FIELD NOTES

# Saleor Field Guide

Saleor checkout, channel and stock fixes.

**51 fixes**, each one a written note with diagrams and a small Python and Node.js script that finds the problem and repairs it. All 51 have a script in the public repo.

**Before you run anything.** Every script starts in a dry run: it reports what it would change and writes nothing. Read that output first, then set `DRY_RUN=false` when you are satisfied it has understood your data.

Scripts: github.com/allanninal/saleor-fixes
Guides: allanninal.dev/saleor
All 14 repos: github.com/allanninal

## The fixes

1 **Abandoned checkout keeps stock reserved past TTL**
A shopper adds items to a Saleor checkout, Saleor reserves that stock in the warehouse right away, and then the shopper closes the tab and never comes back. The reservation is supposed to expire and free the stock on its own. But when the background job that clears expired reservations is misconfigured, delayed, or simply not running, that stock stays debited to a cart nobody will ever finish. Here is why that happens and a small script that finds the stale holds and releases them safely.
`release_stale_reservations.py`
https://www.allanninal.dev/saleor/abandoned-checkout-stock-not-released/
https://github.com/allanninal/saleor-fixes/tree/main/abandoned-checkout-stock-not-released

---

2 **Bulk imported variants missing channel listings**
You ran an import, thousands of variants came in clean, skus matched, attributes matched, stock landed in the right warehouses. Then someone notices a chunk of them are simply not buyable. No error, no crash, just a variant that never shows a price and never shows up at checkout. Here is why Saleor leaves freshly created variants unsellable by default and a script that finds every one of them and repairs only the ones it can safely price.
`find_missing_listings.py`
https://www.allanninal.dev/saleor/bulk-imported-variants-missing-channel-listings/
https://github.com/allanninal/saleor-fixes/tree/main/bulk-imported-variants-missing-channel-listings

---

3 **Bulk stock update leaves stale or partial rows**

You fire one stockBulkUpdate or productVariantStocksUpdate call with a list of warehouse rows and move on, assuming every row landed. Then a customer reports a warehouse still shows the old quantity, or a report shows a variant that never actually changed. The batch call did not fail loudly. It just did not fully agree with what you asked for. Here is why that happens and a script that finds every row that is still stale and repairs only the ones it can safely explain.

`reconcile_bulk_stock.py`

https://www.allanninal.dev/saleor/bulk-stock-update-leaves-stale-rows/

https://github.com/allanninal/saleor-fixes/tree/main/bulk-stock-update-leaves-stale-rows

---

4 **Captured amount doubles the order total**

Finance flags an order where the captured amount is exactly twice what the customer should have paid. Nothing in the Saleor admin threw an error. No mutation failed. But a retried checkout, a redelivered webhook, or a second manual capture call each reported a full charge for the order, and Saleor added both up without ever checking whether the total made sense. Here is why totalCaptured can run past order.total and a script that finds every order where it has, without touching a single dollar until a human says so.

`flag_over_captured.py`

https://www.allanninal.dev/saleor/captured-amount-doubles-order-total/

https://github.com/allanninal/saleor-fixes/tree/main/captured-amount-doubles-order-total

---

5 **Charged amount stays stale after manual capture**

A merchant or an app captures a payment straight at the gateway, or fires off a transactionRequestAction, and the money moves. But the gateway's confirmation only comes back asynchronously, so Saleor never gets a chance to update its own books. Order.totalCharged and the transaction's chargedAmount keep sitting at the pre-capture number, chargePendingAmount stays open, and the order looks unpaid even though the customer's card was already charged. Here is why Saleor's ledger goes stale and a script that finds the mismatches against the gateway and reports the true state back, safely.

`reconcile_charged_amount.py`

https://www.allanninal.dev/saleor/charged-amount-stale-after-manual-capture/

https://github.com/allanninal/saleor-fixes/tree/main/charged-amount-stale-after-manual-capture

---

6 **checkoutCreate returns a stale existing checkout**

A shopper starts a new session, calls checkoutCreate, and gets back a cart with someone else's lines, an old voucher that no longer applies, or metadata from a session that ended hours ago. It looks like Saleor is silently reusing a checkout. It is not, at least not anymore. Saleor 3.x always inserts a fresh Checkout row. The stale cart is coming from one layer up, in the storefront or app that cached a checkout token and keeps replaying it. Here is why that confusion still happens and a script that detects the reused, stale checkouts so a human can fix the real cause.

`flag_stale_checkout.py`

https://www.allanninal.dev/saleor/checkout-create-returns-stale-checkout/

https://github.com/allanninal/saleor-fixes/tree/main/checkout-create-returns-stale-checkout

---

7 **Checkout lines fail to add without a country or pickup**

A guest adds an in-stock item to their cart. No shipping address yet, that comes later in checkout. Saleor answers with something that looks like an out-of-stock error, even though the warehouse is holding plenty of units. The product is fine. The channel is missing a way to tell Saleor which warehouse to even look at.

`flag_checkout_country_risk.py`

https://www.allanninal.dev/saleor/checkout-lines-add-fails-no-country/

https://github.com/allanninal/saleor-fixes/tree/main/checkout-lines-add-fails-no-country

---

8 **Concurrent checkouts oversell the same stock**

Two customers hit checkout on the last unit of a SKU within a second of each other. Both checkouts finish. Both orders look normal. Then the warehouse counts stock and finds one more allocated unit than actually exists. This is a known race in how Saleor checks stock and allocates it, and it is not something you patch by hand, it is something you audit for and triage with a script.

`find_oversold_stock.py`

https://www.allanninal.dev/saleor/concurrent-checkouts-oversell-stock/

https://github.com/allanninal/saleor-fixes/tree/main/concurrent-checkouts-oversell-stock

---

9 **Order confirmation email sent before payment succeeds**

The customer got redirected off to a payment provider, never finished paying, and still landed a cheerful "your order is confirmed" email in their inbox minutes later. Now support is fielding a confused reply about an order that is not actually paid, and possibly never will be. Here is why Saleor sends that email at order creation instead of payment success, and a script that finds every order caught in the gap.

`flag_confirmation_timing.py`

https://www.allanninal.dev/saleor/confirmation-email-sent-before-payment/

https://github.com/allanninal/saleor-fixes/tree/main/confirmation-email-sent-before-payment

---

10 **Digital products not auto fulfilled despite setting enabled**

The shop setting is on. You checked it twice. Every line on the order is a digital download, the order shows as paid, and yet it sits there Unfulfilled like nothing happened. No error, no webhook failure in the logs, just a customer waiting on a download link that was supposed to arrive on its own. Here is why Saleor's automatic digital fulfillment is pickier about how an order became paid than the setting name suggests, and a script that finds every order it quietly skipped.

`flag_unfulfilled_digital_orders.py`

https://www.allanninal.dev/saleor/digital-products-not-auto-fulfilled/

https://github.com/allanninal/saleor-fixes/tree/main/digital-products-not-auto-fulfilled

---

11 **Discount rounding change breaks totals after upgrade**

A store upgrades Saleor, and a few days later finance notices that an order placed just before the upgrade shows a discount that does not match what the same voucher would compute today. Nobody edited a price. Saleor changed how it rounds the last cent of a percentage discount, and every total computed under the old rule is now one cent off from a fresh recalculation. Here is why that happens and a script that finds every order or checkout where it did, without rewriting a single financial record.

`discount_rounding_drift.py`

https://www.allanninal.dev/saleor/discount-rounding-change-breaks-totals-after-upgrade/

https://github.com/allanninal/saleor-fixes/tree/main/discount-rounding-change-breaks-totals-after-upgrade

---

12 **draftOrderComplete drops the applied voucher**

A staff member builds a draft order, applies a voucher code, and the total shows the right discount right there in the admin. Then draftOrderComplete turns it into a real order, and the discount is smaller than it should be, or gone entirely. The voucher was never removed by anyone. Saleor's own price recalculation quietly let it go. Here is why that recalculation path drops the voucher and a script that catches it before finance reconciles the wrong total.

`detect_dropped_voucher.py`

https://www.allanninal.dev/saleor/draft-order-complete-drops-voucher/

https://github.com/allanninal/saleor-fixes/tree/main/draft-order-complete-drops-voucher

---

13 **Draft order taxes reset to zero on completion**

The draft order looks right. Tax shows up on the total, on every line, even on shipping. Then someone calls draftOrderComplete, the order flips to confirmed, and the tax figures that were there a second ago come back zero. Gross and net still hold their value, so nobody notices at a glance, until reconciliation or a customer invoice comes up short. Here is why Saleor drops tax on that transition and a script that catches it before it reaches accounting.

`detect_tax_regression.py`

https://www.allanninal.dev/saleor/draft-order-taxes-reset-on-completion/

https://github.com/allanninal/saleor-fixes/tree/main/draft-order-taxes-reset-on-completion

---

14 **Duplicate address rows created instead of reusing saved address**

A loyal customer ships to the same home address on every order. After a dozen orders, their account address book holds a dozen copies of that one address, each a separate row with its own id, all identical. Nothing is broken, but the address book is a mess and any report that counts addresses is wrong. Here is why Saleor saves a fresh address row per order instead of reusing the one already on the account, and a script that finds every duplicate cluster per customer without deleting anything until you say so.

`find_duplicate_customer_addresses.py`

https://www.allanninal.dev/saleor/duplicate-customer-address-rows/

https://github.com/allanninal/saleor-fixes/tree/main/duplicate-customer-address-rows

---

15 **Duplicate stock checks still allow double selling**

Two customers check out the last unit of the same variant within a second of each other. Both checkouts pass stock validation. Both orders get created. Only one of them should have. Saleor checks available stock again at every checkout step instead of holding one locked reservation for the whole flow, so two non-atomic reads can each look safe on their own while together they oversell the SKU. Here is why it happens and a script that finds the oversold lines after the fact.

`find_oversold.py`

https://www.allanninal.dev/saleor/duplicate-stock-checks-allow-oversell/

https://github.com/allanninal/saleor-fixes/tree/main/duplicate-stock-checks-allow-oversell

---

16 **Editing a confirmed order reverts it to draft status**

Someone fixes a shipping address or tweaks a line on an order that is already confirmed and paid. Nothing looks wrong in the moment. But the order's status silently flips back to DRAFT, and from that second on it is invisible to every normal fulfillment queue, even though the money is already sitting in a Payment or TransactionItem record. Here is why Saleor lets this happen and a script that finds the orders it happened to.

`flag_orphaned_drafts.py`

https://www.allanninal.dev/saleor/editing-order-reverts-to-draft/

https://github.com/allanninal/saleor-fixes/tree/main/editing-order-reverts-to-draft

---

17 **Entire order percentage voucher discount miscalculated**

A shopper adds a product that is already on sale, applies a store-wide percentage voucher on top, and the total should reflect two discounts compounding, one after the other. Instead the discount is too small, or two otherwise identical orders end up with different totals. Nobody edited a price by hand. Saleor's own order-discount pipeline computed the voucher against the wrong base. Here is why that happens and a script that finds every order where it did.

`flag_entire_order_voucher_mismatch.py`

https://www.allanninal.dev/saleor/entire-order-percentage-voucher-miscalculated/

https://github.com/allanninal/saleor-fixes/tree/main/entire-order-percentage-voucher-miscalculated

---

18 **Fulfillment fails when variant stock equals one**

An order comes in for a variant that has exactly one unit sitting in a warehouse. The checkout succeeds, the order is placed, everything looks normal. Then staff try to fulfill it and Saleor throws back INSUFFICIENT_STOCK, on a unit that is sitting right there on the shelf, reserved for this exact order. Here is why Saleor's own allocation math causes that, and a script that flags the boundary case before anyone calls orderFulfill.

`fix_stock_one_fulfillment.py`

https://www.allanninal.dev/saleor/fulfillment-blocked-stock-equals-one/

https://github.com/allanninal/saleor-fixes/tree/main/fulfillment-blocked-stock-equals-one

---

19 **Gift card balance not restored on order cancellation**

A customer paid with a gift card, the order got cancelled for some ordinary reason, stock came back, the order flipped to CANCELED, and everything looked clean. Except the gift card balance never came back. The money the customer had on that card is just gone from Saleor's point of view, and nothing in the dashboard flags it. Here is why Saleor leaves this gap on purpose and a script that finds every affected card and tops it back up safely.

`restore_gift_card_balance.py`

https://www.allanninal.dev/saleor/gift-card-balance-not-restored-on-cancel/

https://github.com/allanninal/saleor-fixes/tree/main/gift-card-balance-not-restored-on-cancel

---

20 **Gift card balance update overwrites initial balance**

A support agent tops up a gift card to refund a customer, or a script "adds" a bit more balance after a promotion, and the mutation looks like it worked. Then someone notices the card's remaining balance no longer matches what the customer had spent. It did not add to the balance. It replaced it, and the card's whole spend history quietly disappeared in the same write. Here is why Saleor's gift card update behaves this way and a script that finds every card it happened to.

`audit_gift_card_balances.py`

https://www.allanninal.dev/saleor/gift-card-balance-update-overwrites-initial/

https://github.com/allanninal/saleor-fixes/tree/main/gift-card-balance-update-overwrites-initial

---

21 **Cannot unfulfill or delete an order paid with a gift card**

A customer wants a return reversed, or a support agent just wants to clean up a test order, and the obvious steps refuse to work. Canceling the fulfillment throws an error. Deleting the line throws a different error. Nothing in the dashboard explains why. The order just sits there, FULFILLED, immovable. Here is why Saleor locks these orders on purpose and a script that finds them and points you at the fix that is actually supported.

`flag_gift_card_blocks.py`

https://www.allanninal.dev/saleor/gift-card-order-blocks-unfulfill-delete/

https://github.com/allanninal/saleor-fixes/tree/main/gift-card-order-blocks-unfulfill-delete

---

22 **Guest checkout order not linked to matching customer account**

A returning shopper checks out as a guest, typing the same email they used to register months ago. The order goes through fine. But it never shows up under their account, their order history looks empty, and support has no easy way to explain why. Here is why Saleor leaves order.user null on purpose in this case, and a script that finds every affected order without ever guessing at who owns it.

`find_unlinked_guest_orders.py`

https://www.allanninal.dev/saleor/guest-order-not-linked-to-customer/

https://github.com/allanninal/saleor-fixes/tree/main/guest-order-not-linked-to-customer

---

23 **Invalid channel slug accepted without error**

You pass channel: "us" to products when the real slug is "us-store", or a channel got renamed and an old script never caught up. Saleor does not complain. It just hands back an empty list, or the pageInfo you expected to see growing stays flat, and nothing in the response tells you the channel argument was wrong. Here is why that gap exists and a script that catches it before your integration trusts an empty result.

`validate_channel_slug.py`

https://www.allanninal.dev/saleor/invalid-channel-slug-accepted-silently/

https://github.com/allanninal/saleor-fixes/tree/main/invalid-channel-slug-accepted-silently

---

24 **Failed concurrent index migration leaves invalid index**

A deploy timed out, a pod got killed, or a lock wait ran long, right in the middle of a migration that was supposed to be safe. Saleor builds its index migrations with Django's concurrent operations precisely so a big table like product_product never locks during an upgrade. But when that concurrent build gets interrupted, Postgres cannot roll it back the way it would a normal index. It leaves a broken index sitting in the catalog, and the very next deploy fails trying to create the same index again. Here is why that happens and a small script that finds the broken index and plans the fix.

`repair_invalid_index.py`

https://www.allanninal.dev/saleor/invalid-index-after-failed-migration/

https://github.com/allanninal/saleor-fixes/tree/main/invalid-index-after-failed-migration

---

25 **Manual order line discount deleted on recalculation**

A staff member applies a manual discount to an order line with orderLineDiscountUpdate, the unit price drops, and everyone moves on. Then something else touches the same draft order, a new line, a shipping address, a voucher, and the price recalculates. The manual discount that was there a minute ago is gone, the line is back to its normal price, and nothing in the response ever said so. Here is why Saleor can silently drop a manual discount during recalculation and a script that catches it before the order ships at the wrong price.

`detect_discount_loss.py`

https://www.allanninal.dev/saleor/manual-line-discount-deleted-on-recalculation/

https://github.com/allanninal/saleor-fixes/tree/main/manual-line-discount-deleted-on-recalculation

---

26 **No available payment gateways despite plugin enabled**

The dashboard shows the payment plugin turned on. The app shows installed and active. And checkout still comes back with no payment gateway to pick, no error, just an empty list. In Saleor, "enabled" is not one switch. It is a per-channel switch for plugins, and a currency-matched webhook response for apps. Here is where that gap actually lives, and a script that finds it channel by channel.

`find_gateway_gaps.py`

https://www.allanninal.dev/saleor/no-available-payment-gateways/

https://github.com/allanninal/saleor-fixes/tree/main/no-available-payment-gateways

---

27 **No shipping methods available for a channel**

A shipping zone exists. It covers the customer's country. The method is sitting right there in the zone. And checkout still comes back with an empty shipping methods list, no error, nothing to click. The zone and the method are only half the story in Saleor. Here is the piece that is usually missing, and a script that checks every channel for it.

`find_missing_shipping_coverage.py`

https://www.allanninal.dev/saleor/no-shipping-methods-for-channel/

https://github.com/allanninal/saleor-fixes/tree/main/no-shipping-methods-for-channel

---

28 **Order can be charged more than its total**

The order total says one number. The finance report says a bigger one. Nobody edited the price, nobody added a line item, and yet the order somehow collected more money than it was ever supposed to. This is not corrupted data. Saleor has a name for this exact state, and it exists because nothing in the system hard-caps what a payment app or a duplicate webhook is allowed to charge. Here is why it happens and a script that finds every order in that state before a refund conversation gets awkward.

`flag_overcharged_orders.py`

https://www.allanninal.dev/saleor/order-charged-more-than-total/

https://github.com/allanninal/saleor-fixes/tree/main/order-charged-more-than-total

---

29 **Order stuck unfulfilled after payment succeeds**

The card was charged, the transaction settled, isPaid reads true. Everything about the money looks correct. But the order still shows UNFULFILLED, sitting in the queue as if nothing happened, and no amount of staring at the payment record explains why. Here is why Saleor keeps those two things apart and a script that finds the orders truly stuck between them.

`flag_stuck_unfulfilled.py`

https://www.allanninal.dev/saleor/order-stuck-unfulfilled-after-payment/

https://github.com/allanninal/saleor-fixes/tree/main/order-stuck-unfulfilled-after-payment

---

30 **Order subtotal uses gross instead of net price**

Finance pulls an order out of Saleor, reads the subtotal, and it does not match what the ERP or the accounting export expects. Nothing errored. No mutation failed. The order's subtotal is a TaxedMoney object that carries a gross amount and a net amount side by side, and somewhere a script, a report, or a migration read the gross figure while the rest of the pipeline was built around net. Here is why Saleor keeps both numbers on purpose and a small script that finds every order where the wrong one was read, without touching Saleor's own correct data.

`audit_subtotal_basis.py`

https://www.allanninal.dev/saleor/order-subtotal-gross-instead-of-net/

https://github.com/allanninal/saleor-fixes/tree/main/order-subtotal-gross-instead-of-net

---

31 **ORDER_UPDATED webhook skipped on metadata change**

Your app subscribes to ORDER_UPDATED and it fires reliably for real order changes, price edits, status moves, address updates. Then someone calls updateMetadata to stamp an order with a CRM id or a fulfillment reference, and nothing arrives at your endpoint. The order was legitimately modified. Saleor just decided that particular change does not count as an order update. Here is why that split exists and a script that finds every app still exposed to it.

`detect_metadata_webhook_gap.py`

https://www.allanninal.dev/saleor/order-updated-webhook-skipped-on-metadata-change/

https://github.com/allanninal/saleor-fixes/tree/main/order-updated-webhook-skipped-on-metadata-change

---

32 **Paid checkout never converts to an order**

The card was charged, the payment provider confirmed it, and by every measure the sale happened. But Saleor still has a Checkout, not an Order, and nothing in the storefront is going to fix that on its own. Here is why Saleor leaves paid checkouts stranded like this and a small script that finds the ones a human should trust and finishes them safely.

`complete_paid_checkouts.py`

https://www.allanninal.dev/saleor/paid-checkout-never-converts-to-order/

https://github.com/allanninal/saleor-fixes/tree/main/paid-checkout-never-converts-to-order

---

33 **Partial fulfillment breaks on a zero quantity line**

You want to ship four of the five lines on an order right now and leave the fifth for later, on purpose. That should be a normal partial fulfillment. Instead the orderFulfill call comes back with an error about a quantity, the whole mutation aborts, and the order is stuck in PARTIALLY_FULFILLED with the remaining line never touched. Here is why a zero quantity line trips this up and a script that finds the stuck orders and reports the exact payload that would actually pass.

`repair_partial_fulfillment.py`

https://www.allanninal.dev/saleor/partial-fulfillment-breaks-zero-quantity-line/

https://github.com/allanninal/saleor-fixes/tree/main/partial-fulfillment-breaks-zero-quantity-line

---

34 **Product published but invisible from a missing channel price**

The dashboard says the product is published to the channel. The API even reports isPublished: true. And yet the storefront never shows it, and checkout will not sell it. In Saleor, publishing a product to a channel and pricing it for that channel are two different steps, and it is easy to finish one without the other. Here is why that gap is silent and a script that finds every product left in it.

`find_mispriced_published_listings.py`

https://www.allanninal.dev/saleor/product-invisible-missing-channel-price/

https://github.com/allanninal/saleor-fixes/tree/main/product-invisible-missing-channel-price

---

35 **Product unavailable from a warehouse zone channel mismatch**

The stock is real. The warehouse has plenty of it. But the storefront shows the product as unavailable, or it just does not show up at all for that channel, and there is no error to point at anything. In Saleor a variant's stock is only reachable from a channel when a small graph of links agrees: warehouse to channel, and sometimes warehouse to zone to channel too. Miss one link and the product goes quiet. Here is why, and a script that finds every stock row a channel cannot actually reach.

`find_orphaned_stock.py`

https://www.allanninal.dev/saleor/product-unavailable-warehouse-zone-channel-mismatch/

https://github.com/allanninal/saleor-fixes/tree/main/product-unavailable-warehouse-zone-channel-mismatch

---

36 **Product without any variant crashes queries**

Someone created a product through the API, saved it, and moved on without ever calling productVariantCreate. The product exists. It has a name, a slug, maybe a description. What it does not have is a single variant, and in Saleor that is where the price, the SKU, the stock, and the channel price all actually live. So defaultVariant comes back null, pricing falls apart, and any storefront or checkout code that assumed a product always has a variant throws or quietly shows nothing. Here is why Saleor lets this happen and a script that finds every product like it before a shopper does.

`flag_variantless_products.py`

https://www.allanninal.dev/saleor/product-without-variant-crashes-queries/

https://github.com/allanninal/saleor-fixes/tree/main/product-without-variant-crashes-queries

---

37 **Queued events become permanently failed while app disabled**

You disable an app, or Saleor's circuit breaker auto-disables it after a run of failed deliveries, expecting things to just pause. Then you re-enable it and new events flow fine, but a chunk of orders and other events from the disabled window are simply gone. Not retried, not queued, gone. Here is why Saleor fails those queued events instead of holding them, and a script that finds and safely retries the ones still worth saving.

`retry_dropped_events.py`

https://www.allanninal.dev/saleor/queued-events-fail-while-app-disabled/

https://github.com/allanninal/saleor-fixes/tree/main/queued-events-fail-while-app-disabled

---

38 **Shipping methods do not refresh after address change**

The customer changes their shipping address. Saleor updates the checkout, the new country is right there in the response. But the shipping method list on screen does not move. It still shows the options for the old country, or an empty list, and the checkout can complete with a delivery method that no longer makes sense for where the order is actually going. Here is why the read goes stale even though the server did its job, and a script that catches it before it reaches an order.

`reconcile_stale_shipping.py`

https://www.allanninal.dev/saleor/shipping-methods-stale-after-address-change/

https://github.com/allanninal/saleor-fixes/tree/main/shipping-methods-stale-after-address-change

---

39 **Stock quantity reads zero despite items in warehouse**

A pallet just came off the truck. Staff can see the boxes. But the storefront says the variant is out of stock, and Saleor agrees, because the number it is reading says zero. Nobody moved the units out, nobody sold them all. The stored quantity and the physical shelf simply drifted apart, and Saleor has no built in alarm for that. Here is why the two numbers split and a script that finds every place they disagree so a human can confirm the real count before anything gets overwritten.

`detect_stock_drift.py`

https://www.allanninal.dev/saleor/stock-quantity-zero-despite-inventory/

https://github.com/allanninal/saleor-fixes/tree/main/stock-quantity-zero-despite-inventory

---

40 **Stock update webhook not triggered on some mutations**

Your app subscribes to PRODUCT_VARIANT_STOCK_UPDATED and it mostly works, until it does not. A staff member runs a manual stock edit and the webhook fires right on cue. Then an order gets fulfilled, Stock.quantity visibly drops in the dashboard, and nothing arrives at your endpoint. The quantity changed. Saleor just never told anyone. Here is why that gap exists and a script that finds every stock change your webhook missed.

`detect_stock_webhook_desync.py`

https://www.allanninal.dev/saleor/stock-update-webhook-not-triggered/

https://github.com/allanninal/saleor-fixes/tree/main/stock-update-webhook-not-triggered

---

41 **Tax calculation off by rounding cents on totals**

Finance pulls an order, multiplies the tax rate by the subtotal, and gets a number that is a cent or two away from what Saleor actually charged. Nothing failed. No mutation errored. Saleor rounded the tax on every order line to the cent, one line at a time, then added up those already-rounded amounts to build the order total. That is not the same arithmetic as taking the rate times the whole subtotal in one shot, and the two can legitimately disagree by a few cents. Here is why that gap exists and a script that audits it the way Saleor itself does the math, instead of flagging every honest cent of rounding as a bug.

`audit_tax_rounding.py`

https://www.allanninal.dev/saleor/tax-calculation-rounding-mismatch/

https://github.com/allanninal/saleor-fixes/tree/main/tax-calculation-rounding-mismatch

---

42 **totalBalance drifts after refund or authorization adjustment**

Finance opens an order and the numbers do not add up. The order shows a partial refund, the amount captured looks right on its own, but totalBalance, the figure everyone reads as "what is still owed or owing," does not match what total, totalCaptured, and totalRefunded would suggest if you did the arithmetic by hand. Nothing errored. No mutation failed. A partial refund or an authorization adjustment simply left the transaction ledger telling two different stories at once, and Saleor reported the derived balance from whichever story it had on hand at read time. Here is why that gap opens up and a script that finds every order where it has, without correcting a single value until a human says so.

`flag_balance_drift.py`

https://www.allanninal.dev/saleor/total-balance-drift-after-refund/

https://github.com/allanninal/saleor-fixes/tree/main/total-balance-drift-after-refund

---

43 **Unpaid orders keep stock allocated indefinitely**

A customer starts checkout, the order line gets created, and Saleor reserves the stock right then. Then the payment never lands. No card charge, no bank transfer, nothing. The order just sits there, and so does the stock it grabbed, because nothing in Saleor ever automatically lets it go unless you told it to. Here is why that allocation outlives the order and a script that finds the stuck ones and safely frees them.

`reconcile_stuck_orders.py`

https://www.allanninal.dev/saleor/unpaid-orders-retain-allocated-stock/

https://github.com/allanninal/saleor-fixes/tree/main/unpaid-orders-retain-allocated-stock

---

44 **Variant cost price miscalculates with multiple stock rows**

A variant sits in two or more warehouses, and someone asks a simple question: what does this thing actually cost us. In old Saleor, that question could crash outright the moment one warehouse never had a cost entered. In current Saleor, the crash is gone, but the same instinct, trying to work out cost from a pile of Stock rows, still produces nonsense: a null cost price sitting next to a real selling price, or a margin so far off it cannot be true. Here is why the original bug happened, why the fix was to move cost off Stock entirely, and a script that audits your variants for the same signature today.

`audit_variant_cost_price.py`

https://www.allanninal.dev/saleor/variant-cost-price-miscalculation/

https://github.com/allanninal/saleor-fixes/tree/main/variant-cost-price-miscalculation

---

45 **Variant has no channel listing after creation**

A new variant just went in, through the dashboard, an import script, or a bulk mutation. The product page loads, the product looks published, and everyone moves on. Then someone tries to buy that exact variant and it is nowhere to be found, or the storefront shows no price at all. The variant exists. It just never got a price on any channel, and Saleor never warns you about it. Here is why that row goes missing and a script that finds every variant left behind.

`fix_missing_channel_listing.py`

https://www.allanninal.dev/saleor/variant-missing-channel-listing/

https://github.com/allanninal/saleor-fixes/tree/main/variant-missing-channel-listing

---

46 **Variant pricing query returns null without a channel argument**

You query productVariants { pricing { price { gross { amount } } } }, the request succeeds, and every single row comes back with pricing: null. So a script built on top of that assumes the whole catalog is unpriced and starts flagging or fixing variants that are, in fact, priced correctly in three channels. Here is why Saleor returns null in this exact shape and a scanner that checks pricing the way Saleor actually resolves it, per channel.

`flag_variant_pricing_gaps.py`

https://www.allanninal.dev/saleor/variant-pricing-null-without-channel-arg/

https://github.com/allanninal/saleor-fixes/tree/main/variant-pricing-null-without-channel-arg

---

47 **Voucher usable past its usage limit under concurrency**

A voucher has a clear usageLimit, and Saleor's own used counter agrees with it right up until launch day. Then two customers check out within the same second, both orders complete, and the voucher that was supposed to cap out at fifty redemptions quietly clears fifty two. Nobody wrote a bad rule. The count and the check that guards it are just not protected from each other. Here is why that race exists and a script that finds every voucher it has already hit and reports exactly which orders are involved.

`detect_voucher_overage.py`

https://www.allanninal.dev/saleor/voucher-usable-past-usage-limit/

https://github.com/allanninal/saleor-fixes/tree/main/voucher-usable-past-usage-limit

---

48 **Voucher usage count double incremented on payment retries**

One order, one voucher code, and yet the usage counter climbed by two. Nobody applied the code twice, the customer only checked out once, but a two-stage payment gateway made the checkout call complete twice behind the scenes, and Saleor counted both. Here is why that double count happens and a script that finds every code where it did.

`reconcile_voucher_usage.py`

https://www.allanninal.dev/saleor/voucher-usage-double-incremented/

https://github.com/allanninal/saleor-fixes/tree/main/voucher-usage-double-incremented

---

49 **Warehouse cannot join a shipping zone without a shared channel**

The warehouse is right there in the shipping zone's warehouse list. It has stock. It should be able to fulfill orders for that zone. But orders in that zone quietly stop pulling from it, and nobody gets an error, because the warehouse and the zone no longer share a single channel. Here is why Saleor ties warehouses to zones through channels, how that link can silently break after the fact, and a script that finds every zone where it already has.

`find_orphaned_warehouse_zone_links.py`

https://www.allanninal.dev/saleor/warehouse-zone-assignment-needs-shared-channel/

https://github.com/allanninal/saleor-fixes/tree/main/warehouse-zone-assignment-needs-shared-channel

---

50 **Webhook deliveries stuck failed past retry limit**

Your endpoint had a bad five minutes. A deploy, a cold start, a database blip, something transient. Saleor tried to deliver the event, failed, backed off, and tried again a few more times. Then it stopped trying. The EventDelivery sits there marked FAILED forever, and nothing in Saleor will ever touch it again on its own. Here is why Saleor gives up for good and a small script that finds the ones worth a second chance.

`retry_stale_failed_deliveries.py`

https://www.allanninal.dev/saleor/webhook-deliveries-stuck-failed/

https://github.com/allanninal/saleor-fixes/tree/main/webhook-deliveries-stuck-failed

---

51 **Webhook payload fields diverge from documented schema**

You built your handler against the sample payload in the Saleor docs, or against the fields your subscription query asked for, and it worked. Then a field you rely on shows up as null, or a whole nested object disappears, or an extra field appears that nothing in the docs mentions. Saleor never raised an error. Nothing failed. The payload just stopped matching what you expected. Here is why that gap exists and a script that diffs a real delivery against what it should contain and tells you exactly what changed.

`detect_webhook_payload_drift.py`

https://www.allanninal.dev/saleor/webhook-payload-diverges-from-schema/

https://github.com/allanninal/saleor-fixes/tree/main/webhook-payload-diverges-from-schema

---