Serhard

30 payment failure modes

The places where payment systems most often lose, duplicate or lose track of money. Each one is a real failure I have seen in production.

Tick an item only if you could show evidence for it today: a test, a query, a dashboard or a runbook. Anything unticked is worth a closer look.

Idempotency

The same request, sent twice, must move money once.

1. ☐ **Every money-moving endpoint takes an idempotency key**
   Reusing a key with a different payload is rejected, not silently processed.
2. ☐ **The key and its result are saved in the same transaction as the money movement**
   A crash between the two cannot leave a charge without a stored result.
3. ☐ **Calls to your PSP or bank carry a stable idempotency key**
   Your own retries reuse the same key instead of generating a new one.
4. ☐ **Keys are kept longer than your longest retry window**
   Including queue redelivery, dead-letter replays and manual re-runs.
5. ☐ **Two concurrent requests with one key cannot both execute**
   One wins; the other waits for, or receives, the same result.

Retries and timeouts

A timeout means "I don't know", never "it failed".

6. ☐ **A timeout puts the payment in an unknown state**
   You query the provider for the real status before retrying or failing it.
7. ☐ **Retries use backoff, jitter and a hard cap**
   A PSP outage does not turn into a retry storm when it recovers.
8. ☐ **Double clicks and resubmits cannot create a second payment**
   The client sends the same key for the same checkout attempt.
9. ☐ **Payments stuck in pending are detected**
   Anything past its expected time raises an alert with an owner.

Webhooks and events

Assume every notification arrives late, twice, out of order, or not at all.

10 ☐ **Webhook signatures and timestamps are verified**
Replayed or forged notifications are rejected.

11 ☐ **Duplicate webhooks are deduplicated by the provider's event ID**
Processing the same event twice changes nothing.

12 ☐ **Out-of-order events cannot move a payment backwards**
An "authorized" event arriving after "captured" is ignored, not applied.

13 ☐ **Webhooks are acknowledged only after state is durably saved**
Or written to an inbox table first and processed from there.

14 ☐ **Missed webhooks are recovered**
A scheduled job polls or backfills status for anything unconfirmed.

Ledger

If it is not in the ledger, it did not happen.

15 ☐ **Every movement is double-entry and balances to zero**
Debits equal credits for each transaction, enforced in code or the database.

16 ☐ **Ledger entries are append-only**
Mistakes are corrected with reversing entries, never by updating or deleting rows.

17 ☐ **Balances can be rebuilt from entries**
A regular job recomputes them and alerts on any difference from stored balances.

18 ☐ **Each entry links to its external reference**
PSP payment ID, bank reference or chain transaction hash.

19 ☐ **Fees, FX spread and chargebacks are booked separately**
Not netted silently into the settled amount.

Reconciliation

Your numbers are only right once someone else's numbers agree.

20 ☐ **Every PSP and bank account is reconciled daily**
Against settlement files or statements, not just your own API responses.

21 ☐ **Unmatched items have an owner, an age and an alert**
Nothing sits in "unreconciled" for weeks without someone noticing.

22 ☐ **Refunds, chargebacks, reversals and fees are reconciled too**
Not only successful captures.

23 ☐ **Importing a settlement file twice does not double-count it**
File ingestion is itself idempotent.

Concurrency and money math

Small rounding errors and race conditions add up at volume.

24 ☐ **Amounts are never stored or calculated as floats**
Integers in minor units, or fixed-precision decimals end to end.

25 ☐ **Decimal places come from the currency, not a hardcoded 2**
JPY has 0, KWD has 3, and crypto assets have up to 18.

26 ☐ **Balance check and debit happen atomically**
Row lock, conditional update or serializable transaction, so two debits cannot overdraw.

27 ☐ **The FX rate and rounding rule are stored with each conversion**
So any amount can be recalculated and explained later.

Operations and audit

When something goes wrong, you can see it and explain it.

28 ☐ **Every payment state change has an audit trail**
What changed, when, and which user or service changed it.

29 ☐ **Manual adjustments need a second approver**
And go through the ledger like any other movement.

30 ☐ **You can say where any customer's money is in under five minutes**
Without asking an engineer to write a query.

More than five unticked?

That is normal for a growing payment platform, and fixable. A Payment Integrity Audit traces these issues in your real code and data, with a fix for each one. Fixed price from €3,500, done in two weeks.

serhii@serhard.com serhard.com