

Selling Through the Outage: An Experience Report on an Offline-First, Exactly-Once Supermarket Point-of-Sale in Duhok, Kurdistan

Serdar Dilshad

Independent researcher, Duhok, Kurdistan Region

hello@serdardilshad.dev

Abstract

Retail software in regions with unreliable grid power and internet links cannot treat the network as part of the checkout. We report on the design, testing and installation of a point-of-sale (POS) and back-office system for a four-lane supermarket in Duhok, Kurdistan Region, built over three months by a single developer. Each till sells from a local SQLite replica and writes every finalized document into a hash-chained outbox in the same local transaction. Documents drain to an in-store Hub and, when the Hub is unavailable, directly to a cloud server. The server applies each document exactly once, keyed by a globally unique document identifier in the same transaction as its stock movements and ledger postings, and it never rejects a money document: anomalies such as oversell are accepted and flagged for human reconciliation. Replication downward uses cursors over a server-assigned sequence and a nightly integrity check. A pure, unit-tested failover function decides when a till may leave the Hub for the cloud, requiring proof before every switch, hysteresis, and monotonic time. We describe these mechanisms; a "direction law" that keeps store-to-cloud replication free of two-way conflicts; a single job-queue pattern through which the owner controls label scales and fingerprint attendance terminals from a phone or tablet; and a catalogue of twenty field failures found during benching and installation, from which we draw six engineering principles. The codebase comprises roughly 310,000 lines across 1,243 commits, 156 schema migrations and 736 test files, including property-based tests of idempotency and replay order-insensitivity. We are explicit about limits: the system is installed but had not yet traded with real customers at the time of writing, so we report design and verification evidence rather than production outage statistics.

Keywords: offline-first, local-first software, point of sale, exactly-once processing, transactional outbox, tamper-evident logs, failover, label scales, fingerprint attendance, developing regions, experience report

1 Introduction

A supermarket checkout is a hard real-time human interface: a queue forms in seconds, and a till that waits on a server is a till that is not selling. In much of the world this is a theoretical concern handled by a redundant network link. In the Kurdistan Region, grid supply is routinely supplemented by neighbourhood and building generators, switch-overs cause short power interruptions, and consumer-grade internet links drop for minutes or hours. Software built on the assumption of a reliable connection to a central server fails in these conditions in ways that cost the shop money and trust.

This report describes a POS and enterprise resource planning (ERP) system built for a four-lane supermarket in Duhok.¹ The system replaces paper and spreadsheet processes with tills, a back office, weighing-scale and label integration, fingerprint attendance and owner dashboards. Its central

requirement was stated by the shop in plain terms: if the internet is cut, the four tills must keep working with the in-store computer until the internet returns, and then all data must reach the cloud "fully accurate"; and if the in-store computer or its cable fails, the tills must continue with the cloud directly.

A second constraint shaped the engineering as much as the network did: the developer is hours away from the store. A failure that does not reach a phone is a failure that runs for days. This pushed the design toward active watchdogs, alerts that leave the building, and remote updates with safety gates.

Our contributions are:

1. An architecture in which the network is never on the checkout path: a durable, hash-chained outbox per till, exactly-once ingest keyed by document identity, and an accept-and-flag reconciliation policy for money documents (§3–4).

2. A dual-home failover function that requires proof before every switch, treats an unrecoverable sequence gap as a failure, and runs on monotonic time (§5).
3. A "direction law" for store-to-cloud replication in which no stream travels both ways, and the rule that identifiers minted from local sequences must never cross a server boundary (§6).
4. A single job-queue pattern for driving store hardware (label scales and fingerprint attendance terminals) from a back office that runs on a phone or tablet, and the safety rules that protect irreplaceable device data (§7).
5. A catalogue of twenty field failures found during benching and installation, distilled into six engineering principles that we believe generalise beyond retail (§8).
6. Concrete engineering evidence for a single-developer system of this size, and an honest account of what has not yet been measured (§9–10).

2 Context and requirements

The store runs four checkout lanes on small desktop PCs, an in-store Hub server, label scales, receipt printers and fingerprint attendance terminals. The cloud server runs on a rented virtual machine. The shop trades in the local dinar and other currencies, with cash rounding; the business day runs from 05:00 to 05:00 local time (UTC+3, no daylight saving), because night-shift sales belong to the previous trading day. From the shop's statement and documented failures of fielded POS systems we derived six requirements:

- **R1 Never block the ring path.** Scanning, pricing, tendering and printing must complete from local state.
- **R2 Zero lost sales, zero double counts.** Every finalized sale reaches the books exactly once, regardless of retries, crashes or outage length.
- **R3 Survive either single failure.** Losing the internet or losing the Hub must not stop trading or stop the upload of the owner's books.
- **R4 Money and time are exact.** Integer minor units for money; every business-day computation names its timezone.
- **R5 Tamper evidence.** A cashier must not be able to silently remove a sale from a till's history.
- **R6 Remote operability.** Updates, alerts and diagnosis must work without a site visit.

3 Architecture

The system has three tiers (Figure 1). *Lanes* are desktop applications with an embedded SQLite database. The *Hub* and the *cloud* run the same server code (Node.js and PostgreSQL), selected by a deployment profile. Lanes talk to the Hub over the store's wired LAN; the Hub replicates to the cloud; and lanes can fall back to the cloud directly. There is no lane-to-lane peer-to-peer channel: an early design using conflict-free replicated data types between tills was rejected because lanes only need to coordinate through a single source of truth, and the shop's accounting must have one.

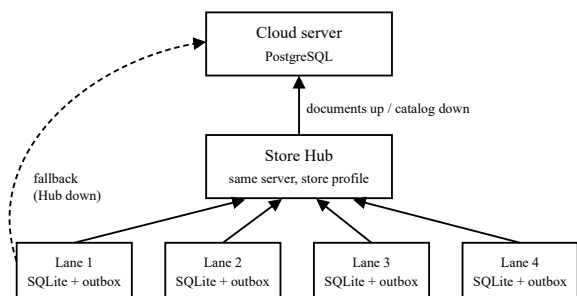


Figure 1. Deployment. Solid arrows are the normal path over the store LAN; the dashed arrow is the dual-home fallback taken only after the failover function of §5 approves it. Each lane sells from its local replica in every state.

The same synchronisation protocol runs at two levels: lane to Hub, and Hub to cloud. Because the protocol uses per-upstream cursors and globally unique document identifiers, moving the lanes from a cloud-only deployment to the Hub was a configuration change rather than a rewrite.

Money is represented as integers in minor units in TypeScript and as 64-bit integers in PostgreSQL; floating-point money is banned. Quantities, which include weighed goods, use exact decimals. Cash rounding is an explicit ledger line within a bounded interval, so invoice totals always reconcile with tenders.

4 The synchronisation protocol

4.1 Durable, hash-chained outbox

Every finalized document (sale, return, shift event, cash event, manager override, no-sale) is written to the lane's outbox in the *same SQLite transaction* that finalizes it. There is no window in which a sale exists but cannot be synchronised; this is the transactional outbox pattern [11] applied on the client. Each outbox row carries a monotonic local sequence number, the document and a globally unique identifier, and rows are linked into a

per-lane hash chain in the spirit of tamper-evident logging [8]. The chain is verified when documents arrive at the server, and a break is escalated for review. This satisfies R5: deleting or editing a past sale on a till breaks a chain that the server checks.

The lane's local database is configured for full durability, so a power cut during tendering leaves the cart either finalized and queued or cleanly absent.

4.2 Exactly-once ingest

The drain loop sends batches in sequence order. The server answers per document: applied, duplicate, or rejected. Exactly-once effect is obtained by an ingest log keyed by document identifier, inserted if absent *in the same database transaction* as the document, its stock movements and its general-ledger postings. A retry after a lost acknowledgement is therefore recognised as a duplicate, and the lane marks the row as synced. This is idempotence by identity [6] rather than distributed atomic commit [5]. Outbox rows are pruned only after their presence on the server has been independently confirmed.

Per-lane order is preserved; cross-lane order is irrelevant by design. Ingest is order-insensitive across lanes, which removes any need for coordination between tills [7].

4.3 Money documents never bounce

A sale that has physically happened cannot be "rejected"; it can only be reconciled. The server therefore accepts every well-formed money document and records anomalies as flags for human review (Table 1). Rejection is reserved for malformed payloads.

Table 1. Reconciliation rules applied at ingest.

Situation	Rule
Same document again	Duplicate; no effect
Oversell against server stock	Accept; negative stock flagged; count task created
Price changed during outage	Cached price honoured; difference noted
Sale for a closed business day	Posted as a flagged adjustment to that day, never a silent edit
Return whose sale is still queued	Retried after earlier lane sequence numbers land
Training-mode document	Never reaches real ledgers

Two exceptions are deliberately stricter. Credit (customer account) sales require a live server check and

are refused on any network failure, because a debt recorded against stale balances is harder to unwind than a declined tender.

4.4 Downward replication without clocks or tombstones

Every replicated table carries a server-assigned monotonic sequence, and each lane stores one cursor per table ("I have everything up to N "). A pull returns changed rows in sequence order; the lane applies a batch in one transaction and then advances the cursor, so a crash mid-batch simply re-pulls the same batch. Cursors rather than timestamps mean clock skew cannot lose rows [10], and "resume after three days offline" is the same operation as "sixty seconds passed".

Replicated data is never hard-deleted, because a deleted row is invisible to a cursor-based replica; removals travel as ordinary updates.

A nightly integrity check compares a compact per-table summary between lane and server. A mismatch is re-pulled once before being called drift, so ordinary lag is never "healed"; confirmed drift raises an alert and is repaired. Self-healing is never silent.

4.5 Behaviour during an outage

The lane's status banner shows *offline* with the number of queued documents; the cashier's workflow does not change. Prices were synchronised as "current and next effective", so the 05:00 business-day boundary still flips correctly offline. Each lane decrements only its own stock snapshot; cross-lane truth resumes on reconnect, where oversell surfaces as flagged negative stock. On reconnect the drain is paced and pull-first, so four lanes returning after a long outage fetch fresh prices before pushing heavy queues.

5 Dual-home failover

With the Hub in place, an internet outage changes nothing about trading. The remaining single point of failure is the Hub itself. Selling never depends on it, but the *upload*, which is the owner's books, does. The lane therefore needs to decide when to send its outbox to the cloud instead.

The decision is a pure function of its inputs (where the lane is pointing, how long that has been failing, and recent probe results), so its safety properties are unit-tested rather than observed in the field.

The first version switched to the cloud after a fixed period of failures without asking the cloud anything,

and returned home on a shallow health check. Two field scenarios broke it. A lane whose *own* cable is cut reaches neither server, so it "failed over" to nothing. And when the cable returned, the shallow check answered successfully even while the Hub's database was down. The revised rules are:

1. **Proof before every switch, in both directions.** A switch requires a real handshake that touches the database and proves this lane is enrolled there. Leave the Hub only after the cloud has answered during this failure. Both unreachable is an explicit offline state: stay pointed at the Hub and re-probe with back-off.
2. **Hysteresis.** Return home only after repeated good handshakes spaced apart in time, and limit how often a lane may switch at all. The outbox is local, so waiting costs only latency, while flapping costs a cursor re-anchor per flap.
3. **A Hub that answers but can never catch up is a failure.** A Hub restored from an old backup or replaced by a fresh database answers every handshake, but if its acknowledged sequence is below the oldest document the lane still holds, the documents in between exist nowhere the lane can reach. This sequence gap opens the same proof-before-switch path as an outage, without waiting. The fallback counts as healthy only if it can *continue* the chain.
4. **Monotonic time.** Every duration is computed from a monotonic clock. A till clock corrected backwards during an outage would otherwise hold a dead Hub for longer.

6 The direction law

Between Hub and cloud, replication is a set of one-way streams, and nothing travels both ways. Upward streams carry everything the store *did*: sales, returns, shifts, cash, stock movements, stocktakes, attendance, order status and each lane's software version. Downward streams carry what the store *sells* and how: website orders, software releases, the catalog, product photos and settings. Stock, sales and shifts are the store's truth and are never written from above.

Settings are the single table holding both sides' truth: the office authors the shop's branding and business rules, while the Hub writes its own operational stamps and owns the lane update window. Settings are therefore split by ownership, and each side refuses entries the other does not own.

A field failure fixed the identity rule for this boundary. Hub and cloud had each minted internal barcodes from a local sequence, and the two databases produced the *same* barcode for *different* items, which wedged the catalog mirror. Similarly, price tiers seeded on both servers had different UUIDs. The rule adopted is that **an identifier minted from a local sequence is per-database vocabulary and must never be assumed unique across servers**; cross-server seams join on natural keys.

7 Store hardware, controlled from a phone

A supermarket is not only tills. Weighed goods need label scales whose prices match the till, and a 24-hour shop with dozens of staff on rotating shifts needs attendance that payroll can trust. The owner also needs to run all of it without standing in the store. This section describes how the system drives that hardware, and how the whole store is operated from a phone or tablet.

7.1 One pattern for every device

The back office never talks to a device. To change a scale or a fingerprint terminal it records a *job*. A small agent inside the store claims the job, drives the device, and writes the result back as a record. Scales and attendance terminals use the same shape.

The pattern has four consequences. The office can be anywhere, because only the agent needs to reach the device, and devices stay inside the store. Every change leaves an auditable result row, including failures, which the office shows to the operator rather than swallowing. And a device that is switched off simply leaves jobs queued, exactly as a till's outbox does during an outage.

7.2 Label scales

The store uses commercial label scales at several counters. From the back office the owner edits an item's price, name and shelf life, and assigns items to the scale's physical keypad in a staged editor. The scale follows within seconds. We verified the full loop end to end: a price and a name were edited on a tablet, the scale updated, a label was printed and the label was scanned at a till.

Scales and tills refresh on different schedules, so a freshly printed label's *name* can briefly run ahead of the till's catalog; the design ensures the amount charged is never affected by that window.

One result was a measured negative. The vendor documentation describes many device settings as

remotely configurable. On our hardware every such call failed, while a positive control on the same connection succeeded, so those settings must be set on the device itself. We record this because it is the kind of fact that vendor documentation does not state and that only the device can answer (principle P2, §8).

7.3 Fingerprint attendance

Attendance uses commercial fingerprint terminals with two roles: one in the office enrolls staff, one on the floor records arrivals and departures. Three hardware facts shaped the design.

The terminal sends no in/out flag. Each punch record carries only a person and a time, so each staff member's punches for a day are paired in order, and an odd count means an open session. The system never infers direction from a status field the hardware does not populate.

Staff are not logins. Every staff member has a name and a fingerprint on the terminal, but most never use the back office. Staff records are keyed on the terminal's own identity for the person, with an optional link to a login. A fingerprint the system does not yet know appears as an inbox card that already carries its banked punches; linking it to a person makes those earlier punches count immediately.

Fingerprint templates are irreplaceable. Losing a terminal would otherwise mean re-enrolling every employee. The agent backs up the terminal's enrolment data regularly, and we verified a byte-identical round trip on real hardware. The database refuses any delete on a terminal that has no backup, and the agent re-checks the terminal before acting.

Time records are evidence: the timesheet is editable only by void-and-supersede, never by deletion. Rotating 24-hour shifts use a per-person business date, so the morning after a night shift belongs to that night. Attendance is recorded only by a finger on a terminal, never by a button.

7.4 The store in a pocket

The back office is a responsive web application of about fifty screens: catalog and pricing, purchasing and receiving, stock, store map and shelf places, delivery, attendance and rota, scales, the till fleet, approvals, alerts and reports. Access is restricted to authorised staff. Owner-facing screens are accepted only after being driven at phone width (430 px) in Kurdish, a right-to-left script, as well as on tablets and desktops.

From a phone the owner can change a price that reaches the tills and the scales; enrol, link and schedule staff; work through approvals and alerts; and see each till's software version. Till updates are signed and rolled out gradually. A till applies an update only outside trading hours, with no open shift and an empty outbox, so a remote update can never interrupt a sale or strand a queued document.

The Hub also monitors and repairs its own services, so most faults recover without a site visit.

8 A catalogue of field failures

Before installation we cross-checked the codebase against documented failures of fielded POS systems (vendor post-mortems, issue trackers of open-source POS projects, loss-prevention literature and platform documentation), verifying each suspected gap against the repository with file and line evidence. The resulting gap list included missing catalog pull on the till, no remote update path, alerts that never reached a phone, a printer failure that silently kept the cash drawer shut, UPC-A/EAN-13 leading-zero lookup misses, scanner input that depended on the keyboard layout, half-configured SQLite durability, and long-run crashes of the desktop runtime. Benching and installation then added further entries. In total twenty failures were documented in the project's field-hardening log. Like studies of production failures in distributed systems [12, 13], most were not exotic algorithmic bugs but gaps between a correct component and the environment it ran in. We group them into six principles.

P1. Only the running process is evidence. A deploy script exited successfully while the service kept running the old version, because a configuration file recorded an intention that something else silently overrode. Every deploy now ends by asking the running system for its version and comparing it with the target.

P2. A fake proves the decision, never the lifecycle. A fingerprint "enrol once" feature passed every test against a fake client with no socket, and failed within minutes of the first device emulator that had one. Device code is gated on the device, in every state it can be put into (empty log, shared socket, garbage record).

P3. A business day is wrong until it names a timezone. Three separate defects computed the business day from the server's clock or the UTC calendar date: one turned the day at 08:00 local time instead of 05:00, and another wrote waste off to the

wrong date. Till-side local time is correct; every server-side boundary must state the store's offset explicitly.

P4. Silence is the failure mode. A single malformed document stalled a lane's upload with no visible symptom, because every ingest test asserted on documents the applier *could* apply. Poison documents are now quarantined and announced. More broadly, a critical alert that ends in a database row is not an alert: it must reach a phone.

P5. Code that runs is not code that is installed. Three of four background daemons had no way to be installed on the machine they were meant to run on. For every component we now ask: on the box it will actually run on, what starts it, what restarts it, and what tells us when it stops?

P6. "Deliberately not wired" is a bug report. The shop's branding was implemented in three finished parts with no connection between them, twice. When a value fans out to several surfaces and one endpoint is a person remembering to act, that endpoint is where the wrong number will live.

The Hub's own back office once failed to load because of a condition specific to that deployment. Machine surfaces were heavily tested because they are easy to script; the one surface that needed a human eye received none. User interfaces are now accepted only after being driven by a person, or a scripted browser, on the real deployment.

9 Engineering evidence

Table 2 summarises the repository as of 9 October 2026. Of 1,243 commits, 1,232 are by the author; the remainder are automated dependency updates.

Table 2. Repository metrics (main branch, 2026-10-09).

Metric	Value
Development period	2026-07-10 to 2026-10-09 (73 active days)
Commits / merged pull requests	1,243 / 172
SQL schema migrations	156
Test files (unit, integration, end-to-end, ops)	736
Test files using property-based testing	24
Lines: server (cloud + Hub)	135,392
Lines: back office	71,236
Lines: lane (till) application	31,662

Lines: shared packages	41,064
Lines: daemons (scale, attendance, labels)	10,004
Lines: operations and provisioning	20,748

Line counts cover TypeScript, TSX and SQL files only. Verification is layered. Six property-based suites (fast-check) express the invariants the books depend on: every ledger entry set balances per currency; invoice lines minus discounts plus rounding equal tenders; stock on hand equals the sum of movements; the audit chain verifies end to end; pushing the same document *N* times in any interleaving yields exactly one invoice, movement set and posting set; and ingest is order-insensitive across lanes. Concurrency tests run against a real PostgreSQL instance rather than mocks, including four lanes racing for the last unit of stock and a lost acknowledgement after commit. A "golden day" fixture scripts a full business day across four lanes, including a long offline window with an oversell, night-shift sales and per-currency closes, and requires reproducible end-of-day reports. Dedicated tests cover the failover function, the nightly integrity check, the lane's hash chain and a two-server failover drill.

Operationally, lanes update over the air with integrity checks and staged roll-out, and only under the safe conditions described in §7.4. All device-to-server traffic is authenticated and encrypted.

10 Limitations

This is an experience report from a single site and a single developer. At the time of writing the system is installed and benched with real hardware, but the store had not yet opened to customers, and all transaction data so far is test data. We therefore cannot yet report production outage frequency, outage durations, drain times under real load, or reconciliation-flag rates; these are the subject of planned follow-up work. The offline-stock design accepts oversell during outages as a deliberate trade-off; its real cost depends on outage length and product mix, which we have not measured. Finally, the failure catalogue reflects one team's blind spots; a different team would find a different twenty.

11 Related work

Disconnected operation has a long history in distributed systems: Coda allowed clients to keep working against cached files and reintegrate later [3], and Bayou

managed update conflicts in weakly connected replicated storage [4]. The local-first movement argues for software that keeps working and keeps users' data on their own devices [1], often using CRDTs [2]. Our design borrows the local-first stance for the checkout but deliberately rejects peer-to-peer merging: a shop's books need a single authority, and document-level idempotence [6] with accept-and-flag reconciliation turned out simpler to reason about than general merge semantics. Helland's argument for building on idempotent, entity-keyed messages rather than distributed transactions [5] and Bailis et al.'s coordination avoidance [7] describe the theory behind order-insensitive ingest. The outbox follows the transactional outbox pattern [11], and the hash chain follows tamper-evident logging [8]. Brewer et al. argued that technology for developing regions must be designed for intermittent power and connectivity rather than adapted afterwards [9]; this report is one data point in favour of that view for commercial software.

12 Conclusion

A supermarket in a region with unreliable power and connectivity can run modern, cloud-connected retail software if the network is removed from the checkout path and every hop to the books is idempotent, ordered where it must be, and loud when it fails. The mechanisms are not new individually; the contribution is their combination in a fielded system, the specific failover rules that field scenarios forced, and the catalogue of failures that lived between correct components and their environments. We intend to follow up with production measurements once the store has traded through its first months.

¹ The client is not named. No sales, customer, network or security details are disclosed; implementation specifics are summarised rather

than reproduced.

References

1. M. Kleppmann, A. Wiggins, P. van Hardenberg, M. McGranaghan. Local-first software: you own your data, in spite of the cloud. *Onward! 2019*, ACM.
2. M. Shapiro, N. Preguiça, C. Baquero, M. Zawirski. Conflict-free replicated data types. *SSS 2011*, LNCS 6976, Springer.
3. J. J. Kistler, M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems* 10(1), 1992.
4. D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, C. H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. *SOSP 1995*, ACM.
5. P. Helland. Life beyond distributed transactions: an apostate's opinion. *CIDR 2007*.
6. P. Helland. Idempotence is not a medical condition. *ACM Queue* 10(4), 2012.
7. P. Bailis, A. Fekete, M. J. Franklin, A. Ghodsi, J. M. Hellerstein, I. Stoica. Coordination avoidance in database systems. *PVLDB* 8(3), 2014.
8. S. A. Crosby, D. S. Wallach. Efficient data structures for tamper-evident logging. *USENIX Security 2009*.
9. E. Brewer, M. Demmer, B. Du, M. Ho, M. Kam, S. Nedeveschi, J. Pal, R. Patra, S. Surana, K. Fall. The case for technology in developing regions. *IEEE Computer* 38(6), 2005.
10. L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM* 21(7), 1978.
11. C. Richardson. *Microservices Patterns*. Manning, 2018.
12. D. Yuan, Y. Luo, X. Zhuang, G. R. Rodrigues, X. Zhao, Y. Zhang, P. U. Jain, M. Stumm. Simple testing can prevent most critical failures: an analysis of production failures in distributed data-intensive systems. *OSDI 2014*.
13. H. S. Gunawi, M. Hao, T. Leesatapornwongsa, T. Patanake, T. Do, J. Adityatama, K. J. Eliazar, A. Laksono, J. F. Lukman, V. Martin, A. D. Satria. What bugs live in the cloud? A study of 3000+ issues in cloud systems. *SoCC 2014*.