FIELD NOTES

# Shopware Field Guide

Shopware 6 order, stock and message queue fixes.

**59 fixes**, each one a written note with diagrams and a small Python and Node.js script that finds the problem and repairs it. All 59 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/shopware-fixes
Guides: allanninal.dev/shopware
All 14 repos: github.com/allanninal

## The fixes

1 **availableStock much lower than stock after a SW5 to SW6 migration**

The warehouse count looks right. Physical stock in Shopware 6 says two hundred units are sitting on the shelf. But the storefront thinks the product is sold out, or checkout refuses the quantity the customer wants, because availableStock is far below stock, sometimes deep into negative territory. Nothing changed in the warehouse. What changed is the migration, and thousands of old SW5 orders that Shopware 6 still thinks are open.

`reconcile_available_stock.py`
https://www.allanninal.dev/shopware/available-stock-low-after-migration/
https://github.com/allanninal/shopware-fixes/tree/main/available-stock-low-after-migration

---

2 **availableStock not restored after order cancellation**

A customer's order gets cancelled. The line items are gone, the order clearly shows cancelled, and the stock those items were holding should be free again. But the product still reads the old, lower availableStock, sometimes for minutes, sometimes until someone notices the storefront is refusing to sell units that are actually free. Nothing is broken forever, the number is just behind, because the recompute that frees the stock does not run inside the cancel request itself.

`reconcile_available_stock_after_cancel.py`
https://www.allanninal.dev/shopware/available-stock-stale-after-cancellation/
https://github.com/allanninal/shopware-fixes/tree/main/available-stock-stale-after-cancellation

---

3 **Bulk order status editor leaves orders mid transition**

You select forty orders in the Shopware 6 admin, pick a status from the bulk editor, and move on. A little while later someone notices one of those orders never actually moved. Its siblings from the same batch went from in progress to completed just fine, but this one is still stuck in the middle, and nothing retried it. Here is why the bulk editor drops orders like this and a small script that finds the stuck ones and finishes the job.

`retry_stuck_bulk_transitions.py`

https://www.allanninal.dev/shopware/bulk-order-transition-stuck/

https://github.com/allanninal/shopware-fixes/tree/main/bulk-order-transition-stuck

---

4 **Delayed flow does not trigger on payment failed transition**

You built a Flow Builder flow on order_transaction.state.state_enter for failed, added a delay so the customer gets a grace period before the follow up email goes out, tested it without the delay and it worked fine. With the delay in place, it just never fires. No error in the log, nothing in Flow Builder's own history that looks broken. The payment really did fail, and the merchant never found out. Here is why the delay breaks the flow, and a script that finds every order this has already happened to.

`flag_missed_delayed_flow.py`

https://www.allanninal.dev/shopware/delayed-flow-not-triggered-on-failed/

https://github.com/allanninal/shopware-fixes/tree/main/delayed-flow-not-triggered-on-failed

---

5 **Duplicate guest customer accounts accumulate per sales channel**

The same shopper checks out as a guest five times over six months, and Shopware quietly creates five separate customer records for the same email. No error, no warning, just a customer table that keeps growing with rows that all belong to the same person. Here is why Shopware never deduplicates guest checkouts and a script that finds every duplicate group without touching a single order.

`group_duplicate_guest_customers.py`

https://www.allanninal.dev/shopware/duplicate-guest-customer-accounts/

https://github.com/allanninal/shopware-fixes/tree/main/duplicate-guest-customer-accounts

---

6 **Duplicate number range configs for a sales channel desync counting**

A sales channel is only ever supposed to have one number range per type, one for orders, one for customers, one for order transactions. But that rule lives only in the admin screen. Underneath it, nothing stops a sales channel from being bound to two different number ranges for the same type at once. Once that happens, the "next number" preview you see in Settings quietly stops matching the number that is actually being handed out, and it never fixes itself. Here is why the two disagree and a script that finds every sales channel this has already happened to.

`find_duplicate_number_range_bindings.py`

https://www.allanninal.dev/shopware/duplicate-number-range-sales-channel/

https://github.com/allanninal/shopware-fixes/tree/main/duplicate-number-range-sales-channel

---

7 **Non-atomic number range storage can duplicate order numbers**

Two customers, two orders, and somehow the exact same order number on both invoices. It looks impossible, because Shopware's number range system is supposed to guarantee a unique value every time. It only keeps that promise while the increment storage behind it is a single, consistently configured, atomic backend. Move a number range between storages mid-flight, or misconfigure the Redis eviction policy it depends on, and the same number can go out twice. Here is why that guarantee breaks and a script that finds the duplicates without touching a single live order number.

`find_duplicate_order_numbers.py`

https://www.allanninal.dev/shopware/duplicate-order-numbers-race-or-eviction/

https://github.com/allanninal/shopware-fixes/tree/main/duplicate-order-numbers-race-or-eviction

---

8 **Duplicate seo_url rows accumulate for the same path in Shopware 6**

A category gets renamed, a template changes, a bulk reindex runs one more time than it needed to, and the seo_url table quietly grows a second, third, or fourth row that all claim the same storefront path. Only one of them is marked canonical, but the rest do not disappear, they just sit there, confusing anyone who queries the table directly and slowing down every future lookup on that path. Here is why Shopware leaves the old rows behind and a small script that groups them by path and reconciles each group down to the one row that should stay.

`reconcile_duplicate_seo_urls.py`

https://www.allanninal.dev/shopware/duplicate-seo-url-rows/

https://github.com/allanninal/shopware-fixes/tree/main/duplicate-seo-url-rows

---

9 **Shopware Elasticsearch alias not refreshed after a full reindex**

You ran a full reindex, waited for it to finish, and the console gave no error. But the storefront still shows an old price, a product that should be gone is still searchable, and a new one nobody can find. Nothing crashed. The read alias just never moved. Here is why Shopware can finish building a new Elasticsearch index and still leave the storefront pointed at the old one, and a small script that tells you when that has happened.

`check_alias_stale.py`

https://www.allanninal.dev/shopware/elasticsearch-alias-stale-after-reindex/

https://github.com/allanninal/shopware-fixes/tree/main/elasticsearch-alias-stale-after-reindex

---

10 **Flow Builder applies a payment status change to the wrong transaction**

A payment gets retried, a customer switches payment method mid-checkout, or an order is split into partial payments, and the order ends up with more than one order_transaction row. A Flow Builder flow fires its "Change payment status" action and blows up with an IllegalTransitionException, or worse, silently transitions the wrong one. Here is why Shopware cannot always tell which transaction the flow meant, and a small script that detects the mismatch and reports the correct transaction id for review before anyone forces a change.

`flag_wrong_transaction.py`

https://www.allanninal.dev/shopware/flow-builder-wrong-transaction-id/

https://github.com/allanninal/shopware-fixes/tree/main/flow-builder-wrong-transaction-id

---

11 **Guest checkout reuses an email already tied to a registered account**

A shopper has a registered account with an email you have on file, but at checkout they never log in, they just check out as a guest with that same email. Shopware does not stop them. It quietly creates a second customer row, guest true, tied to the exact same address, sitting right next to the real registered account. No error, no warning, just a customer table that now has two rows claiming the same person. Here is why the uniqueness check does not catch this and a script that finds every collision so a human can decide what to do about it.

`find_guest_email_collisions.py`

https://www.allanninal.dev/shopware/guest-order-reuses-registered-email/

https://github.com/allanninal/shopware-fixes/tree/main/guest-order-reuses-registered-email

---

12 **Install queue message for an app or service silently lost**

You install an app, or a data-intelligence service, and the admin says it worked. A few minutes later it is still greyed out, still inactive, and there is no error anywhere in the log. Shopware installs extensions by dispatching an asynchronous message rather than installing them on the spot, and in rare cases that message is dispatched and then simply never handled. Here is why that happens and a small script that finds the stuck rows and repairs them the same way Shopware's own workaround does.

`install_message_lost.py`

https://www.allanninal.dev/shopware/install-message-lost/

https://github.com/allanninal/shopware-fixes/tree/main/install-message-lost

---

13 **Invoice document created without a number or date**

A staff member opens an order, clicks Invoice, clicks Create document, and everything looks fine. Then later someone opens that same invoice and Shopware throws an error, or the PDF shows a blank where the invoice number should be. The document exists, it is attached to the order, but it never received a real documentNumber or documentDate. Here is why that gap opens up and a small script that finds every invoice document stuck this way so a human can review it before it goes anywhere near accounting.

`find_orphaned_invoice_documents.py`

https://www.allanninal.dev/shopware/invoice-document-missing-number/

https://github.com/allanninal/shopware-fixes/tree/main/invoice-document-missing-number

---

14 **Media path regeneration on 6.6 upgrade can break file references**

You upgraded to Shopware 6.6, the migration ran clean, and a chunk of your product images now 404 on the storefront. Nothing in storage moved. The files are still sitting exactly where they always were. What changed is how Shopware decides the URL to point at them, and for any media whose file name and stored object drifted apart before the upgrade, that new decision points at the wrong key. Here is why the rewrite breaks these references and a small script that finds the mismatches and repairs only the ones it can prove are safe.

`reconcile_media_path_drift.py`

https://www.allanninal.dev/shopware/media-path-regeneration-6-6-breaks-references/

https://github.com/allanninal/shopware-fixes/tree/main/media-path-regeneration-6-6-breaks-references

---

15 **Message queue backlog when only the admin worker runs**

Shopware 6 ships with a convenient trick for local development: while someone is logged into the Administration, the browser's own polling quietly drains the message queue and runs scheduled tasks. It is not meant for production. If a store has no real CLI consumer running in the background, messages and overdue scheduled tasks only get processed in short bursts whenever an admin happens to have a tab open, and everything else piles up the moment they log out. Here is why that happens and a small script that detects the backlog and raises an alert, without ever trying to start a worker itself.

`detect_queue_backlog.py`

https://www.allanninal.dev/shopware/message-queue-backlog-admin-worker/

https://github.com/allanninal/shopware-fixes/tree/main/message-queue-backlog-admin-worker

---

16 **Failed messages accumulate in the messenger failure queue**

A webhook payload comes in malformed, an indexing job times out, a third party API in a flow action errors out. Symfony Messenger retries the message a few times with backoff, and when it still fails it quietly moves into the separate failed transport. Almost nobody runs a worker against that queue, so the failures just sit there, growing the messenger_messages table and hiding real recurring problems behind an ever bigger pile. Here is why it happens and a small script that flags the backlog before it gets out of hand.

`messenger_failed_queue_backlog.py`

https://www.allanninal.dev/shopware/messenger-failed-queue-backlog/

https://github.com/allanninal/shopware-fixes/tree/main/messenger-failed-queue-backlog

---

17 **Migrated customers end up with multiple accounts across sales channels**

A store migrates from Shopware 5 or a third party platform like Magento into a Shopware 6 setup with several sales channels, and support starts hearing the same complaint: a returning customer logs in and their old orders are gone, or they cannot log in at all because Shopware is not sure which account is theirs. The email is right there in the system, more than once. Here is why the migration split one customer into several rows and a script that finds every split account so a human can decide what to do with it.

`report_customer_duplicates.py`

https://www.allanninal.dev/shopware/migrated-customer-duplicate-accounts/

https://github.com/allanninal/shopware-fixes/tree/main/migrated-customer-duplicate-accounts

---

18 **Number range hitting its maximum value stalls generation**

An order number range has been quietly counting up for years, and one day new orders start failing with a raw database error instead of a normal Shopware message. Nothing in the Admin warned anyone this was coming. Here is why a number range's counter can outgrow its own pattern or its own database column, and a small script that tells you exactly how much headroom every range has left before it happens to you.

`number_range_max_value_overflow.py`

https://www.allanninal.dev/shopware/number-range-max-value-overflow/

https://github.com/allanninal/shopware-fixes/tree/main/number-range-max-value-overflow

---

19 **Filtering Shopware 6 orders by cancelled payment status returns wrong results**

Someone runs a criteria filter for orders whose payment is cancelled, expecting to see the orders that genuinely failed to pay. Instead the list includes orders that are sitting there fully paid, or still waiting to be paid, mixed in with the ones that actually failed. Nothing is broken on the order itself. The filter is asking the wrong question of a collection that remembers every payment attempt an order ever had, not just the one that matters now. Here is why it happens and a small script that finds the false positives and reports them for review.

`report_cancelled_filter_mismatch.py`

https://www.allanninal.dev/shopware/order-filter-cancelled-mismatch/

https://github.com/allanninal/shopware-fixes/tree/main/order-filter-cancelled-mismatch

---

20 **Orders can be force cancelled outside allowed transitions**

A support script, an impatient admin user, or a retry loop calls the order cancel endpoint on an order that is already completed, or otherwise nowhere near a legal cancel edge. It should fail. For a long stretch of Shopware 6's life, it did not. A hardcoded escape hatch inside the state machine let the cancel action through no matter what, quietly corrupting the state graph. Here is why that happened, how Shopware fixed it, and a small script that audits your order history for every order this force cancel touched.

`audit_force_cancels.py`

https://www.allanninal.dev/shopware/order-force-cancel-illegal-transition/

https://github.com/allanninal/shopware-fixes/tree/main/order-force-cancel-illegal-transition

---

21 **Shopware 6 order line item cover references deleted media**

Someone tidies up the Media Library, removes a batch of images that look unused, and everything seems fine. Then an old order shows a broken thumbnail in the admin, or a generated invoice PDF is missing the product picture it used to have. The image was never really unused. An order line item still had its own cover pointing straight at it, and deleting the media file from the library does not check that reference the way deleting a product does. Here is why it happens and a small script that finds the dangling references and clears only the ones that carry no financial risk.

`repair_deleted_media_cover_references.py`

https://www.allanninal.dev/shopware/order-line-item-deleted-media-cover/

https://github.com/allanninal/shopware-fixes/tree/main/order-line-item-deleted-media-cover

---

22 **Shopware 6 order line item references a deleted promotion**

Somebody cleans up old promotions in the admin, selects all, and deletes them in one bulk action. Nothing looks wrong at first. Then a support agent tries to edit an old order that used one of those promotions and the save fails outright. The order is frozen. The cause is that the bulk delete skipped the safeguard that normally blocks deleting a promotion still referenced by an order, so the promotion row is gone while the order's line item still points straight at it. Here is why it happens and a small script that finds the dangling references and clears only the ones that carry no financial risk.

`repair_deleted_promotion_references.py`

https://www.allanninal.dev/shopware/order-line-item-deleted-promotion/

https://github.com/allanninal/shopware-fixes/tree/main/order-line-item-deleted-promotion

---

23 **Shopware 6 order line item orphaned after its product is deleted**

Someone hard-deletes a product that has been sold before. Nothing looks wrong at first. Then, weeks later, a support agent tries to edit the quantity on an old order and the save fails with a foreign-key or type error. The order is frozen. The cause is a database cascade that nulled out the line item's link to the product the moment it was deleted, while Shopware's own entity contract still assumes that link can never be null. Here is why it happens and a small script that finds the orphaned line items and relinks only the ones it can match with confidence.

`repair_orphaned_line_items.py`

https://www.allanninal.dev/shopware/order-line-item-orphaned-product/

https://github.com/allanninal/shopware-fixes/tree/main/order-line-item-orphaned-product

---

24 **Admin and API show a stale transaction state on multi-transaction orders**

A customer's card gets declined, they retry with a different payment method, and the second attempt succeeds. Or a shop switches payment method after a failed attempt. Or a partial payment gets recorded alongside the rest. Whatever the path, the order now has more than one order_transaction row, and Shopware 6 has a habit of showing you the wrong one. The order list, the order detail page, even the order history card can all report a payment status that is out of date, while the truth sits quietly in a transaction the admin never looked at. Here is why that happens and a small script that finds these orders and reports the real state for a human to check.

`report_stale_transaction_state.py`

https://www.allanninal.dev/shopware/order-stale-multi-transaction-state/

https://github.com/allanninal/shopware-fixes/tree/main/order-stale-multi-transaction-state

---

25 **Delivery or transaction stateMachineState returns null**

You search an order, read the delivery or the transaction, and stateMachineState comes back null while stateId sits there looking like a perfectly normal foreign key. The order is not broken. It might not even be a bug. Most of the time it is the Admin API quietly skipping an association you did not ask for, and only sometimes is it a state row that really has gone missing. Here is how to tell the two apart before you touch anything, and a small script that only repairs the second case.

`reconcile_state_machine_null.py`

https://www.allanninal.dev/shopware/order-state-machine-null/

https://github.com/allanninal/shopware-fixes/tree/main/order-state-machine-null

---

26 **Order state cannot be patched directly and must use transitions**

A bulk import tool, a custom ERP sync job, or an ad-hoc script tries to fix an order by sending a plain PATCH with a new stateId, straight to /api/order/{id} or /api/order-transaction/{id}. Shopware refuses every time with a 400 error about scope crud. That is not a bug, it is the state machine doing its job. Here is why Shopware locks state changes behind transitions and a small script that finds orders whose state, transaction, and delivery have drifted out of sync, then repairs them the correct way.

`fix_order_state.py`

https://www.allanninal.dev/shopware/order-state-requires-transition-action/

https://github.com/allanninal/shopware-fixes/tree/main/order-state-requires-transition-action

---

27 **Deleting the last line item leaves a Shopware 6 order total stale**

A support agent removes every product from an order, maybe to rebuild it from scratch, maybe by mistake while cleaning up a test order. The delete succeeds. The order now shows zero line items. But the total at the top still reads the old amount, sometimes hundreds of dollars, because nothing told Shopware to add it back up. The order is quietly lying about what it is worth. Here is why that happens and a small script that finds these orders and gets them in front of a human, or recalculated, safely.

`recalculate_stale_orders.py`

https://www.allanninal.dev/shopware/order-total-stale-after-last-item-deleted/

https://github.com/allanninal/shopware-fixes/tree/main/order-total-stale-after-last-item-deleted

---

28 **Shopware 6 order totals not recalculated after manual line item edits**

A support agent bumps a quantity on an order-line-item, or removes a row, straight through the Administration or a PATCH to the Admin API. The edit saves without complaint. But the order's amountTotal, amountNet, and any promotion discount rows never move, because nothing told Shopware to recompute them. The order now shows a total that does not match its own line items, and the mismatch sits there quietly until someone reconciles the books. Here is why it happens and a small script that finds the stale orders and repairs them with Shopware's own recalculation pipeline.

`repair_stale_order_totals.py`

https://www.allanninal.dev/shopware/order-totals-stale-after-edit/

https://github.com/allanninal/shopware-fixes/tree/main/order-totals-stale-after-edit

---

29 **Order transaction missing a defined paid transition**

A refund tool, a manual reconciliation script, or a support agent tries to move an order_transaction from paid_partially or reminded to paid, the same way it works for a plain open transaction. Shopware throws instead, as if the paid state did not exist at all. It exists, the row just is not there for that particular starting state under the action name your code is calling. Here is why some order_transaction state machines are missing that link and a small script that finds the gap and reports it safely.

`check_paid_transition.py`

https://www.allanninal.dev/shopware/order-transaction-missing-paid-transition/

https://github.com/allanninal/shopware-fixes/tree/main/order-transaction-missing-paid-transition

---

30 **Order transaction stuck open despite a completed payment**

A customer pays through an async payment method, the PSP dashboard shows the charge as settled, and support has the confirmation email to prove it. But the order in Shopware still shows the transaction as open. Nothing failed loudly. No error was ever seen, because the request that should have called the transition either lost a race or never ran. Here is why Shopware's async payment flow can drop that call and a small script that finds the mismatch, checks the PSP as the source of truth, and repairs it the correct way.

`fix_stuck_transaction.py`

https://www.allanninal.dev/shopware/order-transaction-stuck-open-payment-race/

https://github.com/allanninal/shopware-fixes/tree/main/order-transaction-stuck-open-payment-race

---

31 **Transition action names renamed on 6.7 migration**

A store upgrades to Shopware 6.7 and a plugin, app, or webhook that used to call the pay, pay_partially, or do_pay transition on an order transaction suddenly gets a 400 error, as if the transition no longer exists. It does not exist under that name any more. A core migration renamed it, and on early 6.7 builds it sometimes renamed the same action name on a completely unrelated custom state machine too. Here is why that happens and a small script that finds every integration still calling the old name and reports a safe remap.

`check_renamed_transitions.py`

https://www.allanninal.dev/shopware/order-transition-actionname-renamed-6-7/

https://github.com/allanninal/shopware-fixes/tree/main/order-transition-actionname-renamed-6-7

---

32 **Orphaned media cannot be cleaned up non-interactively**

Your media library only grows. Products get re-photographed, variants get replaced, CMS blocks get redesigned, and the old images stay behind on disk forever. Shopware ships a command for exactly this, bin/console media:delete-unused, but it stops and asks you a yes or no question before it will delete anything. Put that command in a cron job or a CI pipeline and it just fails, every single run, because nothing is there to answer the prompt. Here is why Shopware built it this way and a small script that does the same safe cleanup without ever needing a human at the keyboard.

`purge_orphaned_media.py`

https://www.allanninal.dev/shopware/orphaned-media-cleanup-non-interactive/

https://github.com/allanninal/shopware-fixes/tree/main/orphaned-media-cleanup-non-interactive

---

33 **Out of stock products remain addable to cart**

Stock hits zero, the product page still shows a live buy button, and a customer orders something you cannot ship. It looks like a bug in the storefront. It is not. Shopware only disables purchasing when a product has the closeout flag set, and if that flag was never turned on, the product stays purchasable no matter how far stock drops. Here is why that happens and a small script that finds every product currently in that state.

`find_falsely_purchasable.py`

https://www.allanninal.dev/shopware/out-of-stock-still-purchasable/

https://github.com/allanninal/shopware-fixes/tree/main/out-of-stock-still-purchasable

---

34 **Overselling of closeout items under concurrent checkout**

A closeout item, marked isCloseout because there is no restock coming, has exactly three left. Then a sale email goes out, or the product lands on a deal site, and within the same second two or three shoppers all hit checkout. Every one of them sees the item in stock. Every one of them gets a confirmation email. But there were only three units, and now availableStock reads negative and someone paid for a unit that does not exist. Here is why Shopware lets this happen and a small script that finds it and tells you exactly which paid order is the unfulfillable one.

`reconcile_overselling_closeout.py`

https://www.allanninal.dev/shopware/overselling-closeout-concurrency/

https://github.com/allanninal/shopware-fixes/tree/main/overselling-closeout-concurrency

---

35 **Payment state history not updated after switching payment method post-failure**

A customer's card is declined at PayPal, or the external provider times out, and the checkout lets them pick a different payment method to try again. The order goes on to get paid, or it does not, but the audit trail never says so. The order_transaction's stateMachineState and its state_machine_history rows fall out of sync, flow-builder's "Payment status changed" trigger never fires, and nobody can reconstruct what happened just from looking at the history table. Here is why Shopware 6 leaves that gap and a small script that finds it.

`flag_history_gaps.py`

https://www.allanninal.dev/shopware/payment-method-switch-history-gap/

https://github.com/allanninal/shopware-fixes/tree/main/payment-method-switch-history-gap

---

36 **Active assigned products still absent from storefront listings**

The product is active, it is assigned to the sales channel, its product_visibility row says visibility: 30, and it sits in the right category. Pull it with GET /api/product/{id} and everything checks out. Yet the storefront listing and search pages act like it does not exist. Here is why the storefront's read side can fall behind the true state and a small script that finds the exact products affected and forces them back in sync.

`reconcile_absent_products.py`

https://www.allanninal.dev/shopware/product-absent-despite-active-assigned/

https://github.com/allanninal/shopware-fixes/tree/main/product-absent-despite-active-assigned

---

37 **Product available forced to zero when isCloseout is null**

A product gets created through the Admin API, or imported by a tool that never touches the closeout flag, and isCloseout sits at null. Stock is fine. minPurchase is fine. But the storefront still says the product is not available, and nothing about the product data looks wrong at a glance. Here is why Shopware's own availability math breaks on a null closeout flag, and a small script that finds every product this hits and repairs it safely.

`reconcile_null_closeout.py`

https://www.allanninal.dev/shopware/product-available-zero-null-closeout/

https://github.com/allanninal/shopware-fixes/tree/main/product-available-zero-null-closeout

---

38 **Shopware product export scheduled task silently fails or produces an incomplete feed**

A price comparison portal is showing stock that sold out days ago. A storefront feed a marketplace app reads has not changed since last week. Nothing in the admin is red, no error banner, no failed job in sight. The product export cronjob just quietly stopped keeping one or more of your feeds current. Here is why Shopware lets this happen and a small script that finds the stale feeds and repairs the one piece that is safely repairable through the API.

`check_product_export.py`

https://www.allanninal.dev/shopware/product-export-task-silent-failure/

https://github.com/allanninal/shopware-fixes/tree/main/product-export-task-silent-failure

---

39 **Products missing a product_visibility row for a sales channel**

The product is active, it has stock, it sits in the right category, and it is assigned to the sales channel in the Admin. Everything about it looks correct. Yet it never shows up in that channel's storefront, search, or category pages. Here is why Shopware leaves these products invisible and a small script that finds the exact products and channels missing the row, then repairs it safely.

`repair_missing_visibility.py`

https://www.allanninal.dev/shopware/product-missing-visibility-row/

https://github.com/allanninal/shopware-fixes/tree/main/product-missing-visibility-row

---

40 **Product streams filtering on available stock return empty with Warehouses**

You built a dynamic product group with a plain stock condition, something like availableStock greater than or equal to one. It worked fine before. Then the commercial Warehouses extension went live, stock moved to per-location records, and the same stream now returns nothing, even though every warehouse clearly shows units on the shelf. Nothing about the stream changed. What changed is which stock number Shopware is actually reading.

`flag_stream_stock_mismatch.py`

https://www.allanninal.dev/shopware/product-stream-stock-filter-empty-warehouses/

https://github.com/allanninal/shopware-fixes/tree/main/product-stream-stock-filter-empty-warehouses

---

41 **Promotion applies even though its qualifying rule is not met**

A promotion has one job, a rule such as "Grand total over 49.99", spend enough and the discount shows up. Except sometimes it shows up on a cart that, once the discount itself is subtracted, no longer clears that bar. The customer got a discount they should not have gotten, or an order sits with a discount line item that quietly violates its own rule. Here is why Shopware's cart engine lets this happen and a small script that finds the orders where it did.

`flag_rule_violating_promotions.py`

https://www.allanninal.dev/shopware/promotion-applies-rule-not-met/

https://github.com/allanninal/shopware-fixes/tree/main/promotion-applies-rule-not-met

---

42 **Lowercase promotion code entry is not marked as redeemed**

A customer types summer2026 at checkout, the discount applies, and the order completes without a hitch. The code was actually stored as SUMMER2026, and it was supposed to be a one-time code. Nobody sees a problem until you notice the same code applied on a dozen orders, each with slightly different capitalization. Shopware happily accepted every one of them, and never flagged a single one as used. Here is why the checkout and the bookkeeping disagree, and a small script that finds every order this happened to.

`detect_case_mismatch_reuse.py`

https://www.allanninal.dev/shopware/promotion-code-lowercase-not-redeemed/

https://github.com/allanninal/shopware-fixes/tree/main/promotion-code-lowercase-not-redeemed

---

43 **Cancelled orders do not release their reserved promotion code**

A customer checks out with a one-time promotion code, then the order gets cancelled, whether by the customer, a failed payment, or a support agent cleaning up test data. The order is gone. The code is not free. It sits marked as redeemed forever, unusable by the next customer who tries it, even though nothing is actually holding it anymore. Here is why Shopware never lets go of that code and a small script that finds the ones stuck this way and reports them for review.

`release_orphaned_promotion_codes.py`

https://www.allanninal.dev/shopware/promotion-code-not-released-on-cancel/

https://github.com/allanninal/shopware-fixes/tree/main/promotion-code-not-released-on-cancel

---

44 **Promotion maximum usage limit not enforced**

A promotion is set to a hard cap, say two hundred redemptions total, or one per customer. Weeks later someone notices order after order still carrying that code well past the cap. Nothing crashed and no error was logged. Shopware was checking the limit against a number it had not caught up on yet. Here is why the cap gets checked against a stale cached count instead of the real one, and a script that finds every promotion that actually went over.

`evaluate_promotion_usage.py`

https://www.allanninal.dev/shopware/promotion-max-usage-not-enforced/

https://github.com/allanninal/shopware-fixes/tree/main/promotion-max-usage-not-enforced

---

45 **Quote creation consumes the shared order number range**

Your order numbers should run 1001, 1002, 1003, one per order, with nothing missing. Then you notice the sequence has raced far ahead of how many orders you actually have. Numbers are being handed out and never used. The culprit is often the B2B quote flow: every time a quote is created or recalculated, it reserves the next value from the very same order number range that real orders draw from, and that value is never given back. Here is why a quote quietly eats an order number, and a script that measures the drift without touching a single live order.

`detect_order_number_drift.py`

https://www.allanninal.dev/shopware/quote-consumes-order-number-range/

https://github.com/allanninal/shopware-fixes/tree/main/quote-consumes-order-number-range

---

46 **Refresh token race condition causes random invalid_client 401s**

An integration has been refreshing its Shopware Admin API token for weeks without a problem, then out of nowhere a call fails with a 401 and the body says invalid_client. Retrying the exact same refresh a second later works fine. Nothing about the credential changed. What changed is timing: two refresh requests went out with the same cached refresh token at nearly the same instant, and Shopware's OAuth layer only lets one of them win. Here is why that happens and a small client that detects the race and recovers without ever hammering the token endpoint.

`refresh_race_guard.py`

https://www.allanninal.dev/shopware/refresh-token-race-invalid-client/

https://github.com/allanninal/shopware-fixes/tree/main/refresh-token-race-invalid-client

---

47 **Return processing does not restore product stock**

A customer sends an item back. Support processes the return in the Administration, the return line items flip to returned, and the order shows the refund went out. Everyone assumes the unit is back on the shelf and sellable again. It is not. product.stock never moved, so availableStock is still short by exactly the quantity that just came back through the door. Nothing crashed, Shopware simply never wired returns to stock in the first place.

`restore_return_stock.py`

https://www.allanninal.dev/shopware/return-stock-not-restored/

https://github.com/allanninal/shopware-fixes/tree/main/return-stock-not-restored

---

48 **Shopware 6 scheduled task run interval resets after deploy**

Someone tuned a scheduled task to run less often, maybe once a day instead of every five minutes, and it worked fine for weeks. Then a deploy went out and the task is back to running every five minutes, with nothing in the changelog explaining why. Here is why scheduled-task:register keeps doing this and a small script that finds the drifted tasks and restores your tuning without ever fighting the vendor defaults.

`scheduled_task_interval_reset.py`

https://www.allanninal.dev/shopware/scheduled-task-interval-reset/

https://github.com/allanninal/shopware-fixes/tree/main/scheduled-task-interval-reset

---

49 **Shopware scheduled task stuck in queued status forever**

A cache warmer, a cleanup job, or a mail queue task sat down at status queued and never moved again. The store keeps running, but that one job silently stopped, and nothing in the admin tells you why. Here is why Shopware leaves a task behind like this and a small script that finds the stuck row and resets it so it can run again.

`reset_stuck_task.py`

https://www.allanninal.dev/shopware/scheduled-task-stuck-queued/

https://github.com/allanninal/shopware-fixes/tree/main/scheduled-task-stuck-queued

---

50 **SeoResolver can pick a soft-deleted URL as canonical in Shopware 6**

Two rows share the same storefront path, one is live and one was soft-deleted, and the storefront serves the wrong one. It happens because the sort that chooses which row is canonical looks at flags like isCanonical and when the row was written, but it never pushes soft-deleted rows to the bottom. So a row marked isDeleted that still carries a stale isCanonical flag, or that simply sorts ahead of the live row, can win the ordering and get handed back as the canonical URL. Here is why the resolver misses isDeleted, and a small script that flags the bad rows and can clear the stale flag without deleting anything.

`fix_soft_deleted_canonical.py`

https://www.allanninal.dev/shopware/seo-resolver-canonical-soft-deleted/

https://github.com/allanninal/shopware-fixes/tree/main/seo-resolver-canonical-soft-deleted

---

51 **SEO URL not generated for a product on a shared-language sales channel**

A product already ranks fine on your main sales channel, with a clean SEO path in every link. Then you assign it to a second sales channel that happens to use the same language, expecting the same thing to happen there. Instead the second channel silently gets nothing. No SEO URL is written for it, and every storefront link for that product on that channel falls back to the raw /detail/{id} route. Here is why Shopware's own uniqueness rule causes this and a script that finds every product it has already happened to.

`find_missing_seo_urls.py`

https://www.allanninal.dev/shopware/seo-url-missing-shared-language-channel/

https://github.com/allanninal/shopware-fixes/tree/main/seo-url-missing-shared-language-channel

---

52 **Stale SEO URLs remain non-canonical after a template change**

A merchant opens Settings, SEO, and edits the URL template for products or categories, expecting every page to pick up the new pattern right away. Some pages do. Many do not. Weeks later the old slugs are still resolving, some as quiet redirects and some as the only URL the page has, and nobody can tell from the storefront alone which pages actually caught up and which are just sitting on outdated links. Here is why Shopware leaves this half-finished and a small script that finds the stale rows and safely triggers the indexer to fix them.

`reindex_stale_seo_urls.py`

https://www.allanninal.dev/shopware/seo-url-stale-after-template-change/

https://github.com/allanninal/shopware-fixes/tree/main/seo-url-stale-after-template-change

---

53 **HTTP cache shows stale max purchasable quantity after stock changes**

An order goes through, stock drops, and the database is correct the moment the transaction commits. But the next anonymous visitor to that product page still sees the old max purchasable quantity in the quantity selector, sometimes for minutes. Nothing is broken in the product data. Shopware's HTTP cache is just serving a page it rendered before the stock changed, and by default it does not rush to throw that page away. Here is why the delay exists, how to prove a mismatch is really a stale cache and not bad data, and a small script that flags it safely.

`flag_stale_cache_max_quantity.py`

https://www.allanninal.dev/shopware/stale-cache-max-quantity/

https://github.com/allanninal/shopware-fixes/tree/main/stale-cache-max-quantity

---

54 **Deleting a product variant leaves stale configurator setting rows**

A merchant opens the variant overview, right clicks one row that is no longer needed, and picks Delete. The variant disappears from the grid, the product count drops, and everything looks clean. But the row Shopware kept to remember which property group options that variant used, its product_configurator_setting row, is still sitting in the database, pointing at a productId that no longer exists. Here is why the single row delete and the regenerate variants workflow disagree, and a small script that finds every orphaned row without touching anything by default.

`find_orphaned_configurator_settings.py`

https://www.allanninal.dev/shopware/stale-configurator-settings-after-variant-delete/

https://github.com/allanninal/shopware-fixes/tree/main/stale-configurator-settings-after-variant-delete

---

55 **Stock and availableStock drift after direct writes bypass the stock API**

A bulk import runs, an ERP sync pushes numbers straight into the database, or someone patches product.stock by hand outside the normal admin flow. The physical stock count updates, but the storefront keeps selling from an availableStock number that never moved. Nothing looks broken in the Administration, the product page just quietly oversells or undersells until someone notices the mismatch in a support ticket. Here is why Shopware leaves that second number behind and a small script that finds every drifted product and repairs it the way Shopware's own pipeline would.

`reconcile_stock_drift.py`

https://www.allanninal.dev/shopware/stock-drift-bypassing-stock-api/

https://github.com/allanninal/shopware-fixes/tree/main/stock-drift-bypassing-stock-api

---

56 **Sync API re-runs create duplicate configurator setting entries in Shopware 6**

An external PIM or ERP pushes variant configuration through the Sync API, and it works. Then the same sync runs again the next day, and the same product ends up with two rows for the same option in product_configurator_setting. Run it a third time and there are three. Nothing errors, the storefront still looks fine at first glance, but the table quietly fills with rows that all claim to configure the same option on the same product. Here is why Shopware's upsert lets this happen and a small script that finds every duplicate group so you can clean it up on your terms.

`find_duplicate_configurator_settings.py`

https://www.allanninal.dev/shopware/sync-api-duplicate-configurator-settings/

https://github.com/allanninal/shopware-fixes/tree/main/sync-api-duplicate-configurator-settings

---

57 **Thumbnail marked generated but the physical file is missing**

Somewhere on your storefront a product image loads fine, but the thumbnail it should shrink down to comes back as a 404. Shopware's own records say the thumbnail exists. The size is right, the width and height are in the database, everything about the row looks done. But the file itself was never actually written to disk or to S3, and because Shopware only checks whether the record exists, not whether the file exists, it never tries to fix it again on its own. Here is why it happens and a small script that finds the orphaned records and repairs them.

`repair_missing_thumbnail_files.py`

https://www.allanninal.dev/shopware/thumbnail-generated-but-file-missing/

https://github.com/allanninal/shopware-fixes/tree/main/thumbnail-generated-but-file-missing

---

58 **Variant listing config points at a removed main variant**

A parent product's variant listing config decides which variant the storefront shows on a category or listing page, and it makes that decision by holding one specific variant's product.id in mainVariantId. Delete that variant, or clone the parent, and the pointer can be left dangling or aimed at a variant that belongs to someone else now. Shopware never checks. The config just sits there until the storefront tries to resolve it and something breaks quietly.

`fix_stale_main_variant.py`

https://www.allanninal.dev/shopware/variant-listing-config-stale-main-variant/

https://github.com/allanninal/shopware-fixes/tree/main/variant-listing-config-stale-main-variant

---

59 **Webhook event logs stuck in queued state pile up**

Every webhook Shopware sends starts the same way: a row lands in webhook_event_log with deliveryStatus set to queued, right as the message is handed to the message queue. Most of the time a consumer picks it up moments later and flips it to success or failed. But when that message never gets consumed, no worker running, a dropped connection, a deleted receiver, an app deactivated mid-flight, the row just stays at queued. Forever. Here is why Shopware left that gap open until recently, and a small script that finds the truly orphaned rows and clears them out safely.

`cleanup_webhook_event_log.py`

`https://www.allanninal.dev/shopware/webhook-event-log-backlog/`

`https://github.com/allanninal/shopware-fixes/tree/main/webhook-event-log-backlog`

---