FIELD NOTES

Medusa Field Guide

Medusa v2 storefront, inventory and workflow fixes.

88 fixes, each one a written note with diagrams and a small Python and Node.js script that finds the problem and repairs it. All 88 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/medusa-fixes
Guides: allanninal.dev/medusa
All 14 repos: github.com/allanninal

The fixes

1 **Abandoned carts pile up**
A shopper starts a cart, adds a few items, then never comes back. A bot crawls the storefront and triggers an empty cart on every visit. A flaky network makes a checkout retry and leaves a duplicate cart behind. None of these carts ever convert into an order, and none of them ever go away on their own. Here is why Medusa v2 keeps every one of them forever and a small script that finds the truly stale ones and reports them for review, without deleting anything automatically.
`report_stale_carts.py`
https://www.allanninal.dev/medusa/abandoned-carts-pile-up/
https://github.com/allanninal/medusa-fixes/tree/main/abandoned-carts-pile-up

---

2 **Admin invite fails when the email already belongs to a customer**
You send the invite through POST /admin/invites, the email arrives, and everything looks fine. Then the person you invited clicks the link, fills in a password, hits accept, and gets a 401 that says the identity with that email already exists. The invite is dead on arrival, and it will stay dead no matter how many times they retry, because the email was already used to register as a customer. Here is why Medusa cannot tell those two identities apart and a script that flags the collision before you ever send the invite.
`check_invite_collision.py`
https://www.allanninal.dev/medusa/admin-invite-email-collision/
https://github.com/allanninal/medusa-fixes/tree/main/admin-invite-email-collision

---

3 **Backfill external id for reconciliation**

A finance or ops export from the old system lists orders by their legacy id. In Medusa, that same order has no idea it ever had one. Nobody wrote it down, or the flow that created the order never threaded it through, and now the two systems cannot be joined without a person opening spreadsheets side by side. Here is why Medusa v2 has nowhere built in to keep that mapping, and a script that finds the gap and fills it in only when it is sure.

`backfill_external_id.py`

https://www.allanninal.dev/medusa/backfill-external-id-for-reconciliation/

https://github.com/allanninal/medusa-fixes/tree/main/backfill-external-id-for-reconciliation

---

4 **Reserve inventory step fails even with backorders allowed**

A shopper checks out on a variant you deliberately configured to allow backorders. Checkout should sail through even at zero stock. Instead the cart never completes, and the server log shows Not enough stock available for item, thrown from deep inside the reservation step. The setting says backorders are fine. The workflow disagrees. Here is why that gap opens up and a small script that finds the stuck carts and only retries the ones that are genuinely safe to retry.

`retry_backorder_reservation.py`

https://www.allanninal.dev/medusa/backorder-reservation-step-fails/

https://github.com/allanninal/medusa-fixes/tree/main/backorder-reservation-step-fails

---

5 **Broken Medusa product image links**

A product that had a perfectly good thumbnail last week now shows a broken image icon in the storefront and the Admin. Nothing about the product changed. What changed is the file that URL points to, or the host that used to serve it. Here is why Medusa never notices a stored image URL has gone stale, and a small script that scans every product, tells you exactly which images are dead, and only ever clears or replaces the ones you confirm.

`find_broken_images.py`

https://www.allanninal.dev/medusa/broken-product-image-links/

https://github.com/allanninal/medusa-fixes/tree/main/broken-product-image-links

---

6 **Buy X get Y promotions fail to apply during cart updates**

The buyget promotion is active, the code is on the cart, and the customer really did buy enough of the qualifying product. But the free or discounted item never shows up, and it stays missing through every cart update. Nothing errors. The promotion just quietly produces nothing every time Medusa recomputes the cart. Here is why a buyget promotion's application_method can look complete in the admin and still be structurally unable to ever discount anything, and a script that flags it and rebuilds a safe, working payload.

`fix_buyget_application_method.py`

https://www.allanninal.dev/medusa/buyget-promotion-not-applied-on-update/

https://github.com/allanninal/medusa-fixes/tree/main/buyget-promotion-not-applied-on-update

---

7 **Campaign budget exceeded but still applies**

The campaign dashboard shows used past limit. Support has already flagged three orders that got a discount they should not have. And right now, this minute, more carts are computing that same promotion onto their totals. Medusa is not broken here, it is doing exactly what its docs say it will do. Here is why a campaign budget can be crossed and still keep discounting, and a script that finds every over-budget campaign, the orders that slipped through, and stops the bleed without touching a single completed order.

`flag_over_budget_campaigns.py`

https://www.allanninal.dev/medusa/campaign-budget-exceeded-still-applies/

https://github.com/allanninal/medusa-fixes/tree/main/campaign-budget-exceeded-still-applies

---

8 **Campaign budget usage never increments for Buy X Get Y**

The campaign has a spend cap or a usage cap, it is tied to a Buy X Get Y promotion, and the admin dashboard keeps showing budget.used at 0 no matter how many orders redeem it. Nothing is throwing an error. The counter is just never being told anything happened. Here is why buyget promotions slip past Medusa's own usage accounting, and a script that recomputes real usage from your orders and tells you exactly which campaigns have quietly gone over budget.

`reconcile_campaign_budget.py`

https://www.allanninal.dev/medusa/campaign-budget-usage-not-tracked/

https://github.com/allanninal/medusa-fixes/tree/main/campaign-budget-usage-not-tracked

---

9 **Cannot cancel a fulfillment once stock has gone negative**

Someone clicks Cancel fulfillment in the admin, or a script calls the same route, and it just will not go through. The order sits there, the fulfillment is neither canceled nor usable, and the logs point at an inventory step. Here is why Medusa v2's cancel-fulfillment workflow chokes once the location level behind it has already gone negative, and a script that finds every fulfillment stuck this way so you can fix the stock first and retry the cancel.

`find_blocked_cancels.py`

https://www.allanninal.dev/medusa/cancel-fulfillment-negative-stock/

https://github.com/allanninal/medusa-fixes/tree/main/cancel-fulfillment-negative-stock

---

10 **Cart completion fails, no payment provider**

The cart looks complete. Line items are priced, the shipping option is picked, the region matches the customer's country. Then the last click, the one that turns a cart into an order, fails with something like no payment provider available for this region. Here is why Medusa can let a cart get all the way to the final step and only then discover there is nothing to actually take the payment with, and a small script that finds every region missing a working payment provider before a real customer does.

`find_regions_without_payment.py`

https://www.allanninal.dev/medusa/cart-completion-fails-no-payment-provider/

https://github.com/allanninal/medusa-fixes/tree/main/cart-completion-fails-no-payment-provider

---

11 **Checkout blocked for carts split across stock locations**

A shopper adds two items to the cart, one is stocked in the East warehouse and the other in the West warehouse, and both locations are linked to the sales channel. Total stock is fine. Each item on its own is fine. But checkout fails at the reservation step, with no obvious reason why. Here is why Medusa v2 picks the wrong location for a split cart and a small script that finds the carts stuck on it.

`detect_multi_location_cart.py`

https://www.allanninal.dev/medusa/checkout-blocked-multi-location-cart/

https://github.com/allanninal/medusa-fixes/tree/main/checkout-blocked-multi-location-cart

---

12 **continueOnPermanentFailure skips compensation and leaves partial state**

A step is deliberately flagged so a permanent failure will not stop the whole workflow. That flag also quietly turns off its own rollback. If that step already committed something, an order, a captured payment, a reservation, and it later fails, the workflow keeps going and nothing ever undoes what that step did. If a sibling step then fails too and the saga rolls back, it still will not touch the flagged step, so the record sits there in the database with no matching downstream state ever created. Here is why that happens and a small reconciler that finds the gap and reports it safely.

`reconcile_skipped_compensation.py`

https://www.allanninal.dev/medusa/continue-on-failure-skips-compensation/

https://github.com/allanninal/medusa-fixes/tree/main/continue-on-failure-skips-compensation

---

13 **CSV import ignores the variant inventory quantity column**

You upload a product CSV with a proper Variant Inventory Quantity column, the import finishes with a green checkmark, and every product shows up in the catalog looking exactly right. Then the storefront says everything is out of stock. Nothing errored. Nothing warned you. The quantity column was simply never read. Here is why Medusa v2's import leaves every variant with no usable stock, and a small script that finds every variant this happened to and repairs it from the same CSV.

`repair_import_inventory.py`

https://www.allanninal.dev/medusa/csv-import-ignores-inventory-quantity/

https://github.com/allanninal/medusa-fixes/tree/main/csv-import-ignores-inventory-quantity

---

14 **Product import batch job hangs at preprocessing with no event**

You upload a CSV of products, the admin says it is preprocessing, and then nothing. No completion event, no error, no product.created fires, and the summary you were supposed to review never shows up or nobody notices it. Hours later the import is still preprocessing. Here is why Medusa leaves it there on purpose and a small script that finds the transactions stuck this way and reports them for a human to resume or discard safely.

`flag_stuck_import.py`

https://www.allanninal.dev/medusa/csv-import-stuck-preprocessing/

https://github.com/allanninal/medusa-fixes/tree/main/csv-import-stuck-preprocessing

---

15 **Custom module links do not cascade delete with the linked product**

You wired a custom module to a product with defineLink, the product got deleted, and your custom link table still has a row pointing at that old prod_ id. The relation looks broken because it is. Here is why Medusa v2 treats link cascades as opt-in and only honors them through its own delete paths, and a small script that finds the rows this leaves behind.

`find_dangling_links.py`

https://www.allanninal.dev/medusa/custom-link-no-cascade-delete/

https://github.com/allanninal/medusa-fixes/tree/main/custom-link-no-cascade-delete

---

16 **Custom provider capture skips creating the order transaction**

Your custom payment provider returns captured straight from authorizePayment, to say the charge already went through, for example a cash on delivery or a synchronous gateway that settles instantly. Medusa marks the Payment record as captured. But the order still shows the full amount outstanding, because the workflow that would have recorded the money against the order never ran. Here is why that step gets skipped and a small script that finds the orders stuck like this and repairs them safely.

`find_missing_order_transactions.py`

https://www.allanninal.dev/medusa/custom-provider-capture-skips-transaction/

https://github.com/allanninal/medusa-fixes/tree/main/custom-provider-capture-skips-transaction

---

17 **Custom provider capture leaves outstanding amount desynced**

Your custom payment provider captures the money and everyone agrees it went through, the provider dashboard, the webhook log, even the Payment record. But the order in the Medusa admin still shows a balance due. outstanding_amount refuses to move even though paid_total looks like it should have caught up. Here is why that number stops tracking reality and a small script that finds the orders stuck like this and repairs them safely.

`find_outstanding_desync.py`

https://www.allanninal.dev/medusa/custom-provider-outstanding-desync/

https://github.com/allanninal/medusa-fixes/tree/main/custom-provider-outstanding-desync

---

18 **Draft orders cannot get a payment collection created**

A merchant creates a draft order for a phone quote or a custom wholesale deal, expects to collect payment on it the same way as a normal order, and calls the payment-collection route. Nothing happens, or the call fails outright. The draft order sits there with an empty payment_collections relation and no way to register the payment. It is not a bug in your store. A draft order never has a cart_id, and the payment-collection route everyone reaches for first expects one. Here is why that happens and a script that finds the stuck draft orders so you can route them to the fix that actually works.

`fix_draft_order_payment.py`

https://www.allanninal.dev/medusa/draft-order-no-payment-collection/

https://github.com/allanninal/medusa-fixes/tree/main/draft-order-no-payment-collection

---

19 **Draft orders reject valid promotion codes**

The promo code is real. It is active, it applies cleanly to a normal storefront cart, and there is nothing wrong with it. But the moment a script or an admin call tries to add that same code to a draft order, Medusa throws back "An active Order Change is required to proceed," and the code never attaches. Here is why draft orders check for something a regular cart never needs, and a script that tells apart a missing edit session from a genuinely inactive promotion, then repairs only the safe case.

`fix_draft_order_promo_code.py`

https://www.allanninal.dev/medusa/draft-order-rejects-promotion-code/

https://github.com/allanninal/medusa-fixes/tree/main/draft-order-rejects-promotion-code

---

20 **Draft orders never completed**

A sales rep starts a draft order for a phone quote, then the deal goes cold and the draft is left as it was. Someone spins up a test draft to check a workflow and never cleans it up. A quote gets built, sent, and never confirmed. None of these drafts ever get completed into a real order, and none of them ever go away on their own. Here is why Medusa v2 keeps every unfinished draft forever and a small script that finds the truly stale ones and reports them for cleanup, without deleting anything automatically.

`report_stale_draft_orders.py`

https://www.allanninal.dev/medusa/draft-orders-never-completed/

https://github.com/allanninal/medusa-fixes/tree/main/draft-orders-never-completed

---

21 **A leftover workaround subscriber duplicates order confirmation emails**

A customer emails support asking why they got two order confirmations for the same purchase. Nothing in the order looks wrong, the total is right, the payment captured once. But somewhere in the events pipeline, order.placed is firing, or being acted on, twice, and the notification subscriber is happily sending the confirmation email each time it runs. The usual cause is not a bug in the notification code at all. It is an old workaround subscriber that nobody removed after the bug it was written for got fixed. Here is why that happens and a script that finds the orders it hit.

`find_duplicate_confirmations.py`

https://www.allanninal.dev/medusa/duplicate-emails-from-leftover-workaround/

https://github.com/allanninal/medusa-fixes/tree/main/duplicate-emails-from-leftover-workaround

---

22 **Duplicating a product fails on a unique variant barcode constraint**

You click Duplicate on a product in the Medusa admin because it is the fastest way to spin up a new variant of something you already sell. The clone starts, and then it fails, or worse, it succeeds and you only find out later that two products now claim the same barcode. Every variant field got copied over exactly as it was, including the one field that Postgres will not let two live variants share. Here is why Medusa never clears it and a script that finds every barcode, ean, and upc collision in your catalog before it causes a real problem.

`find_barcode_conflicts.py`

https://www.allanninal.dev/medusa/duplicate-product-barcode-conflict/

https://github.com/allanninal/medusa-fixes/tree/main/duplicate-product-barcode-conflict

---

23 **Duplicate product handles from a Medusa import**

A CSV import ran, maybe twice, maybe with a few blank titles in the sheet, and now two or three products in your store share the exact same handle. Nothing errored. The import finished green. But /store/products/{handle} can only ever resolve to one of them, so whichever product the storefront happens to pick first quietly wins the canonical URL, and the others become orphaned duplicates sitting in the catalog. Here is why Medusa lets this happen and a small reconciler that finds every collision before you decide what to do about it.

`find_duplicate_handles.py`

https://www.allanninal.dev/medusa/duplicate-product-handles-from-import/

https://github.com/allanninal/medusa-fixes/tree/main/duplicate-product-handles-from-import

---

24 **Duplicating a product scrambles variant option pairings**

You duplicate a product to save time on a similar listing. The copy comes back with the right number of variants and the right SKUs, so at a glance everything looks fine. Then a customer orders a Small in Red and gets a Large in Blue, because somewhere in the copy, a variant got wired to the wrong option values. Here is why Medusa's duplication logic can mismatch a variant's title and value pairing, and a script that compares a source product against its duplicate and tells you exactly which variants are wrong.

`diff_variant_options.py`

https://www.allanninal.dev/medusa/duplicate-product-scrambles-variants/

https://github.com/allanninal/medusa-fixes/tree/main/duplicate-product-scrambles-variants

---

25 **Duplicate promotion codes**

A customer types SAVE10 at checkout and gets a discount, but it is not the discount marketing set up for this campaign. It is an old one nobody deactivated, or a copy that slipped in from a seed script months ago. Both promotions are active, both match the code the customer typed, and Medusa never complained because, to Postgres, the two codes were never quite the same string. Here is why the database's own uniqueness check can miss a live duplicate, and a script that finds every colliding pair without touching a single one of them for you.

`find_duplicate_promotion_codes.py`

https://www.allanninal.dev/medusa/duplicate-promotion-codes/

https://github.com/allanninal/medusa-fixes/tree/main/duplicate-promotion-codes

---

26 **Events lost without the Redis event bus**

An order comes in, the payment captures, and by every log you can find the checkout worked. But the customer never got a confirmation email. Inventory never synced. Fulfillment never heard about it. Nothing crashed and nothing logged an error, because in Medusa v2 the default Event Module lives entirely in one process's memory, and a process that never saw the event has no way to complain that it missed it. Here is why that happens and a script that finds the orders it happened to.

`find_missing_notifications.py`

https://www.allanninal.dev/medusa/events-lost-without-redis-event-bus/

https://github.com/allanninal/medusa-fixes/tree/main/events-lost-without-redis-event-bus

---

27 **Events fire before subscribers finish loading on boot**

A redeploy or a horizontal-scale restart brings a Medusa instance back up, and for a brief window right after boot, an event that was already sitting in Redis gets processed with zero subscribers attached. No error, no retry, no trace. The webhook that should have fired never fires, the notification that should have gone out never goes out, and BullMQ has already marked the job complete. Here is why the Redis event bus can race ahead of your own subscribers, and a script that finds the gap in your boot logs and reports it safely.

`find_missed_events.py`

https://www.allanninal.dev/medusa/events-race-subscriber-boot/

https://github.com/allanninal/medusa-fixes/tree/main/events-race-subscriber-boot

---

28 **Price list keeps serving prices past its end date**

The sale ended last week. The admin even shows the price list as Expired. But a shopper checks out today and the cart still charges the sale price, not the real one. Nothing crashed and nothing logged an error, the expired list just kept getting picked. Here is why Medusa's pricing module can keep serving an expired price list and a script that finds every affected product and reports, or carefully corrects, the exact fix.

`flag_expired_price_lists.py`

https://www.allanninal.dev/medusa/expired-price-list-still-served/

https://github.com/allanninal/medusa-fixes/tree/main/expired-price-list-still-served

---

29 **Fulfillment created event never fires for untracked items**

The fulfillment shows up on the order. Tracking looks right. Everything in the admin says it shipped. But the customer never got a shipment email, and nothing in your logs says why. When every item on that fulfillment is untracked inventory, Medusa v2 can create the fulfillment and quietly skip the event that was supposed to tell your notification subscriber about it. Here is why it happens and a script that finds every fulfillment it happened to.

`find_missed_fulfillment_events.py`

https://www.allanninal.dev/medusa/fulfillment-event-skipped-untracked-items/

https://github.com/allanninal/medusa-fixes/tree/main/fulfillment-event-skipped-untracked-items

---

30 **Fulfillment status stays Delivered after a full return and refund**

Every item on the order came back. The warehouse checked it in, support issued the refund, the customer got their money. By every measure the order is closed out. But open it in the Medusa admin and the fulfillment status still reads Delivered, as if none of that happened. Reports built off fulfillment status count it as a completed sale that shipped clean, and nobody can tell just by glancing at the order that the whole thing was reversed. Here is why Medusa v2 leaves fulfillment status frozen like this and a small script that finds the fully returned orders and fixes the picture safely.

`flag_stuck_fulfillment.py`

https://www.allanninal.dev/medusa/fulfillment-status-stuck-delivered/

https://github.com/allanninal/medusa-fixes/tree/main/fulfillment-status-stuck-delivered

---

31 **Fulfillment without a tracking number**

Someone clicked Mark as Shipped in the admin. The order looks done. But the fulfillment carries no tracking number at all, so nobody, not support, not the customer, can trace the package once it leaves the building. Here is why Medusa v2 lets a fulfillment become shipped with an empty labels array, and a script that finds every one of these so a human can chase down the real tracking number with the carrier.

`flag_untracked_shipments.py`

https://www.allanninal.dev/medusa/fulfillment-without-a-tracking-number/

https://github.com/allanninal/medusa-fixes/tree/main/fulfillment-without-a-tracking-number

---

32 **Guest checkout creates a duplicate customer on registration**

A shopper checks out as a guest, gets their order, and later comes back to make an account with the same email so they can track things properly. Instead of picking up where the guest checkout left off, Medusa quietly creates a second customer. Their old order is still there in the database, but it is tied to a customer row the new account cannot see. To the shopper it looks like their order history vanished. Here is why Medusa v2 leaves these rows split apart and a small script that finds every pair so a human can decide what to do about them.

`detect_duplicate_customers.py`

https://www.allanninal.dev/medusa/guest-registration-duplicate-customer/

https://github.com/allanninal/medusa-fixes/tree/main/guest-registration-duplicate-customer

---

33 **Inventory not managed, item never sells out**

Real stock hit zero days ago, but the storefront still lets people add the item to their cart and check out. Nothing in the Medusa logs looks broken. The variant simply has its inventory flag turned off, so Medusa never checks how much is actually left. Here is why that flag ends up off without anyone choosing it, and a small script that finds every variant at risk before it oversells again.

`classify_inventory_risk.py`

https://www.allanninal.dev/medusa/inventory-not-managed-never-sells-out/

https://github.com/allanninal/medusa-fixes/tree/main/inventory-not-managed-never-sells-out

---

34 **Medusa inventory decremented at the wrong stock location**

An order comes in through your marketplace channel, and the confirmation email looks completely normal. But the warehouse tied to your main storefront just lost a unit it never shipped, while the warehouse that actually owns that channel never noticed the sale. Nothing errors. Nothing logs a warning. The only sign is a stock count that slowly drifts away from what is actually on the shelf. Here is why a reservation can land at a location that has nothing to do with the channel the order came from, and a small script that finds every order where that happened.

`find_wrong_stock_location.py`

https://www.allanninal.dev/medusa/inventory-wrong-stock-location/

https://github.com/allanninal/medusa-fixes/tree/main/inventory-wrong-stock-location

---

35 **Renaming a linked module or model orphans existing link rows**

You renamed a custom module, say blog to article, or renamed a linked data model, and ran a migration like you always do. Now the products that used to carry a linked blog post do not show one anymore, and somewhere in Postgres a table full of real rows is just sitting there, unreachable. Here is why Medusa's Module Links system does this on a rename, and a small script that finds exactly which link tables were left behind and reports what to do about them.

`classify_link_rename.py`

https://www.allanninal.dev/medusa/link-table-orphaned-on-rename/

https://github.com/allanninal/medusa-fixes/tree/main/link-table-orphaned-on-rename

---

36 **Linked data missing after a Medusa link migration**

The code looks right. A brand module exists, a product-brand link is defined with defineLink(), and both sides independently hold real rows. But every product you fetch with the brand relation expanded comes back with brand set to null, as if the link were never made. Here is why that happens and a small script that tells you, from the outside, whether the link's own database table was ever built.

`detect_unmigrated_link.py`

https://www.allanninal.dev/medusa/linked-data-missing-after-link-migration/

https://github.com/allanninal/medusa-fixes/tree/main/linked-data-missing-after-link-migration

---

37 **Publishable key with multiple channels reports zero stock**

The admin shows real stock sitting at real stock locations. The storefront, using a publishable key scoped to more than one sales channel, insists the variant has zero available. No error, no exception, just a quantity of 0 where a number should be. In Medusa v2 this shows up when the code that resolves a key's stock locations only knows how to handle a single linked sales channel, and silently narrows or drops the filter the moment a key maps to two or more. Here is why that narrowing happens and a small diagnostic that proves it, product by product, so you know exactly which keys and variants are affected before you touch anything.

`diagnose_multi_channel_zero_stock.py`

https://www.allanninal.dev/medusa/multi-channel-key-zero-stock/

https://github.com/allanninal/medusa-fixes/tree/main/multi-channel-key-zero-stock

---

38 **Multiple customer group membership breaks price list resolution**

A customer gets added to a second customer group, maybe a loyalty tier on top of a wholesale tier, and their price list discount just disappears. No error, no warning. The cart quietly charges the base price to a customer who is supposed to be getting an override. Here is why Medusa's pricing module fails to match a price list once a customer belongs to two or more groups, and a script that finds every customer this is happening to.

`detect_multi_group_price_mismatch.py`

https://www.allanninal.dev/medusa/multi-group-price-resolution-fails/

https://github.com/allanninal/medusa-fixes/tree/main/multi-group-price-resolution-fails

---

39 **Multi-part product reservations go negative after fulfillment**

A bundle sells, gets fulfilled, and everything looks routine. Then someone opens the inventory item behind it and reserved_quantity on the location level reads a negative number. No error, no failed webhook, nothing in the logs. It only shows up when a bundle is modeled as one inventory item with a required_quantity above one, and the workflow that reserves it and the workflow that fulfills it do not agree on the multiplier. Here is why that mismatch happens and a script that finds every drifted row and resyncs it safely.

`resync_negative_reserved.py`

https://www.allanninal.dev/medusa/negative-reserved-quantity-bundles/

https://github.com/allanninal/medusa-fixes/tree/main/negative-reserved-quantity-bundles

---

40 **New payment collection ignores amounts already captured**

A customer already paid for their order. Then a price edit bumps the total, Medusa opens a new payment collection to cover the difference, and that new collection is sized for the entire new total, not the small amount actually left owing. The customer either gets asked to pay twice or a staff member sees a scary number and assumes nothing was ever collected. Here is why the order-edit path builds the new collection off the wrong number, and a script that finds every order where this happened and can reconcile it safely.

`reconcile_new_collection.py`

https://www.allanninal.dev/medusa/new-collection-ignores-prior-capture/

https://github.com/allanninal/medusa-fixes/tree/main/new-collection-ignores-prior-capture

---

41 **No API to hard delete link table rows after entity removal**

You deleted a product, a sales channel, or a variant, and the link table row that connected it to something else is still sitting there. You look for a way to purge it for good and there is not one. Medusa's own link.dismiss only soft-deletes the row, and link.delete only cascades when the link was configured to. Here is why Medusa keeps it that way on purpose, and a small script that finds every row this leaves behind and clears the confirmed ones safely.

`classify_link_rows.py`

https://www.allanninal.dev/medusa/no-hard-delete-for-link-rows/

https://github.com/allanninal/medusa-fixes/tree/main/no-hard-delete-for-link-rows

---

42 **No shipping option for the cart**

The region is set up. The product has a price in that currency. The customer picks a country the storefront clearly supports. Then the shipping step comes up blank, no carrier, no rate, nothing to select, and the checkout button has nowhere to go. Here is why Medusa can leave a cart with zero shipping options even when everything upstream looks fine, and a small script that finds exactly which countries are missing coverage.

`find_uncovered_regions.py`

https://www.allanninal.dev/medusa/no-shipping-option-for-the-cart/

https://github.com/allanninal/medusa-fixes/tree/main/no-shipping-option-for-the-cart

---

43 **Editing an order cancels its payment collection and blocks capture**

A support agent adds a line, bumps a quantity, or corrects a unit price on an order that already has money attached to it. The edit saves fine. But now the order's payment collection shows canceled, payment_status has dropped back to not_paid, and every attempt to capture a payment on that order fails with a plain "has been canceled" error. The order is not canceled. The customer still owes money. Here is why the order edit workflow leaves orders in this state and a script that finds them safely, without guessing at a fix.

`flag_edit_cancels_payment.py`

https://www.allanninal.dev/medusa/order-edit-cancels-payment-collection/

https://github.com/allanninal/medusa-fixes/tree/main/order-edit-cancels-payment-collection

---

44 **Order edit after capture computes the refund direction backwards**

A support agent swaps a line item for a cheaper variant on an order that already has a captured payment. The edit confirms fine, the new total is lower, and the customer is owed money back. But the UI, or a cached balance field, reports a positive amount to collect instead of a negative amount to refund. Nobody asked to collect more from a customer who is due a refund. Here is why the sign flips after capture and a read only audit script that catches the mismatch before anyone acts on it.

`flag_wrong_balance_direction.py`

https://www.allanninal.dev/medusa/order-edit-wrong-balance-direction/

https://github.com/allanninal/medusa-fixes/tree/main/order-edit-wrong-balance-direction

---

45 **Order stuck not fulfilled past SLA**

The payment captured days ago. The customer is waiting. But the order's fulfillment_status still reads not_fulfilled, and nobody noticed because nothing in Medusa raised a flag on its own. Here is why placing an order and fulfilling it are two separate things in Medusa v2, how the automation that is supposed to bridge them can quietly fail, and a script that finds every paid order sitting past your SLA so a human can catch it before the customer complains.

`flag_sla_breached_orders.py`

https://www.allanninal.dev/medusa/order-stuck-not-fulfilled-past-sla/

https://github.com/allanninal/medusa-fixes/tree/main/order-stuck-not-fulfilled-past-sla

---

46 **Order summary drops tax from totals, triggering bogus refunds**

A customer pays the full, correct total on a tax-inclusive order. Nothing is owed, nothing is overpaid. But the order's summary in Medusa reports the exact opposite: it thinks the customer overpaid by the tax amount, because the summary's own math never added the tax in. If anything in your stack, an automation, a reconciliation job, an order-edit workflow that "refunds the difference", is wired to that summary number, it will issue a refund for money the customer never actually overpaid. Here is why the summary drops tax and a script that catches it before it costs you real money.

`detect_tax_dropped_from_summary.py`

https://www.allanninal.dev/medusa/order-summary-tax-excluded-bogus-refund/

https://github.com/allanninal/medusa-fixes/tree/main/order-summary-tax-excluded-bogus-refund

---

47 **Orders stay linked to the guest record after registration**

A shopper checks out as a guest, then comes back later and registers with the exact same email so they can track their order properly. They expect to see it waiting for them. Instead their account is empty. The order is still there in the database, still paid, still real, but it is foreign-keyed to the old guest customer row, and the new registered account has no way to see it. Here is why Medusa v2 leaves it that way on purpose and a small script that finds every stuck order so you can drive Medusa's own consent-based transfer flow instead of guessing.

`reconcile_guest_orders.py`

https://www.allanninal.dev/medusa/orders-stuck-on-guest-customer/

https://github.com/allanninal/medusa-fixes/tree/main/orders-stuck-on-guest-customer

---

48 **Failed invite acceptance leaves an orphaned auth identity**

An invitee opens the link, sets a password, clicks accept, and something goes wrong on the last step: a duplicate email, a role conflict, a database hiccup. Medusa rolls the invite back to pending like nothing happened. But the invitee tries again and gets a flat 401, identity with that email already exists. Nothing about the invite looks different. The account that was half created during the first attempt is still sitting there, and it is now the only thing standing between the invitee and a working login. Here is why Medusa leaves that row behind and a script that finds and repairs it without touching anyone's real account.

`repair_orphaned_identity.py`

https://www.allanninal.dev/medusa/orphaned-auth-identity-blocks-reinvite/

https://github.com/allanninal/medusa-fixes/tree/main/orphaned-auth-identity-blocks-reinvite

---

49 **Orphaned records after a link delete**

Someone deleted a product, a sales channel, or an inventory item straight through its own module service, and the admin still shows a link pointing at it somewhere. The far side is gone, but the link row is still there, and nothing tells you until a query tries to expand a relation that no longer resolves. Here is why Medusa v2 lets this happen and a small script that finds the confirmed orphans and clears only those.

`find_orphaned_link_rows.py`

https://www.allanninal.dev/medusa/orphaned-records-after-a-link-delete/

https://github.com/allanninal/medusa-fixes/tree/main/orphaned-records-after-a-link-delete

---

50 **Outstanding amount does not update after the first refund**

A customer gets a partial refund, and the order's outstanding amount drops exactly the way it should. A second refund goes out through the same payment provider, Stripe shows the money leaving your account, but the order's outstanding amount in the Medusa admin does not move at all. It just sits there reporting the balance from after refund number one, as if nothing happened since. Here is why the order's summary stops recomputing after the first refund, and a script that finds every order where the reported number and the real one have drifted apart.

`detect_stale_outstanding.py`

https://www.allanninal.dev/medusa/outstanding-amount-stale-after-refund/

https://github.com/allanninal/medusa-fixes/tree/main/outstanding-amount-stale-after-refund

---

51 **Oversold variant goes negative**

A support ticket says a customer got an order confirmation for something that was not actually in the warehouse. You open the inventory item in the Medusa admin and the available quantity reads a negative number. Nothing crashed. No error was logged. The stock just went underwater. Here is why Medusa lets reserved quantity outrun stocked quantity under concurrent checkout traffic, and a script that finds every oversold location level and helps you repair it safely.

`repair_oversold_inventory.py`

https://www.allanninal.dev/medusa/oversold-variant-goes-negative/

https://github.com/allanninal/medusa-fixes/tree/main/oversold-variant-goes-negative

---

52 **Payment captured but order not paid**

Stripe confirms the charge. Your custom provider confirms the charge. The money is gone from the customer's card and it is sitting in your account. But the Medusa order still shows not_paid, the summary still says nothing was paid, and anything downstream that gates on payment status just sits there waiting. Here is why a successful capture does not always mean the order agrees, and a script that finds the mismatch and repairs it the safe way.

`reconcile_payment_status.py`

https://www.allanninal.dev/medusa/payment-captured-but-order-not-paid/

https://github.com/allanninal/medusa-fixes/tree/main/payment-captured-but-order-not-paid

---

53 **Medusa price list not applied at checkout**

You set up a sale price list in the Medusa admin, gave it a start date, and moved on. Now a shopper opens the cart and the price is still the default one, no discount in sight. Nothing crashed, nothing logged an error, the list just was not used. Here is why Medusa's pricing module silently skips a price list and a script that audits every list against the storefront context it is supposed to serve.

`audit_price_lists.py`

https://www.allanninal.dev/medusa/price-list-not-applied-at-checkout/

https://github.com/allanninal/medusa-fixes/tree/main/price-list-not-applied-at-checkout

---

54 **A price list suppresses all default variant prices once active**

You built a price list for one customer group, or one short campaign window. The moment it goes active, shoppers who were never supposed to see it start getting the price list's number instead of the regular price, and it does not matter whether the default price was actually lower. Here is why Medusa's pricing engine stops looking at the default price the instant any price list matches, and a small script that finds every variant this is quietly happening to.

`find_suppressed_default_price.py`

https://www.allanninal.dev/medusa/price-list-suppresses-default-price/

https://github.com/allanninal/medusa-fixes/tree/main/price-list-suppresses-default-price

---

55 **Product has no price in a region**

A shopper switches the store to a new region and the product page acts like the item does not exist, or the cart throws a does not have a price error at checkout. The variant is fine everywhere else. It just never got a price in that region's currency. Here is why Medusa lets that gap open up and a small script that finds every variant missing a price before a customer runs into it.

`find_missing_region_prices.py`

https://www.allanninal.dev/medusa/product-has-no-price-in-a-region/

https://github.com/allanninal/medusa-fixes/tree/main/product-has-no-price-in-a-region

---

56 **Promotion ignores its sales channel condition and applies anyway**

The promotion has a rule that says it only belongs to one sales channel. The dashboard shows it. The rule is saved. And it still shows up as a discount on carts from a completely different channel. Nobody edited the rule and nobody removed the condition. Somewhere between the rule and the cart, the channel check just never ran. Here is why that happens and a script that finds every order where it actually did.

`find_channel_leaks.py`

https://www.allanninal.dev/medusa/promotion-ignores-channel-condition/

https://github.com/allanninal/medusa-fixes/tree/main/promotion-ignores-channel-condition

---

57 **Medusa promotion not applying**

The promotion is active, the code is correct, the campaign has budget left, and the cart still shows full price. No error, no warning, nothing in the logs. Somewhere in the promotion's rules or its target rules, one condition does not match this cart, and Medusa treats that as a normal, silent no-op. Here is why a single rule mismatch can hide a promotion in plain sight, and a script that diffs every rule against a real cart and points at the exact one that is wrong.

`audit_promotion_rules.py`

https://www.allanninal.dev/medusa/promotion-not-applying-rules-mismatch/

https://github.com/allanninal/medusa-fixes/tree/main/promotion-not-applying-rules-mismatch

---

58 **Medusa publishable key not linked to a sales channel**

The key looks fine. It has the right prefix, it is not revoked, and it is sitting in the header of every storefront request. But /store/products keeps coming back empty, while the same catalog shows up just fine from the Admin API. Nine times out of ten, the key was never linked to a sales channel, or the channel it was linked to got disabled later. Here is why that link matters so much and a small script that finds keys stuck in that state.

`link_publishable_key.py`

https://www.allanninal.dev/medusa/publishable-key-not-linked-to-a-sales-channel/

https://github.com/allanninal/medusa-fixes/tree/main/publishable-key-not-linked-to-a-sales-channel

---

59 **Redis event bus intermittently drops or delays subscriber execution**

A customer completes checkout, the order shows up fine in the Admin, and then no confirmation email ever goes out. Nothing errored. Nothing retried loudly. In Medusa v2, the default Redis Event Bus Module queues every emitted event, such as order.placed from the complete cart workflow, as a BullMQ job for a worker to consume, and if the worker that owns the subscriber starts late, restarts, autoscales, or crashes mid-job, that job can be picked up with nothing listening, retried past its limit, or dropped outright. Here is why that timing gap exists and a script that finds exactly which orders it happened to.

`reconcile_event_delivery.py`

https://www.allanninal.dev/medusa/redis-event-bus-drops-events/

https://github.com/allanninal/medusa-fixes/tree/main/redis-event-bus-drops-events

---

60 **Refund rejected on a captured order showing zero outstanding**

The card was charged, the capture went through, and the money is sitting in your account. But when support tries to refund the customer, Medusa throws back "Order does not have an outstanding balance to refund" as if nothing was ever paid. The payment is real and it is refundable. The order's own summary just does not agree. Here is why that gate can be wrong, and a small script that decides refunds from the payment ledger instead.

`refund_from_ledger.py`

https://www.allanninal.dev/medusa/refund-blocked-zero-outstanding/

https://github.com/allanninal/medusa-fixes/tree/main/refund-blocked-zero-outstanding

---

61 **Refund not reflected on the order**

Someone refunds a customer, the money moves, the payment provider confirms it. But the order in Medusa still shows the old paid total, the old outstanding balance, as if nothing happened. Support reopens the ticket, finance cannot reconcile the books, and nobody can tell just by looking at the order that the refund ever occurred. Here is why Medusa v2 lets orders and payments drift apart like this, and a small script that finds the gap and closes it safely.

`reconcile_refunds.py`

https://www.allanninal.dev/medusa/refund-not-reflected-on-the-order/

https://github.com/allanninal/medusa-fixes/tree/main/refund-not-reflected-on-the-order

---

62 **Region scoped price ignored in favor of currency only price**

You set up a variant with a cheaper (or pricier) price for one region, and a plain currency only price as the fallback for everyone else. The storefront in that region should show the region price. Instead it shows the plain currency price, as if the region rule was never there, even though the correct region_id and currency_code were passed in. Here is why Medusa's price selection resolver picks the wrong row and a script that finds every variant and region pair where this is happening.

`find_ignored_region_price.py`

https://www.allanninal.dev/medusa/region-price-ignored/

https://github.com/allanninal/medusa-fixes/tree/main/region-price-ignored

---

63 **Existing reservation blocks fulfillment of its own order**

A customer buys the last unit of a variant. The order is real, paid, and sitting there waiting to ship. But when staff try to fulfill it, Medusa says there is no available stock, so the fulfillment button is a dead end. The stock is not actually missing. It is held by the order's own reservation, and Medusa's availability check never carves that reservation back out before deciding whether to let the fulfillment through. Here is why that deadlock happens and a small script that finds only the orphaned reservations causing it and clears them safely.

`clear_blocking_reservations.py`

https://www.allanninal.dev/medusa/reservation-blocks-own-order-fulfillment/

https://github.com/allanninal/medusa-fixes/tree/main/reservation-blocks-own-order-fulfillment

---

64 **Reservation records have no order id to trace back to**

A customer emails asking where their order is. You pull up the inventory item, see a reservation holding the stock, and go looking for the order number on it. There isn't one. The reservation just sits there with a line_item_id and no way to click through to the order that made it. Here is why Medusa v2 built it this way and a small script that resolves the trace for you and reports the reservations it cannot.

`trace_reservation_orders.py`

https://www.allanninal.dev/medusa/reservation-missing-order-id/

https://github.com/allanninal/medusa-fixes/tree/main/reservation-missing-order-id

---

65 **Reservation update event never fires**

You change a reservation's quantity, move it to a different line item, or shift it to another location, and your subscriber never wakes up. No error, no log line, nothing. Here is why Medusa v2 silently fired the wrong event on every reservation update and a small reconciler that finds every reservation that drifted out of sync because that event never arrived.

`reconcile_reservation_sync.py`

https://www.allanninal.dev/medusa/reservation-updated-event-not-firing/

https://github.com/allanninal/medusa-fixes/tree/main/reservation-updated-event-not-firing

---

66 **Sub-cent rounding mislabels a paid order as partially captured**

The customer paid in full. Stripe or your provider settled the charge to the cent, the money is in your account, and by any normal measure the order is done. But Medusa still shows payment_status as partially_captured, and anything downstream that gates on a fully captured order just sits there. Here is why a sub-cent remainder trips Medusa's own rounding tolerance and a small script that clears the false positive the same way Medusa itself would.

`clear_rounding_mislabel.py`

https://www.allanninal.dev/medusa/rounding-partial-capture-mislabel/

https://github.com/allanninal/medusa-fixes/tree/main/rounding-partial-capture-mislabel

---

67 **Medusa sales channel not linked to a stock location**

Someone created a new sales channel, maybe for a marketplace app, a headless storefront, or a seed script, and every product through it is suddenly unpurchasable. The inventory items look fine. The location levels look fine. But carts fail, backorders error out, and the message that eventually surfaces reads something like "Sales channel xxx is not associated with any stock location." Here is why that link matters so much and a small script that finds channels stuck in that state.

`link_stock_location.py`

https://www.allanninal.dev/medusa/sales-channel-not-linked-to-a-stock-location/

https://github.com/allanninal/medusa-fixes/tree/main/sales-channel-not-linked-to-a-stock-location

---

68 **Scheduled job did not run**

You wrote a file in src/jobs, exported a function and a schedule, deployed it, and waited. The price list never expired on time. The campaign never got swept. The nightly sync never happened. Nothing crashed, nothing logged an error, and there is no dashboard anywhere that says the job was even supposed to run. Here is why Medusa v2 lets a scheduled job vanish without a trace, and a script that finds the gap in your own data and repairs it safely.

`find_missed_runs.py`

https://www.allanninal.dev/medusa/scheduled-job-did-not-run/

https://github.com/allanninal/medusa-fixes/tree/main/scheduled-job-did-not-run

---

69 **Scheduled jobs execute twice per configured interval**

You wrote one job in src/jobs, gave it one cron schedule, and every tick it runs twice. Sometimes three or four times. No code path in the job itself loops or retries, and the schedule string is correct. The cause is not in the job at all. It is in how many processes are alive and doing background work at the same time. Here is why Medusa v2 lets every one of those processes fire the same tick on its own, and a script that proves it from the Workflow Engine Module's own execution history.

`find_duplicate_ticks.py`

https://www.allanninal.dev/medusa/scheduled-job-runs-twice/

https://github.com/allanninal/medusa-fixes/tree/main/scheduled-job-runs-twice

---

70 **Scheduled jobs run sporadically on Redis backed deploys**

The job in src/jobs works every time on your laptop. In production, split across a server tier and a worker tier the way Railway and similar hosts expect, it runs most ticks fine and then, every so often, just fails. Redis ends up holding a failed BullMQ job with a stacktrace that says Workflow with id "<job-id>" not found, and nobody touched the code. Here is why the Redis backed workflow engine lets this happen, and a script that reads the queue directly and tells you which jobs are actually broken.

`classify_stuck_jobs.py`

https://www.allanninal.dev/medusa/scheduled-jobs-sporadic-redis-deploy/

https://github.com/allanninal/medusa-fixes/tree/main/scheduled-jobs-sporadic-redis-deploy

---

71 **Scheduled jobs stop firing after long uptime**

Everything ran fine for days. Then, sometime after a long stretch of uptime, every scheduled job in your Medusa store just stops. No crash in the logs, no error anywhere, the process is still up and answering requests. It just quietly stopped ticking. Here is why one stuck workflow step can silently freeze the entire scheduler, and a small monitor that catches it early instead of finding out from a customer.

`check_scheduler_heartbeat.py`

https://www.allanninal.dev/medusa/scheduled-jobs-stop-after-uptime/

https://github.com/allanninal/medusa-fixes/tree/main/scheduled-jobs-stop-after-uptime

---

72 **Only one refund per order or payment ever succeeds**

A customer needs two partial refunds on the same order. The first one works fine. The second one, on the same order, sometimes even on the same payment, comes back with "Order does not have an outstanding balance to refund." The payment provider still shows money that could be returned. The customer is still owed it. But Medusa refuses to let anyone touch that payment again. Here is why the refund workflow checks the wrong number, and a small script that finds every order this happened to and tells you exactly how much is still owed.

`refund_shortfall.py`

https://www.allanninal.dev/medusa/second-refund-fails/

https://github.com/allanninal/medusa-fixes/tree/main/second-refund-fails

---

73 **Shipping discount uses a stale shipping amount after cart changes**

A customer has a percentage-off shipping promotion applied, then adds another item or changes a quantity. The shipping price recalculates the way it should. But the discount sitting next to it does not move with it, it is still computed off the shipping price from before the change. The cart total ends up wrong by exactly the difference, and nothing in the UI tells anyone why. Here is why Medusa's own recompute steps can disagree with each other, and a script that finds the carts where this happened and safely re-triggers the fix.

`reconcile_shipping_discount.py`

https://www.allanninal.dev/medusa/shipping-discount-stale-amount/

https://github.com/allanninal/medusa-fixes/tree/main/shipping-discount-stale-amount

---

74 **Stale cart prices after a price change**

You update a variant's price or push a new price list, check the storefront on a fresh page load, and everything looks right. But a shopper who added the item to their cart before the change is still looking at the old number, and it will ride all the way to checkout unless something touches that cart again. Here is why Medusa never goes back to fix an open cart on its own, and a script that finds the carts still holding stale prices and repairs them safely.

`reconcile_stale_cart_prices.py`

https://www.allanninal.dev/medusa/stale-cart-prices-after-a-price-change/

https://github.com/allanninal/medusa-fixes/tree/main/stale-cart-prices-after-a-price-change

---

75 **Fulfillment sometimes leaves a stale inventory reservation**

The order says fulfilled, or even cancelled, but the inventory level does not agree. A handful of units at one location are still shown as reserved, and nobody purchased them anymore. Here is why Medusa v2 sometimes fails to clean up a reservation after a line item is fulfilled and a small reconciler that finds only the stale rows and deletes them safely.

`find_stale_reservations.py`

https://www.allanninal.dev/medusa/stale-reservation-after-fulfillment/

https://github.com/allanninal/medusa-fixes/tree/main/stale-reservation-after-fulfillment

---

76 **Medusa store requests blocked by CORS**

The storefront calls the Medusa Store API and the browser console shows a CORS error, or a blank network response with no data. The backend looks fine, the publishable key looks fine, but the request never completes. Here is why Medusa rejects the origin, how to prove it is really CORS and not a masquerading 401, and a small script that probes every candidate origin and tells you the exact line to change.

`diagnose_store_cors.py`

https://www.allanninal.dev/medusa/store-requests-blocked-by-cors/

https://github.com/allanninal/medusa-fixes/tree/main/store-requests-blocked-by-cors

---

77 **Medusa storefront shows no products**

The catalog is full of products in the admin, but the storefront renders an empty grid. No error in the console, no failed request, just nothing. In Medusa v2 this is almost always a publishable API key that exists and is valid, but is not linked to any sales channel, so it authenticates fine and matches zero products. Here is why Medusa answers with an empty list instead of an error, and a small script that finds the broken keys and safely repairs the one case that is safe to repair.

`fix_publishable_key_sales_channel.py`

https://www.allanninal.dev/medusa/storefront-shows-no-products-missing-publishable-key/

https://github.com/allanninal/medusa-fixes/tree/main/storefront-shows-no-products-missing-publishable-key

---

78 **Stripe capture succeeds but no Medusa order is created**

Stripe's dashboard shows the charge as succeeded. The customer's card was debited. But there is no order in Medusa Admin, the cart is still sitting there with no completed_at, and support has no idea what to tell the customer who is asking where their confirmation email is. The money moved. Medusa never turned that cart into an order. Here is why that gap exists and a reconciler that finds it and repairs it the safe way.

`reconcile_orphaned_captures.py`

https://www.allanninal.dev/medusa/stripe-captured-no-order-created/

https://github.com/allanninal/medusa-fixes/tree/main/stripe-captured-no-order-created

---

79 **Order edit blocked by a stuck active order change record**

You open an order to edit a line item, process a return, or start an exchange, and Medusa refuses with "An active Order Change is required to proceed." The Admin UI shows nothing in progress. No one on the team is editing this order right now. But every attempt fails the same way, on every workflow, forever. Here is why a single orphaned OrderChange row can permanently block an order and a small script that finds and safely clears it.

`reconcile_stuck_order_change.py`

https://www.allanninal.dev/medusa/stuck-active-order-change/

https://github.com/allanninal/medusa-fixes/tree/main/stuck-active-order-change

---

80 **Stuck reservations after cancelled carts**

A shopper adds the last unit to their cart, then closes the tab, or the checkout times out, or someone voids the cart by hand outside the normal flow. The cart is gone, but the stock is not free. Reserved quantity keeps sitting against that inventory item as if a real order still needed it. Here is why Medusa v2 leaves these reservation rows behind and a small script that finds only the stale ones and releases them safely.

`release_stuck_reservations.py`

https://www.allanninal.dev/medusa/stuck-reservations-after-cancelled-carts/

https://github.com/allanninal/medusa-fixes/tree/main/stuck-reservations-after-cancelled-carts

---

81 **Subscriber fails silently on order.placed**

The order is there. The payment captured. Everything in the Admin looks correct. But the confirmation email never went out, or the warehouse sync never ran, and nobody can point to an error that explains why. The subscriber attached to order.placed failed, but it failed in a place nothing was watching, so the order finished as if everything had gone perfectly. Here is why a Medusa subscriber can fail without a trace and a script that finds the orders it happened to.

`find_missing_notifications.py`

https://www.allanninal.dev/medusa/subscriber-fails-silently-on-order-placed/

https://github.com/allanninal/medusa-fixes/tree/main/subscriber-fails-silently-on-order-placed

---

82 **Tax-inclusive pricing shows wrong totals**

A cart looks fine at a glance, but the numbers do not reconcile. The subtotal plus the tax total does not equal the total, or a shipping line seems to have tax baked in twice, or a discount only makes sense if you assume the wrong basis. Nothing crashed and no error was logged. Here is why tax-inclusive settings in Medusa v2 live in more places than most teams realize, how they drift apart, and a script that finds every context where they disagree so you can fix the setting, not guess at a number.

`find_tax_inclusivity_mismatches.py`

https://www.allanninal.dev/medusa/tax-inclusive-pricing-shows-wrong-totals/

https://github.com/allanninal/medusa-fixes/tree/main/tax-inclusive-pricing-shows-wrong-totals

---

83 **manage_inventory false variants still lose stock quantity**

You turned off inventory tracking on a variant because you never want it to sell out, or a supplier drop-ships it and stock does not apply. Then months later someone notices its stocked_quantity has been quietly falling anyway, sale after sale, refund after refund, with no error anywhere. Here is why Medusa does not always honor manage_inventory false across every workflow step, and a script that finds every variant this has happened to.

`detect_untracked_drift.py`

https://www.allanninal.dev/medusa/untracked-variant-quantity-drift/

https://github.com/allanninal/medusa-fixes/tree/main/untracked-variant-quantity-drift

---

84 **Medusa variant not purchasable, no inventory level**

The variant looks fine everywhere you check. It is active, it has a price, and the admin shows it sitting in the catalog like every other variant. But the storefront refuses to add it to the cart, or the Store API reports it out of stock no matter what you do. Nine times out of ten, the variant tracks inventory but never got an InventoryLevel row at a stock location the storefront's sales channel actually uses. Here is why that missing row is enough to make a variant unbuyable and a small script that finds it and fixes it safely.

`create_missing_levels.py`

https://www.allanninal.dev/medusa/variant-not-purchasable-no-inventory-level/

https://github.com/allanninal/medusa-fixes/tree/main/variant-not-purchasable-no-inventory-level

---

85 **Medusa variant options mismatch blocks creation**

You add a variant, or import a CSV, or edit a product's options after variants already exist, and Medusa refuses to save it. The error talks about option lengths not matching, or a variant that already exists, and it is not obvious which variant or which option is actually wrong. Here is why Medusa v2 is strict about every variant carrying a complete, valid set of option values, and a script that scans your catalog and tells you exactly which product and variant to fix.

`find_variant_options_mismatch.py`

https://www.allanninal.dev/medusa/variant-options-mismatch-blocks-creation/

https://github.com/allanninal/medusa-fixes/tree/main/variant-options-mismatch-blocks-creation

---

86 **Long running workflow executions get stuck in invoking state**

You go looking at a long running workflow days after it started and its workflow_execution row still says invoking. It never moved to done, never moved to failed, it is just sitting there. The step waiting on an external signal, a webhook, a worker, a subscriber, never got its answer, and Medusa has no built in clock to give up on it. Here is why that row gets stuck and a small script that finds the stuck ones and reports them for an operator to retry safely.

`flag_stuck_invoking.py`

https://www.allanninal.dev/medusa/workflow-execution-stuck-invoking/

https://github.com/allanninal/medusa-fixes/tree/main/workflow-execution-stuck-invoking

---

87 **Workflow left half done**

A workflow reserves stock, then a later step throws while charging a payment or booking a fulfillment. You expect Medusa to undo the reservation and leave things clean. Sometimes it does. Sometimes it does not, because the step that reserved the stock was never given a way to undo itself, or the process that was running the workflow died before it got the chance. Here is why a Medusa v2 workflow can be left half done and a small script that finds the leftover reservations and tells you, safely, what to do about them.

`reconcile_half_run_workflows.py`

https://www.allanninal.dev/medusa/workflow-left-half-done/

https://github.com/allanninal/medusa-fixes/tree/main/workflow-left-half-done

---

88 **Wrong currency shown for a region**

A merchant sets the region's currency to INR and adds INR prices, but the storefront keeps showing EUR. Nothing crashed and no error was logged, the number on the page just belongs to a different currency than the label next to it. Here is why a Region's currency_code and the price a customer actually sees can disagree in Medusa v2, and a script that finds every variant where that has happened so you can fix it with real numbers, not a guess.

`find_currency_mismatches.py`

https://www.allanninal.dev/medusa/wrong-currency-shown-for-a-region/

https://github.com/allanninal/medusa-fixes/tree/main/wrong-currency-shown-for-a-region

---