Sellify
User Manual

Point of sale for restaurants, retail and services

Every module, every page, every switch — written for the owner setting a shop up, the manager keeping it right, and the cashier at the till.

2 September 2026

Contents

— About this manual

1 Getting started

2 Branches

3 Users

4 Settings

5 Taxes

6 Categories and Products

7 Inventory

8 Recipes

9 Commissary and production

10 Printers

11 The kitchen

12 Terminals (tills)

13 POS — taking a sale

14 Price at the counter

15 Orders and corrections

16 Shifts

17 Day close

18 Campaigns and promo codes

19 Call centre

20 Riders and delivery

21 External channels

22 Employees and staff meals

23 Who can do what

24 Import and export

25 The Android till

26 Reports

27 Insights (AI)

28 Billing

About this manual

Written for the people who actually use the system: the owner setting a shop up, the manager keeping it right, and the cashier at the till. Every field name here is the label you see on the screen.

Read in this order the first time

Setting a shop up has an order, because each step needs the one before it. Following it takes about an hour for a small shop.

#	Step	Page	Why it comes here
1	Sign in and learn the top bar	—	Getting started
2	Check your branches	Branches	Everything else belongs to a branch
3	Add your staff	Users	So people can sign in, with the right powers
4	Set how the shop behaves	Settings	Order types, approvals, receipt, day close
5	Set your taxes	Taxes	A sale cannot be priced correctly without them
6	Build the menu or catalog	Categories → Products	What the POS sells
7	Set up stock	Units → Suppliers → Items	Only if you track stock
8	Enter opening stock	Purchases	Your first purchase is your opening stock
9	Write recipes	Recipes	Restaurants only — what a dish costs and consumes
9b	Set up a central kitchen	Branches → Production	Only if one kitchen makes for several shops
10	Set up the printer	Printers	So the receipt comes out right
10b	Set up the kitchen	Departments → Printers	Restaurants — so the cooks get a slip
11	Pair your tills	Terminals	A cashier cannot sign in without one
12	Start selling	POS	
13	Close the shift, then the day	Shifts → Day close	Every trading day, before going home

Everything after that is added when you need it: home delivery, an aggregator, a staff canteen. None of it is in the way if you never turn it on.

Every page

Running the shop

- Getting started — signing in, roles, the top bar, the menu
- POS — taking a sale — the till, holds, discounts, FOC, payments, receipts

- Price at the counter — selling at a price somebody agreed
- Orders and corrections — reviewing sales, and correcting a paid one
- Shifts — one cashier's turn on one till, and the drawer count
- Day close — counting the drawer, the Z report, reopening a day

Setting up

- Branches — shops, prefixes, business day
- Users — staff, roles, approval PINs
- Settings — every switch, company-wide or per branch
- Taxes — authorities, cash and card rates, inclusive prices
- Categories and Products — the menu, variants, variant options, modifiers, deals
- Printers — 3 inch, 2 inch, A4, and where each type prints
- Terminals (tills) — pairing a device, approving it, one cashier at a time
- Import and export — filling a list from a spreadsheet, and taking it back

Stock

- Inventory — units, suppliers, items, purchases, adjustments, transfers, counts
- Recipes — what a dish is made of, and what it costs
- Commissary and production — a central kitchen that makes and issues

The kitchen

- The kitchen — departments, kitchen tickets, dine-in holds

Selling more

- Campaigns and promo codes — automatic offers and codes
- Call centre — customers, areas, taking an order down the phone, complaints
- Riders and delivery — the rider book, dispatch, settling their cash
- External channels — Foodpanda and the rest, and what they owe you

Your people

- Employees and staff meals — the employee book, allowances, salary
- Who can do what — every role, side by side

On a device

- The Android till — pairing, selling offline, printing, scanning, the customer screen

Knowing where you stand

- Reports — every report, and the Excel/PDF export
- Insights — the AI summary, reorder suggestions, ask-your-data

Money and the account

- Billing — your plan, invoices, how to pay

For whoever runs Sellify — NOT PUBLISHED

These two are the operator's runbook, not the customer's manual. They are deliberately left out of the PDF and out of sellifypos.com/manual (`"audience": "operator"` in `contents.json`, which both publishers filter on). A shop reading about how its own supplier prices packages and raises invoices is reading the wrong document.

- Platform admin — provisioning customers, suspending, reopening
- Packages (plans) — pricing a package, selling it, editing it, retiring it

If something will not save

The message on the field is the answer. The ones you will meet most:

Message	What to do
"A dine-in order needs a table number"	Type the table number before paying or holding
"Not enough stock"	Receive the purchase first, or switch on `Allow negative stock` for that branch
"This company caps discounts at 50%"	The discount is over the company cap — change the cap in Settings → Discounts, or give less
"2026-08-20 is closed at Gulberg"	The day is closed. Reopen it from Day close, then post
"The Starter plan allows 1 branch"	You are at your plan's limit — see Billing
"Table 1 is already held"	One live bill per table. Recall that order, or pick a free table
"Waiting for Sellify to approve"	The till is registered but not let in yet — ring us. See Terminals
"Rs 240 below cost"	A typed price is under what the item costs you — see Price at the counter
A rider is named at day close	Settle their cash first — see Riders and delivery

Three rules worth knowing before you start

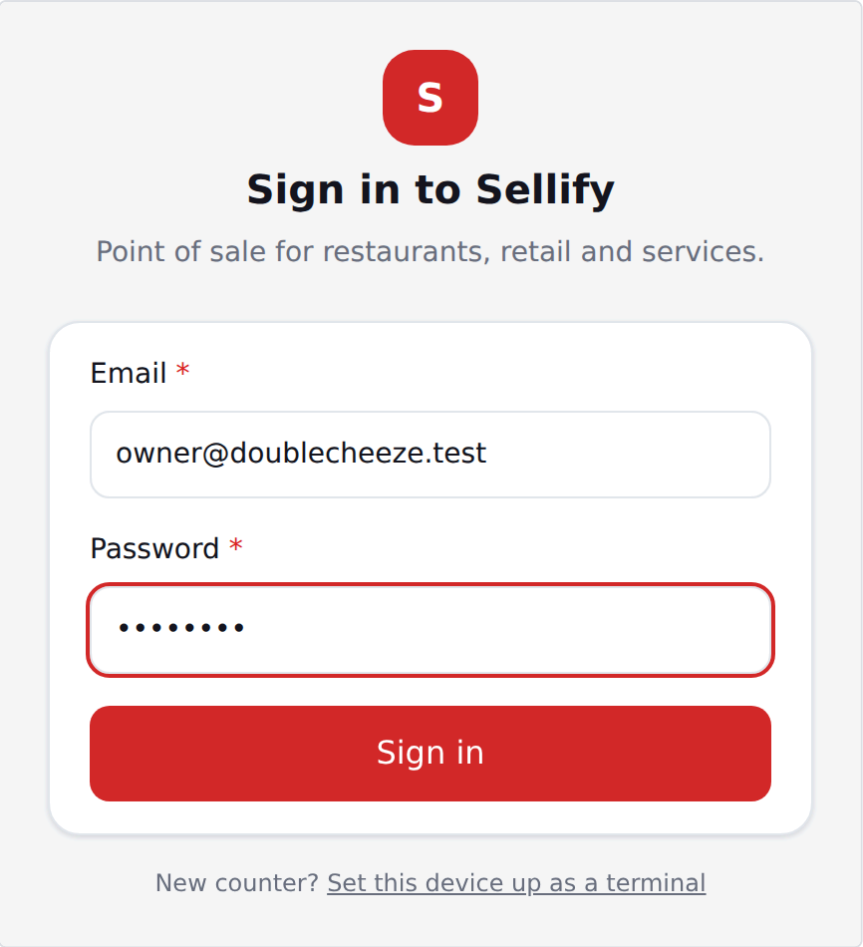
1. **The server decides every figure.** The till never multiplies a price — it asks the server, which is why a discount, a tax rate and a total can never disagree.

2. **Everything is per branch.** Prices are company-wide, but taxes, receipts, stock, the business day and printers all belong to a branch. The branch you are working on is the one in the top bar.
3. **A business day is not a calendar day.** If your day starts at 7am, a sale at 2am belongs to yesterday — on the receipt, in the reports and at day close.

1

Getting started

Signing in



The sign-in screen. Email and password only — there is no company code.

Go to your Sellify address (for example `app.sellifypos.com`) and enter the email and password you were given. Nothing else is needed — there is no company code to remember, because your email belongs to exactly one company.

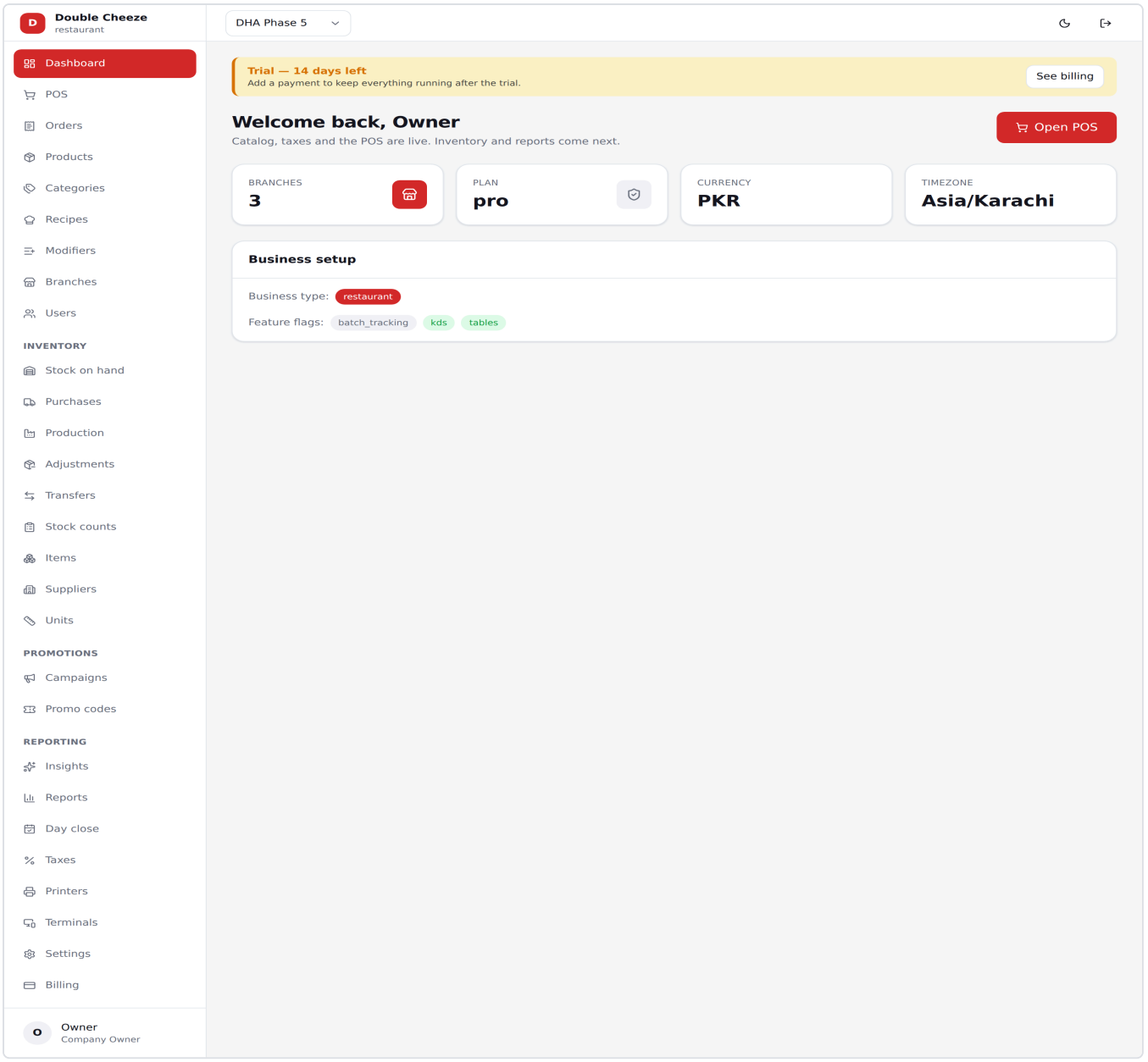
If you see **"This company is suspended"**, the account is unpaid or has been stopped; see Billing.

Who can do what

Role	Can do
Company owner	Everything in your company: all branches, all settings, reports, billing
Branch manager	Their own branch only: sell, stock, reports, day close, approve discounts
Cashier	The till. No reports, no settings, no stock
Kitchen	The kitchen screen (a later phase)
Platform admin	Sellify's own staff. No till, no catalog, no reports of yours

A cashier who signs in lands straight on the POS; there is nothing else for them to see.

The top bar



The dashboard: branch dropdown and dark-mode toggle top right, the menu down the left.

Branch dropdown	The branch everything you do applies to. Change it before entering stock or reading a report — this is the most common mistake
Trial badge	Shown while the company is on trial
Moon / sun	Dark mode. It is remembered on this device only — the POS opens in light mode by default because shop lighting is bright
Sign out	

The menu

The left menu only shows what your role and your business type allow. A retail company never sees Recipes or Modifiers; a restaurant never sees the margin report. That is deliberate — you cannot switch a feature on by hunting for it, and the server refuses it too, not just the screen.

Sections:

- **(top)** Dashboard, POS, Products, Categories, Recipes, Modifiers, Branches, Users
- **Inventory** — Stock on hand, Purchases, Adjustments, Transfers, Stock counts, Items, Suppliers, Units
- **Promotions** — Campaigns, Promo codes
- **Reporting** — Insights, Reports, Day close
- **(bottom)** Taxes, Printers, Settings, Billing

Small things that save time

- Every list has a **search box** at the top left of the table.
- **Delete never destroys history.** A deleted product, supplier or recipe is hidden, and past sales keep pointing at what they actually sold. Writing the same one again brings the old row back rather than failing.
- Errors appear **inline, on the field that caused them**. If a form will not save, the red line tells you exactly which field and why.

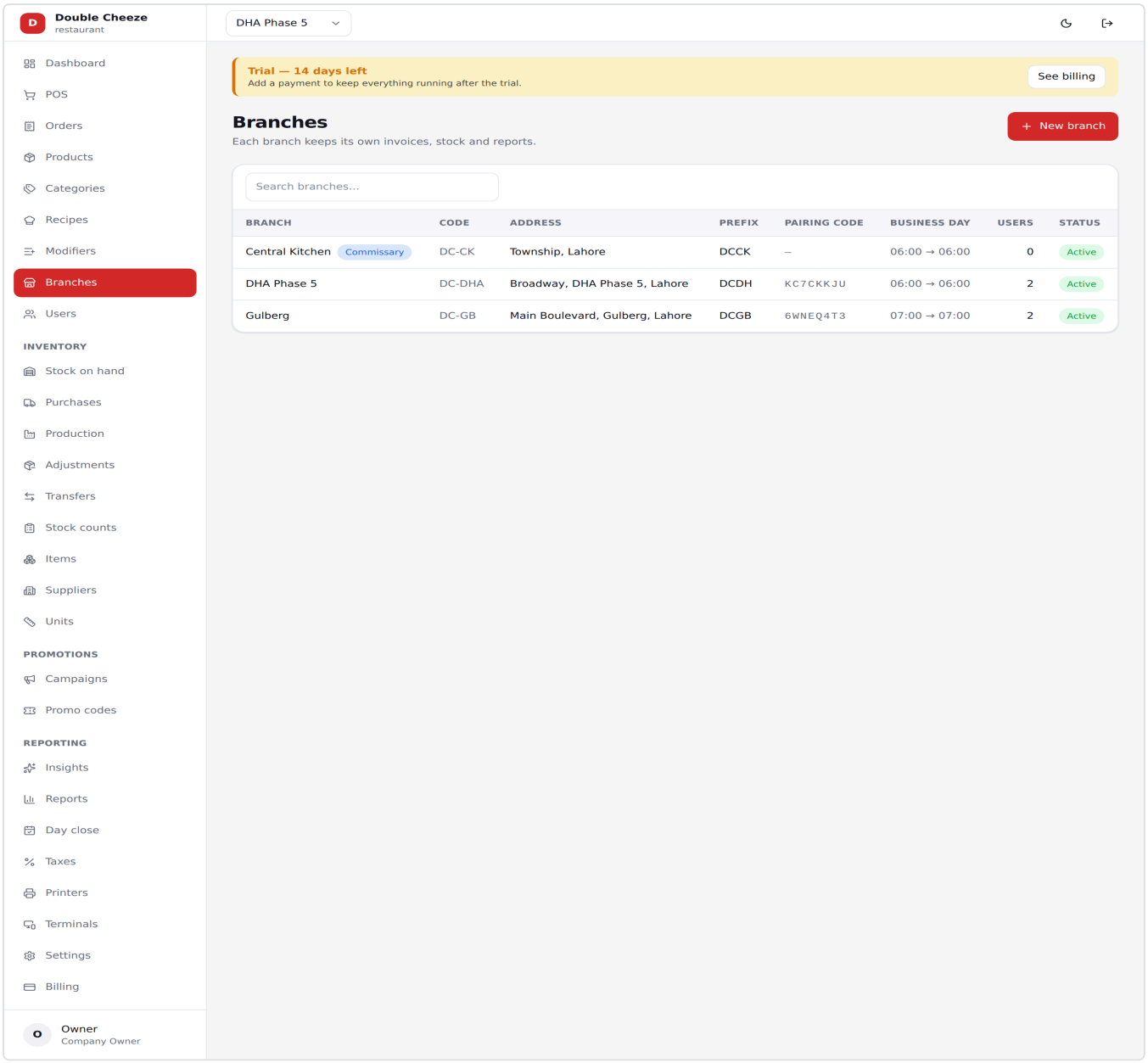
2

Branches

Menu: Branches · **Who:** owner, branch manager

A branch is one shop. Everything that happens — a sale, a stock movement, a day close — belongs to one. A company always has at least one.

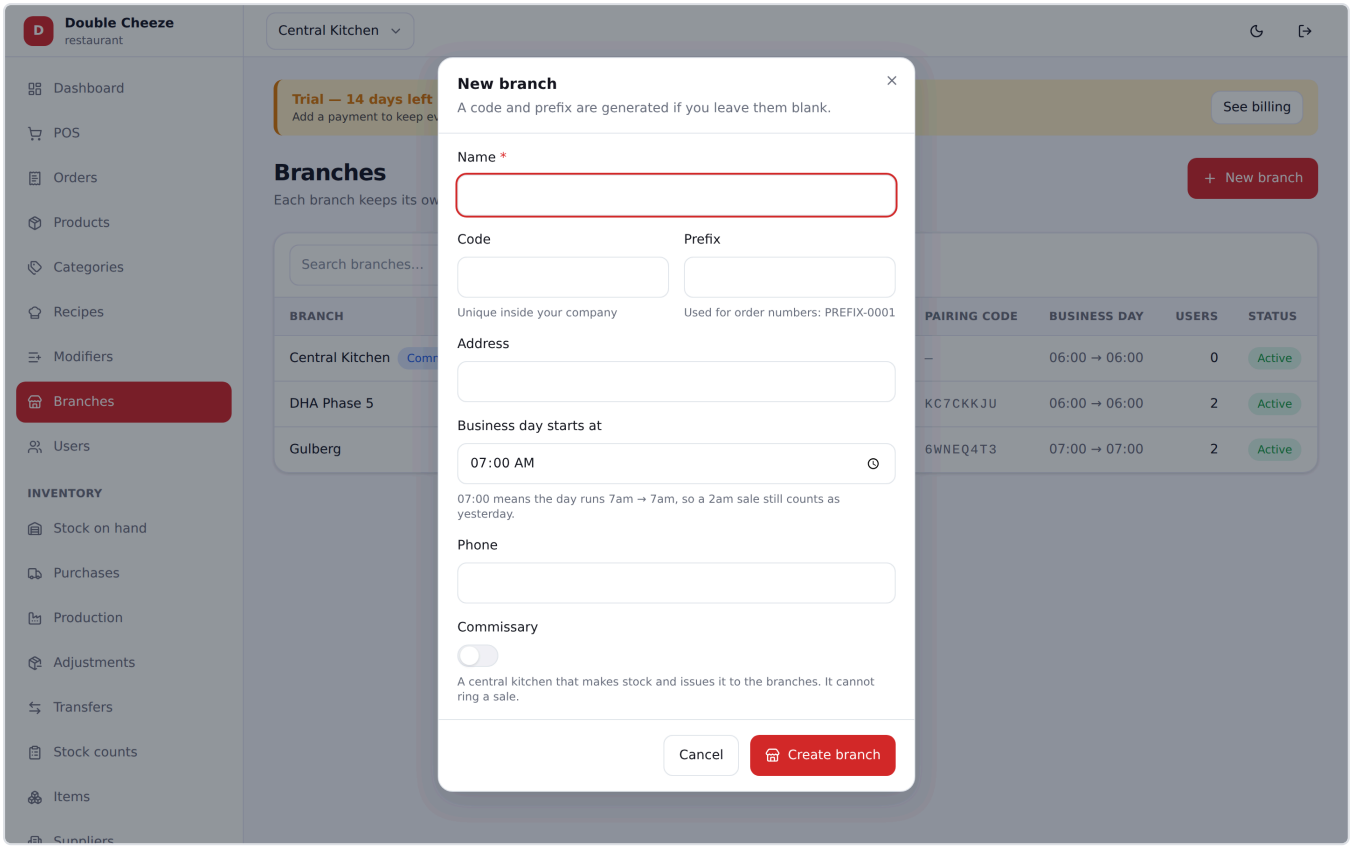
The list



Every branch in the company, with its code, prefix and business day.

Column	What it means
Branch	Name and code
Business day	When the trading day starts and ends, e.g. `07:00 → 07:00`
Users	How many staff are attached
Status	Active or hidden

Adding a branch



Adding a branch. The prefix numbers both its orders and its invoices.

Field	Fill it in like this
Name	What people call it — "Gulberg", "Johar Town"
Code	Short, unique inside your company — `DC-GB`. Leave blank and it is made from the name
Prefix	3–4 letters that start every order number from this branch: `DCGB` → `DCGB-0001`. Leave blank and it comes from the code
City	Its own field, not part of the address — reports and stock can be read for a whole city
Address	Printed on the receipt. Leave empty to print head office's address instead
Business day starts at	Default `07:00`. See below — this matters more than it looks
Phone	Printed on the receipt; falls back to the company's

Branch NTN and STRN are set on the branch too when a branch files its own tax; empty means the company's numbers are printed.

Editing a branch. Press **Edit** on any row to change these later. This is also where a **Commissary** (a central kitchen — see Commissary) is pinned to the branches it may issue stock to, under **Issues to**: pick some and it can transfer to only those, so a van never goes to the wrong city; pick none and it issues anywhere.

The business day

`07:00` **means the day runs 7am → 7am.** A sale rung at 2am belongs to the previous date — on the receipt, in every report, and at day close. Set it to when your shop actually opens, not to midnight, or a late-night sale lands on the wrong day and your cashier's drawer will not tie out.

Each branch sets its own: a bakery opening at 6am and a restaurant closing at 3am can sit in the same company.

What a new branch costs

Your plan includes a number of branches and charges for each one past that. Adding a branch past the plan's ceiling is refused with a message telling you the limit — see Billing.

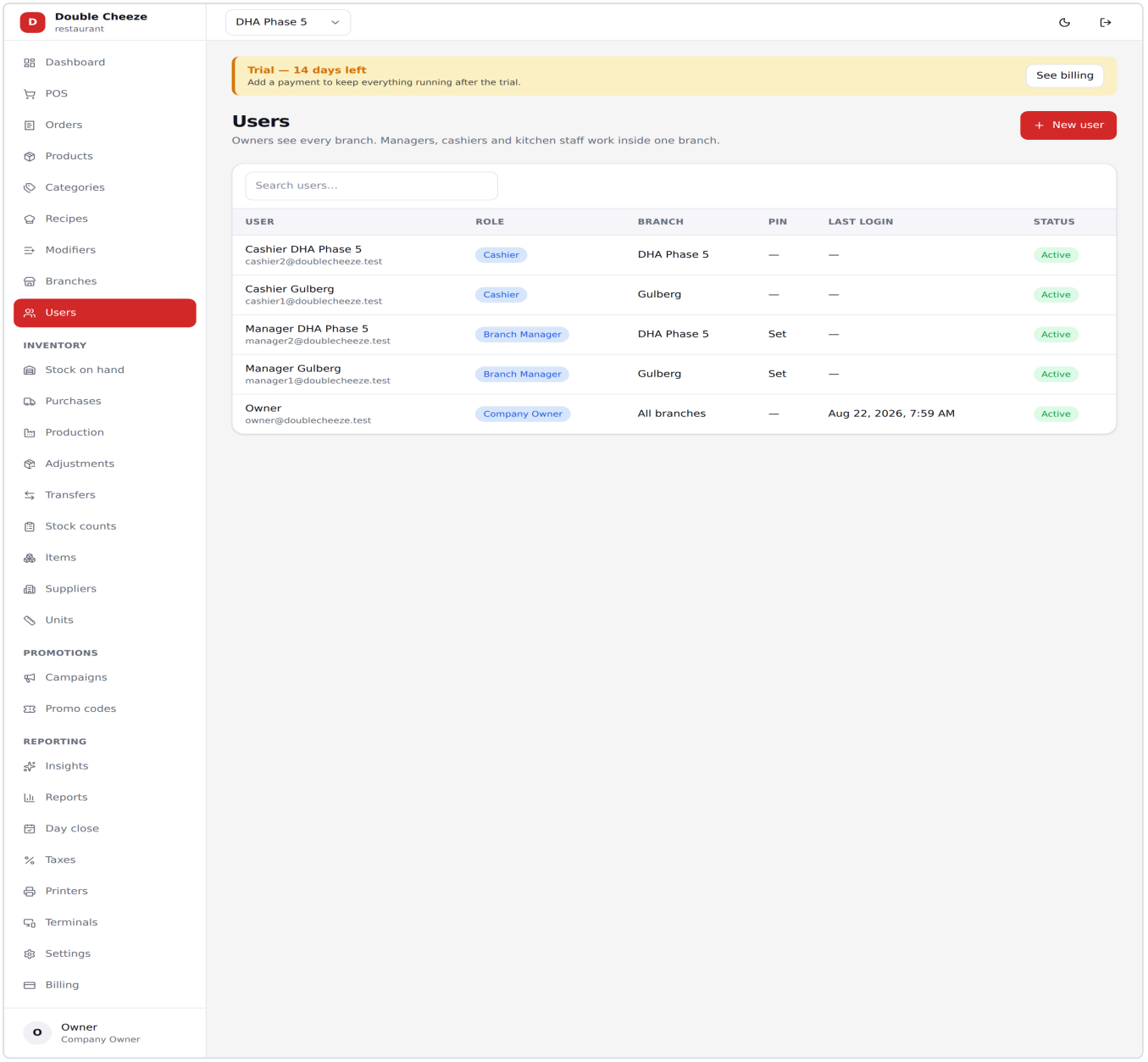
Rules the system enforces

- A branch code is unique **inside your company**, not globally. Two companies may both have `MAIN`.

- The prefix is used for order numbers *and* inventory documents, so keep it short.
- A branch cannot be deleted once it has sold anything; make it inactive instead.

3

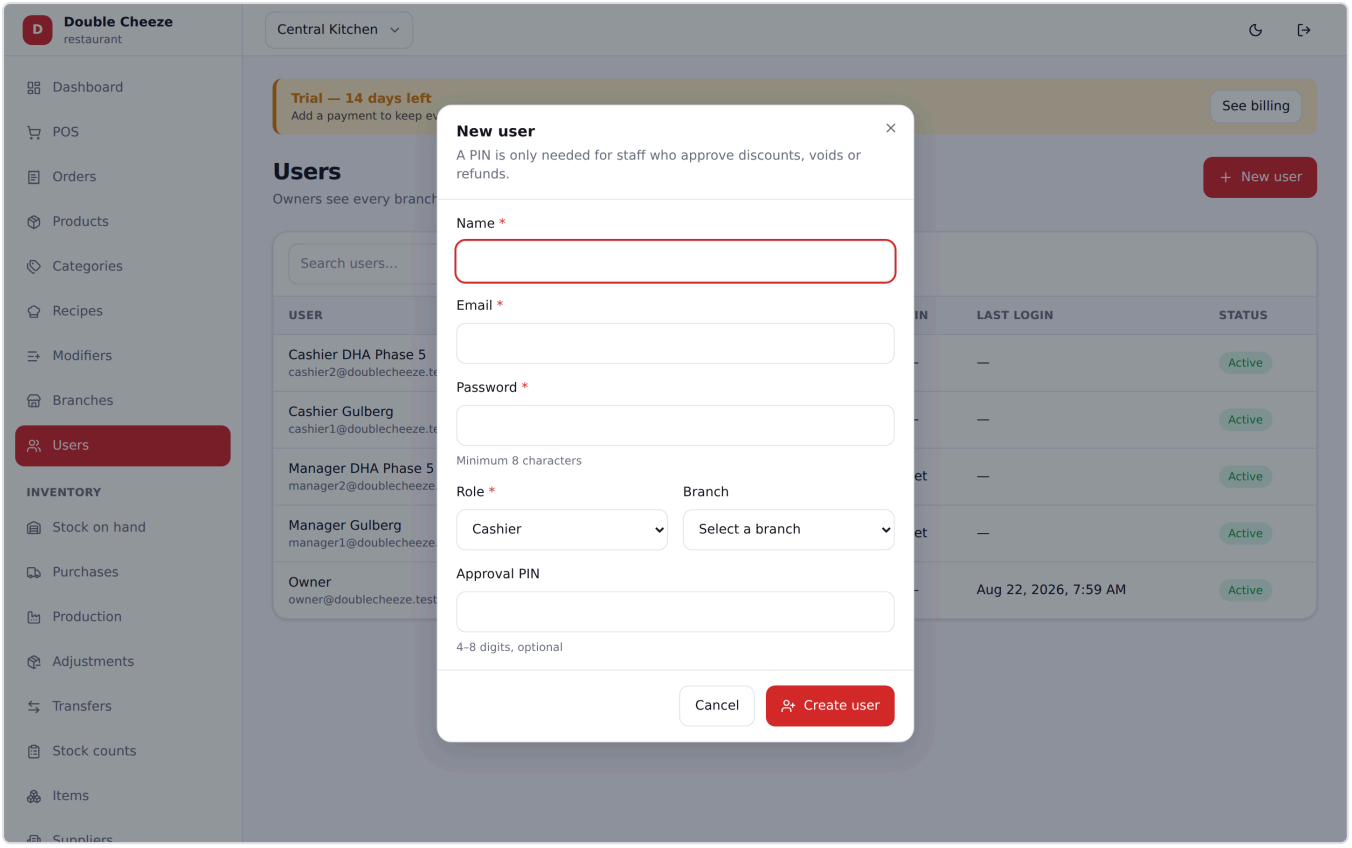
Users



Your staff, their role and the branch each one is bound to.

Menu: Users · **Who:** owner (all branches), branch manager (their own branch)

Adding someone



Adding someone. A manager or cashier must be given a branch; an owner never is.

Field	Notes
Name	As it should appear on the receipt ("Attendant") and in reports
Email	How they sign in. **Unique across the whole platform** — if someone works for two companies they need two addresses
Password	At least 8 characters. They cannot change it themselves yet; a manager resets it here
Role	See below
Branch	Owners are company-wide and have no branch. Everyone else is fixed to one
Approval PIN	4–8 digits, optional. Only needed for people who approve discounts, FOC, voids or a day reopen

Roles, in plain words

In one shop

- **Company owner** — sees every branch, changes settings, reads every report, handles billing. Give this to the person who owns the business, not to a manager.

- **Branch manager** — runs one shop: sells, receives stock, reads that branch's reports, closes the day, dispatches riders, and approves what a cashier cannot do alone.
- **Cashier** — the till and nothing else.
- **Kitchen** — kitchen tickets.

Across the whole company (these have no branch, like an owner)

- **Call centre operator** — takes orders down the phone into any outlet. Takes no money.
- **Call centre manager** — the above, plus the board across every outlet.
- **Call centre HOD** — the above, plus standing discounts and editing complaints. Cannot mint another HOD.
- **HR** — the employee book and the Staff reports. No settings, no billing, no catalog, no stock, no day close, no till.

See Call centre, Employees and staff meals and Who can do what.

Two things that are grants, not roles

"The cashier who is allowed to haggle" and "the accountant who sees the payroll" are not job titles, so they are tick boxes on the user rather than roles. **Only the owner gives them.**

Tick box	What it allows	Who has it by position
Can override price	Type a price at the till instead of the shelf price — Price at the counter	Owner, branch manager
Can view salary	Read salaries in the employee book	Owner, HR

A salary somebody may not see is **absent** from the screen, never blank — so nobody reads "you may not see this" as "nobody set one".

The PIN

The PIN is how a manager approves something at the till without signing the cashier out: a discount over the cashier's limit, a free-of-charge order, a void, reopening a closed day. Which of those need a PIN is set in Settings → Approvals & Security.

The PIN is checked on the server and is never stored in the browser. A manager types it on the till, the till gets a one-time token that is spent on that one action, and it expires after 15 minutes.

Rules the system enforces

- A branch manager can only create users in their own branch, and cannot make an owner.
- Your plan limits how many users a company may have; past it, creating one is refused with the limit named.
- Deleting a user hides them; past sales still show who rang them up.

For the full picture — every screen against every role, and what a manager may not touch — see Who can do what.

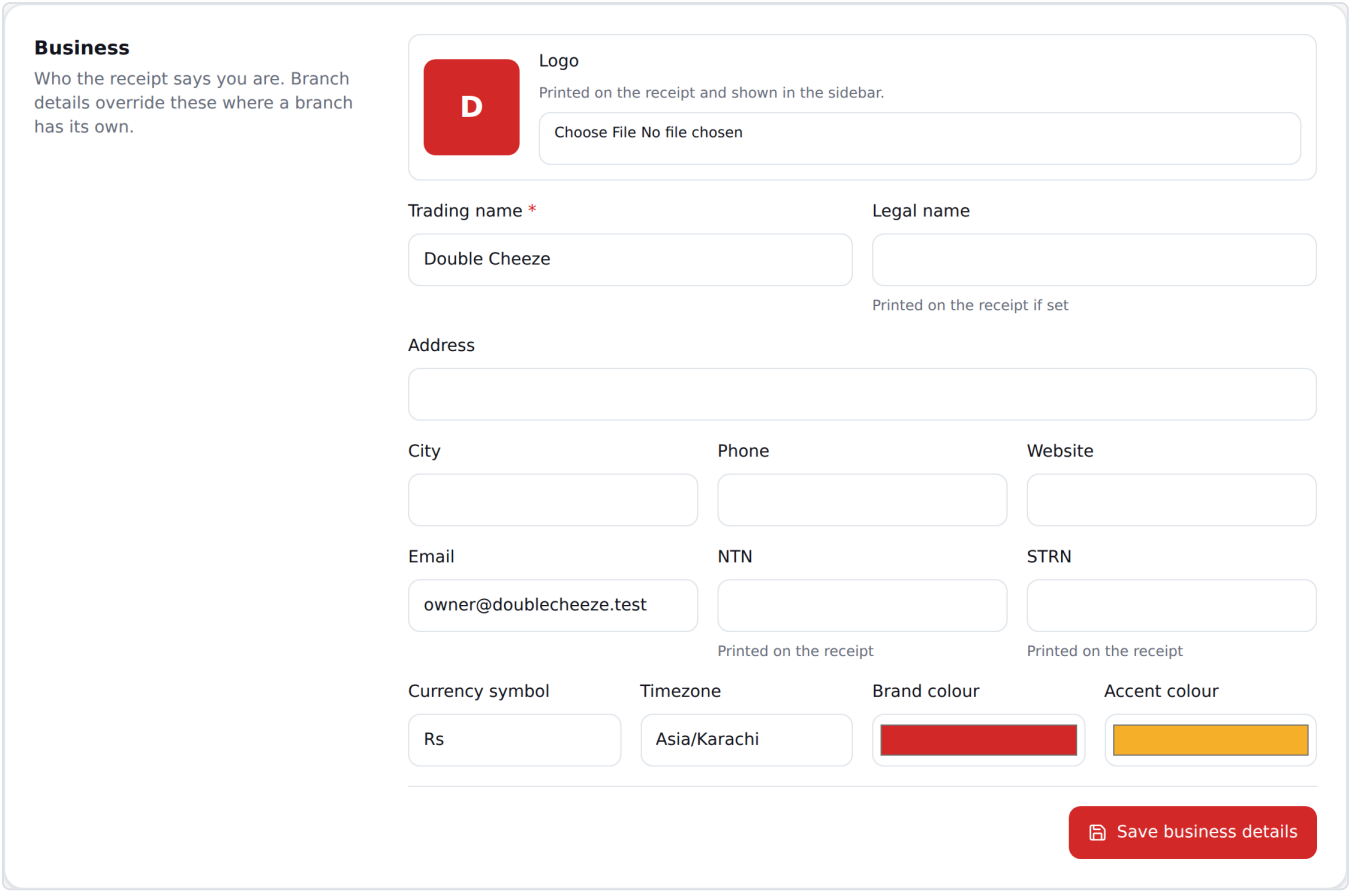
4

Settings

Menu: Settings · **Who:** owner (company-wide and any branch), branch manager (their own branch)

Everything that could differ between two businesses is a setting, not something we hard-code. Every setting has a sensible default, so a brand-new company works before you touch this page.

Company setting or branch setting?



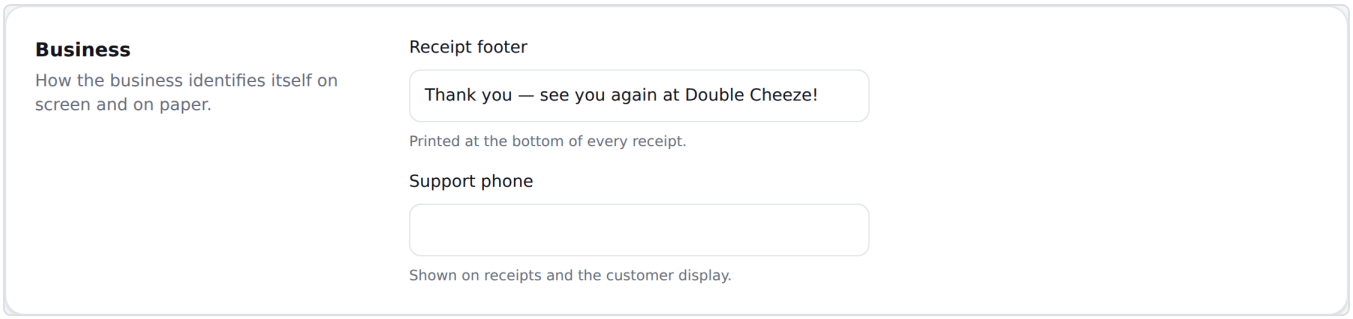
Business details — what the receipt says you are.

At the top is **Settings scope**: *Company* or a branch name.

- **Company** — the value every branch uses unless it says otherwise.
- **A branch** — an override for that shop only. Settings that make no sense per branch (like the discount stacking rule) are only shown at company level.

A value is resolved as: **branch override → company value → built-in default.** Clearing a branch override makes it follow the company again.

Business

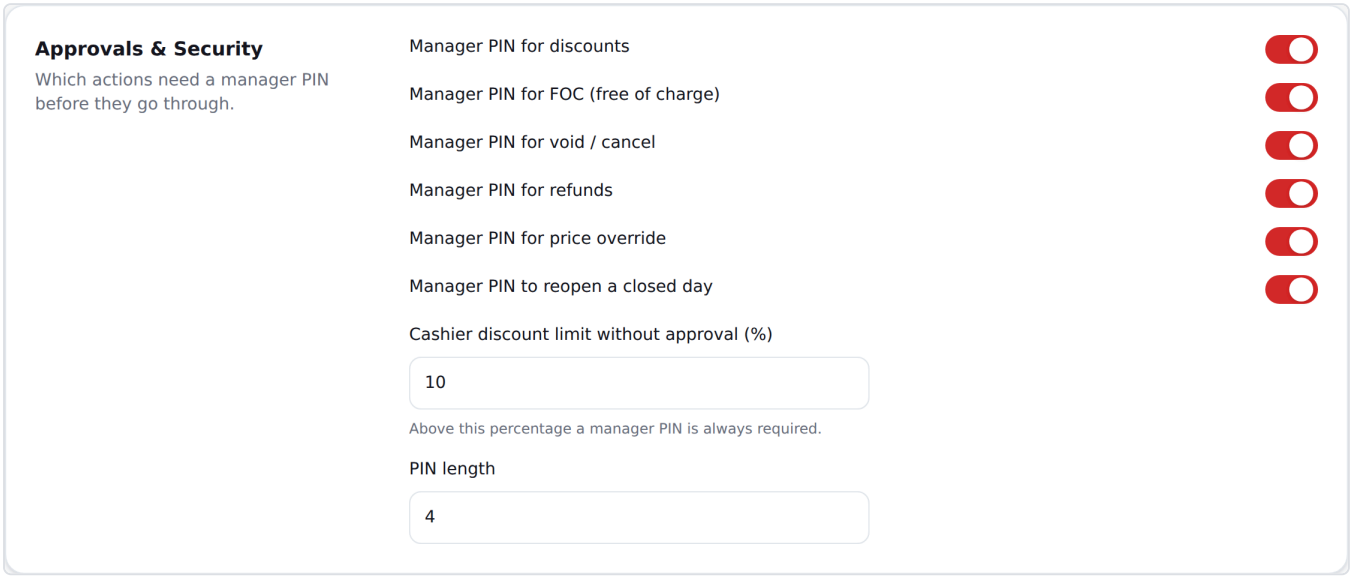


Business: business types, currency and the branch day start.

Setting	What it does
Receipt footer	The thank-you line at the bottom of every receipt
Support phone	Printed under the footer

The company's name, logo, address, NTN/STRN, website and currency are on the same page under the business profile — those are what the receipt and the A4 invoice print.

Approvals & Security

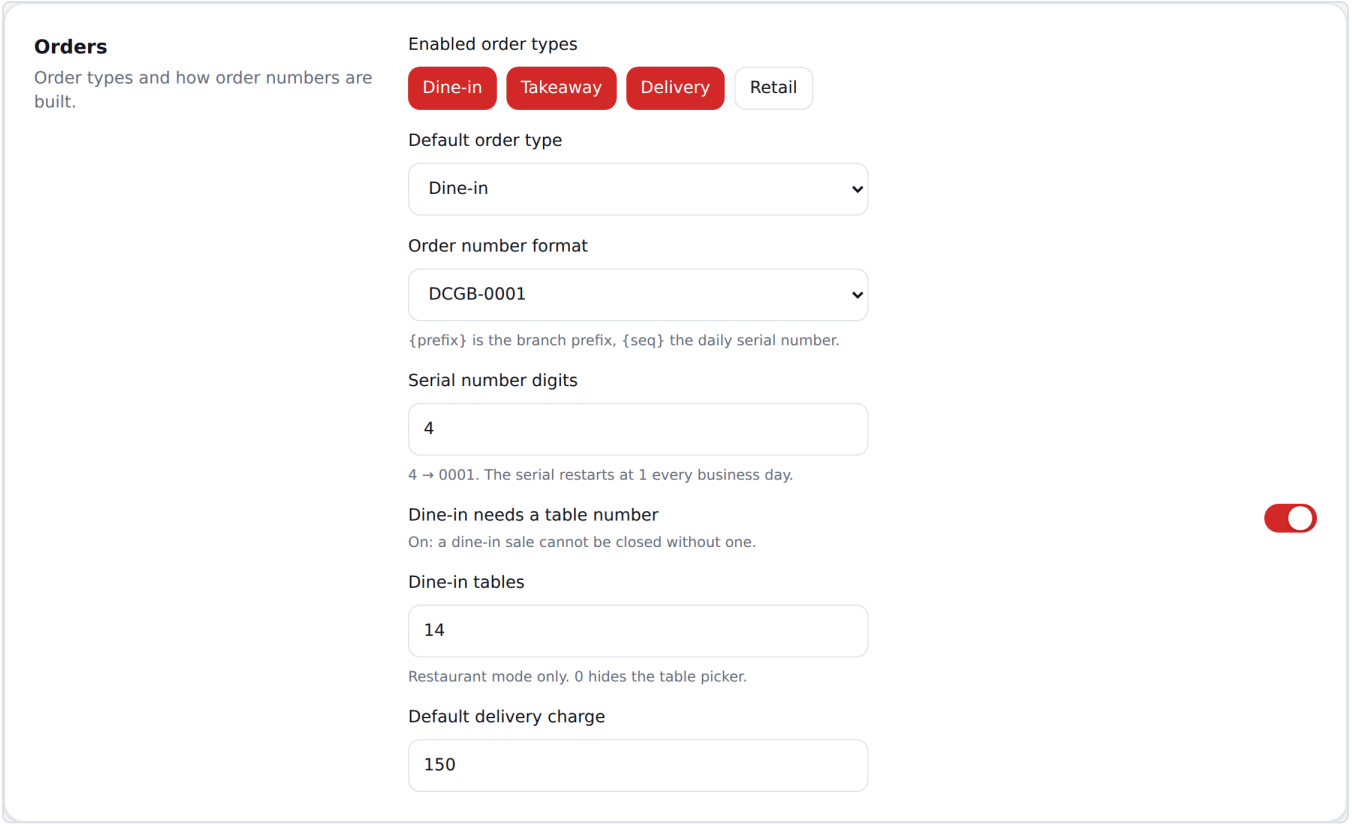


Approvals & Security: what needs a manager PIN.

Setting	Default	What it does
Discount needs a manager PIN	on	A cashier's own discount needs approval
FOC needs a manager PIN	on	Writing a whole order off
Void needs a manager PIN	on	Cancelling a completed sale
Refund needs a manager PIN	on	
Price override needs a manager PIN	on	
Day close reopen needs a manager PIN	on	
Cashier max discount %	0	How much a cashier may give with **no** approval. `0` means always ask
PIN length	4	4–8 digits

A campaign or a promo code never needs a PIN — those are your offers, not the cashier's decision. This setting is only about a discount the cashier types in.

Orders

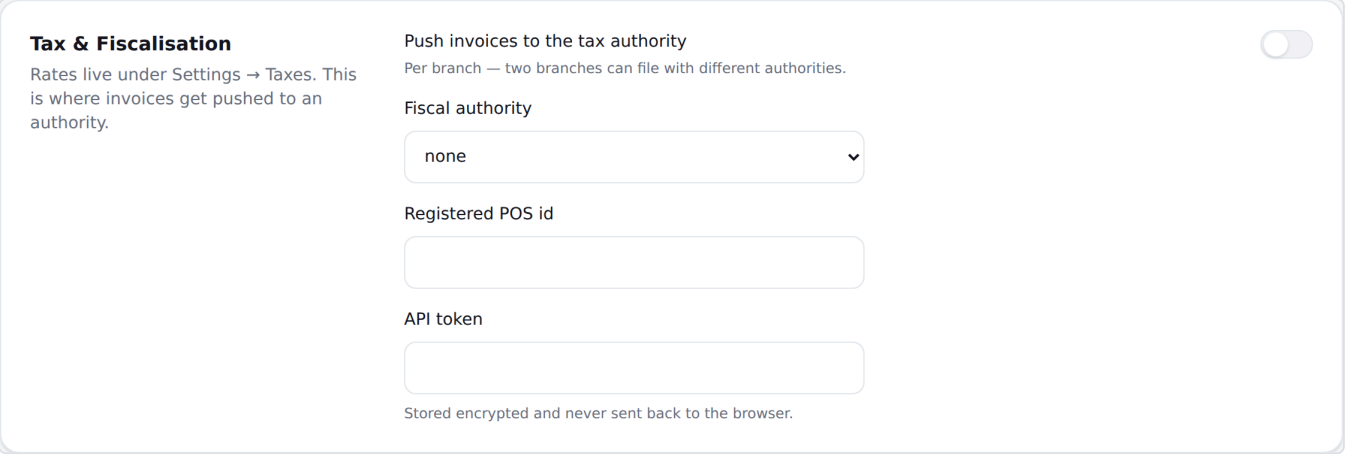


Orders: order types, the table-number rule and the number format.

Setting	Default	What it does
Enabled order types	all four	Which of dine-in / takeaway / delivery / retail this shop takes. A restaurant usually turns "retail" off, a shop turns the other three off
Default order type	takeaway	What the POS opens on
Order number format	`{prefix}-{seq}`	Or `{prefix}-{YYMMDD}-{seq}` to put the date in
Order number padding	4	`0001`
Require a table number	on	A dine-in order cannot be paid **or held** without one
Table count	0	How many tables the shop has
Delivery charge	0	Added to delivery orders only

Order numbers restart every business day, per branch: `DCGB-0001` again tomorrow morning. A held order takes no number — the number is given when the sale is paid.

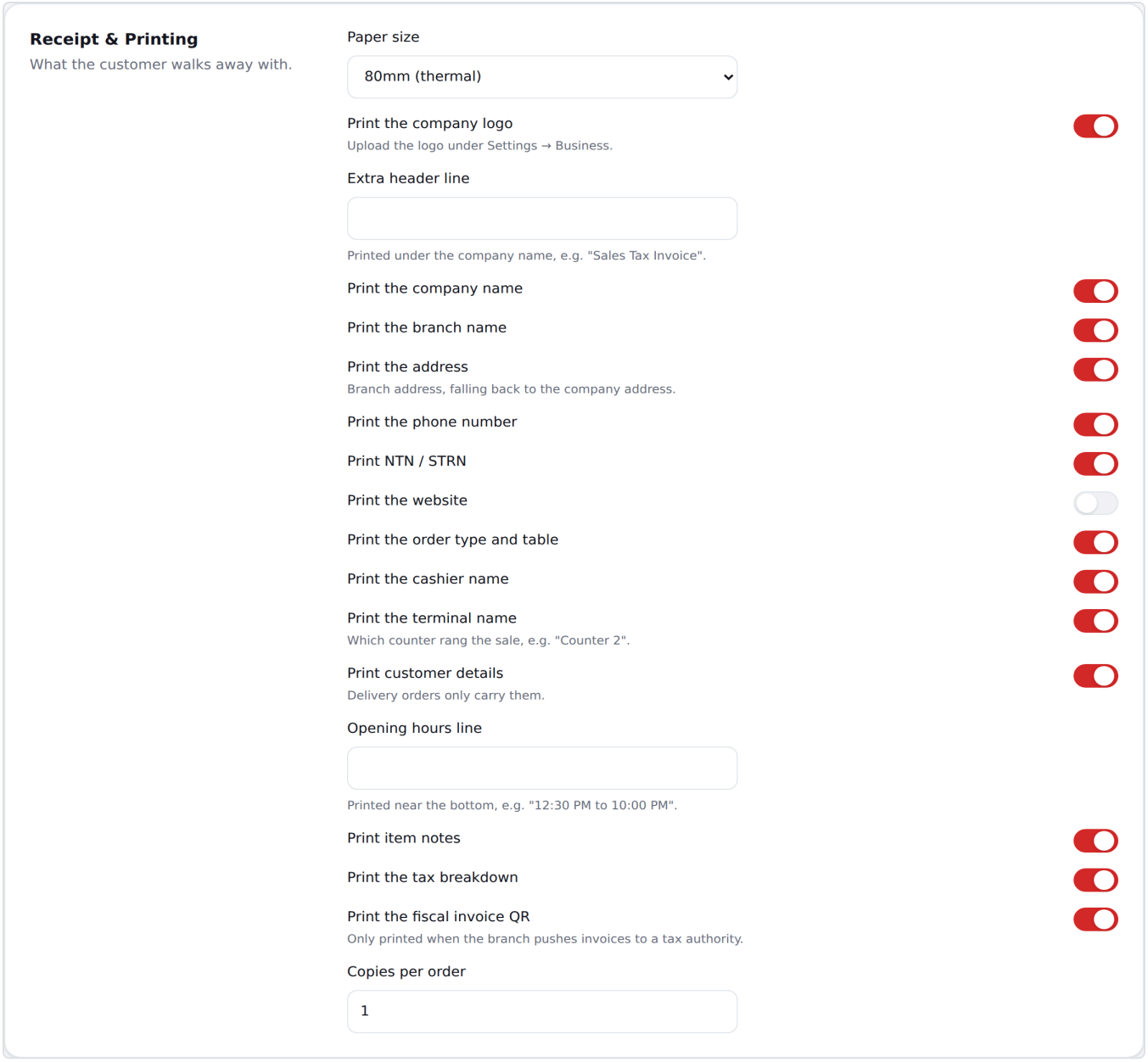
Tax & Fiscalisation



Tax & Fiscalisation: inclusive pricing and the fiscal authority.

Fiscalisation is per branch: switch it on, pick the authority, and enter the POS ID and API token the authority gave you. The token is stored encrypted and is never shown again — the field displays `********` once saved. The tax **rates** themselves are on the Taxes page, not here.

Receipt & Printing

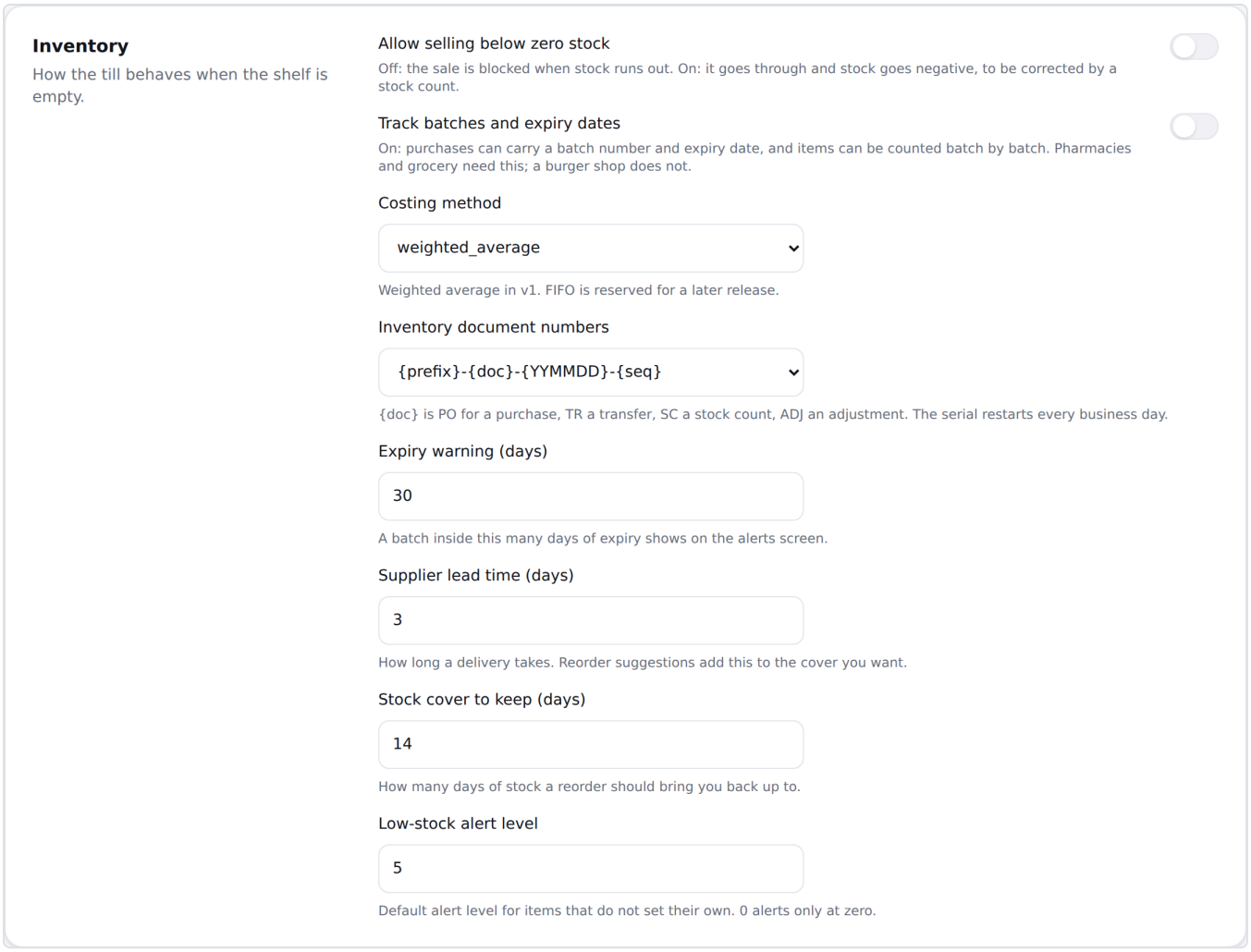


Receipt & Printing: every line the receipt may carry is a switch.

Every line on the receipt is a switch: logo, company name, branch name, address, phone, website, NTN/STRN, order type and table, cashier, customer, item notes, tax breakdown, fiscal QR, an extra header line, an opening-hours line, and how many copies.

Paper size here is only the fallback for a branch with no printer set up. Once you add a printer on the Printers page, that printer's paper wins.

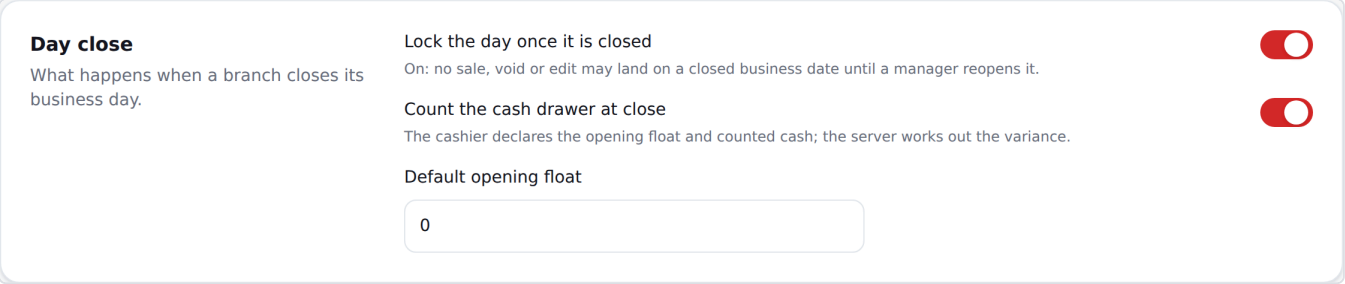
Inventory



Inventory: negative stock, low-stock warnings and document numbering.

Setting	Default	What it does
Allow negative stock	**off**	Off: a sale is blocked when stock runs out. On: the sale goes through and stock goes below zero, to be fixed by a stock count
Track batches and expiry	off	Turn on for a grocery; a burger shop does not need it
Costing method	weighted average	Fixed in v1
Document number format	`{prefix}-{doc}-{YYMMDD}-{seq}`	Purchases, transfers, counts, adjustments
Expiry alert days	30	How early an expiry warning appears
Supplier lead time (days)	3	Used by reorder suggestions
Stock cover to keep (days)	14	Used by reorder suggestions
Low stock alert level	0	The default per item; an item can set its own

Day close



Day close: locking the day and the cash count.

Setting	Default	What it does
Lock the day	on	Once closed, no sale or void may land on that date until a manager reopens it
Require a cash count	on	The drawer must be counted before the day can close
Default opening float	0	What the till usually starts with

Discounts

Discounts: the company cap a manual discount may not pass.

Setting	Default	What it does
Allow a promo code and a campaign together	off	Off: the customer gets whichever is worth more. On: the campaign applies first and the code works on what is left
Maximum discount %	50	A hard cap. A **manual** discount past it is refused outright; a **campaign or code** past it is trimmed to the cap, because a mispriced offer must never stop the till

5

Taxes

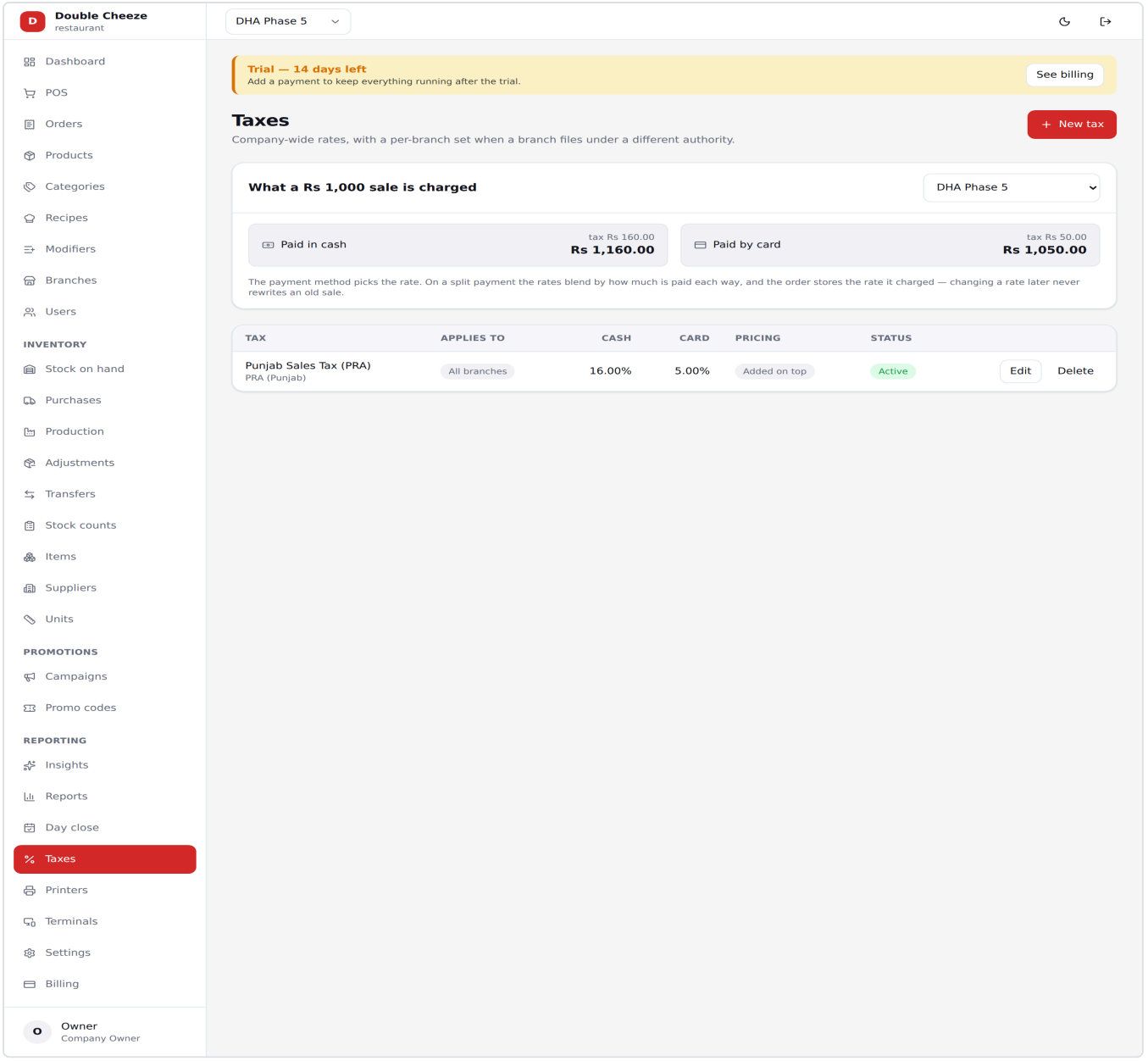
Menu: Taxes · **Who:** owner, branch manager

How tax works here

Two things make Pakistani POS tax different from the usual, and both are built in:

1. **The payment method picks the rate.** Cash and card are taxed differently by most authorities, so every tax has two rates. A split payment blends the two in proportion to how much was paid each way, and the tax collected is reported split by method for your filing.
2. **A branch can file with a different authority.** A company tax set applies everywhere; a branch that defines its own taxes **replaces** the company set entirely — it never merges. That is how a Lahore branch (PRA) and a Karachi branch (SRB) live in one company.

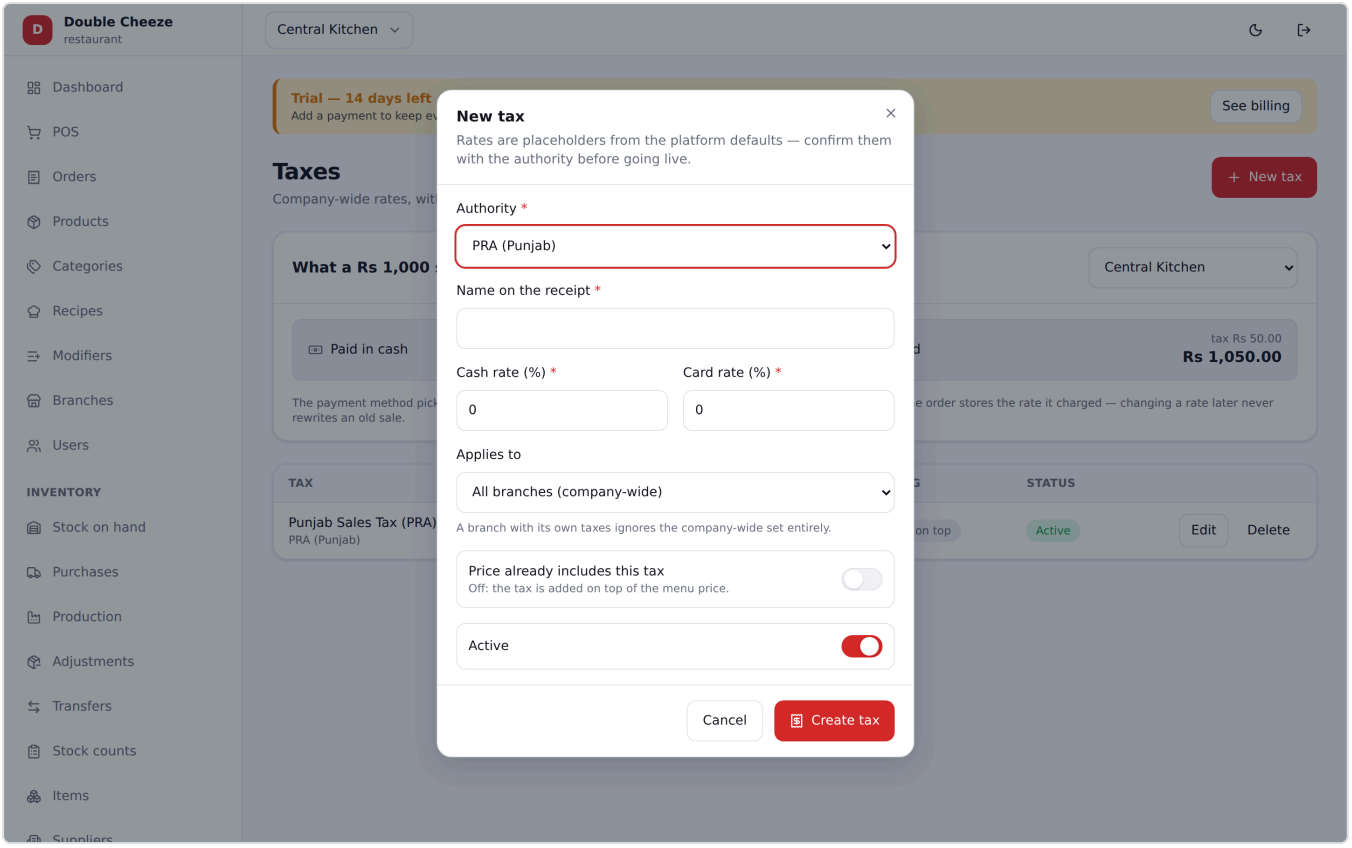
The page



The taxes in force. A branch with its own set replaces the company set entirely.

At the top, **Preview branch** and two boxes — **Paid in cash** and **Paid by card** — showing what a sample amount becomes under the taxes that apply to the branch you picked. Change the branch and watch the numbers change; that is the fastest way to check your setup.

Adding a tax



Adding a tax: a cash rate and a card rate, because the payment method picks the rate.

Field	Notes
Authority	FBR, SRB, PRA, KPRA, BRA, custom, or none. Picking one fills in the name and the placeholder rates
Name on the receipt	What the customer sees: "Punjab Sales Tax (PRA)"
Cash rate (%)	Applied when the customer pays cash
Card rate (%)	Applied when they pay by card
Applies to	Everything, or only certain categories, or only certain products
Price already includes this tax	See below
Active	Turn a tax off without deleting it

Inclusive or exclusive

- **Exclusive** (the box unticked) — the menu says Rs 1,000, tax is added, the customer pays Rs 1,160.

- **Inclusive** (ticked) — the menu says Rs 1,000 and that is what the customer pays; the tax is worked backwards out of it. Most Pakistani restaurants price this way.

Both can sit on the same line; the maths is the same one the receipt and the tax report use.

> **The seeded rates are placeholders.** FBR 15/5, SRB 15/8, PRA 16/5 were entered so a new company can ring a taxed sale on day one. **Confirm them with your authority before your first real customer** and edit them here.

What the system guarantees

- **Old sales never change.** Every order stores its own tax snapshot per line, so editing a rate today does not rewrite last month's filing.
- The tax report groups by **rate**, not just by authority, because a filing is made per rate — 16% cash and 5% card appear as two lines.

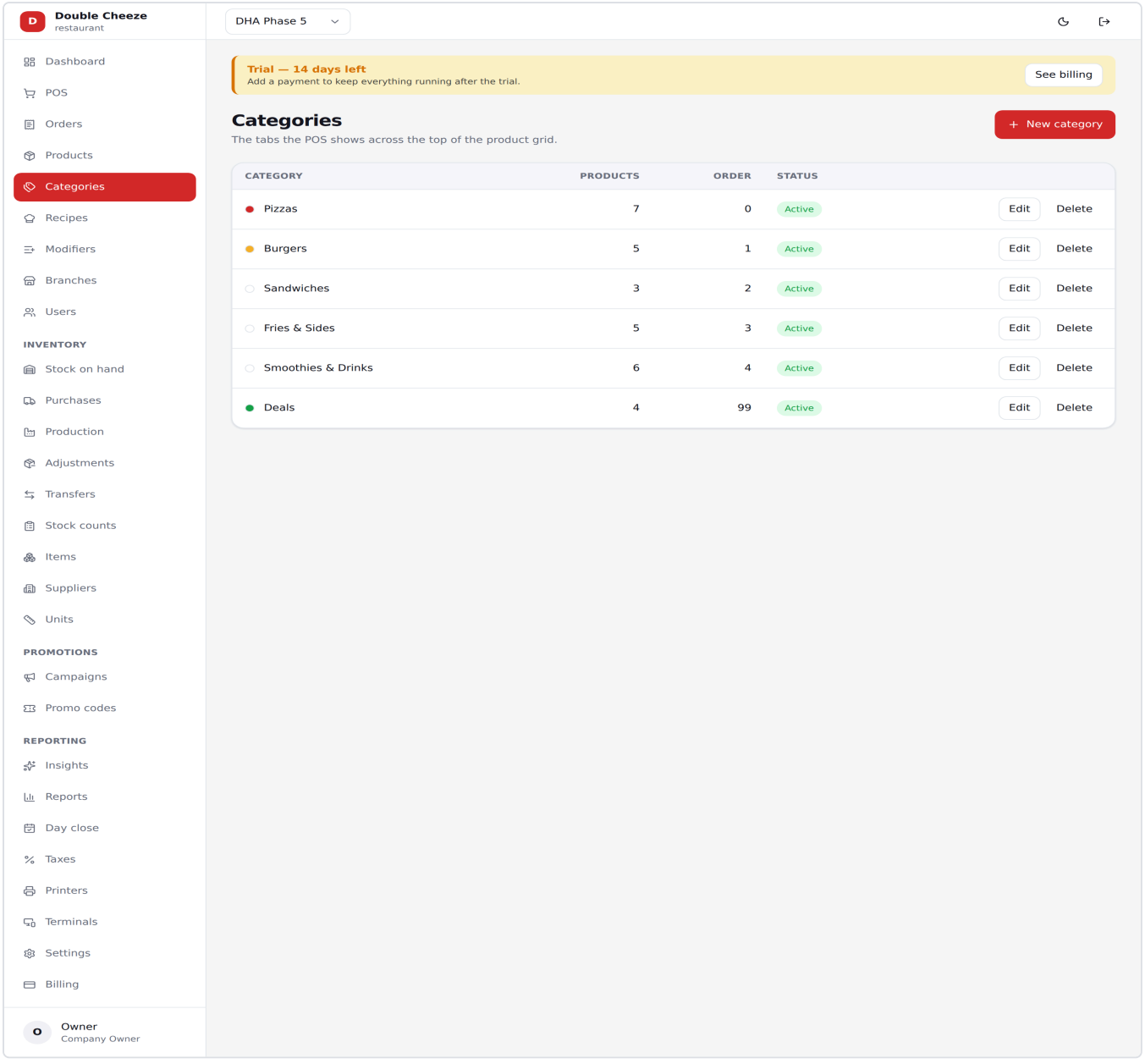
6

Categories and Products

Menu: Categories, Products, Variant options, Modifiers · **Who:** owner, branch manager

Prices are company-wide: one menu serves every branch.

Categories

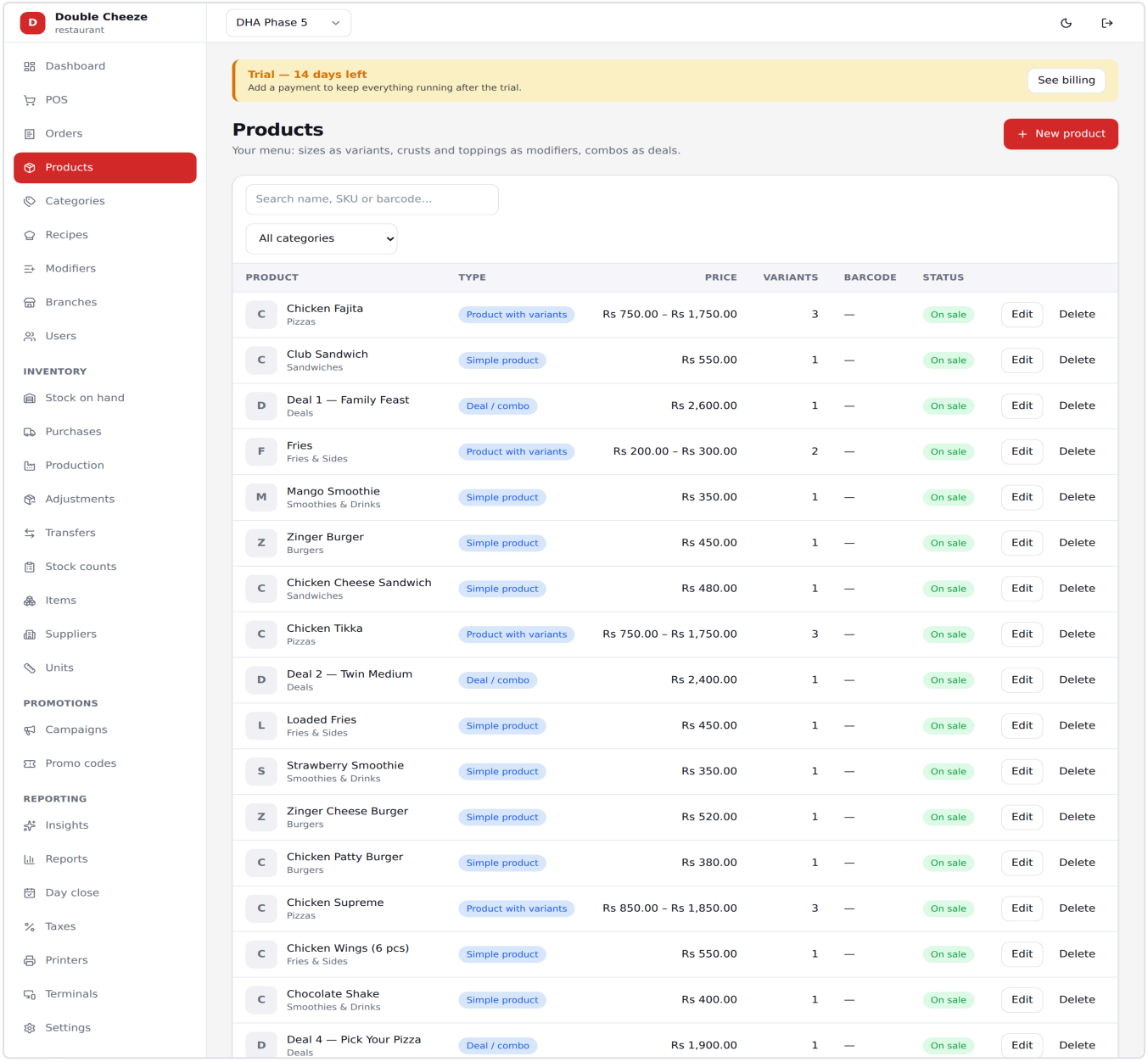


Categories order the POS grid — drag them into the order you want them shown.

The tabs across the top of the POS.

Field	Notes
Name	"Pizzas", "Beverages", "Grocery"
Description	Your own note, not shown at the till
Tab colour	Colours the tab in the POS grid
Sort order	Lower numbers first. Put your best sellers first — it saves taps all day
How the POS browses it	*Grid of products* is the normal till. *Options first* asks for the crust, then the size, then the flavour — see below
Show in the POS	Untick to keep a category for reports but hide it from the till

Products

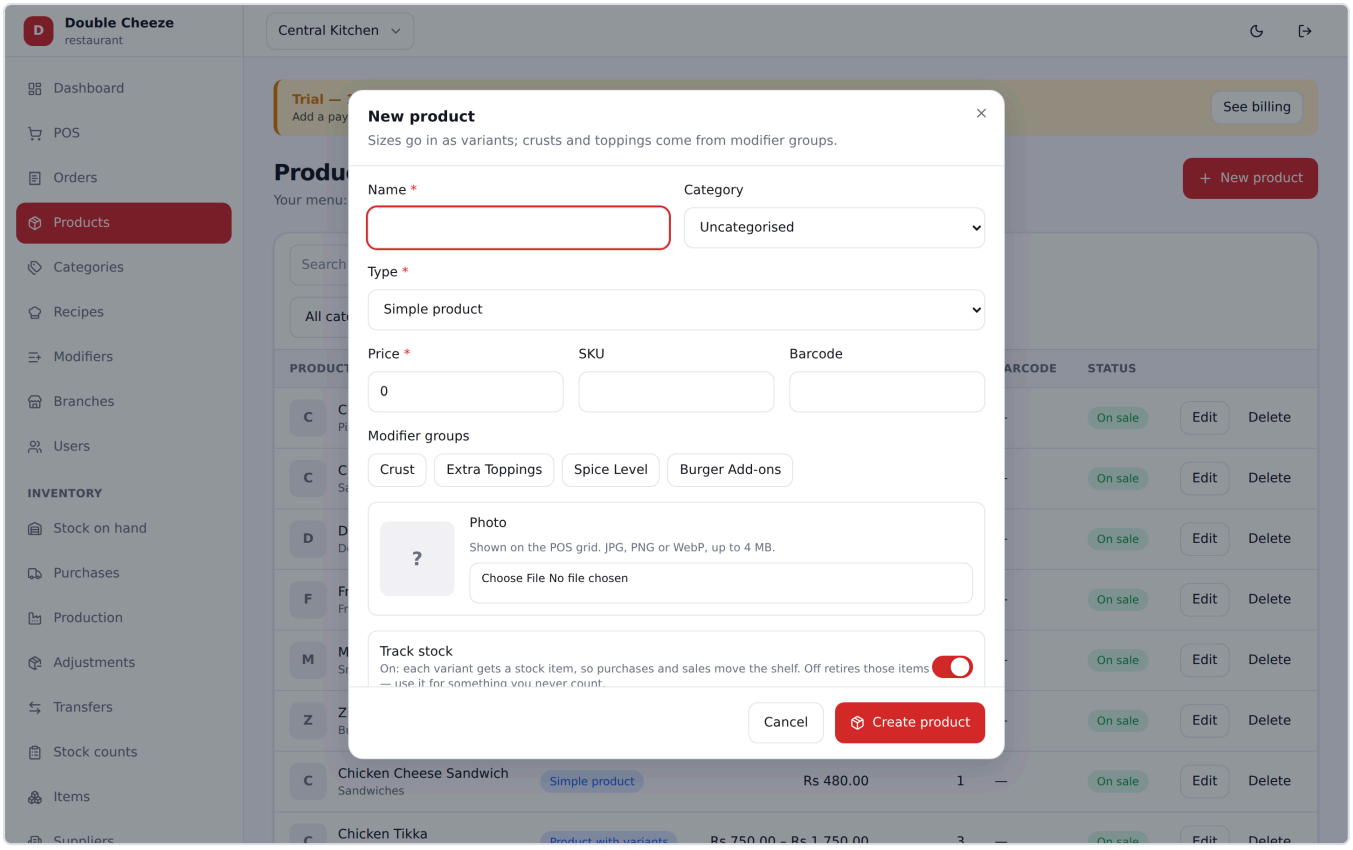


The menu of a restaurant company: each product with its variants and price range.

The one rule to understand

Nothing is sold off a product — a variant is what sells. A simple product still has exactly one variant, called "Regular", created for you. That is why the POS, the stock ledger and every report only ever deal with one shape.

Adding a product



Adding a product. A simple product still carries one variant, called Regular.

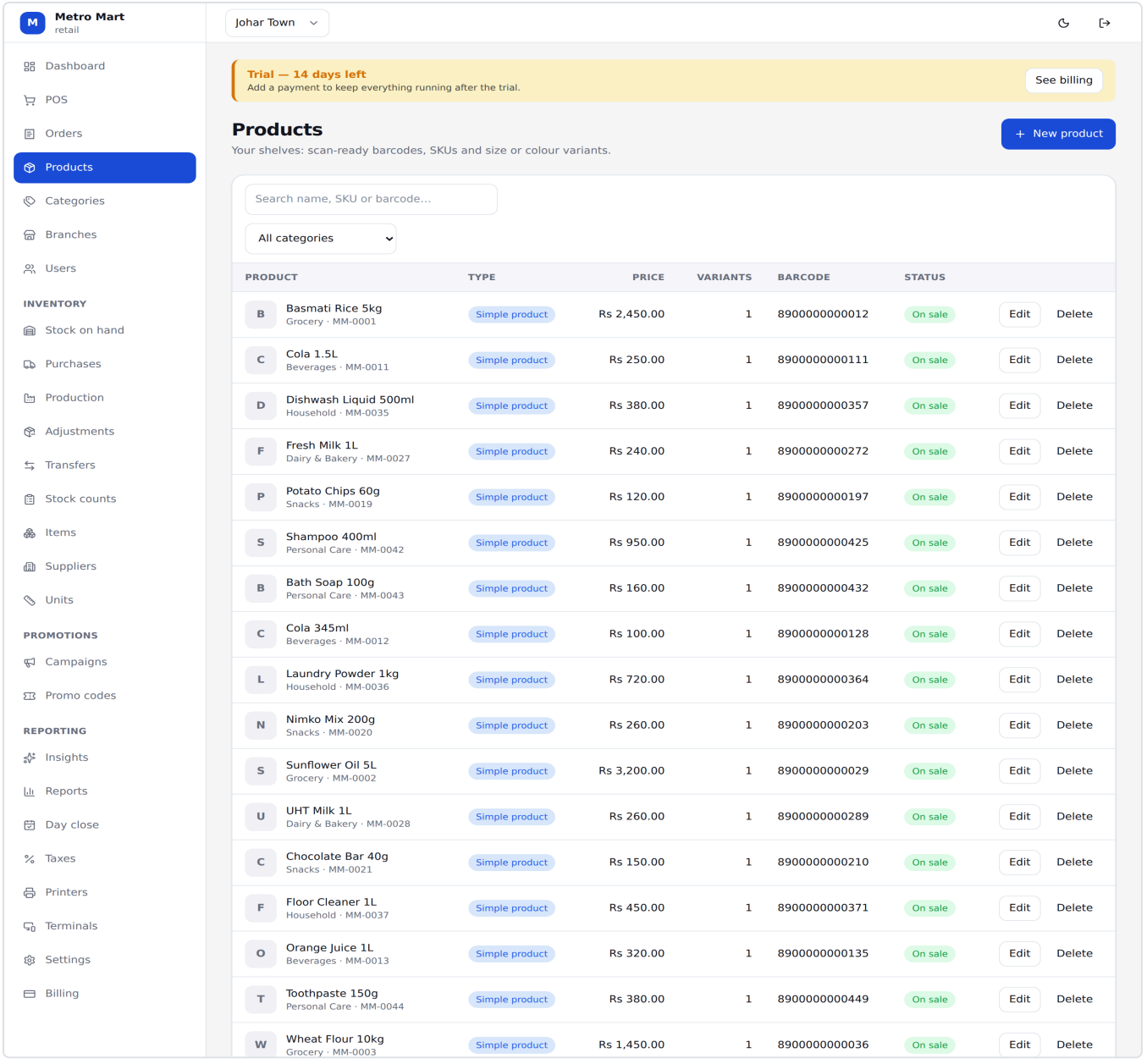
Field	Notes
Name	What the cashier looks for
Category	Which POS tab it appears under
Type	Simple, Variant, Deal, or Service — see below
Photo	Shown on the POS tile; without one the tile shows the first letter
On sale	Untick to pull it off the till without deleting it
Track stock	On by default. On: each variant gets a stock item, so purchases and sales move the shelf. Off retires those items. Not shown for a deal or a service — those never hold stock of their own

Stock happens for you

Track stock is on for a new product, so saving a retail catalog creates a sellable stock item per variant and a 50-product catalog is stock-ready with no second round of data entry. Turn it off for something you never count — and the item it created is retired for you, so nothing is left half-tracked.

A deal or a service never holds stock of its own, so the switch is not offered for them: selling a deal deducts what is inside it.

The four types



The same screen for a retail company — barcodes, SKUs and Track stock.

Type	Use it for	Variants
Simple	One thing at one price — a Coke, a bag of rice	One, "Regular"
Variant	Sizes or flavours — Small / Medium / Large	As many as you like, each with its own price, SKU and barcode
Deal	A restaurant combo — 2 burgers + fries + drink	Holds no stock of its own; selling it deducts what is inside it. A line can be something the cashier chooses at the till
Service	Something with no stock — delivery, a repair charge	One

Deals the cashier answers

A deal is a list of lines, and each line is one of two things:

- **Included** — a fixed item and a quantity. "1 x Large Fajita", "2 x Zinger Burger". Nothing is asked at the till.
- **Cashier chooses** — a name ("Pizza", "Drink"), how many to pick, and the list they pick from. The POS asks for it every time the deal is rung, and nothing is added to the cart until it has been answered.

Each choice can carry an **extra charge**: put 250 against "Double Cheeze Special" and picking it adds Rs 250 to the deal. Leave it at 0 and the deal price stands. Mark one choice as the **Default** and the POS pre-picks it, so the common deal is still one tap.

Whatever is chosen brings its own options with it — the crust and spice level of the pizza inside the deal are asked for too, and a paid option (extra cheese) is charged on top of the deal. What the customer pays is worked out by the server, never by the till.

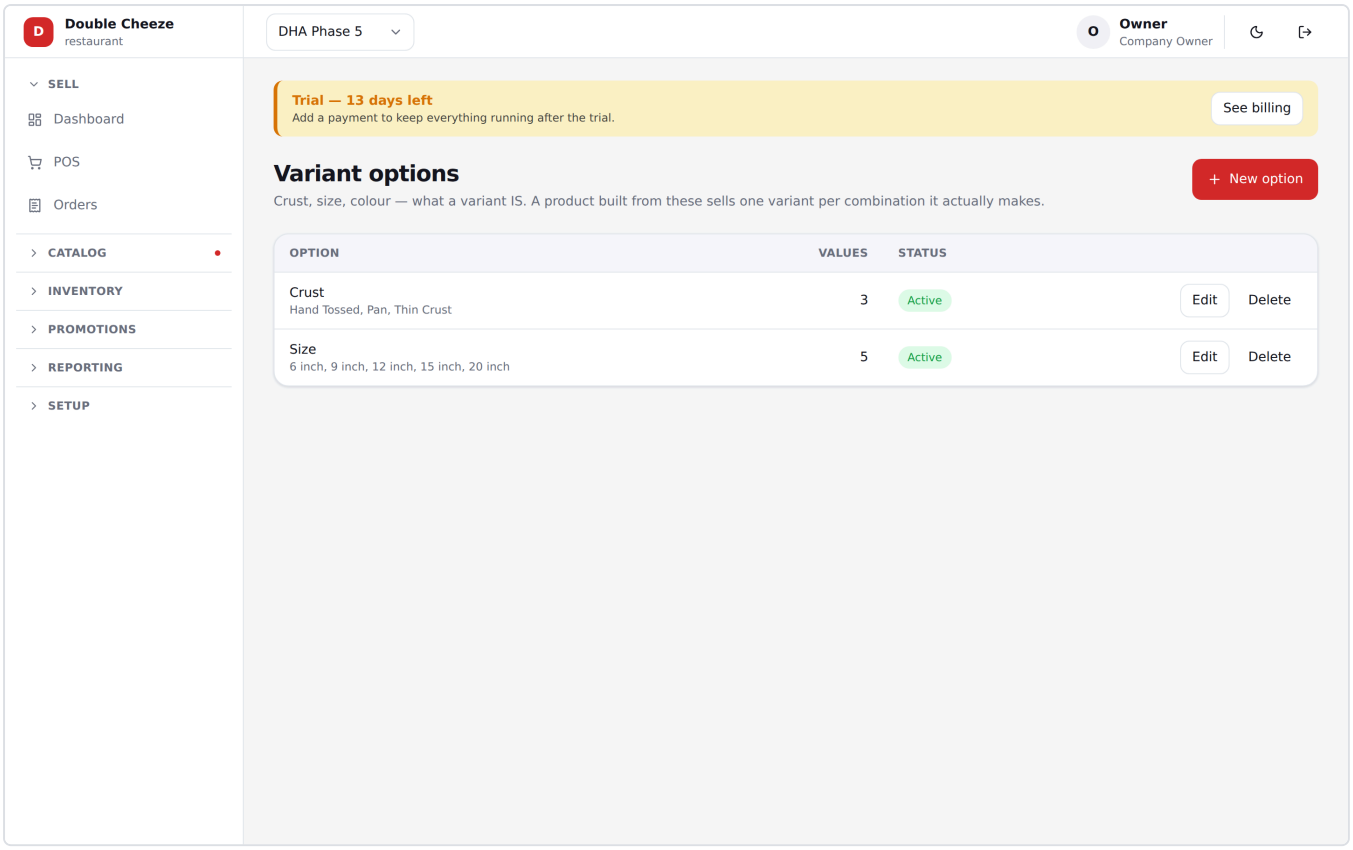
Selling a deal takes what was ACTUALLY chosen off the shelf: pick the water and the water leaves, not the soft drink.

Variants

Field	Notes
Price	What the customer pays (including tax if your taxes are inclusive)
SKU	Your own code, unique per company
Barcode	Scanned at a retail till. Unique per company — two companies may share a barcode

Removing a variant hides it; past orders keep pointing at what they sold.

Variant options — when not every crust comes in every size



Crust and Size, written once for the company. Every pizza reuses them.

Menu: Variant options.

This is the pizza problem. *Hot Chicken Muglai* comes in three crusts and five sizes, but **Pan stops at 12", Thin Crust starts at 9", and only Hand Tossed goes all the way**. Fifteen boxes, ten of them real.

Sellify solves it by making **the combination the thing you sell**. Pan 12" is one variant, Thin Crust 9" is another, and Pan 20" is not there at all — so the till has nothing to grey out and nothing to refuse.

Step 1 — write the options once

Under **Variant options**, add:

Option	Values
Crust	Hand Tossed, Pan, Thin Crust
Size	6 inch, 9 inch, 12 inch, 15 inch, 20 inch

Written once for the whole company. Every pizza reuses them. The order you list the values in is the order the POS shows them in.

Step 2 — price the grid

On a product of type **Variants**, pick **Crust** and **Size** under *Built from options*. The variant list turns into a grid:

	6"	9"	12"	15"	20"
Hand Tossed	900	1400	1900	2600	3400
Pan	1000	1550	2100	—	—
Thin Crust	—	1450	1950	—	—

A box you leave empty is a combination you do not make. No variant is created for it, so it can never be sold, never appears at the till, and never needs a rule of its own. Ten prices typed, ten variants, and the four you skipped simply do not exist.

Each box names itself — `Pan · 12 inch` — so nothing is typed twice. *Opens on* picks which combination the POS starts on.

Untick a box later and that variant is **retired**, not deleted: past orders still read correctly. Tick it again and **the same one comes back**, with its recipe and its stock item intact.

Step 3 — let the till ask in that order

Set the category's **How the POS browses it** to *Options first*. The cashier then taps:

Crust → Size → Flavour.

Each step only offers what you actually make. Tap **Thin Crust** and only 9" and 12" appear, because those are the only two Thin Crust variants on the menu. Change your mind on the crust and the sizes below it refresh.

Anything in the category that is **not** built from options — a garlic bread, a bottle of water — is still shown, as an ordinary grid underneath. Nothing is hidden.

Crust as an option, or as a modifier?

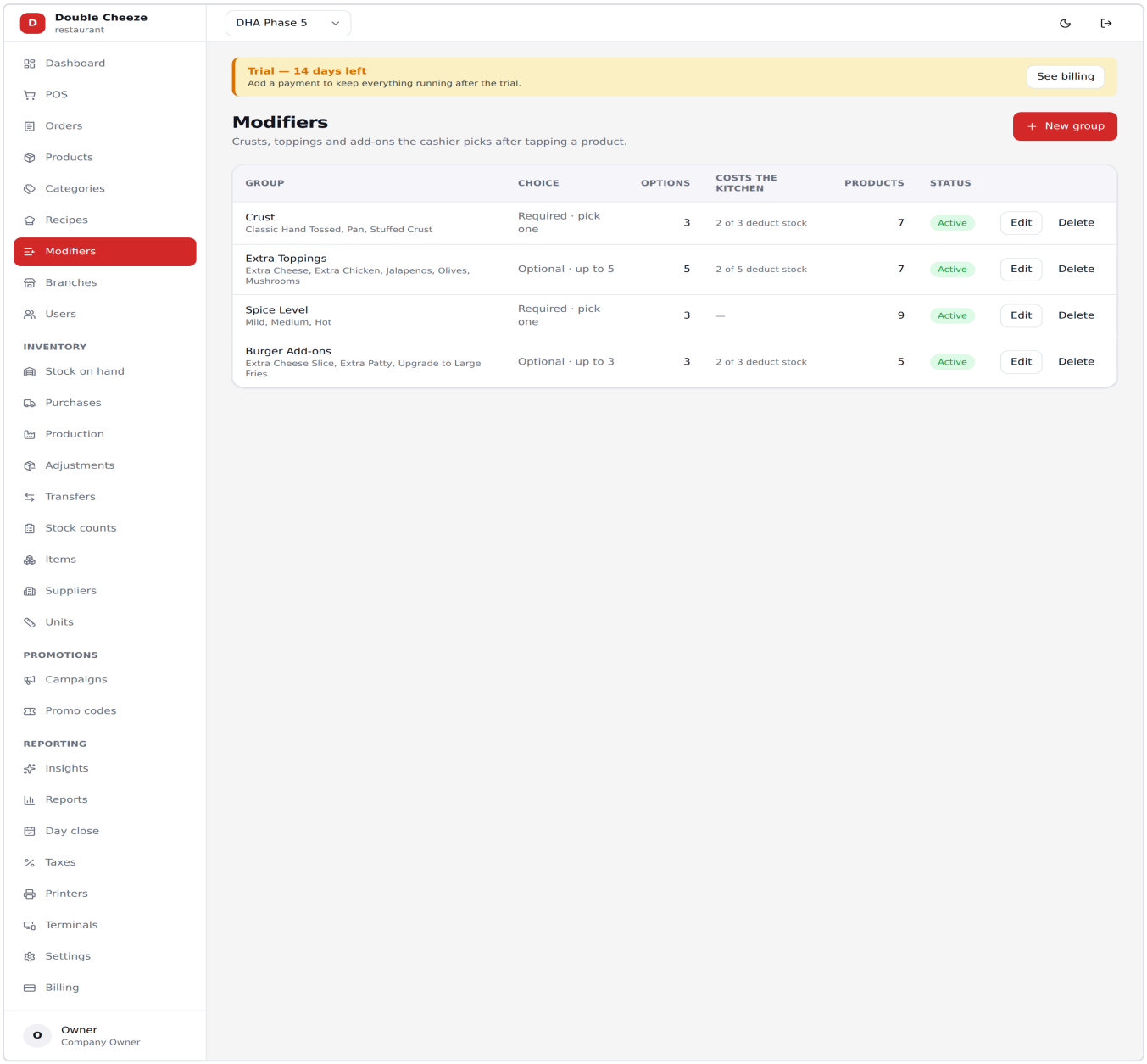
Both exist, and they are for different things.

The question to ask is: **does the shop sell it as a line on the menu, or ask for it at the till?**

Use a variant option when…	Use a modifier when…
It **is** the pizza — Pan 12" is its own line on the menu, at its own price	It is an **upgrade** to the pizza the customer already chose
It has its **own recipe** — a pan base is not a thin base	It is added **on top of** that pizza's recipe
Some crust/size combinations **do not exist** at all	It exists, it is simply not offered on every variant

A recipe belongs to a variant, so a crust that is made from scratch differently has to be a variant option. But a modifier is no longer the poorer choice: an option's charge and its ingredients can follow the size, and it can be linked to the crusts it is offered on — see Modifiers below.

Modifiers (restaurants only)



Modifier groups. An option can be linked to the crusts it is offered on, priced by size, and made of ingredients keyed to either.

Menu: Modifiers. Options attached to a product: crust, size of drink, extra cheese.

Field	Notes
Group name	"Crust", "Add-ons"
Minimum choices	1 makes it compulsory
Maximum choices	1 turns it into a radio button — picking another replaces the first
Cashier must choose	The options popup will not close until they do
Options	Each has a name and a price difference — `+ Rs 150` for extra cheese, `0` for "no onions"
Takes off the shelf	The ingredients this option consumes — **none, one, or several**. Add a row per ingredient
Ingredient / Qty / Unit	Same as a recipe line: 80 **g** of a cheese stocked in kg
Default	Ticked = already chosen when the popup opens, so the common order is one tap
Offered on	Leave everything unticked and the group is asked on every variant. Tick a crust and it is only asked on that crust
Price by	A price per size (or per crust). Leave it empty and the option charges the same everywhere
For (on an ingredient row)	Which variants that row is for — *Every variant*, or one size or crust

An option that costs the kitchen too

"Extra cheese" charges Rs 150 **and** takes 50 g of mozzarella. That deduction is **on top of the dish's own recipe**, and it scales with the line:

```
2 × Chicken Fajita 6" with Extra Cheese
  recipe      90 g × 2  = 180 g
  extra cheese 50 g × 2  = 100 g
                         _________
  mozzarella off the shelf 280 g   (one stock ledger row)
```

The same happens to an option chosen on something **inside a deal** — it scales with that component's own count as well.

An option can consume more than one thing

A stuffed crust is cheese **and** extra dough, so give it two rows:

```
Stuffed Crust   + Rs 250
   Mozzarella Cheese   80 g
   Pizza Dough         50 g
```

There is no limit of two. An option holds as many rows as it really takes — up to 20 — and they can mix freely: some rows written for a size, some for a crust, some for nothing in particular. A row left on *Every variant* applies on all of them, so this is a perfectly ordinary option:

```
Stuffed
   Mozzarella Cheese  100 g   for  12 inch     ← follows the size
   Mozzarella Cheese   60 g   for  9 inch
   Pizza Dough         60 g   for  Pan         ← follows the crust
   Pizza Dough         50 g   for  Hand Tossed
   Chicken Breast      45 g   Every variant    ← always
   Cooking Oil         10 ml  Every variant
```

A **Pan 12 inch** takes four of those rows: 100 g cheese, 60 g dough, 45 g chicken, 10 ml oil.

Anything you do not want to spell out on the option — a filling made of several things — is better written as a **prep item** with its own recipe (see Recipes); point the option at that one item and it explodes into its parts.

An option that belongs on some variants only

A stuffed crust is not sold on a thin base. Under **Offered on**, tick the crusts it belongs on and the group simply is not asked anywhere else — the cashier never sees it, and the API refuses it if something tries.

- Tick **two values of the same option** (Pan and Hand Tossed) and either of them will do.
- Tick **values of two different options** (Pan and 12 inch) and both must match.
- Tick **nothing** and it is asked everywhere. That is how every group behaved before this existed, and it still does.

An option whose price follows the size

A stuffed crust is not one price. Under **Price by**, write a row per size:

```
Stuffed         base price 0
   9 inch    + Rs 150
   12 inch   + Rs 300
   15 inch   + Rs 400
```

The size the cashier chose decides which line is charged; a size with no line of its own falls back to the option's own price. **A price can only vary one way** — by size *or* by crust. Sellify

refuses a price written against both, because a Pan 12 inch would match two rows and neither would be right.

An option whose ingredients follow the size AND the crust

Ingredients are different: they are written **row by row**, so one option can vary several ways at once. That is how a stuffed crust is actually made — the cheese goes by size, the dough goes by crust:

```
Stuffed
   Mozzarella Cheese   60 g   for  9 inch
   Mozzarella Cheese  100 g   for  12 inch
   Pizza Dough         60 g   for  Pan
   Pizza Dough         50 g   for  Hand Tossed
```

A **Pan 12 inch** takes the 12 inch cheese row **and** the Pan dough row — 100 g of cheese and 60 g of dough, which then explodes into flour and oil. A **Hand Tossed 9 inch** takes 60 g and 50 g.

Rows for different ingredients all apply together. Among the rows for the **same** ingredient, the one written for a value the pizza actually is beats the row marked *Every variant*. Two rows for the same ingredient on two different options — cheese for *12 inch* and cheese for *Pan* — are refused, for the same reason a price cannot vary two ways.

Prep items are welcome here

An ingredient that is itself made by a recipe **explodes**, exactly as it does inside a recipe: `Pizza Dough 50 g` never comes off a dough shelf — it becomes flour and oil. So a crust option costs the shelf the same way the pizza does.

When the same thing arrives twice, it is added up, not replaced. If your option lists 10 ml of cooking oil *and* takes dough that is itself part oil, the shelf gives up both — 10 ml plus the 2.4 ml hiding inside the dough is **12.4 ml**. That is what the kitchen really used, so that is what comes off and what the food cost is worked out on.

Two more rules worth knowing:

- The unit list only offers units that convert into how the item is stocked (g for a kilo, ml for a litre) — anything else would write a deduction the shelf cannot take.
- An option with no ingredients is not a mistake. A spice level usually costs nothing extra to make; leave its list empty.

The Modifiers list shows `2 of 5 deduct stock` per group, so you can see at a glance which groups are only charging money.

A company without the restaurant flag cannot create modifiers or deals at all — the API refuses them, not just the screen.

A deal that offers a whole size of the menu

A choice line inside a deal can either **list** what it offers, or **describe** it:

These items	Pick each variant by hand — right for "Coke or Sprite"
Anything matching	A category and the option values it must carry — right for "any 9 inch pizza"

On the product form, a deal line offers three buttons: **Included**, **These items** and **Anything matching**. Pick the last one and the line asks what it should reach:

```
Ask for: Pizza            How many: 1

From: Pizzas                          ← empty = the whole menu
CRUST — none ticked means any    [Hand Tossed] [Pan] [Thin Crust]
SIZE  — none ticked means any    [6"] [9"✓] [12"] [15"] [20"]

Offers 3 items today — and whatever is added later.

Upgrade charge:  ● Per option    ○ From the price difference
   Crust · Pan  [100]
```

The line under the chips is the one to watch: it counts what the rule reaches **right now**, so a rule that offers nothing is obvious before you save it — and the API refuses that one anyway.

Two things follow, and both save real work:

- A flavour added to the menu next month is **in the deal that day** — nobody edits the deal.
- A combination the shop does not make can never be picked, because it has no variant.

The upgrade charge is written one of two ways:

- **Per value** — Pan +100. Charges on different axes add up (a Pan 12 inch pays both).
- **Automatic** — the difference from the cheapest thing the line offers. Hand Tossed 9 inch at 1,400 is included, so a Pan at 1,550 costs +150 with nothing to maintain.

At the till the cashier is asked **crust, then flavour** — two taps instead of hunting through forty-eight buttons.

Common mistakes

- **Two variants with the same name.** Give each a real name; "Regular" is reserved for the automatic one.
- **A deal with no items inside it.** It will sell for its price and deduct nothing.
- **A choice line with no name.** The till has nothing to ask for, so the API refuses it.

- **Putting a whole deal inside another deal.** Refused — a deal has no contents of its own to explode, so the kitchen would get an empty ticket.
- **Forgetting Track stock on retail products.** Without it there is no stock item, so the shelf never moves when you sell.
- **Trying to mark a combination "unavailable".** There is no such switch. Leave the price box empty — a combination with no price is a combination that does not exist.
- **Making crust a modifier when it changes the price by size.** A modifier adds the same charge to every size and cannot carry its own recipe. Use a variant option.
- **Setting a category to Options first when nothing in it is built from options.** The till simply falls back to the grid — harmless, but it did nothing.

7

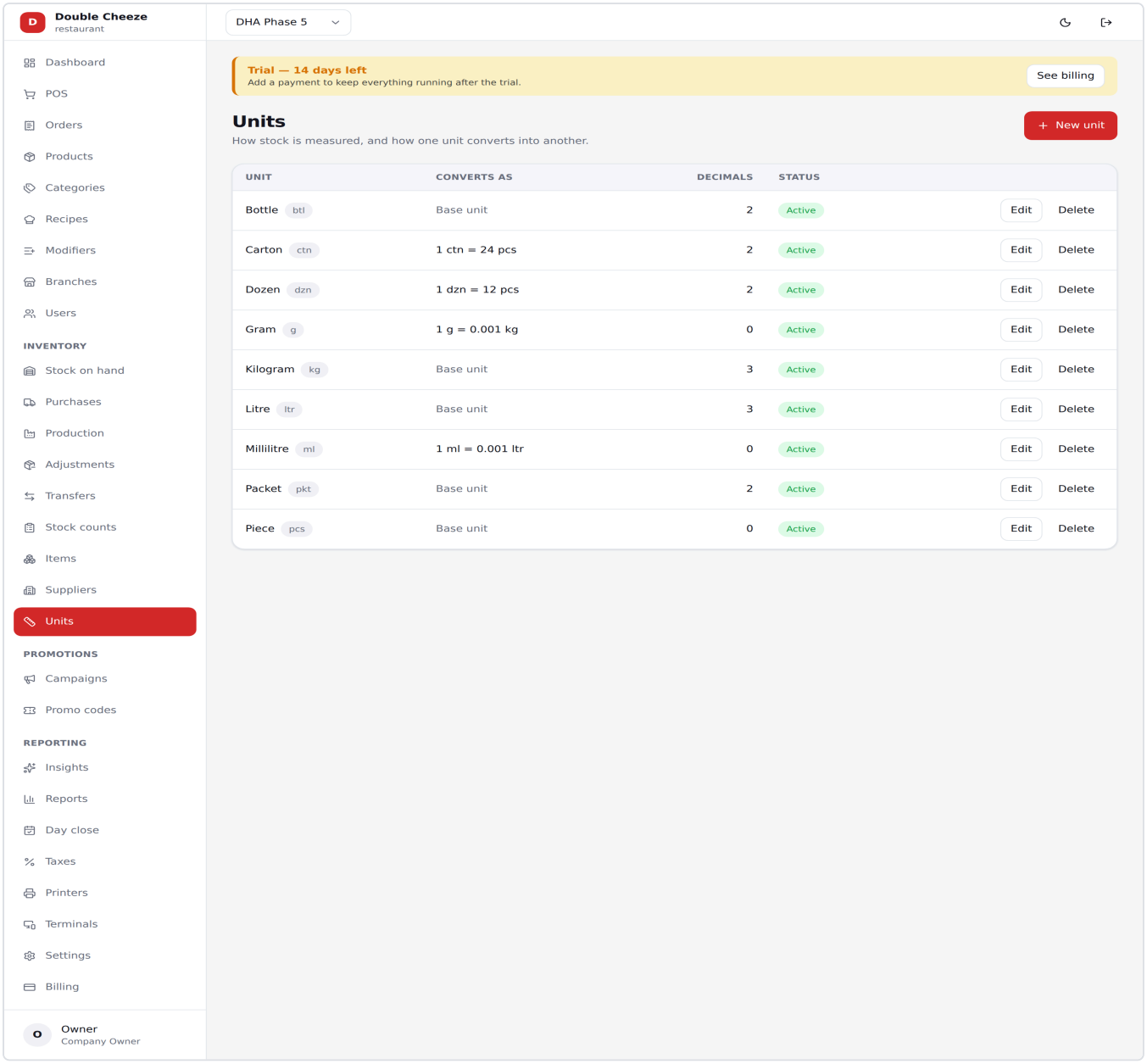
Inventory

Menu: Inventory section · **Who:** owner, branch manager

Stock lives per branch. Both the quantity *and* the cost are kept per branch, so one shop buying cheese dear does not move another shop's food cost.

Set up in this order: **Units → Suppliers → Items → Purchases**.

Units



Units, and the conversion that lets a recipe be written in grams and stocked in kilos.

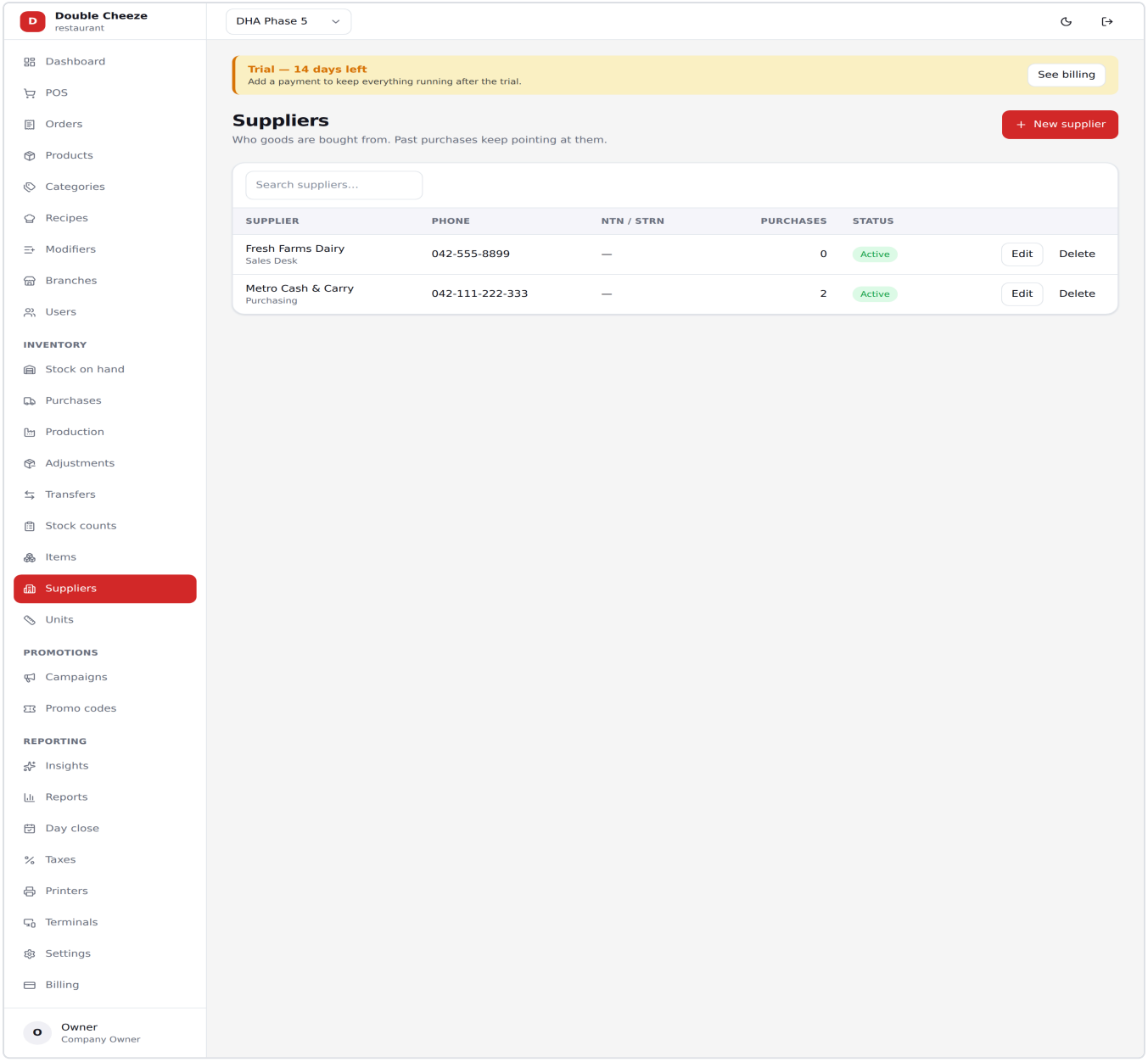
Menu: Inventory → Units

How stock is measured, and how one measure converts into another. Every company starts with a standard set (kg, g, ltr, ml, pcs, dozen, carton); rename or add as you like.

Field	Notes
Name / **Short code**	"Kilogram" / `kg`
Converts to	The base unit this one is expressed in. `g` converts to `kg`
How many	`0.001` — one gram is 0.001 kilos
Decimals shown	3 for weights, 0 for pieces

A unit converts one step, straight to a base unit. You cannot chain g → kg → tonne, and two units cannot point at each other. That keeps every conversion exact.

Suppliers

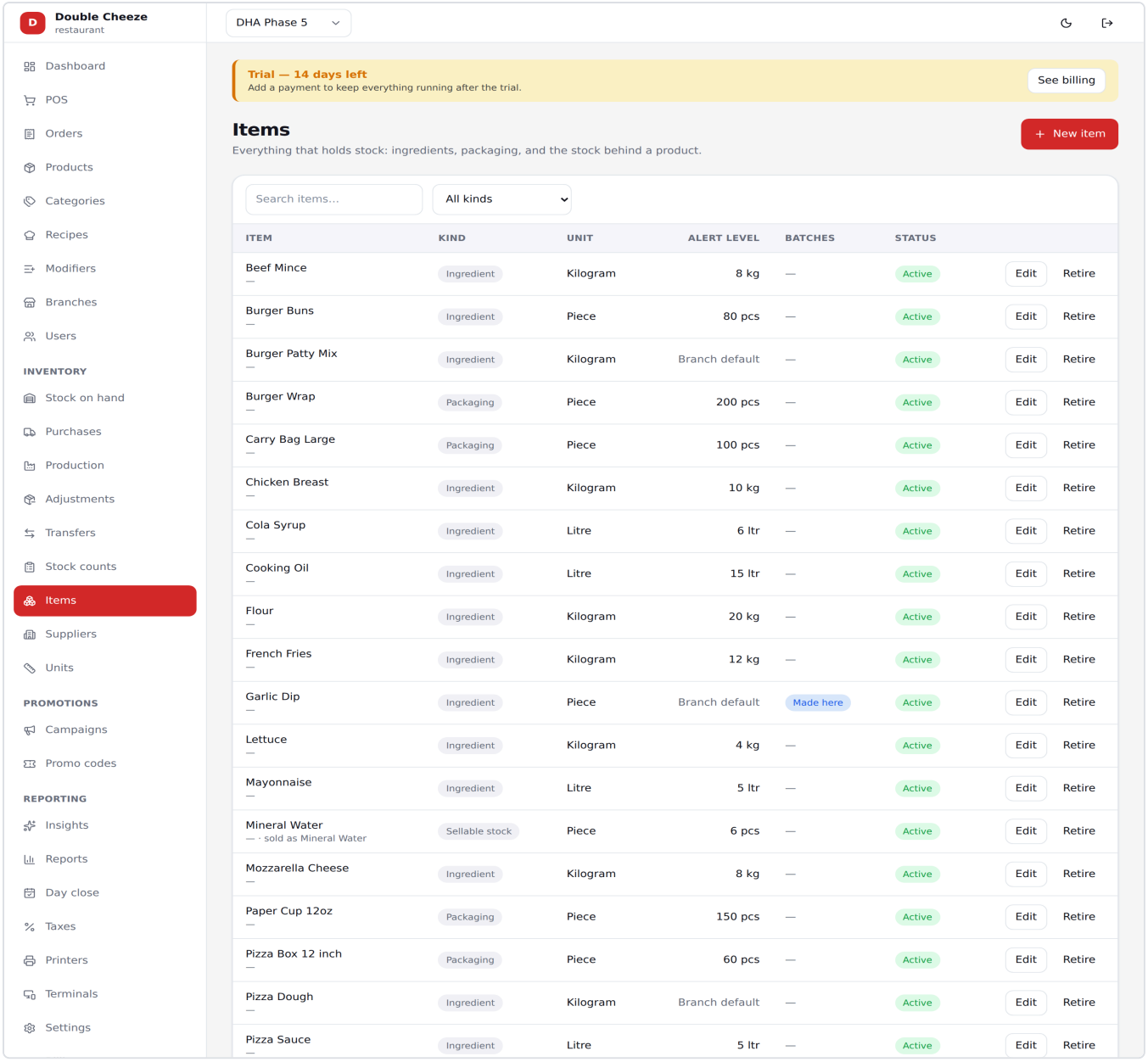


Suppliers. Adding a deleted one back brings its purchase history with it.

Menu: Inventory → Suppliers

Field	**Notes**
Name	Unique in your company
Contact person, **Phone**, **Address**	
NTN, **STRN**	Their numbers, for your input tax records

Items



Stock items. A retail product with Track stock on gets one automatically per variant.

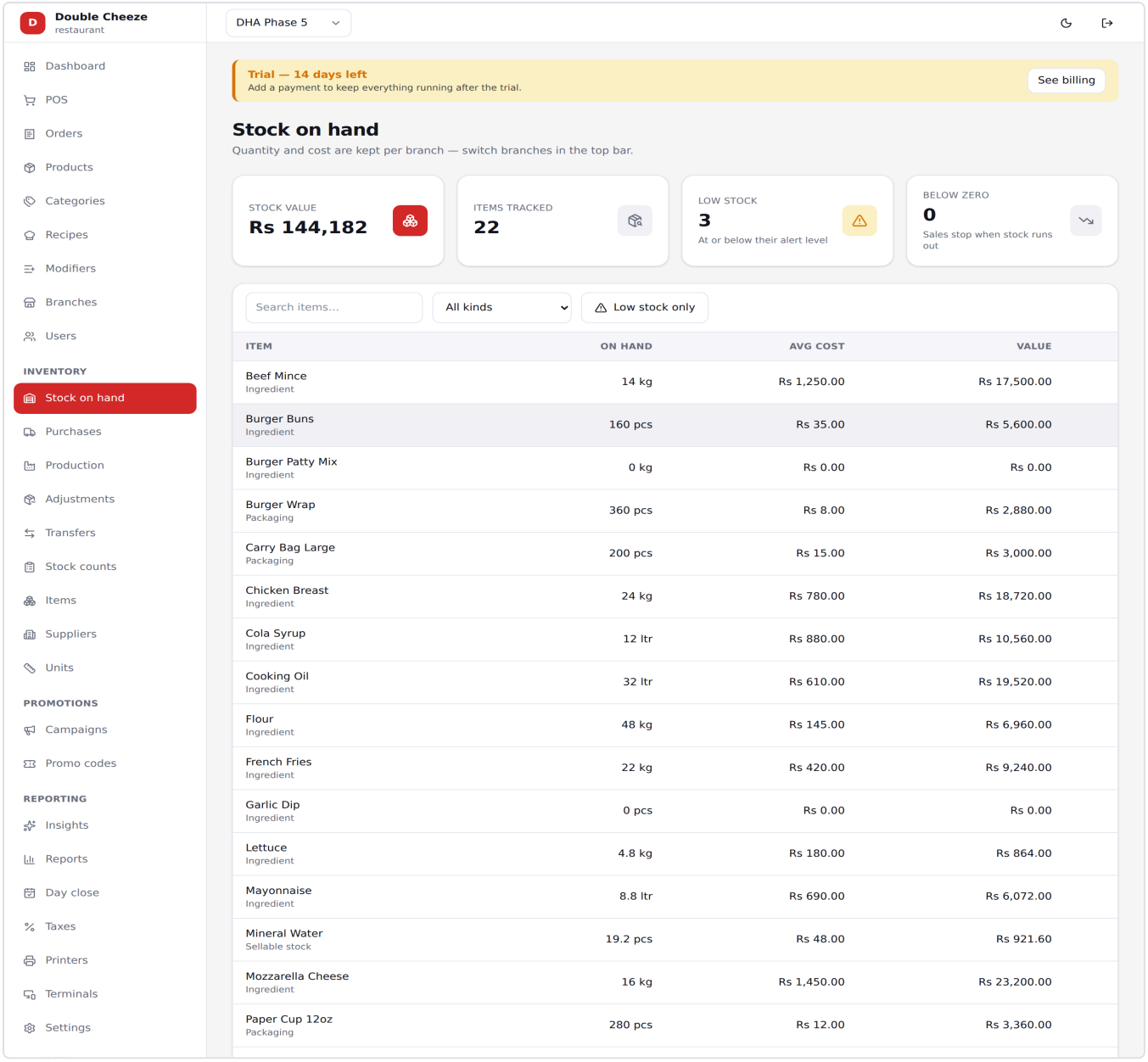
Menu: Inventory → Items

An item is a thing on a shelf. It is not the same as a product: a product is sold, an item is stocked.

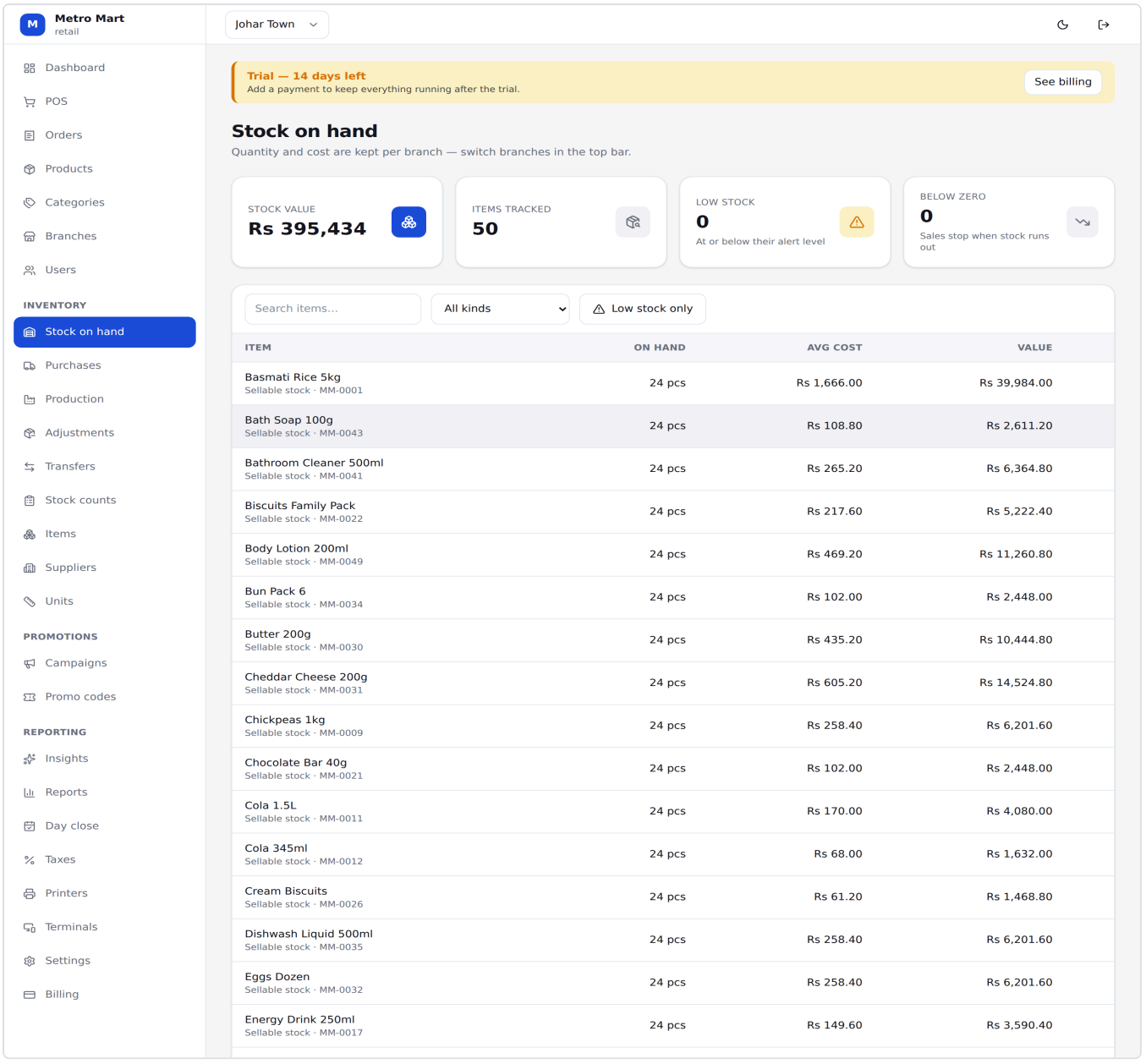
Field	Notes
Name	"Mozzarella Cheese"
Kind	**Ingredient** (used by recipes), **Sellable stock** (sold as-is), **Packaging** (boxes, bags)
Unit	How it is stocked — buy in cartons, stock in pieces
SKU	Optional
Alert level	Below this it shows as Low on the stock page
Track batches and expiry	Per item, if your company has batch tracking on

For retail, ticking **Track stock** on a product creates its sellable item for you. You can still rename it here.

Stock on hand



Stock on hand at the branch in the top bar, with its weighted average cost.



The same screen for a retail branch.

Menu: Inventory → Stock on hand

What is on the shelf **at the branch in the top bar**, valued at that branch's own average cost. Four figures at the top: stock value, items tracked, low stock, below zero.

Click any row to see its **ledger** — every movement that made the number what it is, with the balance and average cost after each one. If a number looks wrong, the answer is always in the ledger.

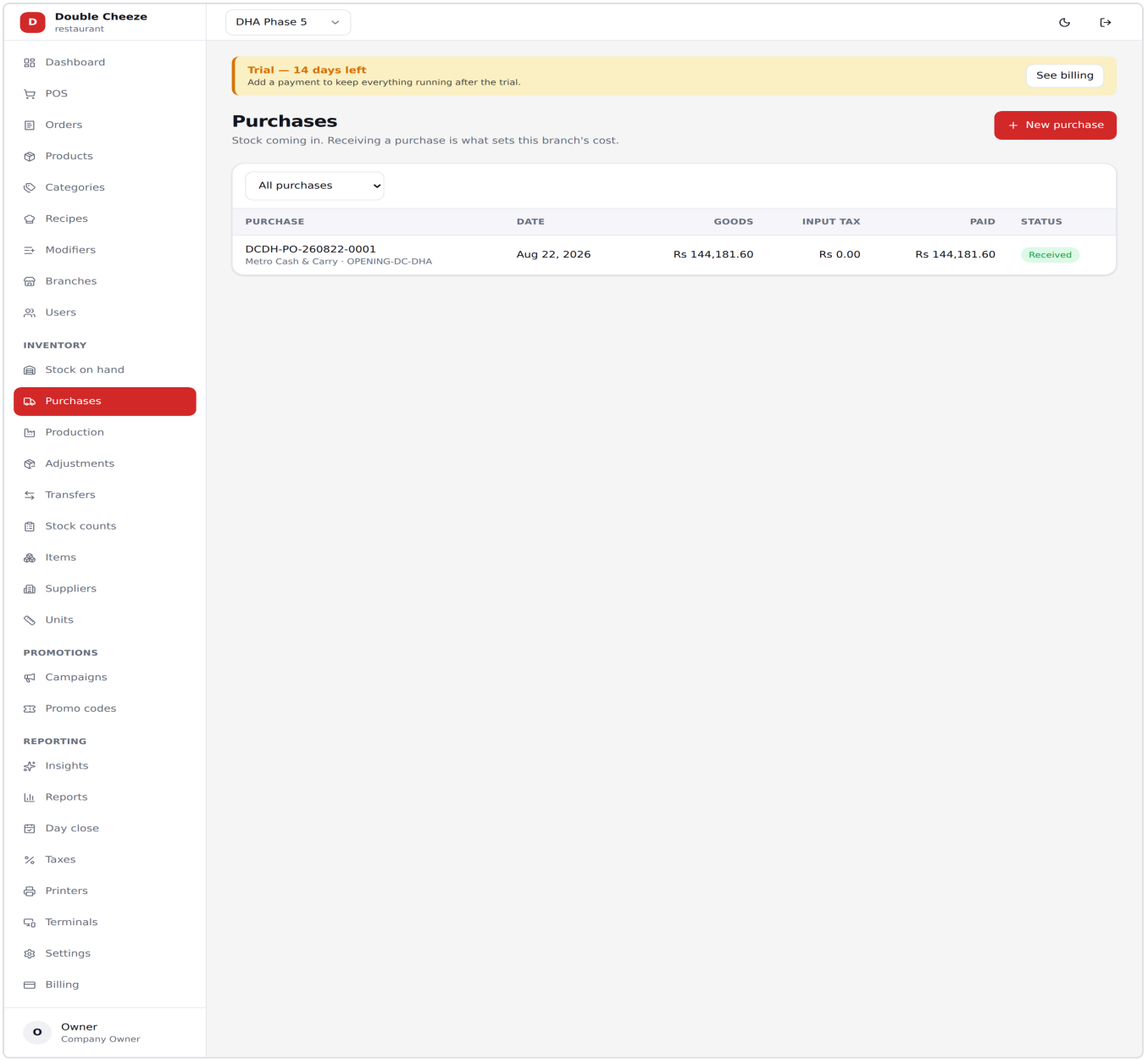
> `stock_ledger` is the only truth about stock. The on-hand figure is a running balance written at the same moment as the ledger row, never edited by hand.

Production

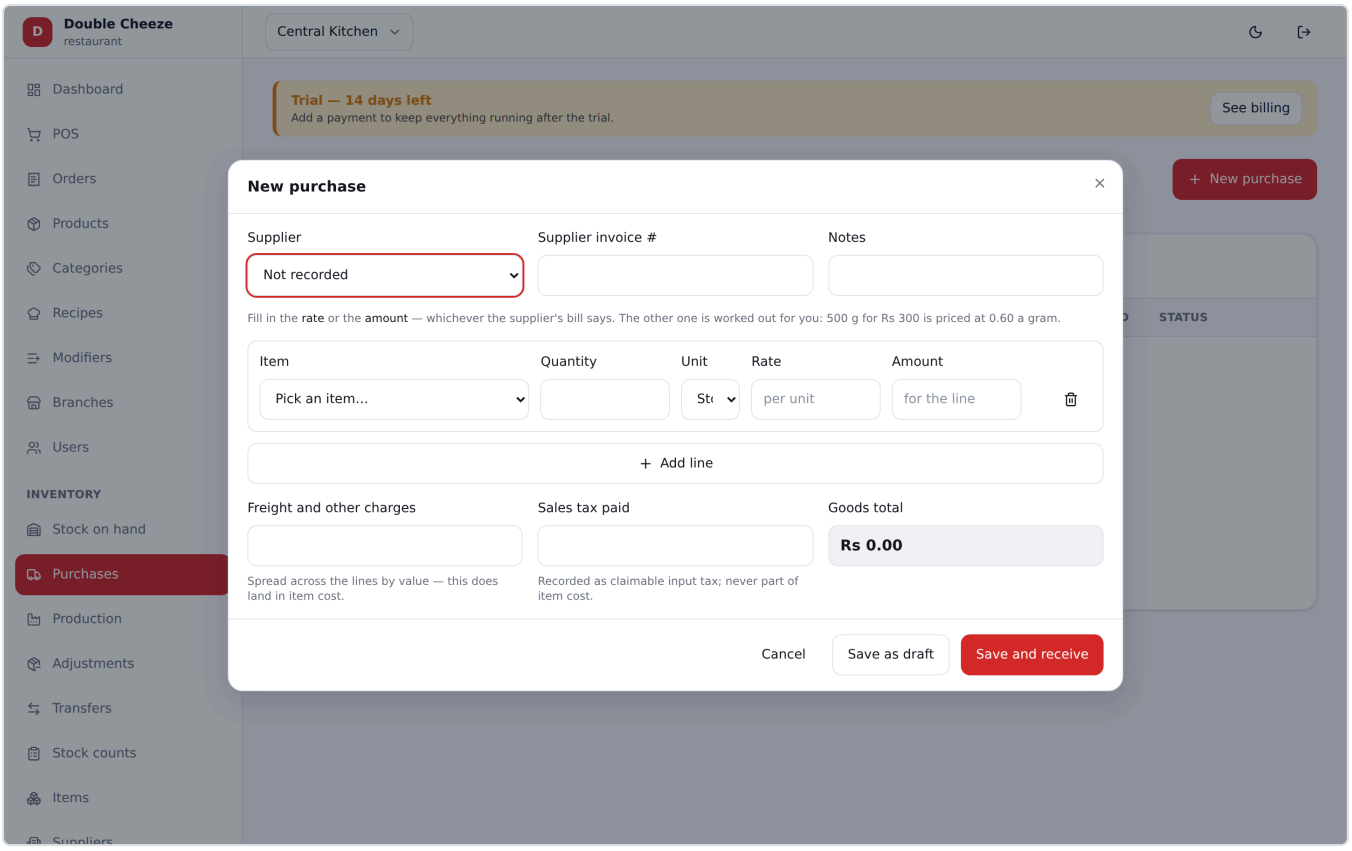
Menu: Inventory → Production. A batch the kitchen made: raw material off this branch's shelf, the finished item onto it, priced by what it actually consumed. It is how a commissary fills the shelf a shop later sells from — and how a shop that makes its own dough records it.

Only items with **Made in batches** on (and a recipe) can be produced.

Purchases



Purchases. A received purchase is corrected, never quietly edited.



Entering a purchase: a line is priced either by rate or by the amount actually paid.

Menu: Inventory → Purchases · *This is also how you enter opening stock.*

Field	Notes
Supplier	
Supplier invoice #	Their bill number, for your records
Expected delivery	The day the supplier promised. Optional — leave it empty if nothing was agreed
Lines	Item, quantity, and the price — see below
Batch number / **Expiry date**	Per line, when the item tracks batches
Freight and other charges	Spread across the lines by value and **added to cost**
Sales tax paid	Your claimable input tax — recorded, but **not** part of item cost

Price: rate *or* amount paid

A shop bill says **"500 g for Rs 300"**, not "Rs 0.60 a gram". So a line takes either:

- a **rate** per unit, or
- the **amount paid** for that line.

Whichever you type, the other is worked out for you, and **the amount you typed is the truth**: 7 kg for Rs 1,000 stores a rate of 142.8571/kg and a line total of exactly 1,000.00, however the rate rounds. The browser never divides it itself.

At the bottom you see **Goods**, **Freight and charges (spread into cost)**, **Input tax (not part of cost)** and **Paid to supplier**.

Order first, goods later

Saving a purchase never touches the shelf. An order is usually placed days before the van turns up, so **Save purchase order** records what you asked for and nothing else. It sits as a **Draft**, and the list shows what you ordered against the **Expected** day — in red, marked *overdue*, once that day has been and gone and the goods still have not.

A draft is edited freely: open it and press **Edit**. Quantities change on delivery, a rate turns out different, the supplier moves the date — all of it is ordinary editing, because nothing has been posted yet.

Receiving

A purchase is a draft until you receive it. Open it once the goods are actually in and press **Receive into stock** — that is what posts the ledger and moves the weighted average cost:

```
old: 10 kg @ 900 = 9,000        new: 5 kg @ 1,100 = 5,500
                    15 kg @ 966.67
```

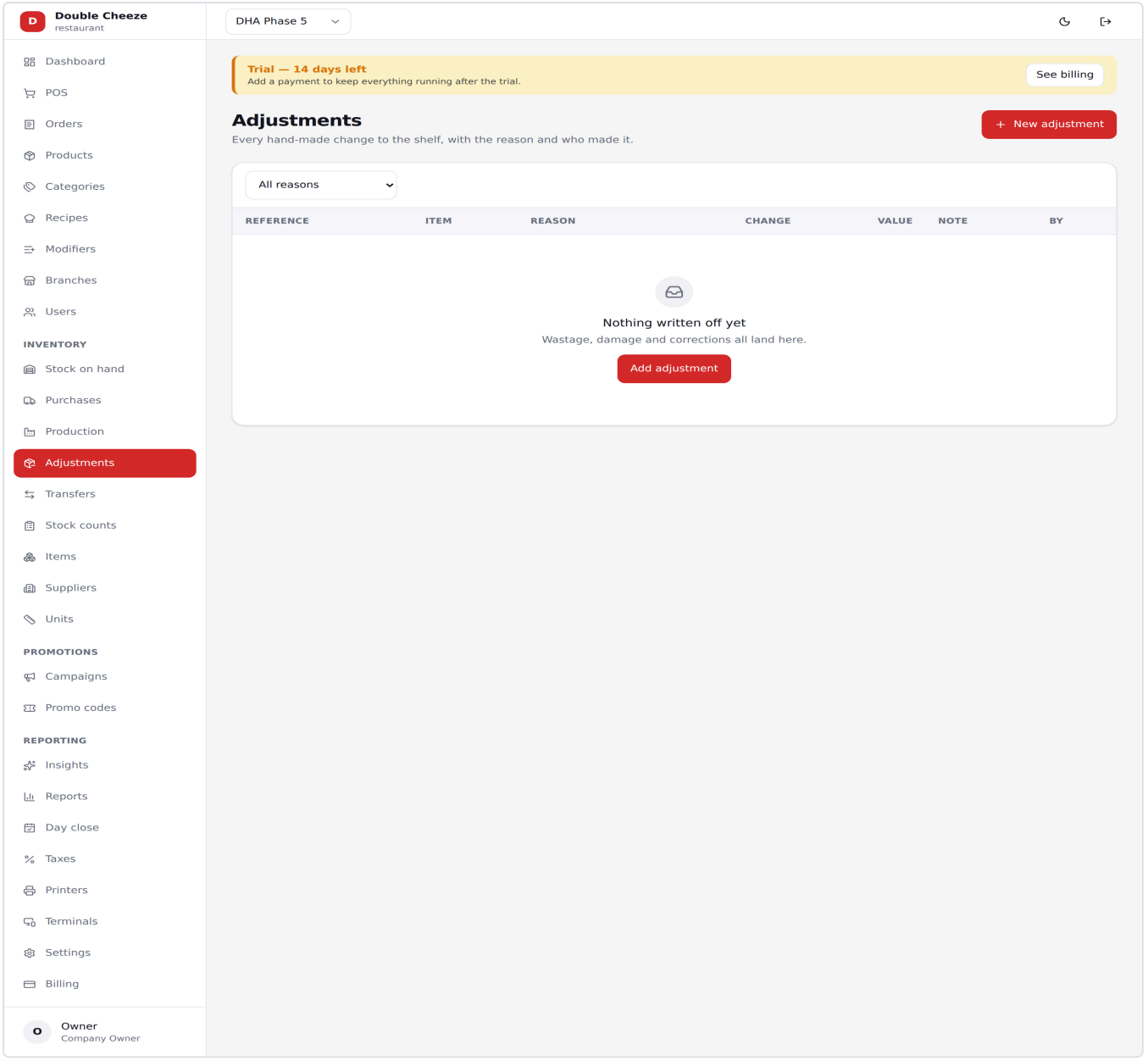
What was promised stays on the document afterwards; what actually happened is the day it was received.

Correcting a received purchase

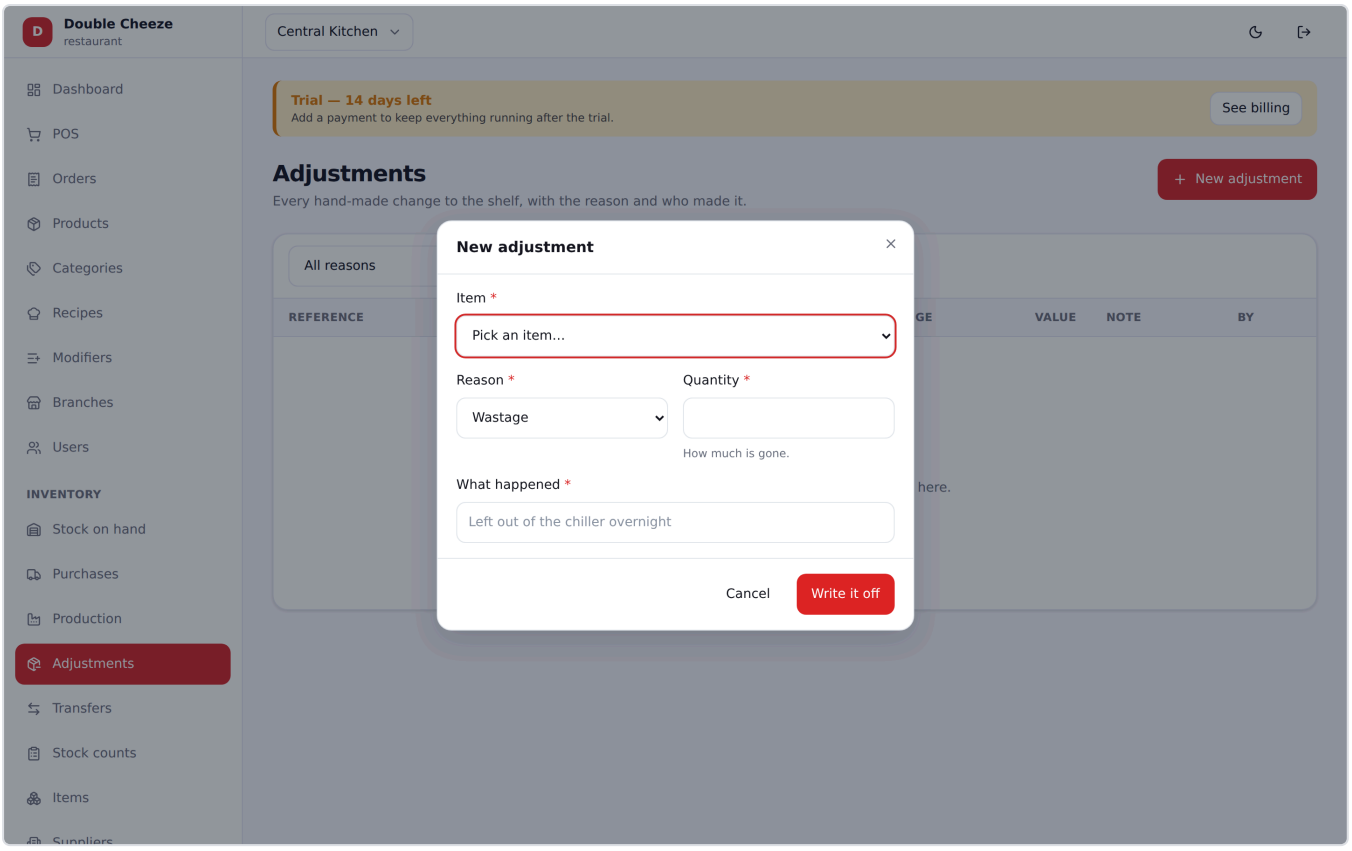
A received purchase is **corrected, never quietly edited**. Open it and press **Correct** (the button only appears when the purchase can still be corrected), fix the lines and give a **Reason for the correction**: the ledger it posted is reversed at the cost the goods came in at and posted again from the corrected lines, under the same document number, with the before/after totals kept on the record. The purchase then shows a **Corrections** list — every correction, who made it and what changed.

It is refused if the day is closed, or if the goods have already been sold or used — that case is a stock adjustment, not an edit.

Adjustments



Adjustments — wastage, breakage and anything the shelf lost without a sale.



Every adjustment needs a reason; it is what the report reads back to you.

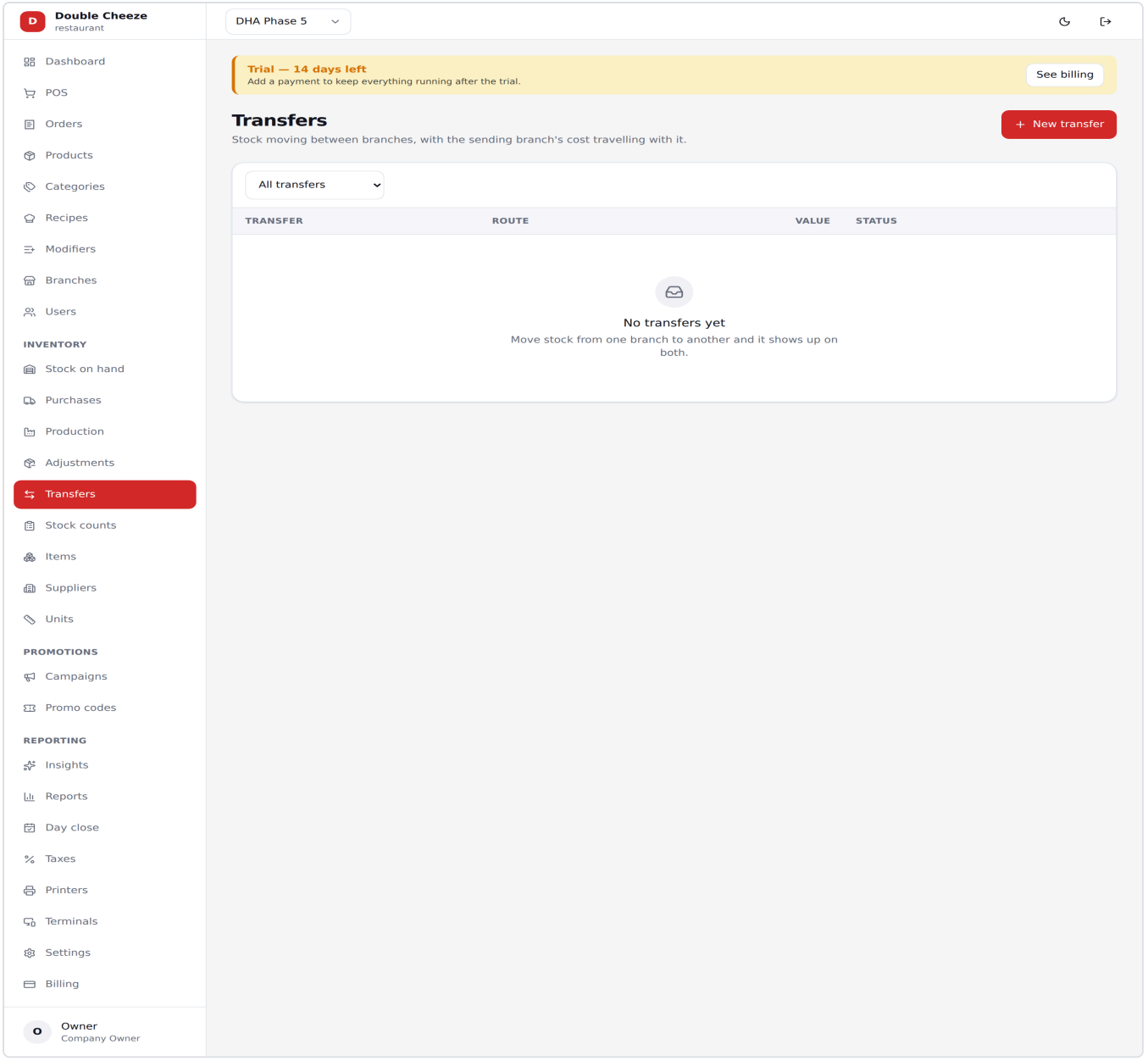
Menu: Inventory → Adjustments

Stock leaving or arriving for a reason that is not a sale or a purchase.

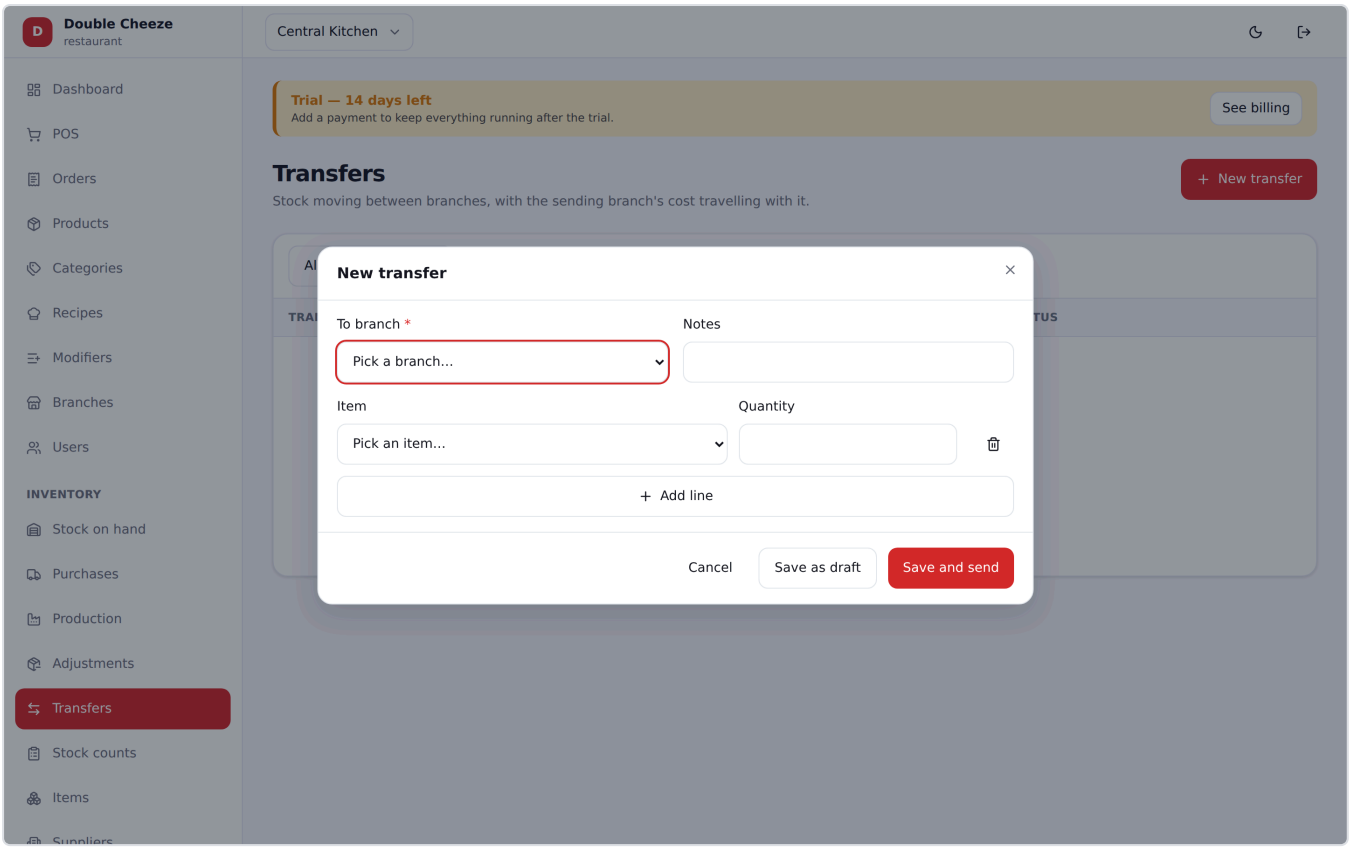
Field	Notes
Item	
Reason	Wastage, Damage, Expiry, Theft, Correction, Opening
Quantity	Negative takes stock off, positive puts it on
What happened	A note — this is what you will read in the wastage report next month

---

Transfers



Transfers between your own branches.



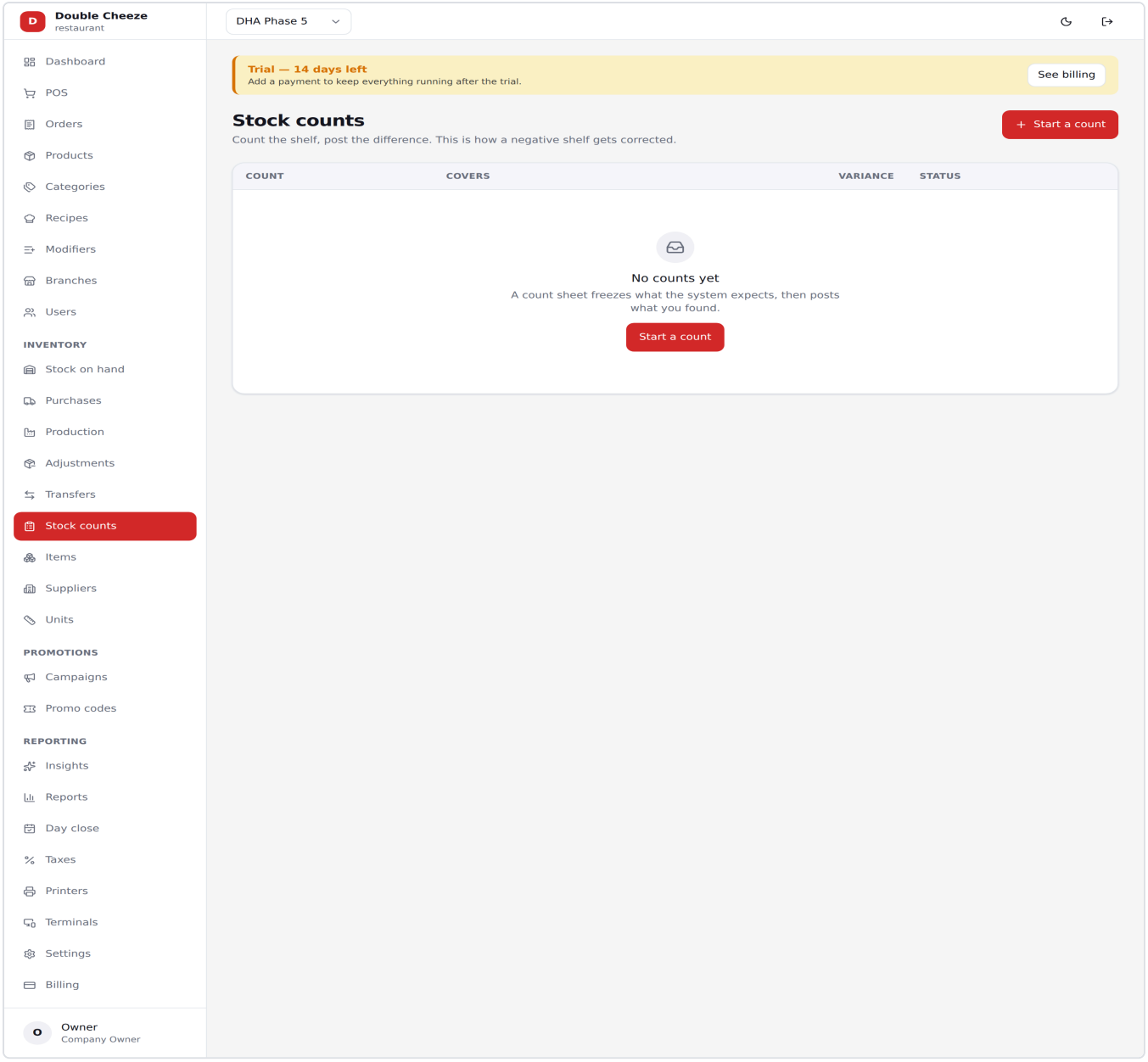
Sending deducts at once; the stock belongs to neither shelf until it is received.

Menu: Inventory → Transfers

Moving stock from the branch in the top bar to another.

A transfer is **two-sided**: sending deducts the source branch immediately (the stock is "in transit" and belongs to neither shelf), and the destination credits it when it is received. The sending branch's cost travels with the goods, so the receiving branch's average is not distorted.

Stock counts



Stock counts. The sheet keeps what was expected at the moment it was opened.

Menu: Inventory → Stock counts

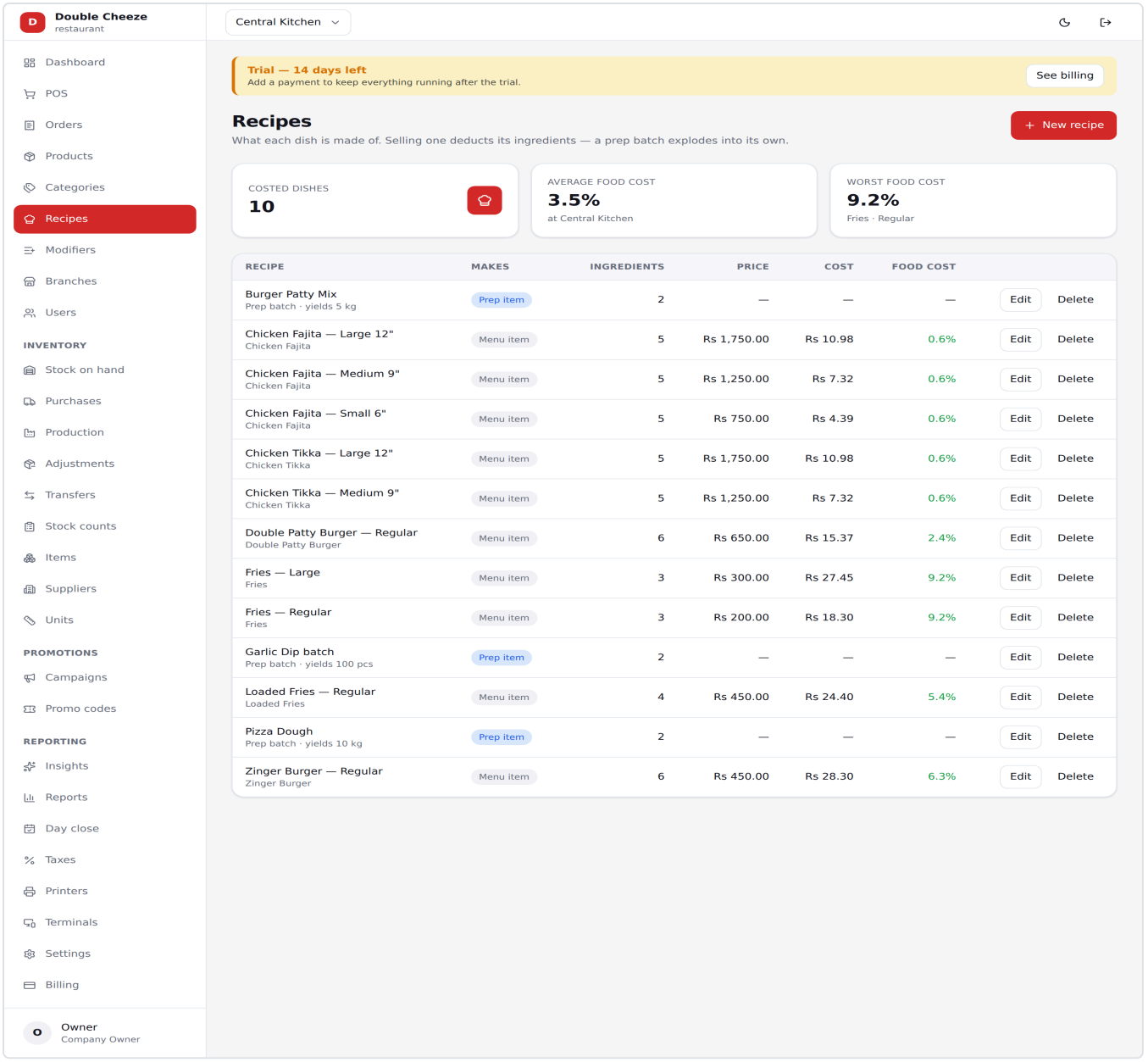
1. Open a count and choose **what is being counted** — everything, one kind, or a few items.
2. The sheet freezes what was expected at the moment you opened it.
3. Count the shelf and type what you actually found.
4. Post it.

Only the **variance** is posted, as one ledger row per item. The sheet keeps the expected figures, so you can always see what the difference was and when.

This is also how you fix negative stock after a day of selling with `Allow negative stock` switched on.

8

Recipes



Recipes, with what each dish costs and the margin it leaves.

Menu: Recipes · **Who:** owner, branch manager · **Restaurants only**

A recipe says what a dish is made of. Selling the dish then takes its ingredients off the shelf — not the dish itself — and tells you what it really costs to make.

The top of the page

Three figures across the top: **Costed dishes**, **Average food cost** and **Worst food cost**. The worst one is the number to look at: it is the dish quietly losing you money.

A recipe makes one thing

Every recipe produces **either**:

- a **Dish** — something the POS sells (a variant: "Chicken Fajita — Large 12″"), or
- a **Prep item** — something the kitchen makes in batches and then uses: dough, sauce, marinade.

Never both. Pick one at the top of the form.

Field	Notes
Recipe name	"Chicken Fajita — Large"
Dish / Prep item	What it produces
Batch yields + Yield unit	What one run makes: `1` pizza, or `10 kg` of dough
Ingredients	Item, quantity, unit, and optional wastage %

Ingredients

Enter quantities **in kitchen units** — grams and millilitres — even if you buy in kilos. The conversion to the stocking unit happens for you.

Wastage % is trim, peel and spillage. 5% on chicken means both the cost and the shelf are charged 1.05 × what the recipe says. Use it; it is usually the difference between the food cost you think you have and the one you actually have.

Prep items explode

If an ingredient is itself made by a recipe, it is not deducted as itself — it explodes through its own recipe, scaled by its yield, until only raw material is left:

```
450 g dough ÷ a 10 kg batch × (7 kg flour × 1.02 wastage) = 0.3213 kg of flour
```

So one large fajita takes flour, oil, cheese, sauce, chicken and a box off the shelf, and the ledger shows each one. A recipe that reaches itself is stopped rather than looping.

A prep item you would rather count

Exploding is right for a mixture nobody counts. When you DO want to count it — because it is made somewhere else, or because you check how many tubs are left — tick **Made in batches** on the item. It then stops exploding: it is deducted as itself, and it arrives by a Production or a transfer instead of by magic.

What it costs

Cost is **theoretical**: every exploded ingredient at that branch's own weighted average cost. An ingredient that has never been purchased is flagged rather than costed at zero, so a half-priced dish can never look profitable by accident.

Two things follow automatically:

- **A dish with a recipe ignores its own stock item** — a pizza is made, not bought.
- The PCA report ranks every dish by food cost % and contribution.

Rules the system enforces

- One recipe per dish, one per prep item.
- The same ingredient cannot be listed twice in one recipe.
- **A unit that cannot convert is refused where you write it**, not later when costing — if an item is stocked in litres you cannot write its line in grams.
- A broken recipe still opens so you can fix it; only the stock deduction stays strict.

9

Commissary and production

Menu: Branches · Inventory → Items, Production, Transfers · **Who:** owner, branch manager

A **commissary** is a central kitchen: it makes things — dips, dough, marinades, sauces — and issues them to the shops. The shops then cook with them, sell them as they are, or both.

Sellify does not treat it as a different kind of business. A commissary is a **branch with a tick box**, production is one new document, and issuing is the transfer you already use.

A commissary **counts as a branch** on your plan, like any other.

The four pieces

Piece	Where
A commissary branch	Branches → New branch → **Commissary** on. It cannot ring a sale
An item that is made	Inventory → Items → **Made in batches** on, and give it a recipe
A batch	Inventory → **Production** — raw material off the shelf, the finished item on
Issuing it	Inventory → **Transfers**, commissary → shop, exactly as between two shops

"Made in batches" is the switch that matters

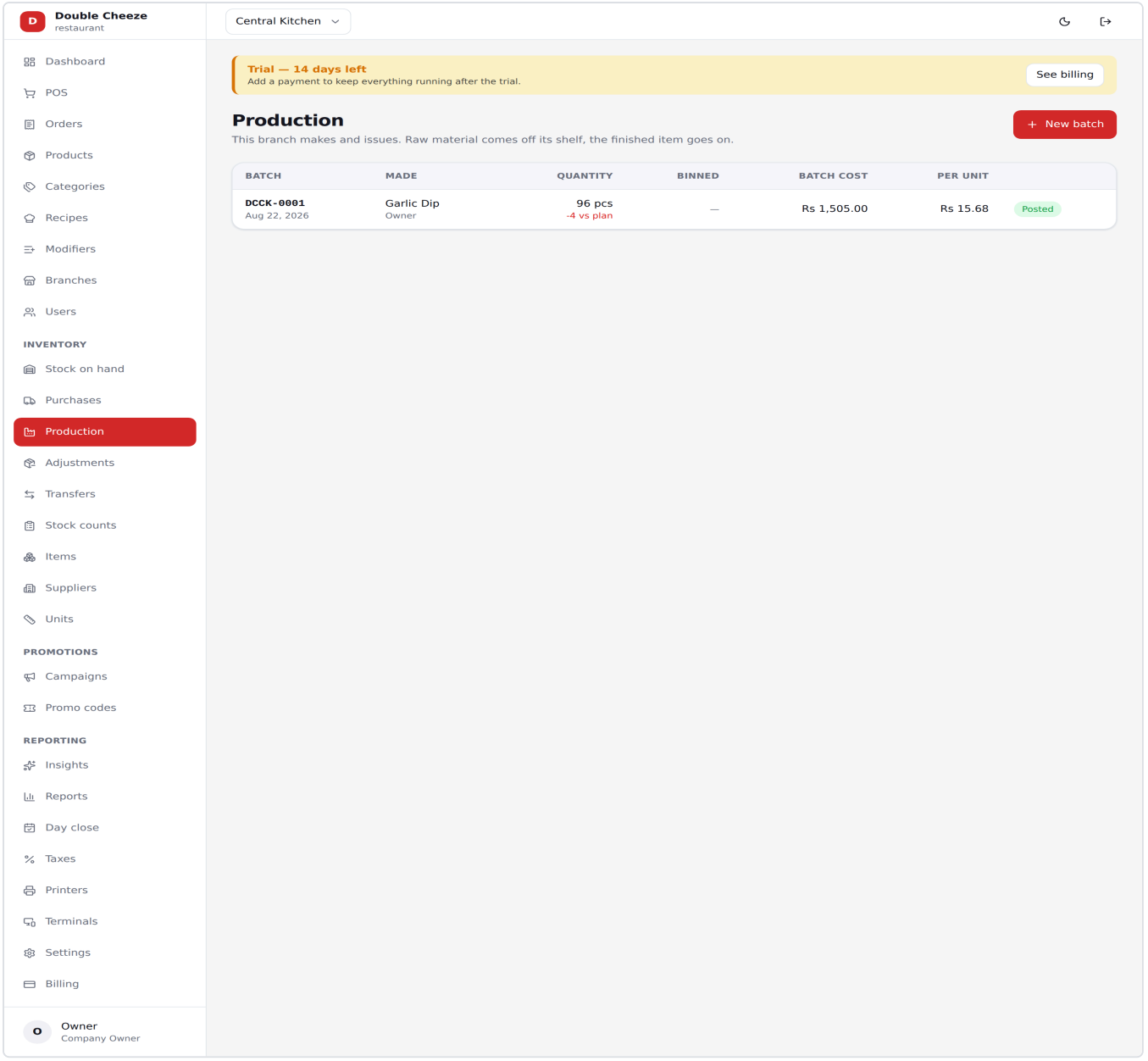
Every item that a recipe can use lives in one of two worlds, and this one tick box decides which:

	Off (the default)	**On**
What it is	**Notional** — a name for a mixture	**Real** — you count it
Stock kept?	No	Yes, per branch
A recipe using it deducts	What it is **made of** (flour, oil)	**The item itself**
How it arrives	It never does	A **Production**, or a transfer

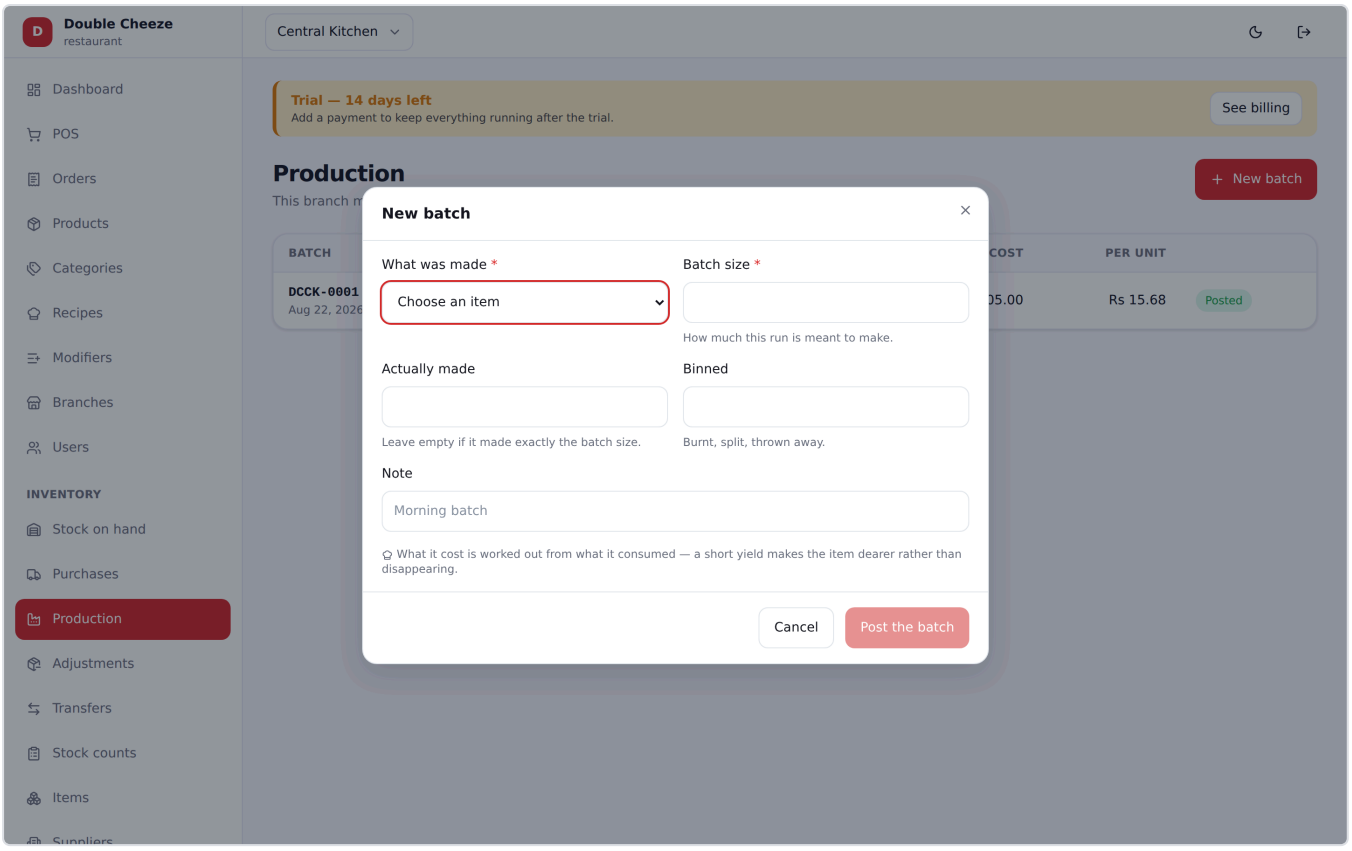
So dough written as a plain prep item deducts flour and oil when a pizza sells. Tick **Made in batches** and the same dough is counted on the shelf: a pizza then takes 180 g of *dough*, and the flour left when the batch was made.

Turn it on when you want to **count** the thing, or when **somebody else makes it** — which is exactly what a commissary is.

Making a batch



Batches the Central Kitchen made. What each cost was worked out from what it consumed.



A new batch. Before anything is posted it shows what will be taken and what is on the shelf.

Inventory → Production → New batch

Field	Notes
What was made	Only items with *Made in batches* on, and a recipe
Batch size	What this run is meant to make, in the item's own unit
Actually made	What really came out. Leave empty if it matched
Binned	Burnt, split, thrown away

Before you post, **This will consume** shows exactly what the batch will take and what is on the shelf — the same figures the sale would use, so the two can never disagree.

What it costs is worked out, never typed

```
Batch of 10 kg dough
   Flour         7.14 kg × Rs 145    = Rs 1,035.30      (7 kg + 2% wastage)
   Cooking Oil   0.40 ltr × Rs 610   = Rs   244.00
                                       ──────────

   Batch cost                          Rs 1,279.30
   Made 9.6 kg, not 10                 ÷ 9.6
   On the shelf at                     Rs 133.26 a kg
```

A short yield does not disappear — it lands in the unit cost, because a bad batch really does make the item dearer. The screen shows `−0.4 vs plan` against the batch.

Issuing to a shop

A normal **transfer**, commissary → shop. Sending takes it off the commissary's shelf; the shop's own receipt puts it on theirs, at the cost it was made for.

If the commissary sells rather than shares, put an **Issue price** on the item (Inventory → Items). A branch is then charged that rate instead of what the batch cost, and it can be overridden line by line on the transfer. Leave it empty and everything moves at cost, which keeps food cost across the chain honest.

Pinning a commissary to its own branches. So a kitchen in one city cannot send a van to another by mistake, open the commissary under **Branches → Edit** and pick its branches under **Issues to**. Once even one is picked, that commissary can transfer to *only* those branches — the till hides the rest, and the server refuses any attempt to reach past them. Pick none and it keeps issuing anywhere, which is how a brand-new commissary starts. Turning the **Commissary** switch off clears the list.

Selling it over the counter as well

Something the commissary makes can also be a menu line — a dip at Rs 60:

1. Create the product as usual (Products → New), with **Track stock** on.
2. Inventory → Items → open the made item → set the **menu line** it is the stock of.

Now the counter and the kitchen come off **one shelf**: selling a dip and cooking a pizza that uses a dip both take from the same number.

What a commissary cannot do

Ring a sale. The POS refuses it — *"Central Kitchen is a commissary — it issues stock to the branches rather than selling."* It has no till, no cashier and no day close of the sales kind; it purchases, produces, transfers and counts.

The order to set it up

1. Branches → the commissary, with **Commissary** ticked.
2. Inventory → Items → the thing it makes, with **Made in batches** on (and an issue price if it charges the shops).
3. Recipes → a recipe for that item, with what a batch yields.
4. Purchases → raw material onto the **commissary's** shelf.

5. Production → the first batch.
6. Transfers → issue it to a shop.
7. Recipes / Products → use it in a dish, or put it on the menu, or both.

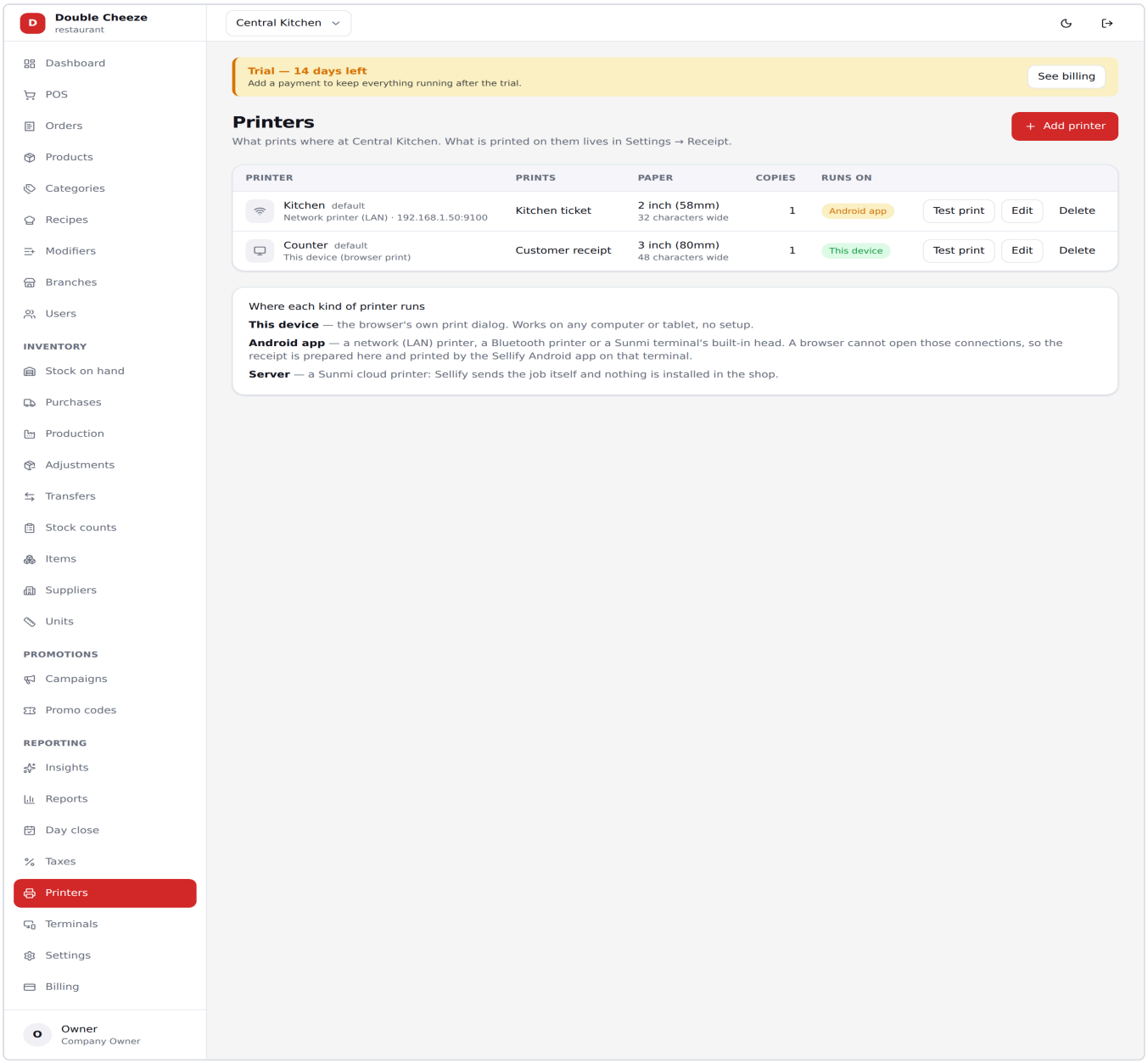
10

Printers

Menu: Printers · **Who:** owner, branch manager

Printers belong to a branch. One shop can have three: the counter receipt printer, a kitchen printer, and an A4 printer for invoices.

Where each type actually prints



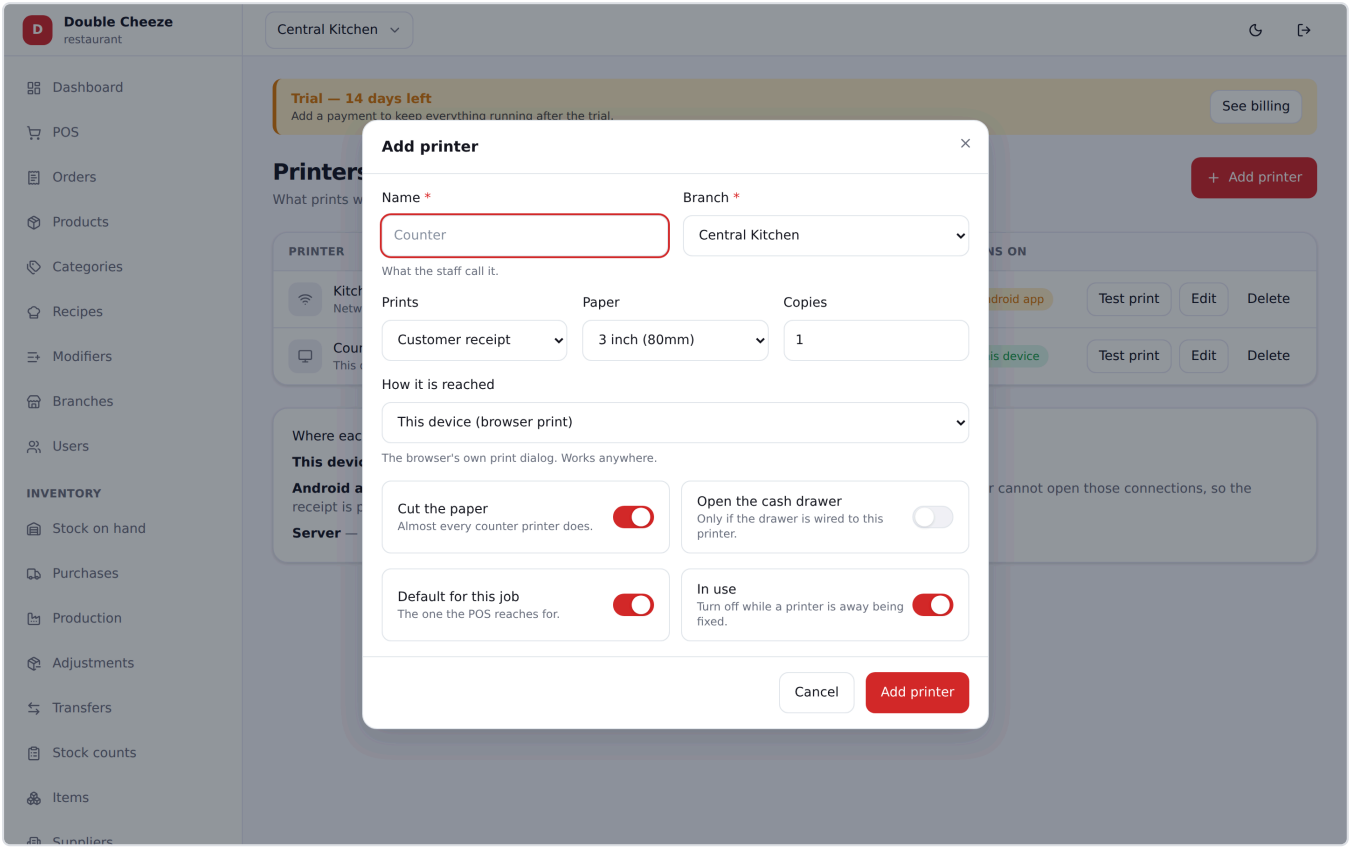
Printers are a list per branch, one default per role.

A browser cannot open a network socket or a Bluetooth port. That is why the transport matters:

How it is reached	Runs on	Available
This device (browser print)	The web POS itself — the browser's print dialog	now
Sunmi cloud printer	Our server sends the job to Sunmi's cloud; nothing is installed in your shop	now, once Sunmi credentials are added
Network printer (LAN)	The Sellify Android app on that terminal	with the Android app
Bluetooth printer	The Android app	with the Android app
Sunmi built-in printer	The Android app on a Sunmi terminal	with the Android app

If you set up a printer the web POS cannot drive, it will not pretend: it hands you the receipt and says the job is waiting for the app.

Adding a printer



A printer row: a role, a transport, a paper size and its connection details.

Field	Notes
Name	What the cashier will see on the Print button: "Counter", "Kitchen"
Branch	Which shop it sits in
Prints	**Customer receipt**, **Kitchen ticket**, or **A4 invoice**
Paper	**3 inch (80mm)**, **2 inch (58mm)**, or **A4**
How it is reached	The transport from the table above
IP address / **Port**	Network printers. Port is 9100 unless yours differs
MAC address	Bluetooth printers
Serial number	Sunmi printers — the SN printed on the device
Cloud key	Sunmi cloud only. Stored encrypted and never shown again
Copies	1 for most shops, 2 where a copy is filed
Cut the paper	Thermal only
Open the cash drawer	Thermal only — kicks the drawer when the receipt prints
Default for this job	The one this branch reaches for. One default per job
In use	Untick to park a printer without deleting it

Test print

Every row has **Test print**. It prints a real receipt marked `*** TEST PRINT ***`, so nobody files it as a sale. Do this before the shop opens, not during a rush — it is the fastest way to know a setting is right.

Paper sizes

	Characters across	What it is
3 inch (80mm)	48	The normal counter receipt
2 inch (58mm)	32	The same receipt, tighter
A4	—	A **different document**: letterhead, item table with tax per line, totals, signature lines

Use A4 for a customer who needs a proper invoice; use 80mm or 58mm at the counter.

Rules the system enforces

- A network printer needs an IP, a Bluetooth one a MAC, a Sunmi one a serial number — a printer nobody could reach is refused when you save it.
- A thermal printer cannot be set to A4, and an A4 printer cannot be a kitchen ticket.
- The cloud key never comes back to the browser once saved.

What the receipt says

Where it prints is this page. **What** it prints — logo, address, NTN, tax breakdown, footer — is Settings → Receipt & Printing.

11

The kitchen

Menu: Catalog → Departments · Setup → Printers · **Who:** owner, branch manager
Restaurants only.

A kitchen ticket is printed **when the order is placed**, whichever button placed it. A dine-in hold, a takeaway paid on the spot, a delivery — the cooks do not care which; they care that the food was ordered.

Turn the paper off entirely for a counter that hands the food over itself: Settings → `kitchen.enabled`, per branch.

Departments — where a dish is made

A **department** is where a dish is made: the grill, the bar, the cold station. Written once for the whole company and reused by every branch.

A department is not a menu category. "Pizzas" and "Burgers" are what a customer reads; Grill is where both are made. Keep the two lists apart, or one kitchen gets two tickets for one section.

Where a dish's department comes from: **the product, else its category, else none.** An unfiled dish is not a hole — it still prints, under *Kitchen*, on the branch's default kitchen printer. A dish reaching the pass uncooked because somebody forgot to file it is not a trade-off worth making.

Which printer

A department is a company list; a printer is a box on a wall. So the mapping is **per branch**: one Grill, three branches, three grill printers. Setup → Printers, on the branch.

Only a **kitchen** printer standing in that branch is offered. A commissary is refused outright — it has no till, so no order is ever rung there.

One slip or several

Settings → `kitchen.ticket_mode`, per branch:

Mode	What comes out
Combined (default)	One slip for the whole order, in sections, in the departments' own order
Per department	One slip per department, each to its own roll

A department with no printer of its own falls back to the branch's kitchen printer rather than going nowhere.

What is on the ticket

No money, ever. A kitchen ticket with prices on it is a receipt somebody will hand a customer by mistake. It says **KITCHEN PRINT** across the top before it says whose paper it is, so a slip lying at the pass beside a receipt is told apart at a glance.

What is on it is every detail of the dish: the quantity, the variant ("Pan · 12 inch"), every modifier under the group it was chosen from, every deal component with its own variant and modifiers, and the line's note.

Adding to a table that is already cooking

Recall a held order, add a drink, place it again — and only **the drink** is sent, headed **ADDED TO ORDER** so nobody starts the pizza again. Sellify works out the difference from what that kitchen has already been told, line by line.

Reprinting

Kitchen print on the order is the try-again button — a printer is a thing in a room, and it jams. A ticket is kept as the paper it was, so a reprint an hour later is the same slip, not a fresh reading of an order that has moved on.

A branch with no kitchen printer at all sends nothing **and records nothing** — so the day one is configured, the cooks get what they never saw.

Dine-in: place first, pay later

`order.dine_in_holds_first` (per branch, on by default) makes the till's big button **Place order** rather than Pay for a dine-in table. The order is held, the kitchen is told, the table is taken, and the bill is settled later by recalling it. A recalled order shows **Pay** again — they are settling, not ordering.

Switch it off for a food court that takes the money first.

Printing never rolls a sale back

A printer that is off, jammed or unplugged is reported on the screen and logged. The sale stands. Fix the printer and press **Kitchen print**.

12

Terminals (tills)

Menu: Setup → Terminals · **Who:** owner, branch manager (Sellify approves)

A branch is a place; a **terminal** is the till standing in it. Every sale is rung on one, and a terminal belongs to exactly one branch. The web POS and the Android app are the same thing here — a terminal row whose platform is `web` or `android`.

Why a till has to be registered

A cashier, a branch manager or a kitchen user **cannot sign in** without a live terminal. An owner and Sellify's own staff can (they read reports from a laptop), but **ringing a sale — quoting, saving, paying, voiding — needs one from everybody**.

Correcting a sale afterwards does not. The Orders panel is office work, and the payment a correction writes is credited to the till the sale was rung on.

Pairing a new till, start to finish

1. **Get the branch's pairing code.** Setup → Branches, open the branch. Press **New code** to roll it if it has been shared too widely.
2. **On the device**, open Sellify, type the pairing code and give the till a name ("Counter 1", "Drive-thru"). The device is not signed in yet — the code is what tells us which branch it is standing in.
3. **The till now says "Waiting for Sellify to approve."** Ring us. Letting a device in is our call, not the shop's.
4. Once approved the device collects its own token and is ready. **The token is handed over once and never shown again** — we keep only a fingerprint of it.

The list

Column	What it tells you
Name and branch	Where the till stands
Platform	Web browser or Android
Status	Pending, active, disabled
Signed in	Who is on it right now, or **Free**
Last seen	**Never** if it has not connected yet
Waiting to sync	How many offline sales that till is still holding

What a shop can do itself

- **Disable terminal** — a lost or stolen tablet, immediately. It loses its token there and then. Do this first and ring us second.
- **Release the counter** — a cashier went home without signing out and nobody else can get on. Releases the session; the release is on the record.
- **Sell without internet** — per till, on by default. Turn it off for a device you would rather refuse a sale than have selling on last-hour prices. See The Android till.
- **Receipt printer** — the till's own default. **Branch default** means it follows the branch.
- **Retire terminal** — the till is gone for good.

Only Sellify approves, rejects or switches a terminal back on. **A re-enabled terminal pairs again from scratch** — switching a till back on is not the same as trusting a device again.

One cashier at a time

While somebody is signed in on a terminal, another user's sign-in is refused. That is deliberate: two cashiers on one drawer is how a shortage becomes nobody's fault. Use **Release the counter** when somebody has genuinely gone.

When the serial does not match

A device is bound to itself — an Android serial, or a random id a browser generates once and keeps. If it does not match, the sign-in is refused and says so. **It never quietly re-binds.** Register the device again and we will approve it again.

A device that still holds its own valid token may re-pair itself without asking (an app reinstall). Every re-pair is on the record.

Everything is on the record

Registered, approved, rejected, disabled, enabled, re-paired, released, signed in, signed out — each one writes a line with who, when and why.

Terminals count against your plan

Every plan caps its tills. The cap is checked when Sellify **approves** a terminal, not hidden in the screen — so a till you cannot have is refused with the plan's name on it. See Billing.

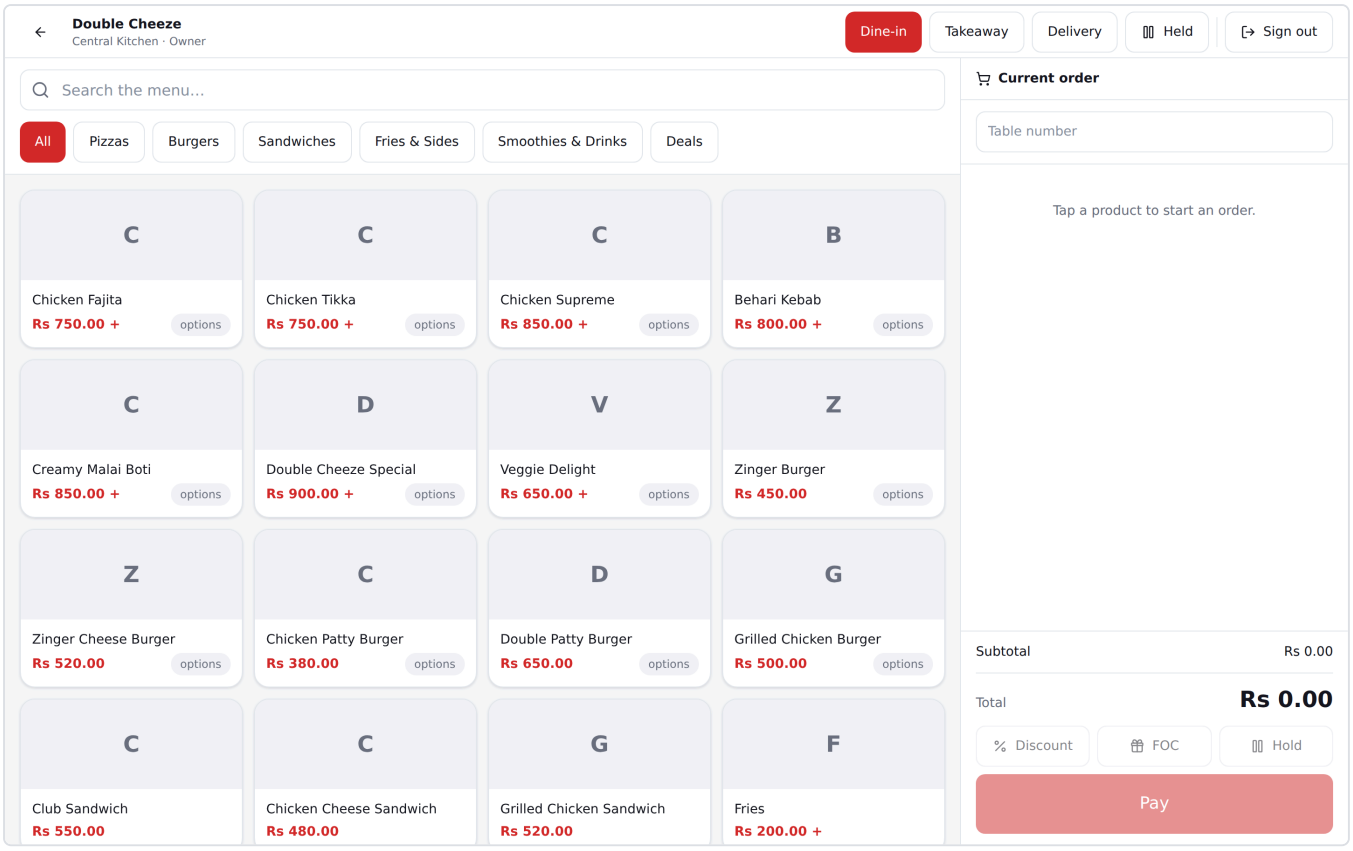
13

POS — taking a sale

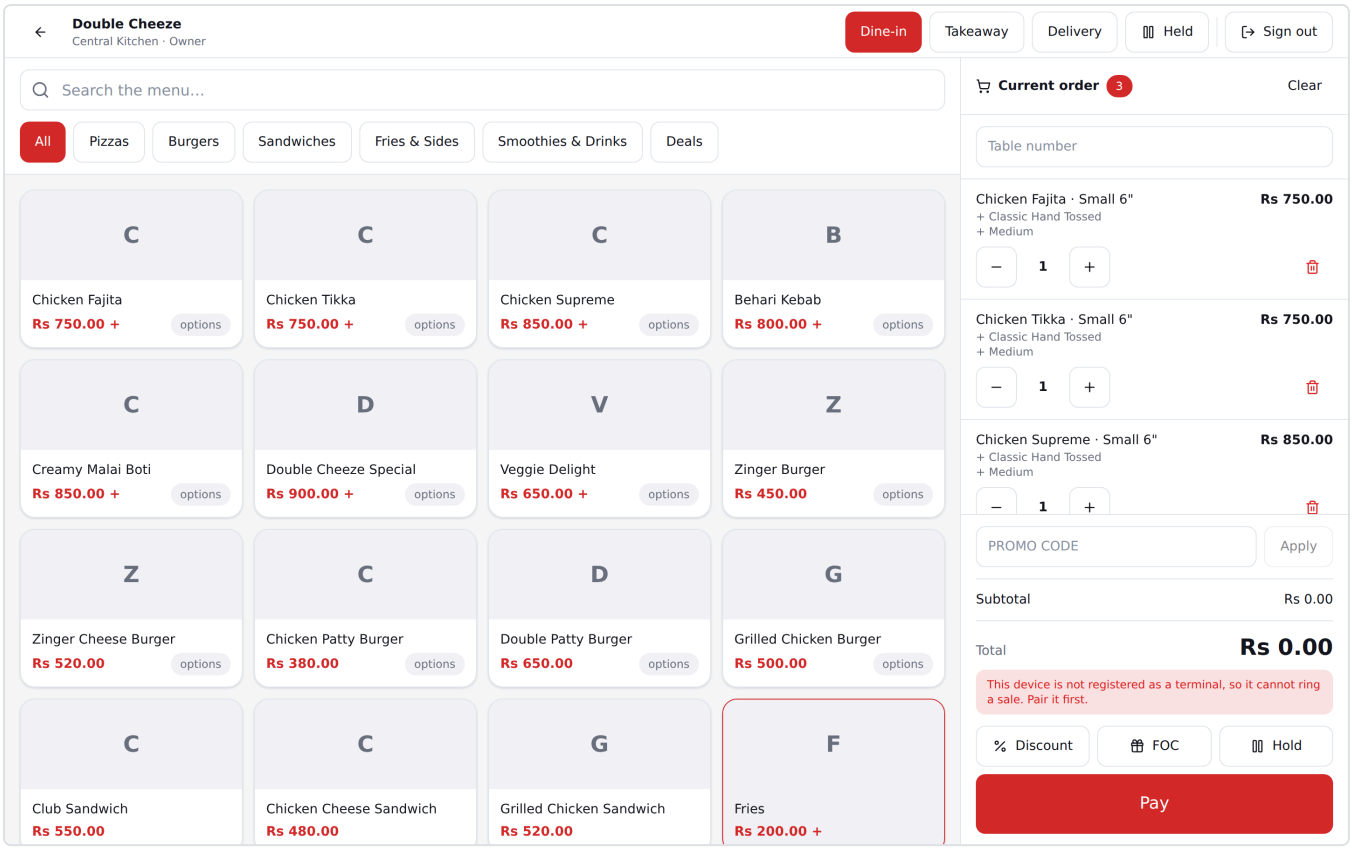
Menu: POS · **Who:** cashier, branch manager, owner

The till is full-screen with no admin chrome. It opens in light mode because shop lighting is bright; the mode toggle is per device.

The screen



The till before anything is rung: the menu on the left, the order on the right.

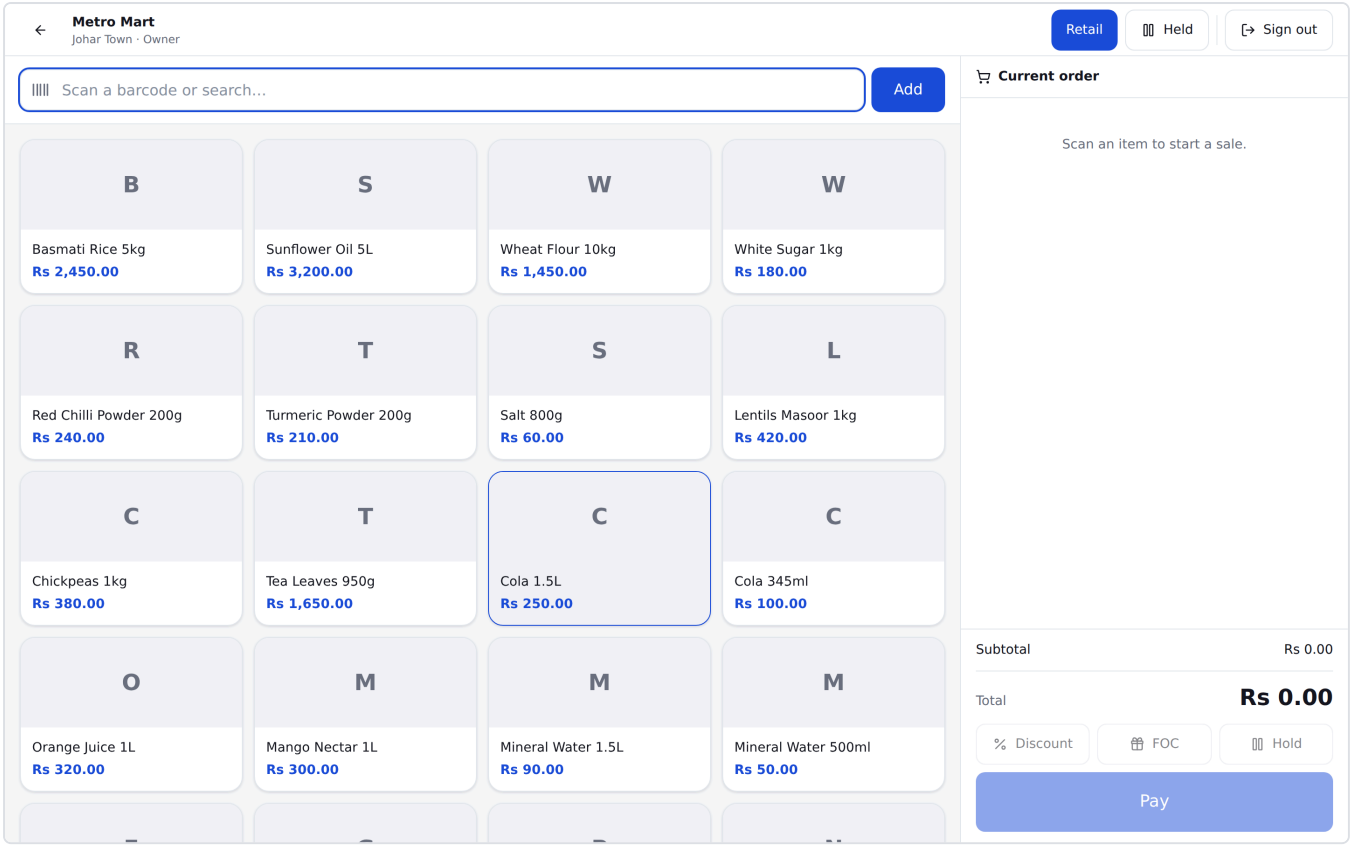


Three lines rung up. Every figure came back from the server, including the offer and the tax.

- **Left** — search box, category tabs, product grid. A product with a photo shows it; the rest show the first letter.
- **Right** — the order type buttons, the cart, offers, the totals, and the big **Pay** button at the bottom.
- **Top right** — **Held** opens parked orders.

Retail branches get a barcode-first layout: the search box takes scanner input straight away, so scanning adds the item without touching the screen.

Ringing a sale



The retail till leads with the barcode box instead of a menu grid.

1. Pick the order type: **Dine-in / Takeaway / Delivery** (restaurant) or **Retail** (shop). Only the types this branch enables are shown.
2. Tap products. A product with options opens a popup — choose crust, size, add-ons, then **Add to order**.
3. Adjust quantities with **−** and **+**, or the bin to remove a line.
4. Dine-in: type the **table number**. Without it the sale cannot be paid *or* held.
5. **Pay**.

Every figure on the screen came back from the server. The till never multiplies a price itself — which is why the total, the tax and the discount can never disagree with the receipt.

Offers

Above the totals:

- **Campaign chips** — offers that fit this cart. One eligible offer applies itself with a tick and can be tapped off; **when two fit, nothing is applied until you pick one** — ask the customer which they want.
- **Promo code box** — type the customer's code and press Apply. If it cannot be used you are told why in one line ("needs an order of at least Rs 2,500", "this number has already used

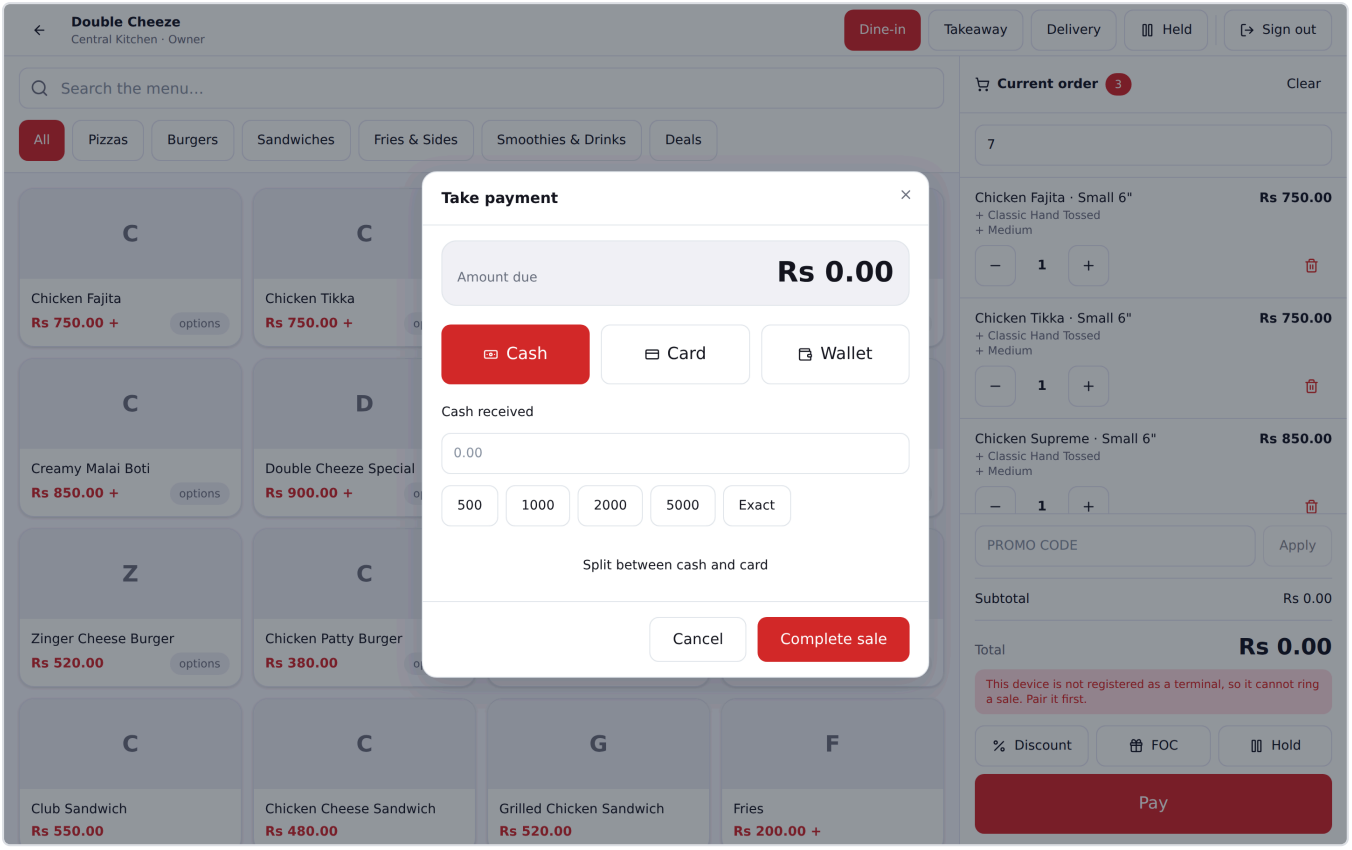
EID500") and the sale carries on at full price.

A **manual discount replaces any offer** — the cashier's decision wins even when the campaign was worth more.

Discount and FOC

- **Discount** — percent or amount, with a reason. Over the cashier's limit it asks a manager for their PIN. Over the company cap it is refused outright.
- **FOC** — writes the whole order off, total and tax zero. It needs a reason and is reported separately from a discount, with the approver's name.

Payment



Taking payment. The method picks the tax rate, so a split re-prices the sale.

The payment dialog shows the amount due and **which rate the tax was worked out at**.

- Tap **Cash**, **Card** or **Wallet**.
- For cash, type what the customer handed over (or tap 500 / 1000 / 2000 / 5000 / Exact) — the change is worked out for you.
- **Split between cash and card** when they pay part each way. The tax rate blends between the two automatically, and the totals update before you confirm.
- **Complete sale**.

Hold and recall

Hold parks the order — for a table still eating, or a customer who has gone to the car. A held order takes **no order number**; the number is given when it is finally paid. Open **Held** to bring one back, add to it, and pay.

The receipt

The receipt appears as soon as the sale is saved, and it is rendered by the server, so what you see is exactly what prints. **Print** sends it to this branch's default receipt printer; the button names it ("Print · Counter"). If that printer is one only the Android app can reach, the dialog says so instead of pretending.

Everything on it is a switch under Settings → Receipt & Printing.

Voiding a sale

Open the order and press Void with a reason. It needs a manager's PIN if the setting says so. Voiding puts the stock back on the shelf **at the cost it left at**, and gives back a promo code use. A voided sale is excluded from every report.

A void is refused once the day is closed — reopen the day first.

When stock runs out

If `Allow negative stock` is off, a sale the shelf cannot cover is blocked and the cart shows exactly what is short. With it on the sale goes through and stock goes negative, to be fixed by a stock count.

14

Price at the counter

Menu: Sell → POS · **Who:** anybody the owner grants it to · **Report:** Reports → Price overrides

A counter sometimes sells at a price somebody agreed: loose goods weighed out, a damaged carton, a wholesale line haggled over. The catalog is the starting point, not the last word.

So a cart line may carry a **unit price the cashier typed**. Everything else about that line is still worked out for you — the modifiers, the deal components, the discount and every paisa of tax — from the price you typed instead of the shelf price.

Who may

It is a **grant, not a job title**. "The cashier who is allowed to haggle" is not a role.

- The **company owner** and the **branch manager** have it by position — they set the shelf price in the first place.
- The owner gives it to anybody else: Setup → Users, **Can override price**.

Somebody without it who tries is **refused in a sentence** — never quietly reverted to the shelf price, which is the failure nobody notices until the day's takings are short.

How it looks at the till

Tap the pencil beside the line price and type the figure. The line shows the shelf price struck through beside what you charged, so a supervisor walking past can see it.

No PIN and no cap

Deliberately. An override is not a discount a cashier talked themselves into — it is the counter's own price. A shop that has decided this person may set it has already made the decision a PIN would ask about again.

`discounts.max_discount_percent` does **not** trim it. That cap is about a cashier's own manual discount, which is a different act.

What guards it is the record, not a challenge at the till.

Below cost is refused

The one floor, and the only one that cannot be argued with: a till that sells at a loss because of a typing slip is worse than a till that asks again. The refusal names the item and what it costs

you.

Cost is what the line actually **consumes** — a dish costs its ingredients at this branch's own average, a carton costs the carton, and the modifiers and deal components on the line are costed with it.

No cost is not a floor of zero. An item this branch has never bought, a product that does not track stock, a service — none of them have a cost, and the line is allowed. The alternative is a till that cannot sell a haircut.

What the line remembers

The shelf price at the moment of the sale, and who typed over it. A price list edited next month never rewrites what was overridden today, and a correction re-pricing a sale keeps the override exactly as it keeps the frozen tax.

It is not a discount

And it is never reported as one. The bill's totals, the tax, and a campaign's minimum order value all read what was actually charged. The discount and FOC reports stay about money somebody **decided to give away** — which this is not.

Reports → Price overrides is its own page: what was sold, at what, against what it should have been, by whom, and what the difference came to. Read it weekly.

It needs the internet

There is no override offline. The floor is counted from this branch's stock ledger, which another till moves while this one is dark — and a floor computed against a stale shelf is not a floor. The till says so in a plain sentence.

15

Orders and corrections

Menu: Sell → Orders · **Who:** owner, branch manager

This is management, not till work. A cashier fixes a mistake by voiding and ringing it again; they never rewrite a sale. **A paid sale is corrected, never quietly edited** — and every correction is on the record.

The list

One branch, a run of **business days**, counted the way the floor thinks about it — by order mode: dine-in, takeaway, delivery, retail.

The screen **opens on today** — both ends of the range default to the business day that branch is trading right now, which the server works out. So "Today" here and "Today" on the Z report are the same day, even at 1am.

Filters: **Every status** (Paid, Held, Voided), **Every mode**, **Every terminal**, and **Discounted only**, **Written off only**, **Corrected only**. Search by order number, name or phone.

A whole range is added up by the database, so a month is as quick as a day, and a range typed backwards is turned around rather than refused.

Opening an order

Everything the sale actually was: the lines with their variants, modifiers and deal components, the discount and where it came from, the tax per line, the payments, the terminal it was rung on and — for a delivery — the area, the promise and the rider.

Correcting a paid sale

Press **Correct**, give a reason. What you may change:

Change	What happens
Remove or reduce a line	The order is re-priced from **its own** frozen lines
Give a discount	Replaces any campaign or promo that was on it, and gives the code back
Write the bill off	Lines go to zero. The **delivery charge stays** — a delivery that was made was made
Remove tax	An **exemption of this customer**, not a change to your tax table
Refund or collect	Money moves as a payment row on the order

It re-prices from the sale's own snapshots — never from today's catalog and never from today's tax table. A shelf price or a tax rate that has changed since cannot rewrite what was sold.

Removing tax parks the lines' frozen rates rather than deleting them, so putting the exemption back charges exactly what the sale charged before.

The money an amendment moves

It is a **payment row** on the order: negative went back to the customer, positive was collected from them. The method defaults to however the sale was paid. The till's expected cash and the Z report already read payments, so nothing has to be told twice.

What is refused

Refusal	What to do
The business day is closed	Reopen the day from Day close first — a PIN'd, recorded act
The discount is over the company cap	`discounts.max_discount_percent` applies here exactly as it does at the till. Nothing about being a manager goes past it
The food has already left on a motorbike	A delivery that is out cannot have its cart changed. Fail or complete it first — see Riders and delivery

Declining a held order

A void. It never took an order number and never took any money, so nothing leaves the shelf or the till.

Every correction is on the record

Who, when, why, and the order's totals on each side of it. That record is the point — it is what makes correcting a sale safe enough to allow at all.

16

Shifts

Menu: Reporting → Shifts · **Who:** owner, branch manager

A shift is one cashier's turn on one till. It is opened with a float, closed with a count, and it is what tells you **which person** was short, not merely which day.

A day close is per branch. A shift is per till, per cashier, inside that day.

Opening a shift

The cashier signs in at the till and is asked for the **opening float** — what is in the drawer before anything is sold. **A sale needs an open shift**; the till says so rather than half-working.

Closing a shift

At the end of the turn the cashier counts the drawer and enters **Counted cash** — what is actually in the drawer. Everything else is the server's:

Figure	Who
Opening float	The cashier, at the start
Cash sales, refunds, payouts	The server
Expected in drawer	The server
Counted cash	The cashier, at the end
Over / short	The server: counted − expected

Short reads as a negative number — the drawer's own convention, the same as a stock count. The browser computes none of it.

The list

Filter by **All shifts**, **Open now** or **Closed**. Each row shows the till, the cashier, when it opened and closed, and the variance.

Closed by a manager means the cashier went home without closing and a manager did it for them. It is marked, because a drawer counted by somebody who did not sell out of it is a different kind of figure.

The Declaration Sales Report

Open any closed shift and press **Declaration Sales Report** — one page for that cashier's turn: what they sold, how it was paid, what was declared and what the difference came to. This is the sheet to sign and file.

Shifts and the day close

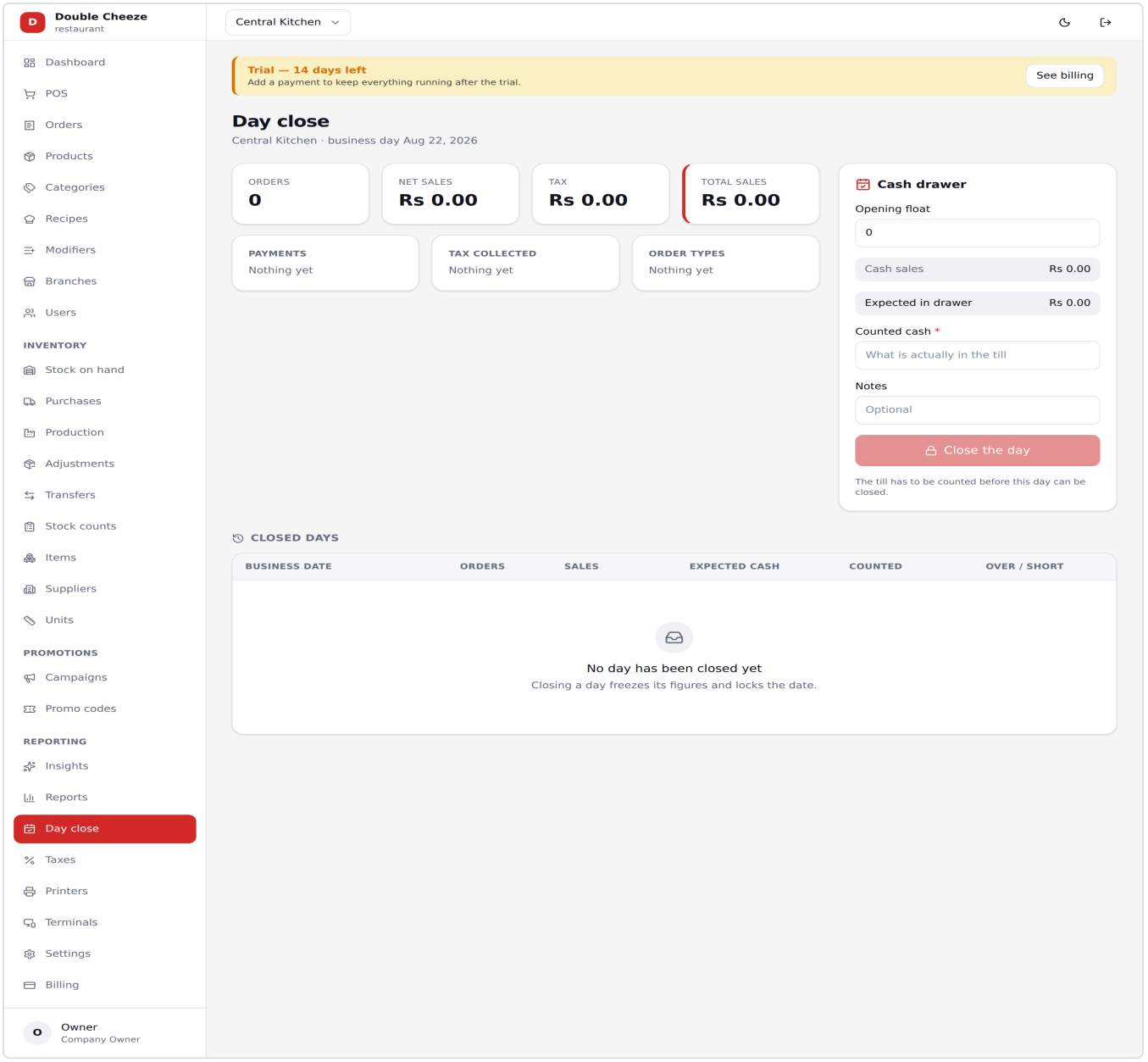
The day close is still per branch. A shift is a breakdown **inside** the day, not a day of its own — so closing a shift does not close the day, and closing the day does not excuse an uncounted drawer.

If a shift will not open

- The till is not approved yet — see Terminals.
- Somebody else is already signed in on that till. Release the counter.
- The business day is closed. Reopen it from Day close.

17

Day close



Day close: the declared float, the counted cash and the over/short variance.

Menu: Reporting → Day close · **Who:** owner, branch manager

Closing the day freezes the branch's figures, reconciles the till and locks the date. Do it at the end of every trading day, before the cash leaves the shop.

While the day is open

The page shows the day as it stands for **the branch in the top bar**:

- Orders, net sales, tax, total sales
- Breakdowns: **Payments**, **Tax collected**, **Order types**
- The **Cash drawer** panel on the right

The cash drawer

Field	Who fills it
Opening float	You — what the till started with
Cash sales	The server
Expected in drawer	The server: float + cash sales
Counted cash	You — what you actually counted
Over / short	The server: counted − expected

The browser never computes the expected figure. You count, the server compares. If the setting requires a cash count, the day cannot close until you have entered it.

Press **Close the day**.

After closing

The page becomes the **Z report**: payments, tax by authority, order types, and the cash drawer with the over/short. Export it as **Excel** or **PDF** for your file.

The date is now locked. No sale, void or edit may land on it. A late sale rung after closing is refused with a message naming the day and the branch.

Reopening a day

Press **Reopen day**, give a reason. The day trades again — the page goes back to the live figures and you can close it a second time. Every reopen is on the record: who, when and why, kept alongside the close.

There is never more than one day-close row per branch per date; a second close updates the same one.

Closed days

The table at the bottom lists every closed day with its orders, sales, expected and counted cash and the over/short. Click any row to read that day's Z report again.

If the drawer does not tie out

- Check **Payments** — a card sale rung as cash is the usual cause.
- Check whether anything was paid out of the till (that is a cash adjustment, not a sale).
- A small daily variance is normal; a large one on one day is worth chasing while people still remember the day.

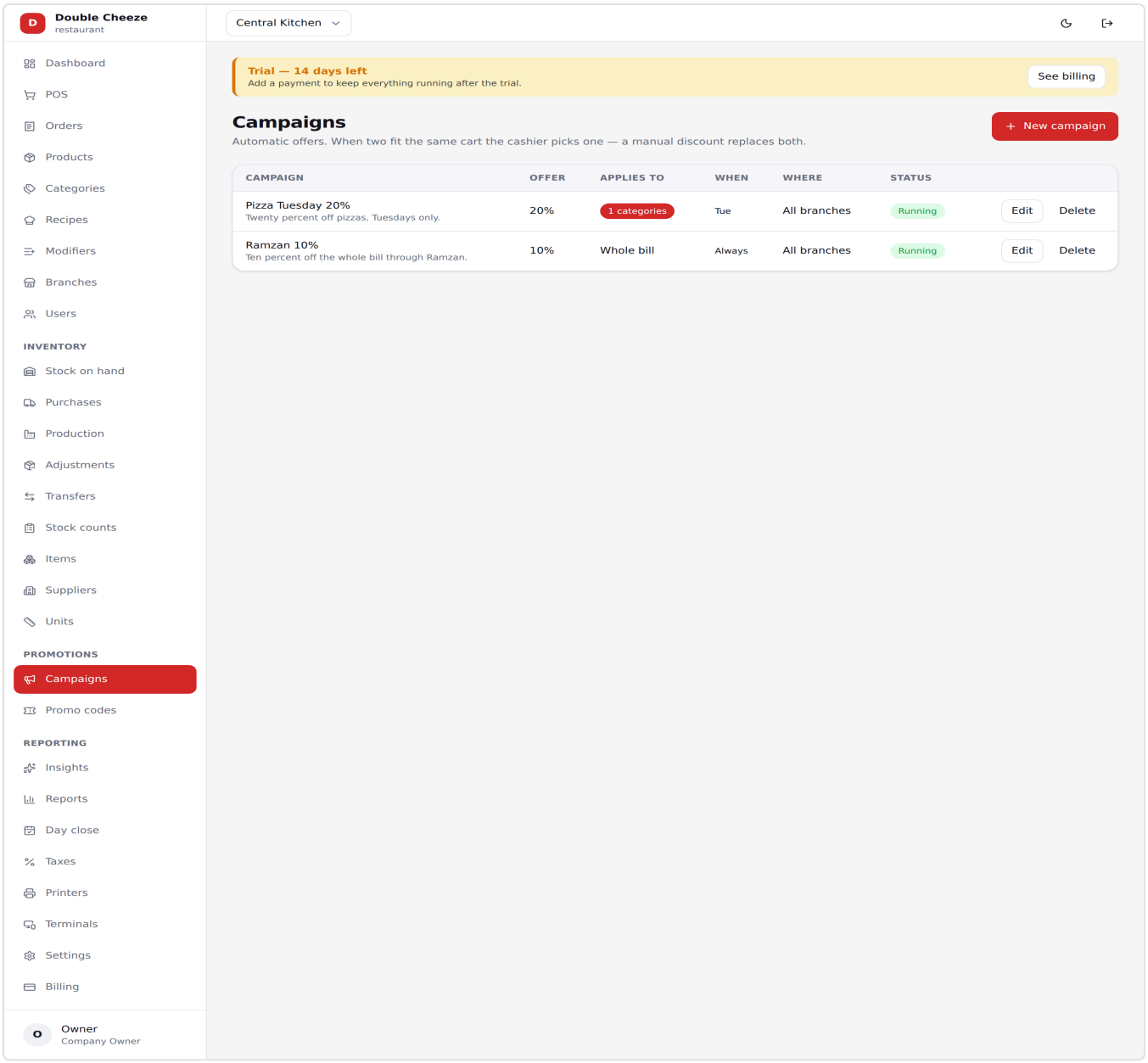
18

Campaigns and promo codes

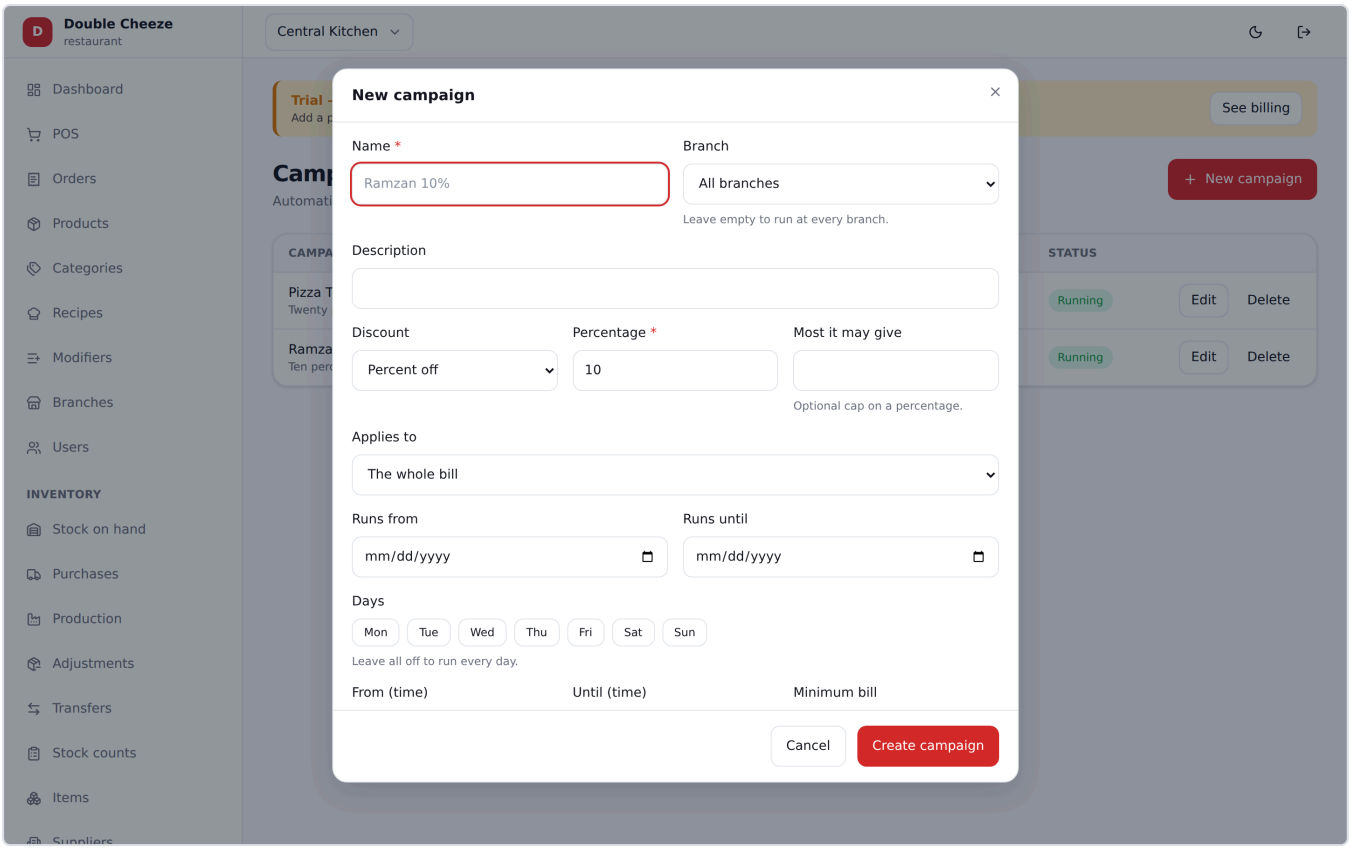
Menu: Promotions · **Who:** owner, branch manager

Two kinds of offer. A **campaign** applies itself; a **promo code** is asked for by name. Both are decided in one place, so they can never double up by accident.

Campaigns



Campaigns apply themselves when the cart fits.



A campaign carries its own scope — all, a category, or named products.

Field	Notes
Name	What the cashier and the receipt see: "Ramzan 10%"
Branch	Empty = every branch
Discount + value	Percent off, or an amount off
Most it may give	A cap on a percentage — "20% off, up to Rs 500"
Applies to	**The whole bill**, **only these categories**, or **only these products**
Runs from / until	Dates
Days	Tuesdays only, weekends only. All off = every day
From / Until (time)	Happy hour, in the branch's own clock. A window that ends before it starts (22:00 → 02:00) runs through midnight
Minimum bill	Checked against the whole bill
Order types	Delivery only, dine-in only… all off = every type

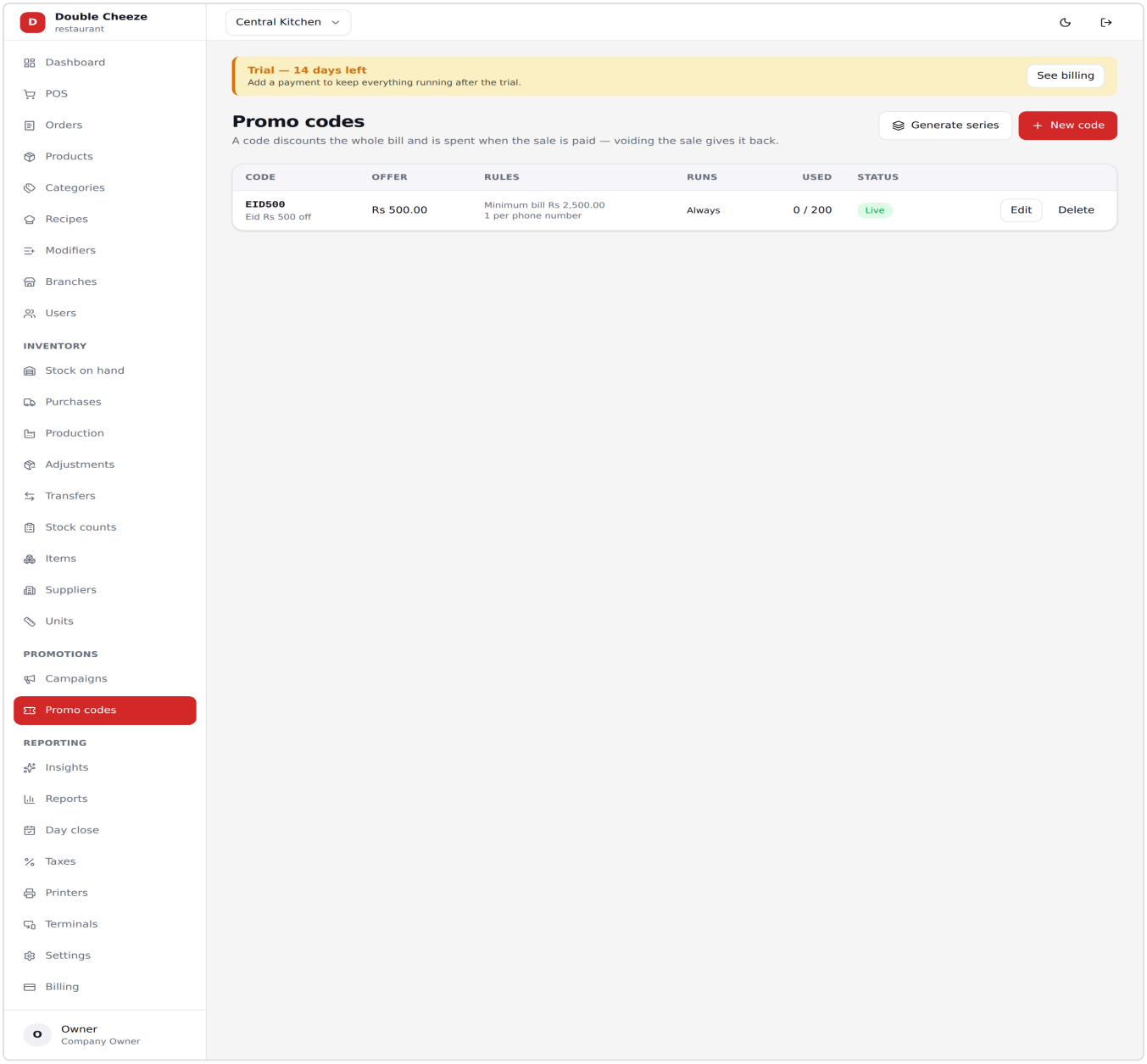
Scope decides what it touches

- **Whole bill** — everything in the cart is discounted.
- **Categories / products** — **only the qualifying lines**. "20% off Pizzas" never touches the cola sitting next to it.

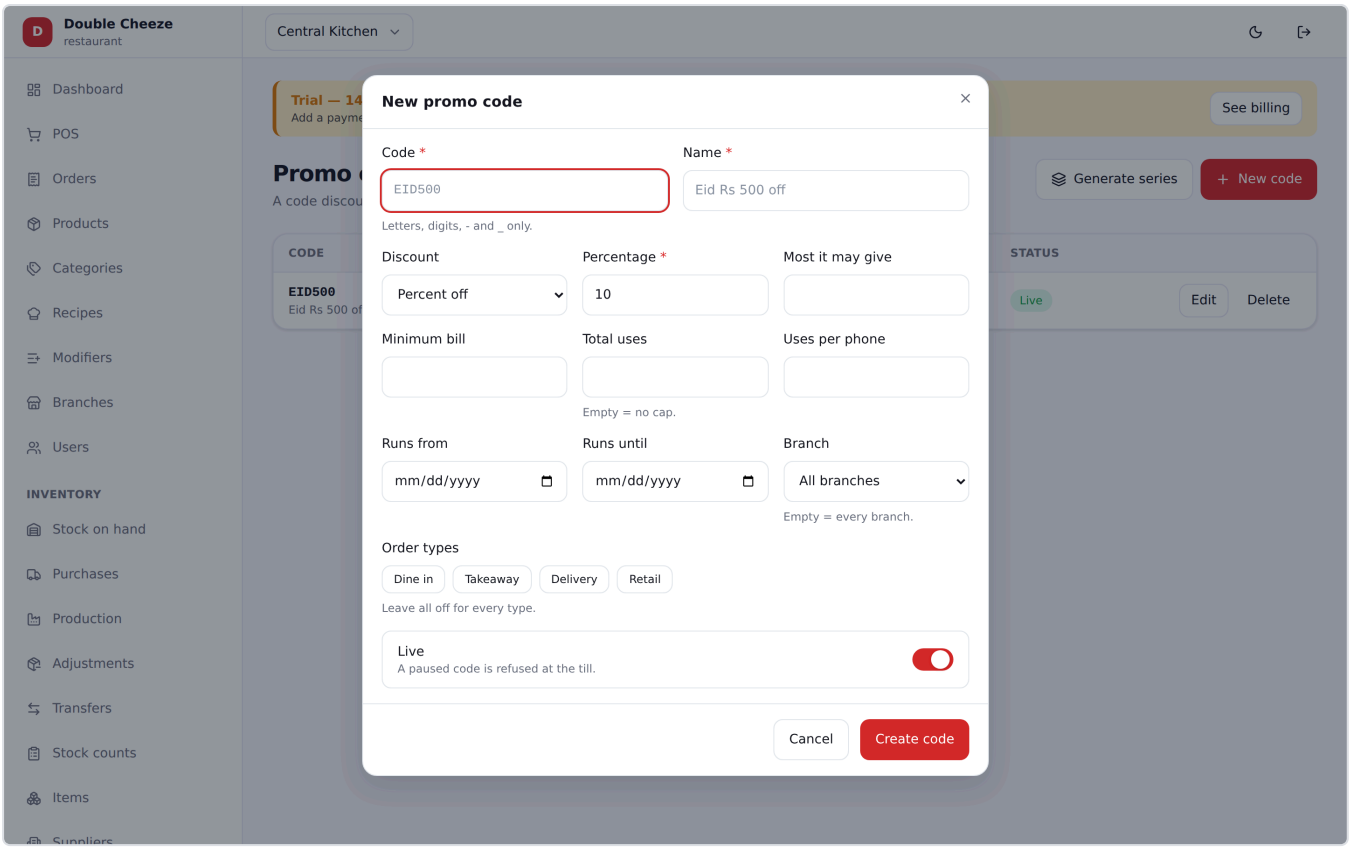
When two campaigns fit

Nothing is applied. The POS shows **PICK AN OFFER** with each campaign and what it is worth, and the cashier asks the customer. A single eligible campaign applies itself with a badge.

Promo codes



Promo codes, with how many times each has been spent.



A code carries dates, a usage cap, a minimum order value and a per-phone limit.

Field	Notes
Code	`EID500`. Letters, digits, `-` and `_`. Unique in your company — another company may use the same code
Name	What the receipt prints: "Eid Rs 500 off"
Discount + value, **Most it may give**	As above
Minimum bill	"Rs 2,500 se ooper"
Total uses	The whole run. Empty = no cap
Uses per phone	One per customer. Needs the customer's phone on the order
Runs from / until, **Branch**, **Order types**	As above

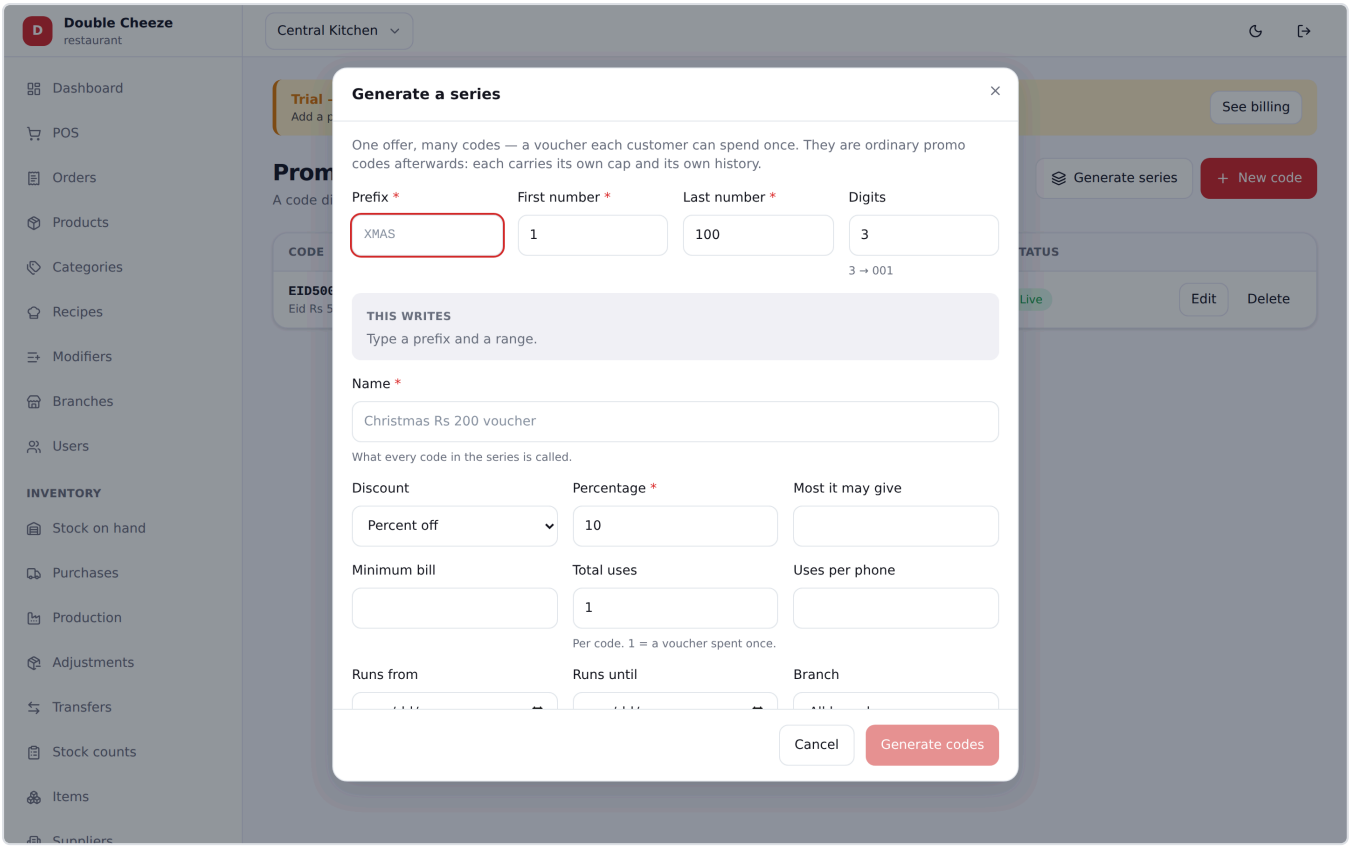
A code discounts **the whole bill**. Line-scoped offers are campaigns, not codes.

When a code is spent

When the sale is paid — not when it is quoted, and not when the order is held. **Voiding the sale gives the use back.** The Used column counts from the record of actual uses, never from a number someone typed.

Writing a deleted code again starts a **new run** of it: its caps count from that moment, and the old uses stay in the reports.

A series of codes — XMAS001 to XMAS500



Writing a numbered series in one go.

One code at a time is right for `EID500`, the code you print on a banner. A **voucher** is the other shape: a different code for every customer, each one good for a single use. **Generate series** writes them all in one go.

Field	Notes
Prefix	The word in front — `XMAS`. Letters, digits, `-` and `_`
First / Last number	The range. 1 to 500 gives five hundred vouchers
Digits	How the numbers are padded. 3 → `XMAS001`; 4 → `XMAS0001`
Name	What every code in the series is called, and what the receipt prints
Everything below	The one offer every code in the series carries — discount, minimum bill, dates, branch, order types

Total uses is per code, and it starts at 1 — a voucher spent once. Set it higher if one voucher may be used more than that; empty means no cap on each code.

This writes shows exactly what will be written — the first code, the last code and the count — before you press anything. Up to **2,000 codes** at a time; run it again for the next block, with the same prefix, starting where you left off.

Once a series exists

It is listed as **one line**, not five hundred, and each code inside it is an ordinary promo code with its own cap and its own history. On the series line:

Codes	Opens the list, with a search box — type `XMAS24` to find that voucher and see whether it has been spent
CSV	Downloads every code in the series, to mail-merge onto cards or send out
Pause / Restart	Stops or restarts the whole series at once
Delete	Retires every code in it. Sales that used one keep their discount

Used / left on the line counts the vouchers actually spent, so you can see a print run being redeemed.

If a code in the range already exists

Nothing is written. The message names the codes that clash — *"XMAS001, XMAS002, XMAS003 and 7 more already exist"* — and you either change the prefix or start the numbering higher. A live voucher is never quietly overwritten, because the cards are already printed.

A whole series that was deleted and written again is a **new run**, exactly like a single code: the old spends stay in the reports and this run's caps start at zero.

Which offer wins

1. FOC writes the whole order off and stops.
2. A **manual discount replaces** any automatic offer.
3. Otherwise the customer gets **whichever is worth more** — unless `Allow a promo code and a campaign together` is on, in which case the campaign lands first and the code works on what is left.

A campaign or a code never needs a manager PIN. And the company discount cap **trims** an over-generous offer instead of refusing the sale, because a mispriced offer must not stop the till.

Seeing what worked

Reports → Promotions shows every campaign and code with the orders it brought, what it gave away, that as a share of gross, and the average order it produced.

19

Call centre

Menu: Sell → Call centre · Sell → Customers · Setup → Areas, Cities **Who:** call centre operator, manager and HOD — plus the owner

An agent takes an order into **any** outlet from a desk that is not a till. Delivery and takeaway only: a table booking is a different thing and is not in Sellify.

Before the first call

Set up	Where	Why
Cities	Setup → Cities	So a customer and a branch can be filed by city
Areas	Setup → Areas	The delivery charge, the promise and the minimum order
Agents	Setup → Users	The three call-centre roles are company-wide, like an owner

Areas

One area belongs to one outlet. An address in an area is served by that outlet and no other, so the agent never picks a branch and no two branches both think an order is theirs. Writing an area name a second time is refused, naming the outlet that already has it. A commissary is refused an area outright — it has no till, so it delivers nowhere.

Field	What it does
Delivery charge	Left blank means **fall back to the branch's charge** — not "free"
Promise (minutes)	What the agent reads out. Worked out on the **branch's** clock
Minimum order	Measured on the **goods after discount**, never on the bill
Radius (km)	A number for a human to read out. No map, nothing is computed from it

Out of radius **warns**. A shop that means it turns on `delivery.block_out_of_radius` for that branch.

Taking a call

1. **Find the customer by phone.** `0300 1234567` and `+92 300 1234567` are one person — Sellify matches the number, not the punctuation.
2. **Pick the address**, which picks the area, which picks the outlet. The charge and the promise appear at once.

3. **Build the cart on the same screen a cashier uses.** That is the point: a till and a call can never price one menu two ways.
4. **Confirm.** The order is placed, not paid.

The order type is stated, not offered — the API refuses a dine-in call order.

What a confirmed order is

It holds no money, so nothing in the day close, the Z report or the sales reports changes. It takes **no order number** either — those restart every business day and are assigned when the money is taken, so a cancelled order must not burn one. It carries a **call reference** instead, and the real number arrives when the branch takes the money.

Refusals worth knowing

Refusal	What it means
"Choose 2 for Drink."	A deal has a choice nobody answered
The minimum names the shortfall	So nobody does subtraction on the phone
The outlet's day is closed	A manager at that branch must reopen it. Not something to retry

The customer

Stars are worked out from their orders, not typed by a manager — a rank somebody has to maintain by hand is a rank that is six months out of date.

A customer's own discount wins. When their standing discount and a campaign or promo both fit, theirs applies and **nothing is asked** — it was promised to them personally, and an agent should not adjudicate that on the phone. Only a cashier's own manual discount replaces it. It discounts the goods, never the delivery charge, and the company cap trims it rather than refusing the sale.

Giving one is the **owner's or the HOD's**, not a branch manager's — it is worth money forever. Every change is on the record: percent before and after, the reason, who and when. Blank dates mean standing; switching it off is a pause, not a deletion.

The branch answering

The branch sees new orders on its POS within about ten seconds and hears an alarm.

- **Accept** — the kitchen is told and the ticket prints. Accepting is what says somebody is standing in that kitchen and has read it. `call_centre.auto_accept` collapses the two for a shop that wants the paper immediately.

- **Reject** — a void with a **required reason**, which the agent reads out to the customer. Nothing leaves the shelf or the till either way.

The alarm needs arming once. A browser will not play a sound until somebody has tapped the page — so the till shows a plain strip until it is armed. Tap it at the start of service. A feature that fails quietly at 8pm on a Friday is worse than no feature.

The board

`Call centre → Board` lists every outlet's unanswered orders, oldest first. Anything unaccepted past `call_centre.unaccepted_alert_minutes` (5 by default) turns red. **Ring the branch.** A till that is off the network hears nothing, and that cannot be fixed from our end — the board is the other half of the answer.

Feedback and complaints

Menu: Sell → Feedback

Anybody who takes the call may record one — an agent, a branch manager, an owner. When it is filed against an order, **the branch and the customer come off the order**, so it cannot land on the wrong outlet.

- **Categories** are your own words for what went wrong (Setup → Complaint categories). A pizza shop's "cold on arrival" and a pharmacy's "wrong strength" are not one list.
- **Resolving** is its own act, so a branch manager can say what they *did* about a complaint against their outlet without being able to rewrite what the customer *said*.
- **Editing and deleting** are the HOD's, the owner's and Sellify's — the words somebody wrote down are evidence. Every edit and delete is on the record, and a delete is soft.

A day will not close over an open confirmed order

Named by its call reference. The kitchen is cooking food nobody has paid for; closing would refuse that payment when it came.

20

Riders and delivery

Menu: Sell → Dispatch · Setup → Employees, Delivery failures **Who:** owner, branch manager (an agent sees it, read-only)

A rider is an employee

Not a user and not a role — most riders never log in. Setup → Employees, tick **Rider**. They get the code, the CNIC and photo, the joining and leaving dates, and the effective- dated salary rows every employee has. Re-hiring somebody brings their history back with them.

A rider belongs to **the outlet they ride out of**. A head-office employee cannot be one, and a commissary dispatches nothing — it has no till.

Where the food is

Dispatch is its own thing, not an order status. Only a **delivery** order has one:

```
pending  →  assigned  →  out  →  delivered
                              →  failed
```

Backwards is refused, and delivered and failed are the end — with one exception: **assigned back to pending**, because a rider who calls in sick has to come off the order and that is not a failed delivery. The food never moved.

Stock leaves when the food does

Stock is deducted when the order goes `out`, not when it is paid. Food in a bag on a motorbike while the shelf still says it is in the building means the till sells the last of a cheese that has physically left.

Dine-in, takeaway and retail are unchanged — they still deduct at payment.

A failed delivery **puts the stock back**.

When the money is taken

`delivery.pay_on_return` (per branch, **on** by default): the sale is paid when the rider comes back. While the food is out the order is placed and unpaid, so the drawer never shows cash the shop has not got, and a failed delivery never has to be un-sold.

Switch it off for a counter that takes the money first.

The card machine goes out with the rider. **Nothing is prepaid** — no card over the phone, no payment link.

The dispatch screen

One branch, one business day. Assign a rider, mark **out**, then **delivered** or **failed**.

A failed delivery needs a **reason from your own list** (Setup → Delivery failures). The order is voided — stock back, no money was taken — but the failure is written down beside it, because a void answers what happened to the money and says nothing about why the food came back. The reports count failures apart, or "on time %" is quietly computed over the deliveries that went well.

A second attempt is not in Sellify. Ring the order in again.

Settling a rider's cash

Per rider, per business day. The rider hands over what they collected and you type **one figure** — **Counted from their hand**.

Figure	Who
Cash expected	The server: the cash total of what that rider delivered
Card taken on the machine	The server, reported **beside** the cash, never inside it
Declared	You
Over / short	The server: declared − expected. **Short is negative**

Settling is per rider, so somebody who finishes at 9pm does not wait for a rider still out. Settling twice moves the same row.

A delivery nobody has rung in yet blocks the settlement, naming how many. Its cash is invisible to us until the cashier takes the money, and settling over it would measure the rider against a figure short by exactly that order.

A day will not close with an unsettled rider, named.

The Z report

Carries **by rider** beside **by terminal**. A variance of blank means nobody counted it — never zero, because "nothing missing" and "nobody counted it" are different answers.

The reports

Reporting → Reports, under **Delivery**:

- **Delivery performance** — on time, late, failed, and the average minutes against the promise
- **Rider performance** — with the cash variance
- **Failed deliveries** — names the customer, so it is never shown to the AI assistant

Two things these refuse to do: a **failure is counted apart**, never inside the on-time count; and a delivery nobody promised anything for is **unmeasured**, with a column of its own, staying out of the average entirely. On-time % reads blank rather than 100 when nothing was measured — a shop that promised nothing was not perfect, it was unmeasured.

What Sellify does not do

No maps, no coordinates, no live rider location, and no rider app. The transitions are an API, so an app can be built on them later.

21

External channels

Menu: Setup → Channels · Reporting → Channel settlements · **Who:** owner

Foodpanda, Bykea, your own website — the food is made in your kitchen, the tax is filed by you, and the dish appears in your food cost. **An order that came from somewhere else is still your sale.** So it is an ordinary order with one extra fact on it, not a second kind of order.

Two different questions

	Answers	Example
Source	Who typed it into Sellify	The counter, the call centre
Channel	Where the customer ordered	Foodpanda

A Foodpanda order typed on the counter till is source *POS*, channel *Foodpanda*. Neither is folded into the other, or whichever question was asked second would be lost.

Setting a channel up

Setup → Channels. Your own list — write a new one the day an aggregator opens in your city, no waiting on us. Writing a name that was deleted brings it back with its orders still on it; a channel orders point at is switched off rather than deleted.

Field	What it decides
Payment method	Cash, Card, Wallet or **Credit — owed to us** (the default)
Settles later	The platform collects from the customer and pays you afterwards
Commission %	What they keep
Charged on	**Order total** (what the customer paid) or **Net of tax** (the goods after discount)
Uses own rider	On by default — their rider, not yours

Why the payment method matters more than the dropdown

An aggregator collects from the customer and remits later. Rung as **cash**, your drawer would be expected to hold money the platform is holding — short by exactly that amount, every single day, with the cashier standing over it.

So **Credit — owed to us** means money you have earned that somebody other than the customer in front of you owes. The drawer stays honest with no new arithmetic anywhere.

The method picks the tax rate, exactly as it always has. A platform that charges the customer's card is set to *Card* and files at the card rate; one that hands over cash-on-delivery takings is *Cash*. **Cash + settles later is refused** — a drawer cannot both hold the money and not hold it.

What commission is charged on is your contract's business, not ours. Guessing it would make every figure in the settlement report quietly wrong for half the shops.

Ringing a channel order

Pick the channel on the till like any other id. The server fills **one payment row** for the whole total at the channel's method — nobody types a figure and the cash/card question is never asked.

A dine-in order refuses a channel. Nobody sits down at a Foodpanda table, and the till would settle it as the platform's money while skipping the hold that occupies the table. The POS hides the control and the API refuses it.

A channel order is completed, not left open. The food was made and handed over; the sale is real. What is outstanding is not the order, it is the **payout**.

What the order remembers

The commission rate and the amount, snapshotted. A rate renegotiated in June must not rewrite what March was owed.

Their order number goes in **channel reference**, so a dispute is traceable from day one.

Settlements — what the platform actually paid

Reporting → Channel settlements. Pick the channel and the period, and Sellify claims every unsettled order on that channel inside it. You type **one figure — Payments received**.

Figure	Who
Gross	The server
Commission	The server
Expected payout	The server
Received	You
Variance	The server: received − expected. **Short is negative**

Still owed across every channel is on the same page.

An order belongs to at most one settlement. Deleting a settlement releases its orders rather than stranding them — so a wrong period is corrected by doing it again, not by editing history.

The reports

- **Channel sales** (Sales) — orders with no channel are the **Counter** row rather than a missing one, so you can read what came through your own door against what came through somebody else's, on one page.
- **Channel settlements** (Money)

What is not in Sellify yet

- **A per-channel price list.** Rates are the same as the counter's. It is a real thing aggregators do and it is a job of its own.
- **An API integration.** Entry is manual today. When we build it, it will write the same rows this screen writes.

22

Employees and staff meals

Menu: Setup → Employees · **Who:** owner and HR (a branch manager reads their own outlet)

An employee is not a user

A cook who never logs in still eats; a cashier who does has both rows. Minting a login for every dishwasher just so the canteen could count their lunch is not a system, it is a chore.

An employee belongs to a company and to at most one branch. **Head office** is a real answer, not a gap.

The book

Field	Notes
Code	What the cashier types at the till. Unique per company
Name, CNIC, photo, designation	
Branch	Or **Head office**
Rider	See Riders and delivery
Meal allowance	Per calendar month
Salary	Read only by HR, the owner, and anybody the owner grants it to
Joined / left	Somebody who has left keeps their history

Allowance and salary are dated rows, not fields

Nothing is ever edited in place. A correction is another row, and the row **in force** on a date is the latest one dated on or before it. So a report for March keeps reading March's figure, and a raise entered today for next month moves nothing today.

The figure on the record is a cache of the row in force right now, rebuilt nightly — which is what brings a dated raise into force with nobody touching anything.

Who reads a salary

- **HR** reads it by definition — that is the job.
- The **owner** grants it to anybody else (Setup → Users, **Can view salary**) — an accountant sees the payroll without being made HR.

Withheld salary is **absent** from the screen, never blank, so nobody reads "you may not see this" as "nobody set one".

HR is not an owner: no settings, no billing, no catalog, no stock, no day close, no till. HR reads the **Staff** reports and nothing else.

Sellify's AI assistant is never asked about staff pay. Salary and what the canteen costs are out of its reach entirely — a scope decision, not a budget one.

A staff meal at the till

The cashier rings the food, attaches the employee by code, and is told the name, the allowance and what is left.

The entitlement outranks every offer. A campaign, a promo code and a customer's standing discount all step aside (the code is reported as not applied, never silently dropped), and a typed manual discount is **refused in a sentence** — the cashier chose one or the other and has to be told which.

What is covered

The lesser of what is left and **the goods**. Allocated across the lines before tax, so a fully covered meal comes to zero with the tax as well. **The delivery charge is never covered.** The till shows *covered* and *to pay* and computes neither.

Two rules that do not apply here

- **The company discount cap does not trim it.** A fully covered meal is the normal case, and this is not a cashier's decision.
- **No manager PIN.** The allowance is the authority. (`approvals.staff_meal_requires_pin` exists for a shop that disagrees.)

Nothing left is not a refusal

The meal is rung, the entitlement covers nothing, and the employee pays for all of it. Refusing a hungry cook at the counter because the month ran out is not the till's job. It is **still a staff meal** — the kitchen slip says so, and the month's report shows that they paid for the whole thing themselves.

What **is** refused, each in a sentence the cashier can read out: an unknown code, somebody who has left (by name and date), the wrong outlet when `staff_meal.own_branch_only` is on, and somebody with no allowance ever set.

The month

The **calendar month**, the 1st to the last day, whether that is the 28th, the 30th or the 31st — measured in business dates, so a meal rung at 1am on the 1st is spent out of the month that just ended.

What is left at the end of the month is lost. No carry forward, ever — an allowance that accumulates is a liability nobody budgeted for.

`staff_meal.period_start_day` makes it a payroll month instead.

What is left is counted, never held in a counter

So it cannot drift, and there is no figure to put back. Paying a recalled held order moves the figure instead of spending the allowance twice; voiding gives it back; a correction that discounts or writes the bill off gives it back, exactly as it gives a promo code back.

Two tills ringing the last of an allowance at the same instant is settled by the database: the second one waits, re-reads what is really left, and covers that.

The reports

Reporting → Reports, under **Staff**. **Everybody with an allowance is a row**, whether or not they ate — a row reading zero is an answer, a missing row is a question. Somebody who has left keeps their line, because the month they left is the month somebody has to settle.

A range is widened out to whole months rather than cut in half: "what was this person allowed for 5–12 March" is not a question with an answer.

Staff meals need the internet

There is none offline. What is left is counted from rows another till changes while this one is dark, and the alternative would be carrying the whole staff book on every tablet.

23

Who can do what

Sellify has nine roles. Eight of them are yours to hand out; the ninth belongs to Sellify.

Role	Belongs to	Sees
Platform admin	Sellify's own staff	The commercial side of your account. No till, no catalog, and none of your reports
Company owner	The customer who pays	Their whole company, every branch
Branch manager	One shop	**Only their own branch**
Cashier	One shop	The till, and what it needs
Kitchen	One shop	Kitchen tickets
Call centre operator	The whole company	The call screen and the customer book
Call centre manager	The whole company	The above, plus the board across every outlet
Call centre HOD	The whole company	The above, plus standing discounts and complaints
HR	The whole company	The employee book and the Staff reports. Nothing else

Company-wide roles have no branch: the owner, the three call-centre roles and HR. An agent orders into any outlet; HR keeps a book that is not one shop's. Everybody else is bound to exactly one branch, and that binding is enforced on the server: a manager who asks for another branch's figures is answered with their own, never with a refusal they could work around.

Two things are **grants, not roles**, because "the cashier who is allowed to haggle" and "the accountant who sees the payroll" are not job titles. The owner gives each on the user:

Grant	What it allows	Who has it anyway
Can override price	Type a price at the till — see Price at the counter	Owner, branch manager
Can view salary	Read salaries in the employee book	Owner, HR

The company-wide roles in one line each

- **A call-centre agent takes orders, and takes no money.** The order is confirmed; the branch takes the cash on its own till. So an agent needs no terminal to quote or to place an order — and still needs one to complete, void or correct anything.
- **A call-centre HOD staffs their own floor** and gives standing discounts. They cannot mint another HOD, and they touch no settings, no billing, no catalog, no stock and no day close.
- **HR keeps the employee book.** No settings, no billing, no catalog, no stock, no day close, no till. A role that exists to keep the book has no business in the sales summary, and asking for one is refused rather than merely hidden.

The short version

- **Sellify handles the account.** Your plan, your invoices, your payments, and approving a new till. **They cannot ring a sale, read your reports or touch your catalog** — that is not a hidden menu, the API refuses it.
- **The owner runs the business.** Everything inside their company, except the commercial terms they are being sold.
- **The manager runs one shop.** Everything operational, in their branch only, except the things that are company-wide by nature.
- **The cashier sells.** Nothing else.

The table

✅ can · ⛔ cannot · 🏬 own branch only

	Sellify	Owner	Manager	Cashier
Selling				
Take a sale on the POS	⛔	✅	✅	✅
Hold, recall, void a sale	⛔	✅	✅	✅
Give a manual discount / FOC	⛔	✅	✅	with a PIN
Approve someone else's discount (PIN)	⛔	✅	✅	⛔
Setting up				
Create a branch	⛔	✅	⛔	⛔
Edit a branch (prefix, day start, NTN)	⛔	✅	⛔ sees own only	⛔
Add staff	⛔	✅ any role	🏬 cashier / kitchen only	⛔
Change company settings	⛔	✅	⛔	⛔
Change branch settings	⛔	✅	🏬	⛔
Edit the company profile, logo, colours	⛔	✅	⛔	⛔
Set up taxes	⛔	✅	⛔ read only	⛔ read only
Set up printers	⛔	✅	🏬	⛔
Catalog				
Categories, products, variants, modifiers	⛔	✅	✅	⛔ read only
Recipes	⛔	✅	✅	⛔
Stock				
Items, suppliers, units	⛔	✅	✅	⛔
Purchases, adjustments, transfers, counts	⛔	✅	🏬	⛔
Production — making a batch	⛔	✅	🏬	⛔
Selling more				
Campaigns and promo codes	⛔	✅	✅	⛔

	Sellify	Owner	Manager	Cashier
Knowing where you stand				
Reports	⛔	✅ all branches	🏬	⛔
Insights (AI)	⛔	✅	🏬	⛔
Close the day	⛔	✅	🏬	⛔
Reopen a closed day	⛔	✅ with a PIN	✅ with a PIN	⛔
Money and the account				
See the plan, usage and invoices	✅ everyone's	✅ own	⛔	⛔
Say "I have paid" with a reference	⛔	✅	⛔	⛔
Change the plan	✅	⛔	⛔	⛔
Mark an invoice paid	✅	⛔	⛔	⛔
Suspend / reopen a company	✅	⛔	⛔	⛔
Switch AI on for a company	✅	⛔	⛔	⛔
Create packages and prices	✅	⛔	⛔	⛔

The kitchen role signs in for kitchen tickets only; it takes no sale and changes no data.

What stays with Sellify

These are the commercial controls, and they stay with us deliberately — an owner who could move their own company onto a bigger plan, or mark their own invoice paid, is not on a plan at all. Asking for one of these is not blocked by a hidden menu: **the API refuses the request**, so it makes no difference what is sent to it.

	Ask us to
Set your company up — the company, its first branch and your owner login	Done once, on the call
Approve a new till	See Terminals
Move you to another plan	
Raise an invoice, and record that you have paid it	
Switch **AI** on for your company, and set its monthly budget	
Suspend or reopen a company	
Give a late account a fresh grace window	

You can see your plan, your usage and your invoices, and you can tell us you have transferred the money — see Billing. That is the whole of the boundary, and it is the same for every customer.

Two things worth knowing, because they surprise people:

- **A company owner cannot change their own plan, status or trial dates.** The API rejects the attempt rather than quietly ignoring it, so nobody is left wondering whether it worked.
- **Shutting a lost till out is yours; letting a new one in is ours.** You can disable a stolen tablet the second you notice, without ringing anybody. Approving a device is our call, and a re-enabled terminal pairs again from scratch.

24

Import and export

Where: the **Import / Export** button on Products, Items, Item categories, Units, Suppliers and Stock on hand · **Who:** owner, branch manager

One mechanism, not a screen per list. Export writes the **same columns** the import reads, so a file exported, edited in Excel and imported again is the round trip a shop actually does.

Starting from scratch

1. Press **Import / Export** on the list you want to fill.
2. **Download the template** — a CSV with the right headings and nothing else.
3. Fill it in Excel and save as CSV.
4. **Preview** it.
5. If it reads right, **Import**.

Preview first, always

Nothing is written until you press Import. The preview tells you, before anything moves:

- how many rows would be **created**
- how many would be **updated**
- every row that is **wrong** — the row number, the column and the reason

Fix the file and preview again. There is no penalty for previewing.

The import is one transaction

A file with a bad row on line 400 leaves your data **exactly as it was**. There is no half-finished import to unpick.

It matches on the code, not the name

List	Matched on
Products	SKU
Items	Item code
Suppliers	Name
Units	Code

A match **updates** that row. No match **creates** it. A match on something you had deleted **brings it back**, with its history still attached — a supplier comes back with its purchases, a retired item comes back to the ledger that still points at it.

An import never deletes. Removing a line from the file removes nothing from Sellify.

Products carry their variants

One CSV row per **variant**, grouped by the product code. So:

```
SKU        Product   Variant      Price
SHIRT-01   Shirt     Red / Small   1200
SHIRT-01   Shirt     Red / Large   1400
```

is two lines of one product. A row whose product has no variant columns is the single "Regular" variant every simple product already carries.

Opening stock

Imported **per branch**, and it lands on the stock ledger as an `opening stock` movement like every other — never written straight into the balance.

Importing the same item twice adjusts to the new figure, it does not add to it. So a count you re-import after a correction gives you the corrected figure, not double.

Exporting

The same button. Everything the list is showing, in the import's own columns. Use it to:

- hand your accountant a price list
- edit a hundred prices in Excel and put them back
- take your data with you. **Leaving is never held over you.**

If a row will not import

Message	Usually means
A required column is missing	The template changed — download it again
"Unit `KG` not found"	Import your units first. Codes must match exactly
"Category not found"	Import item categories first
A price is not a number	Excel has put a comma or a currency symbol in it

The order that works from empty: **units → item categories → suppliers → items → products → opening stock.**

25

The Android till

Who: cashiers, on a tablet or a Sunmi terminal

The Android app is a **terminal**, not a second Sellify: the same terminal row, the same API, the same server-built receipt. Everything in POS applies. This page is only what is different about a device.

Setting one up

1. Install the APK we give you.
2. Type the branch's **pairing code** and name the till.
3. Ring us to approve it — see Terminals.
4. Sign in. Sign in **online at least once**; that is what makes offline sign-in possible later.

The app checks for a new version itself. An update is only **forced** when a build is genuinely unsafe to keep selling on — a shop mid-service is never stopped by an update it could take after closing.

Selling with no internet

The till prices the cart itself from a **price bundle** the server issued, prints what it quoted, and uploads the sale when the network comes back. **The server re-prices it from the ids at that point**, exactly as it always does. If the figures disagree the sale is saved at the server's figure and flagged for a manager — never silently absorbed, never silently kept.

Works offline	Does not
Lines, variants, modifiers, deals	Promo codes (their caps are counted on the server)
Tax, including the cash/card split	A customer's standing discount
Campaigns	Staff meals
A manual discount within your cap	A price override
A manager's PIN for a discount or an FOC	Reopening a closed day
Kitchen tickets	The day close
Held orders, today's sales	The negative-stock refusal

Each refusal is a plain sentence on the screen — never a dead button.

A bundle older than 24 hours refuses the sale. A shop that has been off the network for two days is selling last week's prices.

Selling offline is per till (Setup → Terminals, **Sell without internet**). Turn it off for a device on the shop's wifi that you would rather stopped than half-worked; leave it on for the one on a 4G dongle.

The outbox

Sales waiting to upload. It drains when the network returns, when the app comes to the front, on a timer, on **Sync now**, and — the case none of the others cover — through Android's own job queue, so a tablet swiped away at closing time still uploads.

A sale carries the id the device generated, so **one timed-out response never charges a customer twice**.

A day will not close while a till still has unsynced sales, and the refusal names the tills so a manager knows where to go. A till that is off the network cannot report, so its last figure stands and the day stays open — that is the safe way round on purpose.

A sale for a day that has since closed

Refused, and **kept**. It waits on the outbox screen for a person: a manager reopens that day and it goes through. Retrying it every sixty seconds helps nobody and dropping it loses a sale that was really made. **A backdated sale stays backdated** — it is never quietly re-filed onto today.

Signing in offline

A cashier who has signed in **on this terminal** before may sign in offline for 7 days since their last online sign-in (`pos.offline_login_days`). Nobody else — a dead router at 8am must not close the shop.

Signing in offline does not restart the clock; only an online sign-in does.

Receipts and kitchen tickets

Online, the device prints the bytes the server built. Offline, it builds the same document and prints it. Both are checked against the server automatically, so the two can never drift.

An offline receipt prints a **local ref**; the real order number arrives at sync and shows on the orders list and on any reprint.

Printing never rolls a sale back. The money is taken whatever the paper does; a printer that is off, out of paper or unpaired says so on the same screen with **Try again** and **Set up a printer** beside it.

The device's own printers

Set on the device (role, connection, paper, copies, cut, drawer). The branch's printer rows pre-fill the form, but **the device's own binding wins** — which tablet is paired to which Bluetooth printer is nobody else's business and is pushed nowhere.

Test print is the server's own test receipt when the network is up, and otherwise a plain test page the device builds — the printer's name, where it is, and a column ruler. A till that has never been online must still be provable before a customer is standing there.

Barcode scanning

Three kinds of scanner, all of which just work:

- **A USB or Bluetooth scanner** — it types the code and presses Enter
- **A Sunmi hardware trigger**
- **The camera**, for a tablet with no scanner at all

A scan reads the same menu the grid is drawn from, so it works with the internet off and can never find something the cashier cannot see.

A barcode on the **variant** names exactly what was scanned; a barcode on the **product** names the dish and still asks for the size. Scanning the 500ml never lets the till decide it was the 1.5 litre.

A scan that misses stays quiet unless the code could not have been typed on purpose — telling a cashier that "rice" is not a barcode helps nobody.

A second screen for the customer

Plug one in and it starts by itself. It shows the lines as they are rung, then the total.

While the cashier is ringing, the figure is labelled **ITEMS** — the goods, before tax. **Tax** and **TOTAL** appear only once the sale is priced. A figure called *Total* that then grows when the tax lands reads as a surprise charge, which is the one thing a customer display exists to prevent.

It never touches the sale. A till with one screen, a panel unplugged mid-order — all ordinary.

Sending us a diagnostic

Settings → **Send diagnostics** on the device. It uploads the app's own logs and nothing else: **no passwords, no PINs, no tokens and no customer details**.

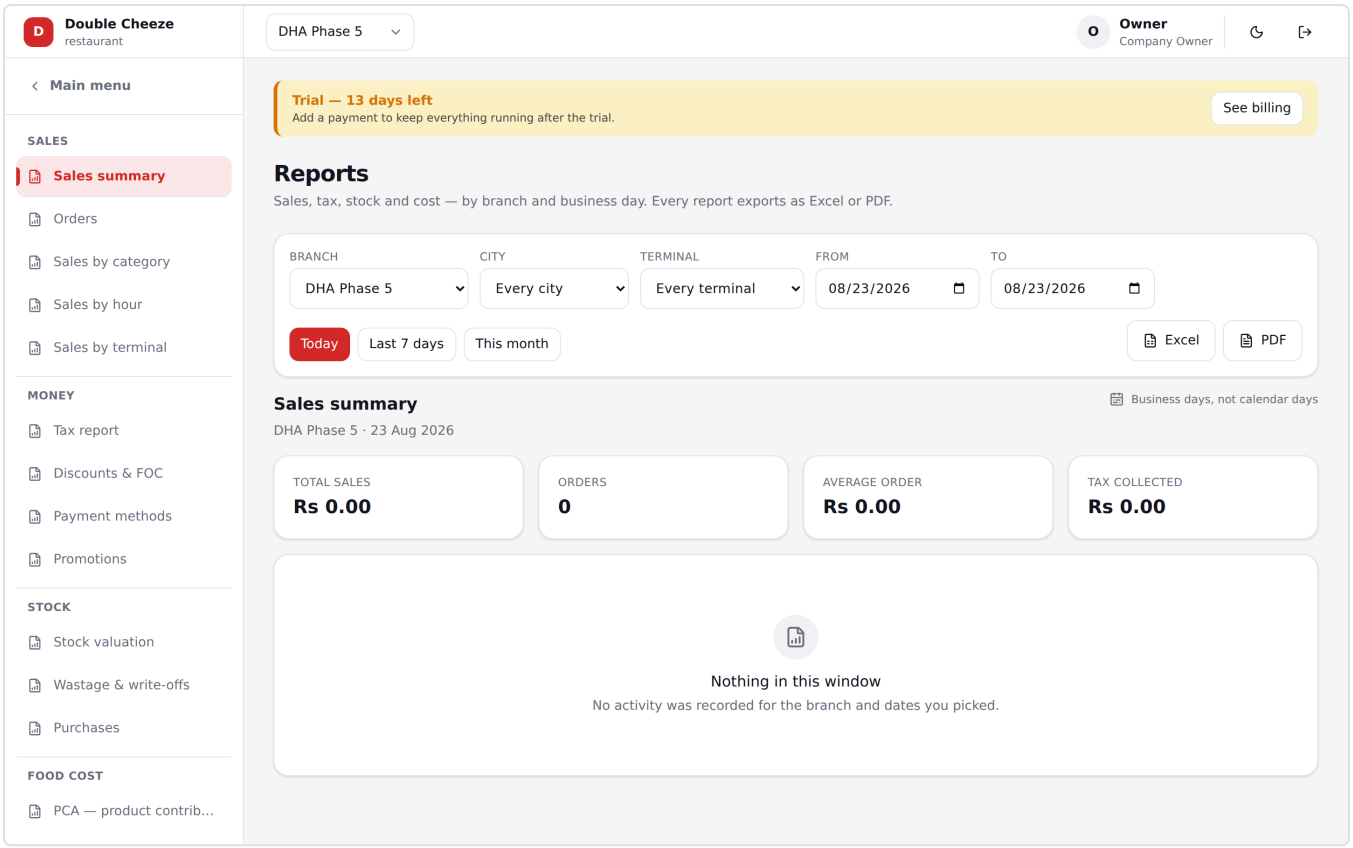
26

Reports

Menu: Reporting → Reports · **Who:** owner, branch manager

A cashier cannot open this page. A branch manager always sees **their own branch**, whatever they ask for.

Using the page



Every report reads the same service the Excel and PDF exports do.

- **The sidebar** — while you are on this page the menu on the left becomes the list of reports your company can run, filed under Sales, Money, Stock and Food cost. The report itself then gets the whole width of the screen.
- **Main menu** — the button at the top of the sidebar puts the ordinary menu back. Press **Reports** in the same place to return to the report list.
- **Top** — branch (owners can pick "All branches"), **From** / **To**, and the quick buttons **Today**, **Last 7 days**, **This month**.
- **Excel** and **PDF** hand you the file.

On a narrow screen there is no sidebar, so the report is picked from a **Report** dropdown at the start of the filter bar instead.

Dates are **business days**, not calendar days: a 2am sale is reported on the day it was actually sold.

> The exports are generated on the server from the very same figures the screen read, so a download can never disagree with what you were looking at.

The reports

The list is filed under headings, and a report your business type or your plan does not earn is **refused by the server**, not merely hidden — a pure retailer has no PCA, a pure restaurant has no margin report.

Sales

Report	What it answers
Sales summary	Orders, gross, discount, net, tax, total and average order, per business day
Orders	Every completed order — number, time, type, cashier, totals, FOC flag
Sales by category	Where the money came from
Sales by hour	Your trading curve — which hours are worth staffing
Sales by terminal	Which till took what
Order segments	Phone work against counter work: source × order type. The only way to tell whether a new call desk is earning its keep
Price overrides	What was sold at a typed price, against what it should have been, by whom — see Price at the counter
Channel sales	Foodpanda and the rest against your own counter, on one page. Orders with no channel are the **Counter** row

Money

Report	What it answers
Tax report	One row per authority + tax + **rate**, with tax split cash vs card, and the invoice list underneath. This is what a filing is made from
Payment methods	Cash / card / wallet / credit, count and amount
Discounts & FOC	What was given away, by reason, with the approver named. FOC counted separately
Promotions	What each campaign and code did
Channel settlements	What each platform owes you, what it paid, and the difference

Food cost *(restaurant)*

Report	What it answers
PCA	Per dish: quantity, net sales, theoretical cost, contribution and **food cost %**
Daily PCA	Per outlet per business day, with a city roll-up
Item consumption	Per stock-room category, and the items under it
Consumption comparison	Per item, against the branch's **own counted** closing
Consumption variance	Should-have-used vs actually-used, per ingredient. The gap is waste, theft or a wrong recipe
Margin *(retail)*	Net sales against the **actual** cost the sale took off the shelf

A dish with no recipe is **not** costed at zero and buried — its sales are reported separately, because a percentage quietly reading low is worse than one that explains itself.

Customers *(call centre)*

Report	What it answers
Customer list	Who they are, what they have spent, their stars
Dormant customers	Who has stopped ordering — the list worth ringing
Customers by city	Grouped on the customer's own city, not the branch's
Customer orders	One customer's history
Frequent customers	Who is worth keeping
Customer discounts	Who has a standing discount, at what, and who gave it
Complaints	By category, by branch, and what was done about them
Call centre agents	What each agent took

These name people, so **none of them is ever offered to the AI assistant**.

Staff *(staff meals)*

Report	What it answers
Staff meals	Per employee per month: allowed, used, what is left — see Employees and staff meals
Staff meals daily	The same, day by day

Read by HR, the owner, and anybody the owner grants it to. **Never offered to the AI assistant** — salary and what the canteen costs are out of its reach entirely.

Delivery *(call centre)*

Report	What it answers
Delivery performance	On time, late, failed, and the average minutes against the promise
Rider performance	What each rider carried, and their cash variance
Failed deliveries	Every one, with its reason. Names the customer, so it is never offered to the AI

Stock

Report	What it answers
Stock valuation	On-hand quantity × that branch's own average cost
Purchases	Supplier documents, goods, freight, and the input tax you can claim
Wastage & write-offs	Adjustments by reason, at cost

Which way is the bad way

Some columns say so. **Variance** means over is bad, so a positive figure reads red. **Shortfall** means short is bad, so a **negative** figure reads red — a count variance is counted − expected, the shelf's own convention, so a missing kilo is a negative number.

A table past eight columns pins its first column, so the row keeps its name while the figures scroll.

Reading the two that matter most

PCA. Food cost % is theoretical cost ÷ net sales. Anything over about 40% deserves a look: either the recipe is wrong, the portion is too big, or the price is too low. The contribution column is what the dish actually leaves you after ingredients.

Consumption variance. If the kitchen should have used 12 kg of chicken and actually used 15 kg, that 3 kg is the number to chase. Wastage entries explain part of it; the rest is portioning or shrinkage.

Reading a whole city

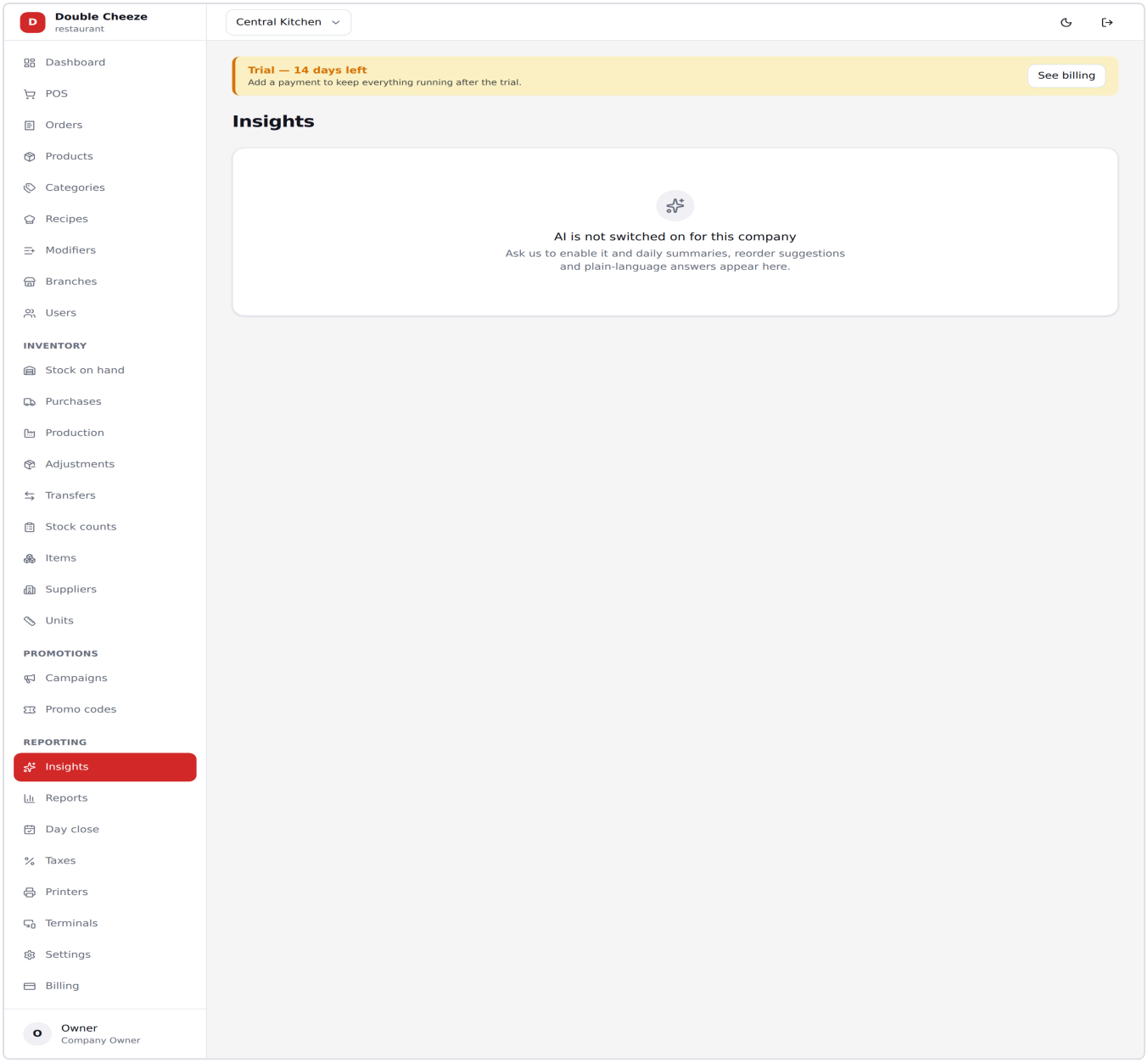
A chain with three shops in Lahore reads its takings by city as often as by shop. The **City** filter beside Branch adds those shops up — the report says *"Lahore · 3 branches"* so there is no doubt what was counted.

Branch wins when both are set: naming one shop answers for that shop. A branch-bound manager is pinned to their own branch, and the city they ask for is ignored exactly as the branch id is.

Stock on hand carries the same filter: pick a city and the quantities add up across its branches, valued at what those branches actually paid.

27

Insights (AI)



Insights. The model writes the sentence; every figure in it came from a report.

Menu: Reporting → Insights · **Who:** owner, branch manager

Plain-language help built on your own reports. If the page says AI is not switched on for your company, ask us to enable it — an owner cannot turn it on themselves.

What it will and will not do

- **It never sees your database.** To answer a question it asks for a *report* by name and a date range; the report runs inside your company's own scope and it writes the sentences

around the figures.

- **It never does arithmetic.** Every number comes from the same reports you can open yourself.
- **Customer names and phone numbers never leave.** They are stripped before anything is sent, so an answer says "Customer #412" and never a name.

Ask about your data

Type a question and press **Ask**:

- "How much did we take today?"
- "What sold best last week?"
- "How does this month compare with last month?"

Under the answer it tells you **which report it looked up**. If the reports cannot answer the question, it says so rather than guessing.

Daily summary

Yesterday against the same weekday last week, the best and worst sellers, and anything that looks off — discounts creeping up, food cost drifting. A restaurant's summary includes food cost; a shop's includes margin.

Press **Generate** to write one now, or ask us to switch on automatic summaries — then it is written by itself each time a day is closed, and is waiting for you in the morning.

Reorder suggestions

What is running out and how much to buy. **These quantities are arithmetic, not the AI's opinion:**

```
used per day = everything that left the shelf ÷ days looked at
days left    = what is on hand ÷ used per day
order        = (cover you want + supplier lead time − days left) × used per day
```

Two settings drive it, per branch: **Supplier lead time (days)** and **Stock cover to keep (days)** in Settings → Inventory. The AI only orders the list and explains it; the table under the text is the real figure to act on. Items already covered past the next delivery are not listed at all.

Campaign ideas

Two or three offers read off the last 30 days — quiet days, quiet hours, categories that already sell. It writes the idea; **you** create the campaign on the Campaigns page. It never launches an offer by itself.

What it costs you

Top right shows **tokens used this month out of your allowance**. When the allowance runs out the feature stops until the 1st rather than billing on. Insights are stored, so re-opening the page costs nothing — only pressing Generate or Ask does.

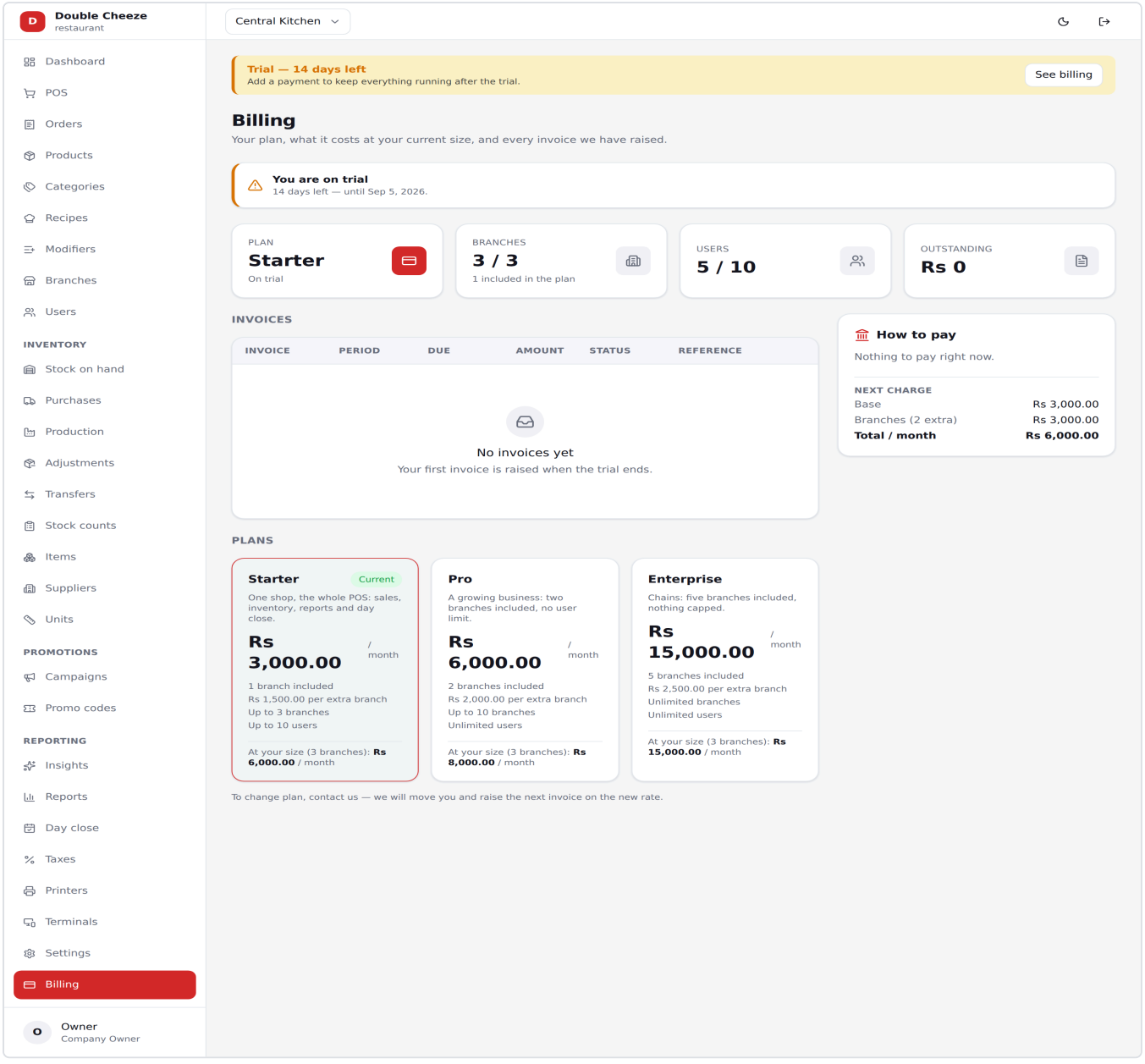
28

Billing

Menu: Billing · **Who:** company owner only

Your plan, what it costs at your size, and every invoice we have raised.

What you see



Your plan, your usage against its limits, and your invoices.

Plan	Which plan you are on, and whether you are on trial, active or payment-due
What it allows	Branches, users and **tills**. A till past your plan's number is refused when it is approved, not hidden
Branches	How many you use out of what the plan allows
Users	The same for staff
Outstanding	What is unpaid right now

Under that: your invoices, the plans priced **at your current size**, and a **How to pay** panel.

How a plan is priced

A plan charges **both ways at once**: a base price for the company, plus a price for every branch past the ones the plan includes.

```
Pro — Rs 6,000/month, 3 branches included, Rs 1,500 per extra branch
Five branches  →  6,000 + 2 × 1,500  =  Rs 9,000 a month
```

The **Next charge** box breaks it into base and branches so there is no surprise.

Paying

We collect by **bank transfer**, and a person confirms it:

1. Transfer the invoice amount to the account shown in **How to pay**.
2. **Quote the invoice number** (`SUB-2026-0001`) as the payment reference.
3. Press **I've paid** on that invoice and enter your bank's transaction reference.
4. We check it against our statement and mark it paid. The invoice stays **Unpaid** until we do — your reference is a claim, not a payment.

Trial, grace and suspension

- A new company gets a **14-day trial**.
- When the trial or a paid period runs out you enter a **grace period** (7 days by default): the till keeps working and a banner appears on every admin screen.
- If grace runs out, the company is **suspended** — nobody can sign in until a payment is recorded.

We do not cut a shop off mid-service; that is what the grace window is for. But do not leave it to the last day.

What you cannot do here

Changing your own plan, status or trial dates is not something the screen hides — the API refuses it. Ask us and we will move you; the new rate applies from the next invoice.

Who can do what sets out the rest, role by role.