPcore is, and what it is not 5

What GAAPcore does for you 5

What it leaves to you 5

The database says no 5

3 How the pieces fit 7

The writer boundary 7

Why a server in the middle 7

4 Decide before you build 8

The owner's worksheet 8

Keep the console general 8

Choose a stack 8

5 Set up GAAPcore 9

Steps 9

For local development 9

6 Register your console as a writer 10

Roles and scope 10

The two calls 10

Looking after the key 10

7 The rules every request follows 12

Headers 12

Money is a string 12

One error shape 12

Idempotency: one key per user action 13

Companies, lists and numbers 13

Pin the version; generate the client 13

8 Your people: sign-in, roles and the audit trail 14

Keep people out of the books' database 14

Roles are yours 14

Record which person asked 14

Session hygiene 14

9 Screens accountants trust 16
What every record screen shows 16
Every number drills down 16
Confirm every write in one plain sentence 16
Let the ledger compute; show its preview 16
Show the tie-outs 16
A command registry from the start 17
10 Controls and approvals belong to your console 18
Where GAAPcore helps 18
How to build an approval 18
Actions that deserve a second person 18
Guard the side doors 19
11 Map your business to GAAPcore 20
Modelling tips from the reference build 21
12 Going live 22
The opening balance 22
Prove it 22
Three go-live lessons 22
13 Month-end close and statements 23
The close checklist 23
Who may close 23
14 A recommended build order 24
When the API is missing something 24
15 Testing and upgrading 25
Test against the real thing 25
Upgrading GAAPcore 25
16 Security checklist 26
17 Common mistakes 27
A A complete request, end to end 28
The authentication checks, in order 28
B Error codes you will meet first 29
C Glossary and further reading 30
Where to read more 30

1 Who this guide is for

You run a business, or build software for one, and you want accounting screens that fit how that business works. You don't want to build an accounting engine to get them.

GAAPcore is that engine. It is a **headless** accounting core: it has no screens of its own. It keeps the general ledger, the subledgers (customers, vendors, inventory, fixed assets, leases, bank) and the financial statements, and it enforces US GAAP accounting rules inside the database. Your **console** is the part people see and use: the screens, the workflow, the sign-in and the approvals.

The guide is written for two readers working together:

- **The owner or operator**, who decides what the console must do, who uses it, and which controls the business needs. Read sections 1–4, 9–10 and 12–14.
- **The builder**, a developer or a coding agent directed by one, who writes the console. Read everything. The technical rules are in sections 5–8 and the appendices.

> **A WORKING EXAMPLE EXISTS**
>
> Everything in this guide has already been built once. The reference console, `accountingkit-console`, runs the books of a demonstration coffee roaster (Lantern Hill Coffee Roasters) and its sister café, covering purchasing, roasting, sales, banking, assets, the close and intercompany trade. When this guide says "the reference console", it means that repository. Its file `docs/patterns.md` is the detailed companion to this guide.

What you will end up with

- A web application your staff sign in to, with screens for the parts of the business that touch the books.
- A small server that holds your console's GAAPcore key and passes requests to the GAAPcore API, recording which person asked for each one.
- An GAAPcore installation (its API and its PostgreSQL database) that holds the books and refuses anything that would make them wrong.

2 What GAAPcore is, and what it is not

GAAPcore is a ledger, not an app. Your software sends it business documents; it turns them into journal entries, inventory layers and statements, or it refuses them and explains why.

What GAAPcore does for you

It owns	What that means for your console
Double entry and balancing	You never compute a journal entry for a sale, a receipt or a shipment. You send the document; GAAPcore posts the entry.
Subledgers that tie out	Customer, vendor, inventory, bank, asset and lease balances always agree with their control accounts, and GAAPcore reports the tie-out.
FIFO inventory cost	Stock is valued in cost layers. Shipments, scrap and production consume the layers in order; your console never sets a cost.
Periods and the close	Closed periods cannot be posted into. The close is previewed, checked and then locked.
Document lifecycles	Drafts, posting, and reversals that point back to the original. A posted document is never edited; a mistake is reversed.
Statements and notes	Trial balance, income statement, balance sheet, cash flow, changes in equity, notes, and group (consolidated) statements.
Who wrote what	Every posting is stamped with the application that wrote it and when.

What it leaves to you

- **Screens and workflow.** GAAPcore has no user interface.
- **People.** GAAPcore authenticates *applications*, never people. Your console signs in its own staff, decides what each may do, and records who asked (section 8).
- **Approvals and separation of duties** between people (section 10).
- **Commercial data the books don't need**: price lists, product photos, delivery preferences, notes for the sales team. Keep these in your own store, not in the ledger.

The database says no

The rules run as database checks, so no caller can get around them, not even a buggy one. When your console asks for something the books cannot accept, it gets a clear refusal. These are real messages:

The console tries to…	GAAPcore answers
Post a journal with $1,000.00 of debits and $990.00 of credits	`journal entry JE-0042 must balance (debits 1000.00, credits 990.00, lines 2)`

Date an invoice into a closed September	`invoice INV-1093 date 2026-09-28 falls in a closed fiscal period`
Post a manual journal straight to a bank's cash account	`CONTROL_ACCOUNT_RESTRICTED` : bank cash moves only through bank documents

This is the main reason to build on GAAPcore. A console can have bugs and still never leave the books wrong. The worst outcome is a refusal that your screen shows to the user.

3 How the pieces fit

There are four parts, and only one of them talks to the books.

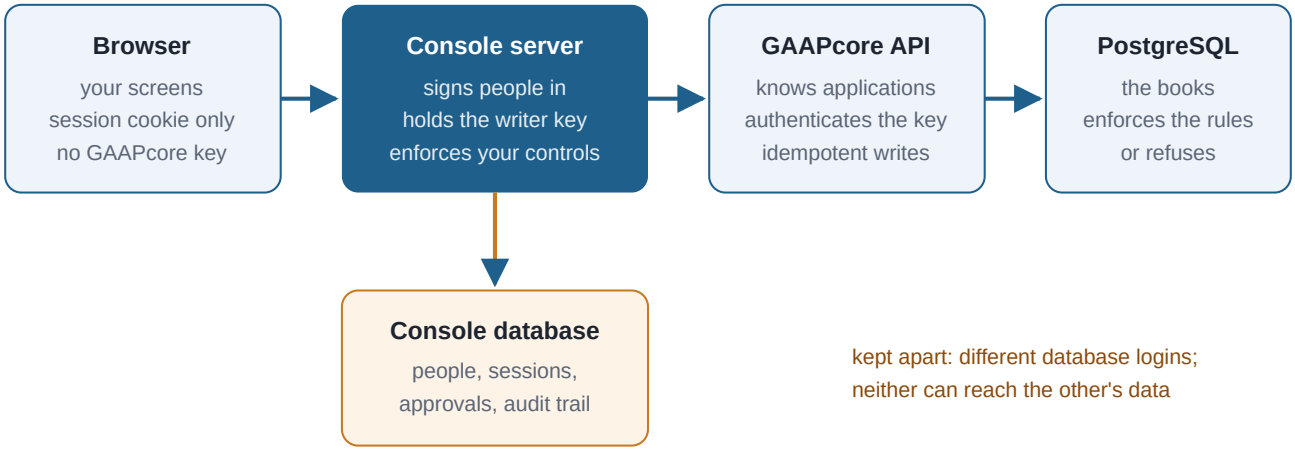


Figure 1. The browser never holds an GAAPcore key. The console server calls GAAPcore as one registered application, called a writer.

The writer boundary

Every program that changes the books is a registered **writer** (in the database, a `source_system`). Your console is one writer. A bank-feed importer, an online-store connector or a nightly script would each be another. Every request a writer sends carries three things:

1. **Who it is**: the `X-Source-System` header with the writer's code.
2. **Proof**: `Authorization: Bearer ak_…`, the writer's API key.
3. **On every write, an `Idempotency-Key`**, so a retried request never posts twice.

GAAPcore stamps the writer on every row the request creates. Your console adds the other half of the story: which person, signed in to the console, asked for it.

Why a server in the middle

The writer key can post to the books and does not expire on its own. If it reached a browser, anyone with that browser could post anything. The console server keeps the key, passes the browser's requests on to `/v1/…`, and adds the headers itself. It is also the one place every person's write passes through, which makes it the right place to enforce your approvals.

4 Decide before you build

A few business decisions shape the console more than any technical choice. Write them down first; a page of notes is enough.

The owner's worksheet

Question	Why it matters	Example (coffee roaster)
Which activities have an accounting consequence?	These are the screens you need. Activities with no accounting effect stay in your own tools.	Buying green coffee, roasting, selling wholesale and online, payroll summaries, equipment
Which companies are on the books?	Each is a separate company in GAAPcore. Related companies can form a group for intercompany trade and consolidated statements.	The roastery, plus a sister café that buys from it
Who uses the console, and in which roles?	Roles are your console's, not GAAPcore's. Start with three: viewer, bookkeeper, administrator.	Owner (administrator), bookkeeper, outside accountant (viewer)
Which actions need a second person?	Approvals are your console's to enforce (section 10).	Purchase orders over $5,000; any credit memo, write-off or refund
When do you go live?	You start from opening balances on a cut-over date (section 12).	Close of business on December 31
What other systems will write?	Each gets its own writer and key, scoped to the companies it needs.	Online store payouts; a bank feed
Where will it run?	The operator runs PostgreSQL, the API and TLS (section 5).	One small cloud server plus managed PostgreSQL

Keep the console general

Build screens around accounting objects (orders, receipts, bills, invoices, recipes, periods), not around one company's quirks. The reference console was built for a roaster, but nothing in it is specific to coffee: the roaster is only its data. A general console is easier to test, and easier to reuse when the business adds a second company or line of trade.

Choose a stack

Any language that speaks HTTP and JSON will do. The reference console uses React and Vite in the browser and a small TypeScript server built with Hono, plus its own PostgreSQL database for people and audit records. It generates typed API calls from GAAPcore's `openapi.yaml`. Use that combination unless you have a reason not to; it lets you copy patterns from the reference console directly.

5 Set up GAAPcore

The operator, whoever runs your servers, installs the database and the API once. This section is a summary; the exact commands are in `api/README.md` and `migrations/README.md` in the GAAPcore repository.

Steps

1. **PostgreSQL 15 or later.** Create an empty database owned by a schema-owner role.
2. **Apply the schema.** As the owner, run `docs/schema.sql` and then every migration in `migrations/`, in order.
3. **Create the API's login role** and grant it with `api/sql/grant-api-role.sql`. The API never owns the schema. Re-run the grants after every migration.
4. **Run the installer** once. It creates the first *administrator* writer, and optionally the first company, and prints the administrator's key one time. Store it in your secrets manager straight away; it is stored nowhere else.
5. **Start the API** with `DATABASE_URL` (the API role's connection) and `PORT`. Check it with `GET /v1/health`.
6. **Schedule the key cleanup** (`npm run cleanup-keys`) daily, to remove idempotency keys past their retention (at least 30 days).
7. **Put TLS in front.** Transport security, certificates and network allow-lists are the operator's job. Ideally only your console server, and your other writers, can reach the API at all.

```
cd api && npm install
DATABASE_URL=postgres://owner@host/books npm run install-db -- \
  --owner-name "Pat Lee" --owner-email pat@example.com \
  --company "Acme Coffee" --base-currency USD --time-zone America/New_York

DATABASE_URL=postgres://ak_api:<password>@host/books PORT=8080 npm start
curl https://books.internal.example/v1/health
```

> **UPGRADES TODAY**
>
> Each GAAPcore migration currently assumes it is applied in order to a database built from those same migrations. Upgrade-safe migrations for databases already holding live books are on the roadmap. Until then, treat an GAAPcore upgrade as a planned event: back up, apply the new migrations in a copy first, run your tests against it, and only then upgrade production.

For local development

The reference console's `npm run dev:setup` does all of the above in a Docker container in one step. It creates the database from a pinned GAAPcore checkout, registers the console's writer, creates demonstration people and seeds a ledger. Copy that script as the starting point for your own.

6 Register your console as a writer

The administrator key from the installer is used for one job: managing writers. It cannot read or post the books. Use it to register your console and issue the console's own key.

Roles and scope

Role	May	Use it for
`reader`	Read the books of the companies in its scope	Dashboards, a reporting tool, an outside accountant's export
`poster`	Read and write the books of the companies in its scope	**Your console**; an importer; an online-store connector
`administrator`	Register, scope, retire writers; issue and revoke keys. No ledger access.	The operator's tooling only. Never the console.

A reader or poster is scoped either to a list of companies or to all companies. Give each writer the narrowest scope that works. GAAPcore enforces some separation of its own: an administrator cannot change its own role or scope, and role changes need a different administrator.

The two calls

```
POST /v1/system/source-systems
X-Source-System: installer
Authorization: Bearer <administrator key>
Idempotency-Key: 0b8c…

{ "code": "console", "name": "Acme accounting console", "source_kind": "app",
  "owner_name": "Pat Lee", "owner_email": "pat@example.com",
  "role": "poster", "company_ids": ["<company id>"] }

POST /v1/system/source-systems/console/credentials
→ { "id": "…", "secret": "ak_7Q2M4XKD9P1C_…", … }
```

The `secret` is shown **once**. GAAPcore keeps only its SHA-256 hash. If you lose it, revoke that credential and issue a new one; even a replay of the same request returns the secret redacted.

Looking after the key

- Keep it in your server's secret store or environment, never in source control, and never in anything sent to a browser.
- Keys start with `ak_`, so secret scanners can find a key that leaked.
- **Rotate without downtime:** issue a second credential, deploy it, then revoke the old one. Revocation takes effect on the very next request, because GAAPcore checks credentials against the database on every call.

- Retiring a writer revokes all of its keys at once.
- When you add a company, an administrator widens the console's scope (`PUT /v1/system/source-systems/console/scope`).

7 The rules every request follows

The API is large (over 400 routes) but uniform. Learn these rules once and every domain behaves the same way.

Headers

Header	When	Notes
`X-Source-System`	Every request	Your writer's code. It must match the key.
`Authorization: Bearer`	Every request	Your writer's key. Never log it.
`Idempotency-Key`	Every POST, PUT, PATCH, DELETE	1–255 printable characters; a UUID v4 per user action.
`X-Request-Id` (reply)	Every reply	Record it with the person who asked (section 8).
`Idempotent-Replayed` (reply)	Replayed writes	`true` when the reply is the stored result of an earlier attempt.

Money is a string

Every amount is a JSON **string** with exactly two decimals: `"1234.50"`. Never `1234.5` (a number), never `"1,234.50"`, never negative; direction comes from the field (`debit_amount` / `credit_amount`). Quantities and unit prices are strings too, at their own precision. In your code, keep amounts as strings or decimals (for example with `decimal.js`), and format them only for display. Never convert an amount to a floating-point number, because a double cannot hold every cent.

One error shape

```
HTTP/1.1 422 Unprocessable Entity
{ "error": {
    "code": "PERIOD_CLOSED",
    "message": "Posting date 2026-08-31 falls in fiscal period 2026-08, which is closed.",
    "details": [ { "code": "PERIOD_CLOSED", "field": "posting_date",
                   "message": "Period 2026-08 is closed." } ],
    "request_id": "req_01J9Z6Q4N8" } }
```

- **Branch on `code`**, which never changes meaning. Show `message` to people, but never parse it.
- A validation failure lists **every** problem in `details`, each with its field path (`lines[1].debit_amount`). Mark each field on the form.
- A **4xx** is an answer about the books. Show it and do not retry. Only a **5xx** or a network failure is retried, with the same idempotency key.
- Make your own server's errors use the same envelope, so the screens handle one shape.

Idempotency: one key per user action

Create the key when the person clicks “Post”, not for each HTTP attempt. If the network drops and you retry, GAAPcore returns the first result instead of posting twice. Two traps:

- **After a refusal, use a new key.** GAAPcore stores refusals too. If the user fixes the cause and you resend with the old key, you get the old refusal back. Keep a key only while the outcome is unknown.
- **A changed request is a new action.** The same key with a different body is refused (`409 IDEMPOTENCY_KEY_REUSED`).

For scripts and imports, derive the key from the content (for example, the source file name and line number), so re-running the script is safe.

Companies, lists and numbers

- **`company_id` is always explicit:** in the body of every create, and as a query parameter on company reads. GAAPcore never infers it. Put the company in your URLs (`/c/<company>/…`) and in every cache key, so switching company never shows another company's data.
- **Lists are paged:** `{ data, next_cursor }` , at most 200 rows a page. Follow `next_cursor` until it is `null` .
- **Document numbers come from GAAPcore.** When a number series exists for a document type, don't send a number: GAAPcore assigns the next one and refuses yours. Show the expected next number as information only.
- **Warnings.** A successful write can carry `warnings` (for example, deactivating an account that still has a balance). Show them after the write.

Pin the version; generate the client

Keep a copy of the `openapi.yaml` you built against, with its commit and schema version (the reference console keeps them in `vendor/accountingkit/PIN.json`). Generate types and a typed client from it, for example with `openapi-typescript` and `openapi-fetch` , so a renamed field fails the compiler, not a customer demo. At run time, compare your pin with `GET /v1/version` and show a banner when they differ.

8 Your people: sign-in, roles and the audit trail

GAAPcore has no user accounts, no passwords and no sessions, and it never will. How your people sign in, whether with passwords, single sign-on or a second factor, is your console's decision.

Keep people out of the books' database

Store people, sessions, approvals and the audit trail in the console's **own** database, owned by its own PostgreSQL login. The GAAPcore API's login must not be able to connect to it, and the console's login must not be able to connect to the books. The reference console's setup script tests both and fails if either connection works.

Roles are yours

A sensible start:

Console role	Typical permissions
Viewer	Read everything; no writes. Good for an outside accountant or an investor.
Bookkeeper	Day-to-day documents: orders, receipts, bills, invoices, payments, journals, bank matching.
Administrator	Everything a bookkeeper does, plus approvals, the period close, issuing statements, reversals of sensitive documents, people management.

Enforce permissions **on the console server** before the call is passed on. Hiding a button in the browser is not a control.

Record which person asked

GAAPcore records the writer, which is your console. Your console records the person. After every write, store a row with the person, the route, GAAPcore's `X-Request-Id`, and the ids in the reply (the document, its journal entry, any reversal). Make the audit table refuse updates and deletes. A record screen then shows both halves:

> "Posted by **Sam Ortiz** through writer `console` at 2026-11-08 11:50 EST."

Where GAAPcore offers a field for a person's name (`prepared_by`, `closed_by`, an approver), your **server** fills it from the signed-in session, ignoring whatever the browser sent.

Session hygiene

- An HttpOnly, SameSite=Strict session cookie; the browser holds nothing else.
- Store sessions server-side as a hash of the token, so they can be revoked. Deactivating a person or changing their password ends their sessions immediately.
- Lock an account briefly after repeated failed sign-ins.

- Require a custom header on writes (the reference console uses `X-Console-Request: 1`), which a cross-site form cannot send.
- Do not relay writer administration (`/v1/system/*`) through the console at all.

9 Screens accountants trust

An accountant decides within minutes whether to trust a system. They look for four things: every number drills down, the books tie out, bad entries are refused, and the close is real.

What every record screen shows

- ☐ The document, its status, and its lifecycle (draft → posted → reversed).
- ☐ The journal entry it produced, with accounts, debits and credits.
- ☐ Links both ways: document ↔ entry ↔ subledger item ↔ report line.
- ☐ The writer and time of every change, the person who asked, and reversals linked to their originals.
- ☐ Refusals shown in GAAPcore's own words, never hidden or reworded.

Every number drills down

A trial balance line opens the account's activity, which opens the journal entry, which opens the source document, and back again. Open drill-downs in a side panel whose state is in the URL (`?panel=je:<id>`), so a reload keeps it and Back closes it. Some source types cover several documents; for example, a `goods_receipt` may be a purchase receipt, a production receipt or a count gain. Try each route in turn (a 404 means “not this kind”) rather than guessing.

Confirm every write in one plain sentence

Before anything changes the books, say what will happen and wait for the click:

> “Post MJ-0090 dated 2026-11-06 for 900.00 to the ledger. A posted journal cannot be edited; a mistake is corrected by reversing it.”

Saving a draft changes no balance and needs no confirmation. Show a refusal inside the same dialog, next to the action that caused it.

Let the ledger compute; show its preview

Don't re-derive in the browser a figure that GAAPcore computes, such as a FIFO cost, a production receipt's share of work in process, or a lease schedule. Where a route offers a preview (the period close, the opening balance, a lease modification), show it. Where only a draft exists, save the draft, show GAAPcore's figures, then post or discard it. The person sees the number the ledger will book.

Show the tie-outs

Give the accountant one page where each control account sits beside its subledger (receivables, payables, inventory layers, goods received not invoiced, fixed assets, leases) and each bank sits beside its statement, every period.

GAAPcore computes these; your screen only lays them out.

A command registry from the start

Register every screen, record, report and action as a command with typed parameters whose values come from lists of valid choices. A keyboard palette (Ctrl+K) uses it on day one. Later, voice or an AI assistant can use the same registry. Because the registry refuses any argument outside the valid choices, a language model cannot invent an account. Commands that write always end at the on-screen confirmation.

10 Controls and approvals belong to your console

An approval is about people: who asked, who approved, and that they were not the same person. GAAPcore authenticates applications, so it cannot hold that rule. Your console holds it.

Where GAAPcore helps

Some controls live in the ledger: price holds placed automatically when a bill disagrees with its receipt, bill approval, a refusal to pay a held or unapproved bill, customer credit holds, closed periods. Read these states back from GAAPcore and show them; don't keep a second copy. Your console decides *who* may approve or release, and writes the approver's name itself.

How to build an approval

1. **Enforce it on the server**, at the one place every person's write passes through, not by hiding a button.
2. **Refuse** the write that would commit without approval (for example, opening a purchase order above the threshold).
3. **Freeze and fingerprint** the document while it waits, so the approver approves exactly what was submitted.
4. **Let the approval make the write**, as your writer, recorded as the approver's.
5. **Cap later changes** at what was approved.
6. **Put "not your own" in a table constraint** as well as in code.

Actions that deserve a second person

Action	Why
Credit memos, write-offs, customer refunds, escheats	This is how a receivable disappears, so it is where cash taken at the counter would be hidden. One person prepares; an administrator who did not prepare it posts.
Asset disposals, impairments, changes in estimate	They take value off the books.
Bill approval and hold release	Never the person who entered the bill.
Period close, issuing statements, editing notes	They change what the books *report*, not only what they record. Administrators only.
Reversing an intercompany sale or transfer	It changes two companies' books at once.

Guard the side doors

Every control has a front door (the release button) and side doors: the setting that decides what gets held (a credit limit, the payables tolerance, the dunning level that holds orders), and other holds that do the same job. Give each side door the same rule as the front door. Then ask someone who did not write the guard to look for more. In the reference console, every review of these rules found at least one.

> **SAY WHAT THE RULE COVERS**
>
> Another writer of the same books, such as an importer script, is not bound by your console's approval rules. Say so in your documentation, and give such writers the narrowest scope that works.

11 Map your business to GAAPcore

GAAPcore covers 121 transaction types. Map each activity from your worksheet (section 4) to the area that handles it, and build only the areas you need.

Area	Routes	What your console uses it for
Foundations	`/v1/companies`, `/v1/ledger/fiscal-*`, `/v1/number-series`	Companies, fiscal years and periods, document numbering
General ledger	`/v1/ledger/accounts`, `control-accounts`, `manual-journals`, `journal-entries`, `trial-balance`	Chart of accounts, control account mapping, manual journals (draft, post, reverse), account inquiry
Banking	`/v1/banking/…`	Bank accounts, money in and out, transfers, statement import, matching, reconciliation
Master data	`/v1/sales/customers`, `/v1/purchasing/vendors`, `/v1/inventory/products`, `units`, `warehouses`, `/v1/tax/…`	Customers, vendors, products, units of measure, warehouses and bins, sales tax jurisdictions and rates
Purchasing and payables	`/v1/purchasing/…`	Purchase orders, goods receipts (FIFO layers), vendor bills with holds and approval, payments, credits, returns, AP aging
Inventory	`/v1/inventory/…`	Put-away, bin and warehouse transfers, adjustments, cycle counts, available to promise, valuation
Production	`/v1/production/…`	Bills of material (recipes), production orders, component issues, finished goods, labor and overhead, variances
Sales and receivables	`/v1/sales/…`, `/v1/ar/…`	Sales orders, credit holds, picks and shipments, invoices with sales tax, receipts, credit memos, returns, dunning, write-offs, AR aging, statements
Fixed assets and leases	`/v1/assets/…`, `/v1/leases/…`	Asset register, depreciation runs, disposals, impairments; lessee leases, schedules, modifications
Reporting	`/v1/ledger/income-statement`, `balance-sheet`, `cash-flow-statement`, `financial-statements`, `subledgers`	Statements, notes, the issued statement set, subledger tie-outs
Groups	`/v1/group/…`	Company groups, intercompany sales and transfers, consolidated statements

The full list of routes and fields is `api/openapi.yaml`. Each transaction type also has a written specification under `catalog/` (business trigger, journal entries, validations, reversal, US GAAP notes), which is the best place to learn what a document does before building its screen.

Modelling tips from the reference build

- **Bank cash moves only through bank documents.** Once a GL account is a bank's cash account, manual journals to it are refused. Show a bank's register as its GL activity, marking what is still outstanding.
- **Give each bank document the bank's reference** (`bank_reference`). Then auto-matching pairs statement lines with book items. For a fee or interest with no book item yet, create the document from the statement line and match again.
- **Receive at the order's cost.** The receiving screen takes quantities, not prices. A different price on the vendor's bill is the bill's to settle.
- **Put process losses in the recipe.** Roasting, baking or trimming needs no “shrink” field: the bill of material lists more input than output, so the output's cost carries the loss.
- **What the books don't need, the console keeps**: price lists, photos, preferences. Don't stretch the ledger to hold them.

12 Going live

A business that already trades starts on GAAPcore with one opening balance, dated in a conversion period that ends on the cut-over date.

The opening balance

The opening balance (`/v1/ledger/opening-balances`) is a document with sections. Each section creates real subledger records that later documents can settle:

Section	What it creates
GL	Balances for accounts that have no subledger
Bank	Each bank account's balance at cut-over
AR, AP, credits	Open customer invoices, vendor bills and unapplied credits, which can later be collected or paid
Inventory	Stock on hand by product, location and cost, as FIFO layers that can later be sold
Received not invoiced, WIP	Receipts awaiting a bill; production in progress
Fixed assets, leases	The asset register and lease schedules at cut-over

Build it as a draft, check it with `preview`, then `post`.

Prove it

Immediately after posting, show the tie-outs: each control account against its subledger, each bank against its statement. This is the first thing an accountant will ask to see.

Three go-live lessons

- **Open assets and leases from their own schedule.** Create the register first, read its schedule at cut-over, and open from those figures, rather than typing totals into the ledger. The register then ties from day one, and later depreciation runs continue the same schedule.
- **Clear go-live items the way the business did.** If you import the months after go-live as summaries, pay and collect the opening items against themselves on their due dates. Otherwise every aging report will show go-live bills as hundreds of days late.
- **Validate an import completely before sending it.** Parse the file, report every bad line by number, guess nothing (dates are `YYYY-MM-DD`), and check that the totals agree. GAAPcore makes the final check, but the user should never meet a refusal that the file itself already showed.

13 Month-end close and statements

The close is where trust is won or lost. GAAPcore does the hard part. Your console turns it into a checklist the administrator can follow.

The close checklist

1. **The month's computed work**: depreciation and lease runs, production closeouts, variance disposition, any reclassifications. Anyone who may write can do these.
2. **Reconcile each bank** against its statement.
3. **Review the tie-outs** page.
4. **Preview the close** (`GET /v1/ledger/fiscal-periods/{id}/close-preview`). It returns the figures, the closing entry, and every blocker in GAAPcore's words ("Draft MJ-0088 is dated in the period; post, re-date or delete it first").
5. **Close** (`POST …/close`), sending back the figures the administrator saw (`expected_net_income_amount`, `expected_result`). If anything moved since the preview, GAAPcore refuses instead of closing on different numbers.
6. **Issue the statements** with their notes.

Read each step's state from the records, not from a tick box someone clicked. Disable the Close button while anything blocks.

Who may close

Closing, issuing statements, editing notes, estimating the allowance for credit losses and mapping control accounts change what the books report. Keep them to administrators, and have the server write the person's name (`closed_by`) from the session.

14 A recommended build order

Build in chunks. Each chunk ends with screens someone can click through, tests, and a short “what to try” note for the owner. Review each before starting the next.

#	Chunk	What gets built
1	Foundations and shell	Console server, sign-in, people and roles, the relay, company switcher, navigation, the command palette, drill-down panel, audit trail
2	General ledger	Chart of accounts, control mapping, periods, manual journals, trial balance with drill-down
3	Banking	Bank accounts, transactions, transfers, statement import, reconciliation
4	Master data	Customers, vendors, products, units, warehouses, tax; opening balance screens
5–7	Procure to pay	Purchase orders with approvals, receiving and put-away, bills, holds, payments, AP aging
8	Production	Recipes, production orders, component issues, finished goods, labor, variances (if you make things)
9–11	Order to cash	Sales orders and credit, picks and shipping, invoices, receipts, credits, dunning, AR aging
12	Assets and leases	Register, depreciation, disposals, leases
13	Reporting and close	Statements, notes, the tie-outs page, the close checklist
14	A realistic year	A simulator that posts a year of activity through the API, so the books look lived-in; reset and snapshot
15+	Optional	Group companies and intercompany trade; voice or AI assistance through the command registry

Skip what your business doesn't do. A services firm might need only chunks 1–4, 9–11 and 13. If you prefer fewer review stops, review procure-to-pay and order-to-cash each as one cycle.

When the API is missing something

Sometimes a screen needs a read the API doesn't offer yet, such as a list, a field it stores but doesn't return, or a typed row. Write the gap down (the reference console keeps `docs/api-gaps.md`), work around it in one place, and raise it with the GAAPcore maintainers. Several GAAPcore routes, for example the journal registers, exist because a console asked for them.

15 Testing and upgrading

Test against the real thing

- **Server tests** for your relay, permissions and approvals, including the side doors (section 10). Test that each refused action is actually refused, not only that allowed actions work.
- **Component and unit tests** for money formatting, the command registry and forms.
- **A scripted browser run per chunk** (for example with Playwright) against a freshly reset demo ledger, following the same “what to try” steps the owner will.
- **A local GAAPcore** built from your pinned commit. Never test against production books.
- **A fast reset**: snapshot the seeded database and restore it in seconds, so every test run starts from known books.

Upgrading GAAPcore

1. Copy the new `api/openapi.yaml` into your pinned copy and update the pin (commit, schema version).
2. Regenerate the client and type-check. The compiler lists every screen the change touches.
3. Run all tests against a database built at the new version.
4. Upgrade production only after a tested backup (see the warning in section 5).

16 Security checklist

Go through this list before real books go on the system.

- ☐ The writer key is only on the console server, in a secret store. It is not in the repository, logs, or anything sent to a browser.
- ☐ The administrator key is stored separately and is not used by the console.
- ☐ The console's writer is a `poster` scoped to its own companies, not all companies unless needed.
- ☐ The API is reachable only over TLS, and only from your writers' networks.
- ☐ The console's database and the books' database use different logins, and neither login can reach the other database.
- ☐ Sessions are HttpOnly, SameSite=Strict cookies, stored as hashes and revocable. Failed sign-ins are rate-limited.
- ☐ Writes require a custom header that cross-site forms cannot send.
- ☐ `/v1/system/*` (writer administration) is not relayed.
- ☐ Every permission and approval is enforced on the server, with “not your own” also as a table constraint.
- ☐ Person-name fields (`prepared_by`, `closed_by`, approver) are written by the server from the session.
- ☐ The audit table refuses updates and deletes, and links each write to GAAPcore's request id.
- ☐ A key rotation has been rehearsed: issue, deploy, revoke.
- ☐ If an AI model is used for voice or search, it sees the command list only, never ledger data, and it cannot write without the on-screen confirmation.

17 Common mistakes

Mistake	What happens	Do this instead
Calling the API from the browser with the writer key	Anyone with the page can post to the books	Relay through your server (section 3)
`Number(amount)` or floating-point totals	Off-by-a-cent totals and refusals	Strings and a decimal library (section 7)
A new idempotency key on every retry	Double postings after a timeout	One key per user action
Reusing the key after a refusal	The old refusal comes back even after the fix	New key once the outcome is known
Rewording or hiding GAAPcore's errors	Users can't tell what to fix; accountants stop trusting the screen	Show the code, message and field details
Computing journal entries or costs in the console	Figures that disagree with the ledger	Send documents; show GAAPcore's preview or draft
Sending your own document numbers when a series exists	Refused	Let GAAPcore assign them
Hiding a button as the only control	Anyone who calls the server directly gets around it	Enforce on the server
Adding user accounts to GAAPcore	Not supported, by design	People live in your console's database
Storing price lists or notes in ledger fields	Clutter and confusion in the books	Keep commercial data in your own store
Reading only the first page of a list	Missing accounts or documents past 200 rows	Follow `next_cursor`

A A complete request, end to end

Posting a manual journal: create the draft, then post it.

```
# 1. Create a draft (no balance changes yet)
curl -X POST https://books.internal.example/v1/ledger/manual-journals \
  -H 'X-Source-System: console' \
  -H "Authorization: Bearer $AK_WRITER_KEY" \
  -H 'Idempotency-Key: 5d7f6a0e-2c1b-4c39-9a9e-3e0b6f1f9a11' \
  -H 'Content-Type: application/json' \
  -d '{ "company_id": "…", "entry_type": "adjusting",
        "posting_date": "2026-11-06", "description": "Accrue November rent",
        "prepared_by": "Sam Ortiz",
        "lines": [ { "gl_account_id": "…rent expense…", "debit_amount": "900.00" },
                   { "gl_account_id": "…accrued liabilities…", "credit_amount": "900.00" } ] }'

# 2. Post it, after the person confirms (a new key: this is a new action)
curl -X POST https://books.internal.example/v1/ledger/manual-journals/<id>/post \
  -H 'X-Source-System: console' \
  -H "Authorization: Bearer $AK_WRITER_KEY" \
  -H 'Idempotency-Key: 9a2e1c44-7b0d-4f6e-8a51-c1d2e3f4a5b6'

# 3. Record in the console's audit table:
#    person, route, X-Request-Id, journal id, journal entry id
```

The fields follow `ManualJournalCreate` in `api/openapi.yaml` at version 0.1.0. No `journal_number` is sent because the company has a number series (section 7). Generate your client from the OpenAPI file rather than copying this example.

The authentication checks, in order

Step	Check	Failure
1	`X-Source-System` present	`400 SOURCE_SYSTEM_MISSING`
2	Bearer credential present	`401 AUTHENTICATION_REQUIRED`
3	Key well-formed and genuine	`401 CREDENTIAL_INVALID`
4	The writer code exists	`403 SOURCE_SYSTEM_UNKNOWN`
5	The header names the key's own writer	`403 SOURCE_SYSTEM_CREDENTIAL_MISMATCH`
6	The writer is active	`403 SOURCE_SYSTEM_INACTIVE`
7	Key not revoked or expired	`401 CREDENTIAL_INVALID`
8	The role allows the route	`403 OPERATION_NOT_PERMITTED`
9	The company is in scope	`403 COMPANY_NOT_IN_SCOPE`

B Error codes you will meet first

Code	HTTP	Meaning and what the screen should do
`VALIDATION_ERROR`	400/422	One or more fields are wrong; mark each one from `details[].field`
`AMOUNT_INVALID`	422	An amount isn't a two-decimal string
`JOURNAL_UNBALANCED`	422	Debits don't equal credits; show both totals
`PERIOD_CLOSED`	422	The date falls in a closed period; suggest a date in an open one
`CONTROL_ACCOUNT_RESTRICTED`	422	That account is fed by a subledger; use the right document (bank, invoice, bill)
`INSUFFICIENT_FIFO_QUANTITY`	422	Not enough stock at that location to issue
`DOCUMENT_NOT_DRAFT`	409	The document was already posted; refresh and show its current state
`DOCUMENT_NUMBER_ASSIGNED`	422	A number series assigns this number; don't send one
`IDEMPOTENCY_KEY_MISSING`	400	A write without a key: a bug in your client
`IDEMPOTENCY_KEY_REUSED`	409	Same key, different body: a bug in your key handling
`IDEMPOTENCY_KEY_IN_PROGRESS`	409	The first attempt is still running; wait and retry with the same key
`OPERATION_NOT_PERMITTED`	403	The writer's role doesn't allow this route
`COMPANY_NOT_IN_SCOPE`	403	The writer isn't scoped to that company; an administrator widens the scope
`INTERNAL_ERROR`	500	Unexpected; safe to retry with the same idempotency key

Codes are permanent: once published, a code is never renamed or reused. The full registry is section C3 of `docs/api-conventions.md`.

C Glossary and further reading

Term	Meaning
Console	Your application: the screens, people, workflow and controls
Writer (source system)	A registered application allowed to call GAAPcore, with a code, a role, a company scope and API keys
Control account	A GL account whose balance is the total of a subledger (receivables, payables, inventory, bank cash)
Subledger	The detail behind a control account: open invoices, open bills, FIFO layers, assets, leases
Tie-out	The check that a control account equals its subledger, or a bank equals its statement
FIFO layer	A quantity of stock at the cost it was received at; issues consume the oldest first
Conversion period	The earliest period, ending on the cut-over date, where the opening balance is posted
Idempotency key	A unique key per write, so a retry returns the first result instead of posting twice
Reversal	A new posting that undoes a posted document, linked to it; posted documents are never edited
Company group	Related companies that trade with each other and report consolidated statements

Where to read more

File	What it holds
`Acctgheadless/docs/overview.md`	What GAAPcore is and what it covers
`Acctgheadless/api/README.md`	Installing and running the API
`Acctgheadless/api/openapi.yaml`	Every route and field
`Acctgheadless/docs/api-conventions.md`	Headers, money, errors, idempotency, restricted accounts
`Acctgheadless/docs/authentication-design.md`	Writers, roles, scope, keys
`Acctgheadless/catalog/`	One specification per transaction type
`accountingkit-console/docs/patterns.md`	The 33 integration patterns behind this guide, each pointing at the code that shows it
`accountingkit-console/docs/build-notes/`	What each chunk built, and what to try