Turning paid web orders into a live competition entry list, without writing code

0 · 2026-09-19

A climbing gym running a competition sells entries like any other product: a line in the web shop, a price, maybe a category per age group. The entries arrive over three weeks. Then, the night before, somebody exports orders, filters them by hand, and pastes names into a spreadsheet so the registration desk has a list.

That last step is the one worth removing. Not because spreadsheets are bad — the desk genuinely wants a spreadsheet — but because the copy happens once, late, from data that has been changing for three weeks. Anyone who enters on the final evening is missing from it.

A webhook fixes the timing rather than the tool. The gym keeps its spreadsheet; the spreadsheet fills itself as each entry is paid.

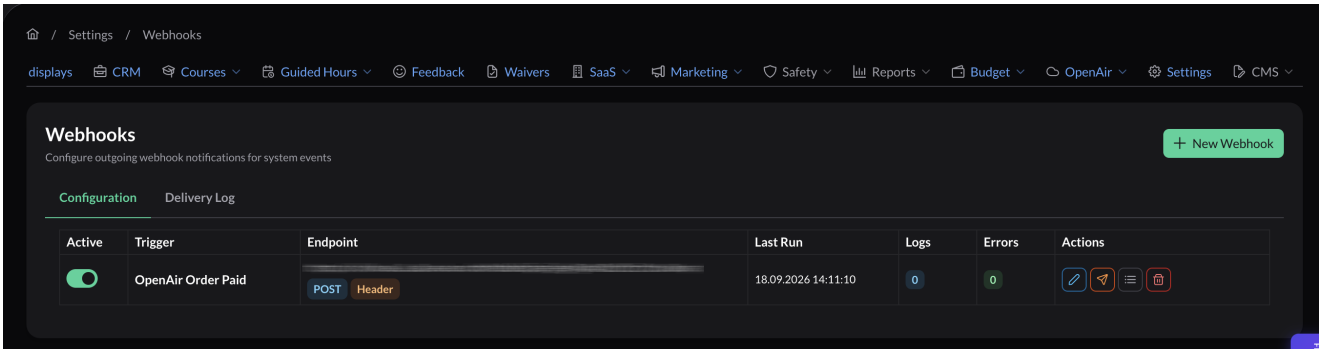
What is a webhook in a gym management system?

A webhook is an HTTP request GymKeeper sends to a URL you choose, at the moment something happens. Nothing polls, nothing is scheduled, and you do not need access to the database. You give GymKeeper an address; it posts JSON there when the event occurs.

GymKeeper fires webhooks for orders, memberships, customers, invoices, courses, tickets and door access. For orders there are five events — created, paid, payment cancelled, abandoned and delivered — and each fires under three names, so you can subscribe to one shop or to all of them at once:

Any shop	OpenAir	Web shop
`order_paid`	`openair_order_paid`	`webshop_order_paid`

The one that matters for a competition is **paid**. An order that exists is not an entry; an order somebody has paid for is. Subscribing to `order_created` would fill the sheet with abandoned carts.



What does the order-paid webhook actually send?

The payload describes the order in plain names rather than database columns:

```
{
  "event": "paid",
  "channel": "openair",
  "fired_at": "2026-09-18T14:11:10+00:00",
  "order": {
    "id": 25315,
    "total": 55,
    "paid": true,
    "paid_at": "2026-09-18 14:11:07",
    "payment_method": "paytrail",
    "status": 101,
    "status_key": "payment_accepted",
    "is_test": false
  },
  "customer": {
    "id": 982,
    "firstname": "Jarmo",
    "lastname": "Annunen",
    "email": "jarmo@annunen.fi",
    "phone": "0401234567"
  },
  "rows": [
    {
      "id": 37909,
      "title": "Boulder Battle 2026 — Adult",
      "product_id": 17,
      "feature_id": 17,
      "quantity": 1,
      "unit_price": 55,
      "vat_percent": 14,
      "total": 55
    }
  ]
}
```

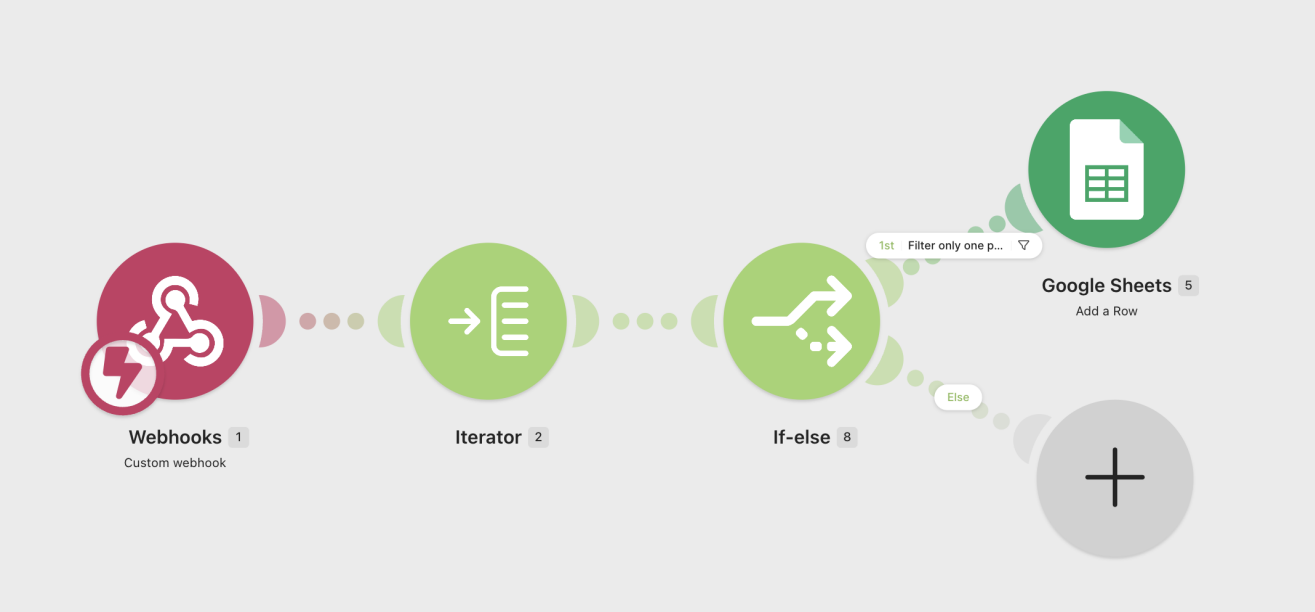

Two fields decide the whole design. **`rows` is an array** — one order can contain a competition entry, a chalk bag and a day pass. And **`channel`** tells you which shop it came from, so one scenario can serve both without guessing.

How do you get one row per competitor into Google Sheets?

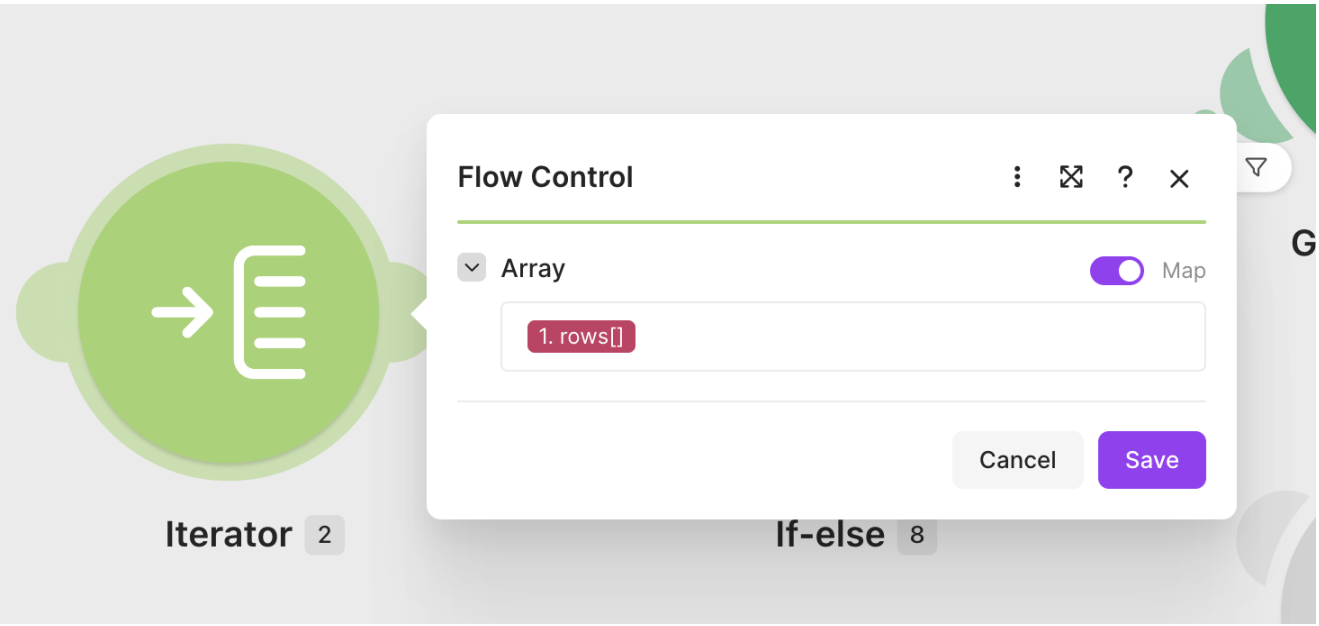
The shape of the problem is that a webhook arrives per **order**, and the spreadsheet wants a line per **entry**. Those are not the same number. In Make.com the chain is three modules:

1. **Custom webhook** — Make gives you a URL. Paste it into GymKeeper as the endpoint for `openair_order_paid`. Send one order through so Make can learn the data structure.
2. **Iterator** — point it at the `rows` array. Everything after this runs once per row rather than once per order, which is what turns a family buying three entries into three lines.
3. **Filter**, then **Google Sheets → Add a row**. The filter decides whether the row in hand is a

competition entry; the Sheets module writes it.



Map the competitor's name and email from `customer` — which stays available after the iterator, because it belongs to the order — and the category from the row's `title`. Add `order.paid_at` if the desk needs to settle who registered first for a capped category.

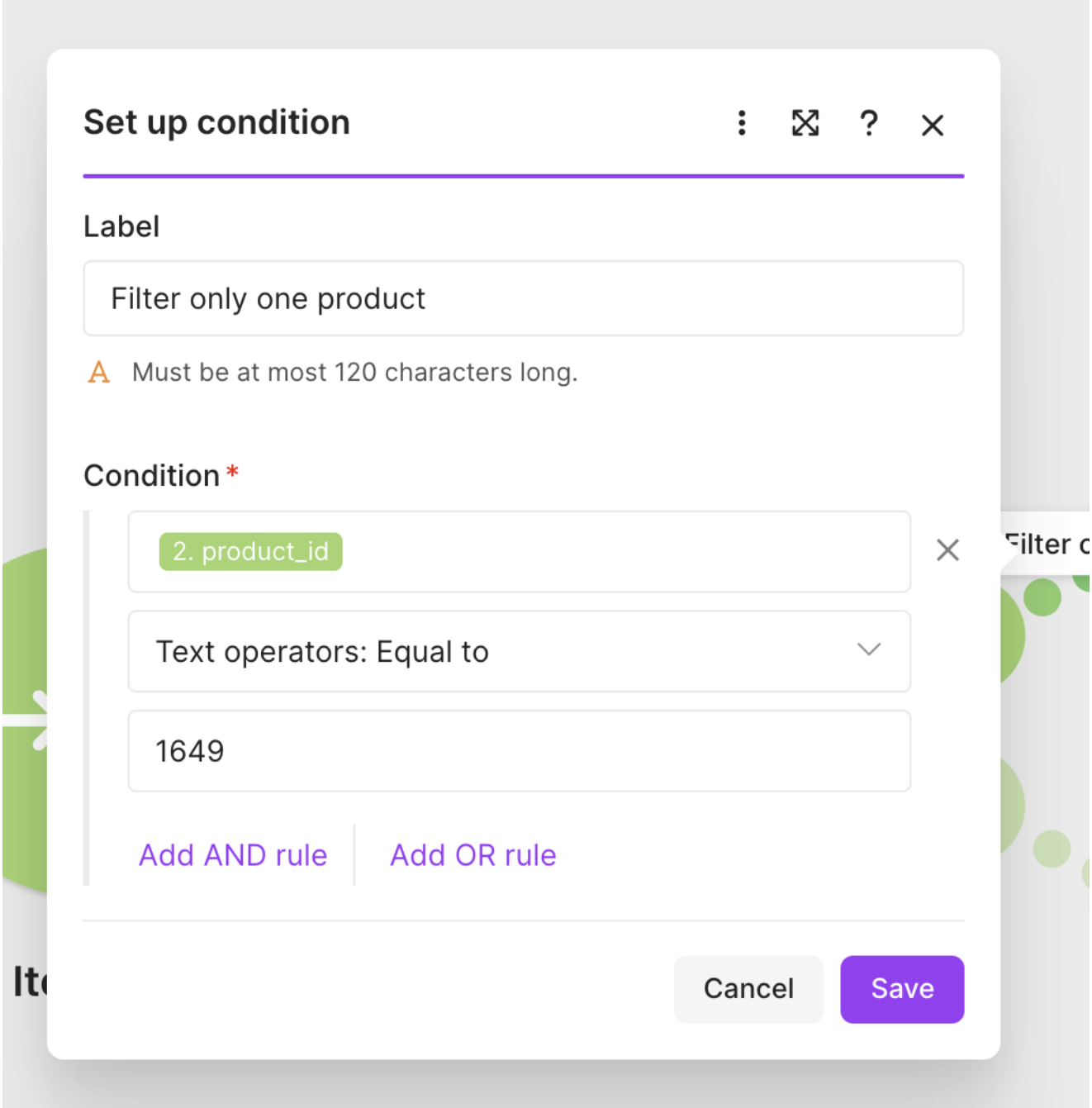


How should the filter identify a competition entry?

Match on **`product_id`**, not on the title.

The title is the product name as it read at the moment of sale, and product names get edited: "Boulder Battle 2026" becomes "Boulder Battle 2026 — Adult" when a youth category is added, and a filter matching `title contains "Boulder Battle 2026"` keeps working right up until someone renames it to "Boulder Battle '26". The failure is silent. Entries keep arriving, the scenario keeps returning success, and the sheet simply stops growing.

`product_id` is the product's identity in GymKeeper and does not change when the name does. Look it up once in the product editor and filter on the number. If the competition has several categories as separate products, filter on the set of ids and let the title carry the category into the sheet — that way the sheet is grouped correctly and the filter still survives a rename.



Four things that will bite you

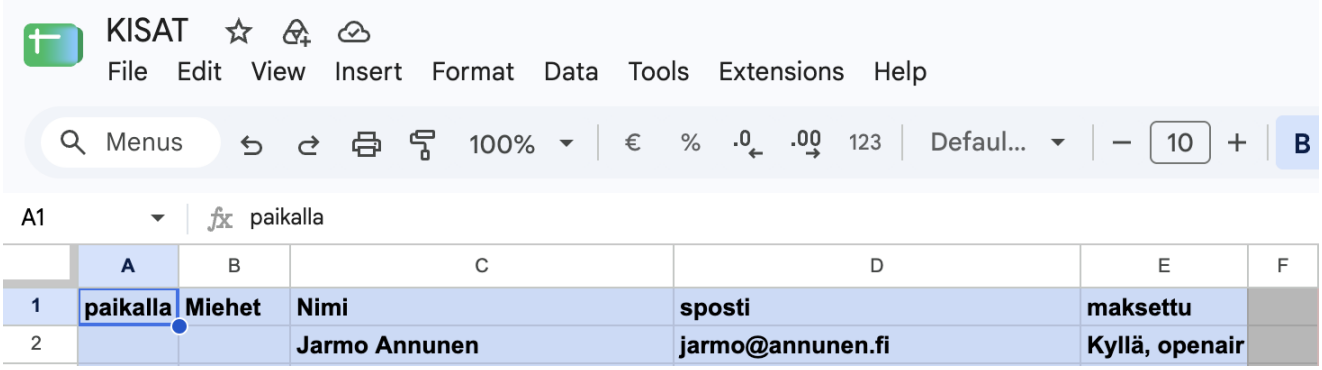
Test orders look exactly like real ones. Every payload carries `is_test`, set from the order's own testing flag. If you try the flow with a test payment — and you should — that entry lands in the sheet alongside the real ones, and nobody notices until a name at the registration desk belongs to nobody. Add `is_test = false` to the filter before the competition opens, not after.

Delivery is a single attempt. GymKeeper posts once and records the result; there is no automatic re-send if your endpoint is down or slow. The stored retry count is not currently

honoured. Make.com has its own error handling and queueing, so let it hold the retry: a scenario that errors is recoverable from Make's history, whereas a webhook that was never accepted is gone. Check `webhooks_log` in the admin if you are unsure whether something was delivered — it records the response and status code of every attempt.

A zero-minute delay puts your endpoint inside the payment redirect. With no delay, GymKeeper sends the request while the customer is being returned from the payment provider, so a slow endpoint makes them wait on the "payment complete" page. Setting a delay of one minute moves delivery onto a scheduled job instead, and the customer's redirect no longer depends on your automation. For a spreadsheet that a human reads the next morning, a delay costs nothing.

One webhook, two shops. If the gym sells through both OpenAir and the web shop, subscribing to `openair_order_paid` alone quietly misses half the entries. Either subscribe to the combined `order_paid` and read `channel` if you need to tell them apart, or subscribe to both per-shop events. Orders placed before the shop channel was recorded count as web shop.



	A	B	C	D	E	F
1	paikalla	Miehet	Nimi	sposti	maksettu	
2			Jarmo Annunen	jarmo@annunen.fi	Kyllä, openair	

Securing the endpoint

A webhook URL is a public address, and anything that knows it can post to it. GymKeeper offers four ways to prove the request is genuinely from your gym: a custom header, a query parameter, a bearer token, or basic auth — set per subscription alongside the endpoint. Make.com can require a matching header on its side. For a competition list the stakes are low; for anything that writes to a finance system, set one.

What this pattern is good for

The same three modules — iterate rows, filter to a product, write somewhere — cover a family of jobs that otherwise become manual work at exactly the wrong moment:

- a course or clinic roster that fills as places are paid for
- a rental waiver checklist for a group booking
- a shoe-size list for a beginners' session, where the size is a product variant
- a merchandise pick list for the shop, filtered to physical products only

None of them need code. They need the event to arrive at the moment the money does.

Frequently asked questions

Does this work for the web shop as well as OpenAir? Yes. Subscribe to `webshop_order_paid`, or to `order_paid` for both shops at once. The payload is identical apart from `channel`.

What happens if someone cancels or refunds after paying? The paid webhook has already fired, so the row is already in the sheet. Subscribe to `order_payment_cancelled` if you need the other half of the story; a refund handled by staff moves the order's status, which you can watch through `order_delivered` and the status field rather than assuming the sheet stays correct on its own.

Can I send this somewhere other than Make.com? Yes. The webhook is plain JSON over HTTP, so Zapier, n8n, a Google Apps Script web app or your own endpoint all work the same way. Make.com is used here because its iterator makes the one-row-per-entry step explicit.

Do I need a developer? No. Everything above is configuration: a URL pasted into GymKeeper, three modules in Make.com, and a product id copied from the product editor.

GymKeeper is climbing gym management software developed by GymKeeper Oy in Finland, covering memberships, billing, access control, POS, the web shop and integrations. If you are new to the wider system, start with what climbing gym management software covers, or see how memberships and punch cards are sold through the same web shop these orders come from.