

Hydra Point-of-Sale (PoS) & Invoicing: Technical architecture and design

Purpose

This document describes the technical architecture for hydra point-of-sale and invoicing, including how the system processes payments using hydra (layer 2) with automatic fallback to cardano layer 1, and how payment state is observed, persisted, and surfaced to the merchant UI and external integrations.

The included diagrams are inspired by the concepts shown in your reference images (hydra head session + middleware + settlement), but adapted for a retail PoS + invoicing system.

Architecture goals

- Non-custodial by default (merchant controls keys and infrastructure).
 - Works in retail environments (fast checkout, clear payment status).
 - Supports ADA and CNTs.
 - Uses hydra for instant confirmation when feasible; falls back to layer 1 when hydra is unavailable or unsupported by the payer.
 - Self-hosted and open-source friendly (simple deployment, observable, debuggable).
 - Extensible via API + webhooks.
-

System overview

Primary actors

- **Merchant (cashier/owner)** using the pos web app

- **Customer** using a cardano wallet (qr scan + payment)
 - **Merchant backend** (api + payment orchestration)
 - **Hydra head** (optional, for instant settlement sessions)
 - **Layer-1 chain access**
-

Components

Client-side

PoS web app (merchant ui)

- cart, checkout, invoice creation, transaction list, exports
- shows qr (hydra or l1) and live payment status
- role-based screens (owner/manager/cashier/viewer)

Invoice portal (customer view, optional)

- a lightweight page opened from an invoice link that shows the qr + status
- can be used when invoice is remote (not in-store)

Customer wallet

- scans qr and pays on layer 1
- for hydra mode: may use a companion web flow or hydra-capable wallet tooling (depending on ecosystem support)

Server-side

Api service

- rest endpoints for invoices, payment requests, products, transactions
- auth, roles, rate limiting
- returns qr payloads and live status

Payment orchestrator (background worker)

- owns payment state machine for each payment request

- talks to:
 - **hydra connector** (hydra head api/websocket)
 - **layer-1 connector** (node/indexer/provider)
- emits events (webhooks, ui updates)

Webhook dispatcher

- signs and sends webhooks (invoice.paid, payment.confirmed, etc.)
- retries with idempotency and backoff

Reporting/export service

- exports csv/excel for transactions
- generates pdf receipts/invoices (server-side or client-side)

Database

- invoices, payment requests, payment attempts, rails used, tx references, audit log, users/roles

External dependencies

Hydra head

- optional runtime component for layer-2 sessions
- used for instant payments when both parties can participate

Cardano layer-1 access

- either:
 - self-hosted node + ogmios/kupo (recommended for full sovereignty), or
 - blockfrost or another provider (simpler to start)

Deployment options

Option A: Fully self-hosted

- docker compose:

- pos-ui (static)
- api
- worker
- db (postgres)
- cardano node + ogmios + kupo (or an indexer)
- hydra node/head (optional)

Option B: hybrid (fast start)

- self-host ui + api + db
 - use a third-party layer-1 provider for chain reads
 - add self-hosted node/indexer later
-

Payment rails and decision logic

Rail selection

- **hydra rail** when:
 - merchant has hydra configured and reachable, and
 - the checkout/invoice is flagged “hydra enabled”, and
 - customer can participate (or uses the supported hydra flow)
- **layer-1 rail** otherwise

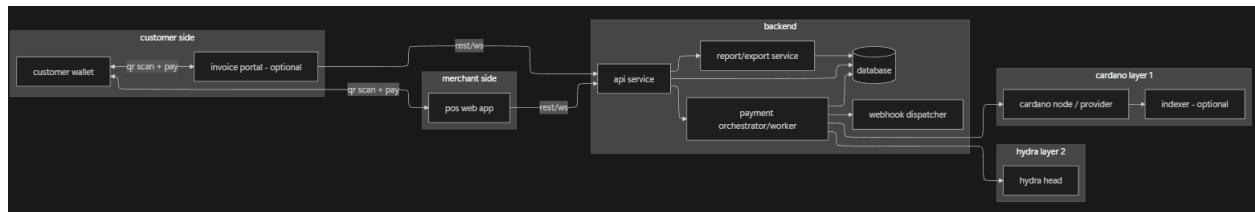
Fallback behavior

- If a payment request is created as hydra and the head becomes unavailable before payment:
 - pos displays an immediate “switch to layer 1” option
 - backend generates a layer-1 qr tied to the same invoice/order reference
 - first successful payment locks the request (prevents double-payment)
-

Data model

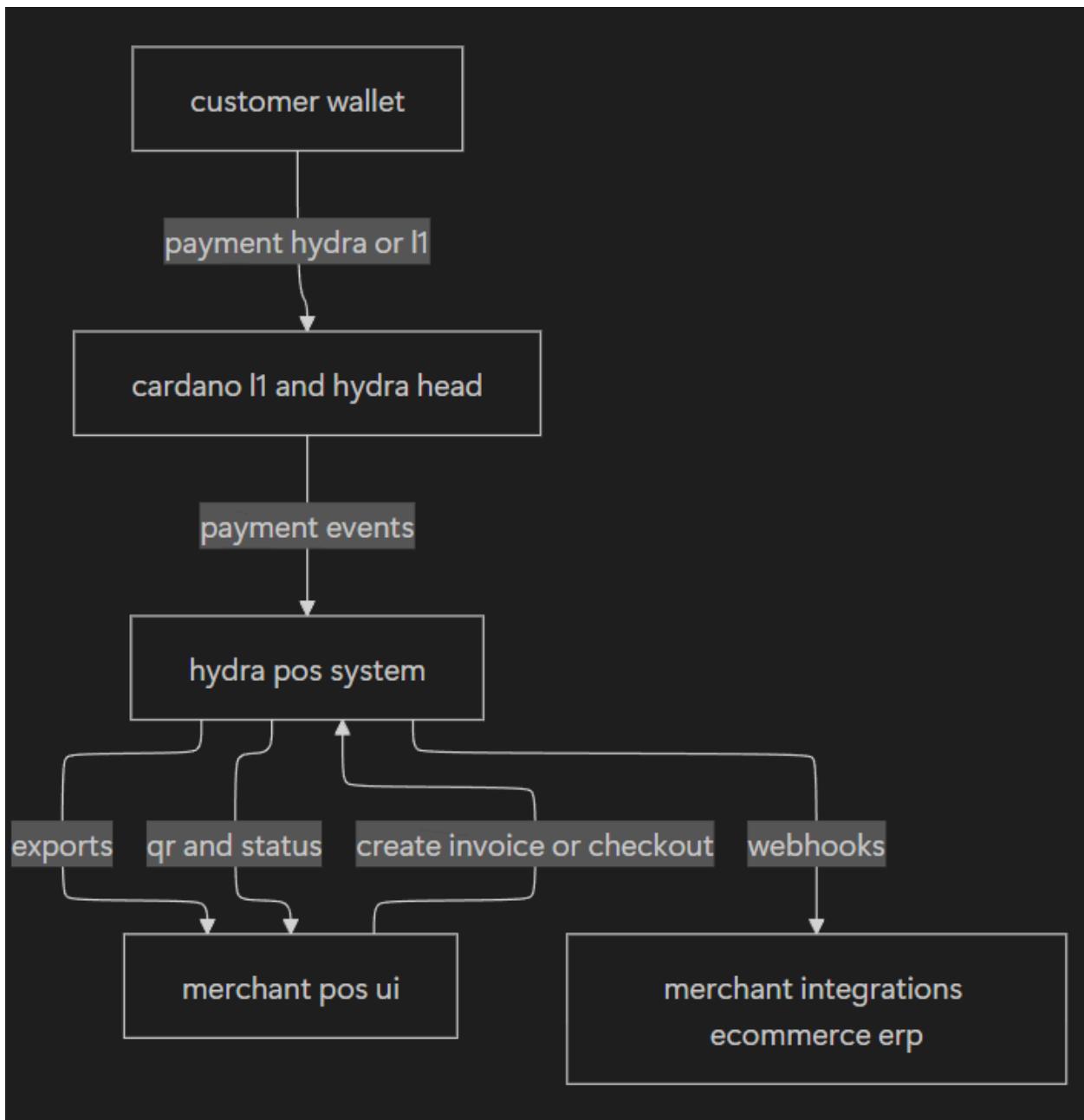
- **invoice**
 - id, reference, status, asset, amount, expiry, metadata, created_by
- **payment_request**
 - id, invoice_id (optional), order_id (optional), rail_preference, status, expiry
- **payment_attempt**
 - id, payment_request_id, rail (hydra/l1), status, identifiers (l1 tx hash or hydra tx id), observed_at
- **transaction**
 - normalized record for reporting (invoice ref, asset, amount, rail, timestamps)
- **audit_log**
 - user/action/object/time

Technical architecture diagram

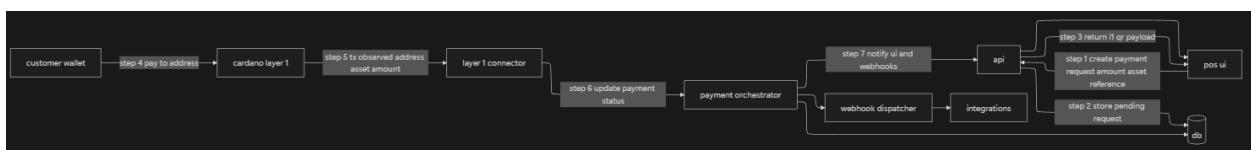


Data flow diagrams

DFD level 0 (context)



DFD level 1a: layer-1 checkout

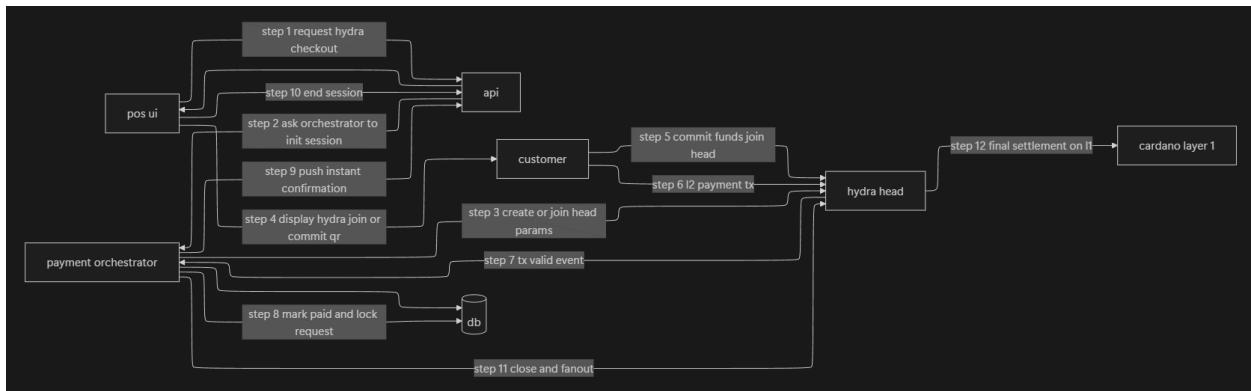


Key notes:

- the l1 connector can be implemented using a node+indexer or a provider.
- matching rules use address + asset + amount + time window (+ optional metadata if used).

DFD level 1b: hydra checkout

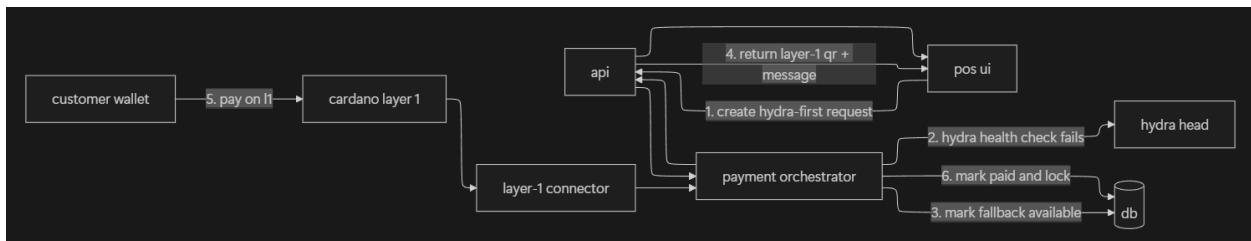
This is the pos adaptation of your reference flow (commit funds → transact inside head → close/fanout → settle on l1).



Practical behavior in retail:

- Hydra mode can be used as a “fast lane” when both sides can participate.
- Otherwise, the system immediately offers layer-1 qr (see fallback flow below).

DFD level 1c: hydra to layer-1 fallback

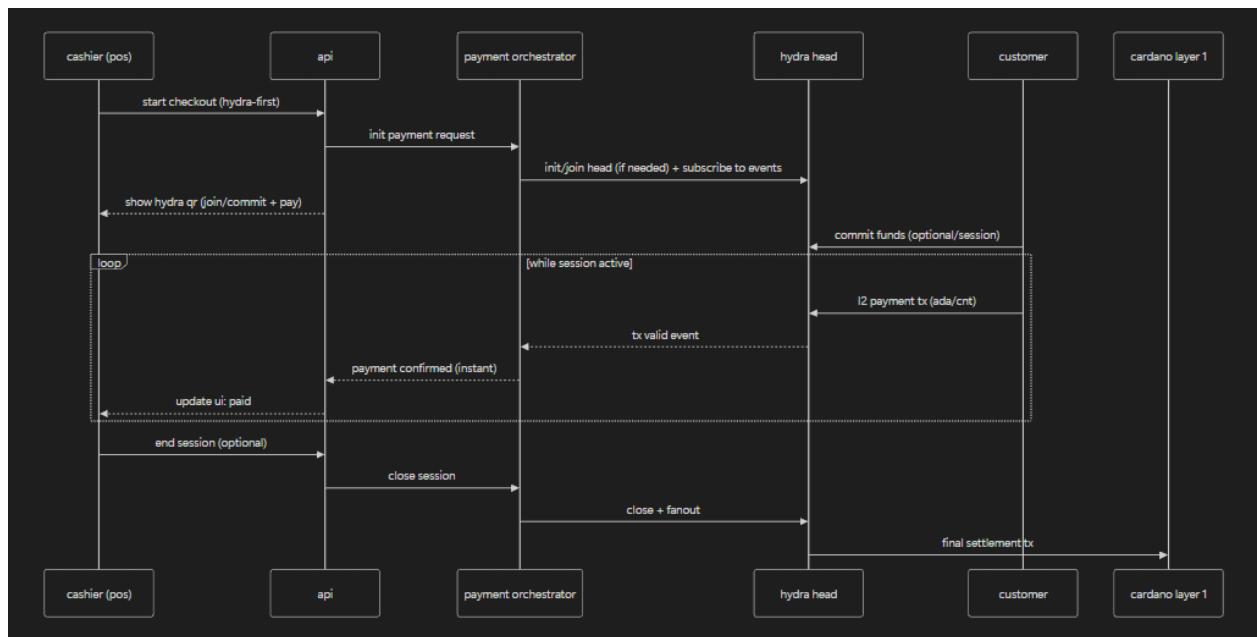


DFD level 1d: invoice flow (create → share → pay → export)

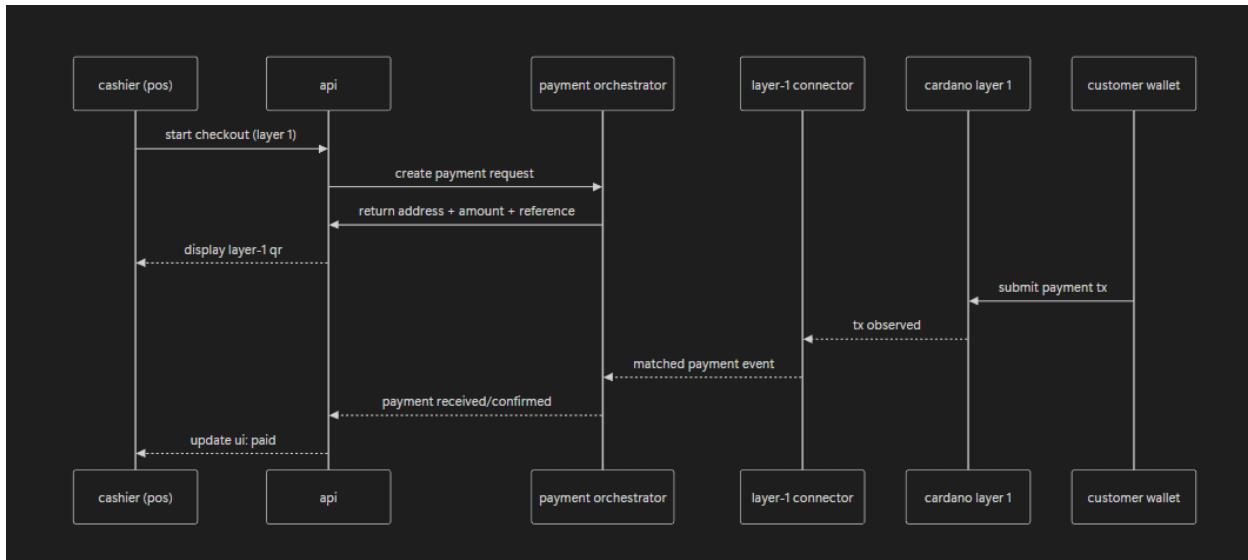


Sequence diagrams

Hydra checkout



Layer-1 checkout



Interfaces

Api surface (example)

- `POST /payments` create payment request (pos checkout or invoice)
- `GET /payments/{id}` fetch status + rail + tx refs
- `POST /invoices` create invoice
- `GET /invoices/{id}` invoice details + status
- `POST /webhooks` register webhook endpoint (admin)
- `GET /exports/transactions?from=&to=` generate export

Event model

- `payment.requested`
- `payment.rail_selected` (hydra/l1)
- `payment.received`
- `payment.confirmed`
- `payment.expired`
- `payment.fallback_available`

- `invoice.paid`
-

Operational considerations

Observability

- structured logs for payment lifecycle transitions (request id, invoice ref, rail, tx id)
- health endpoints:
 - hydra connectivity
 - layer-1 connectivity
 - db connectivity
- metrics (optional): payment confirmation latency by rail, success rate, fallback rate

Security

- secrets encrypted at rest
- webhook signing with shared secret
- role-based access for settings
- idempotency keys on webhook retries and payment status updates

Edge cases

- partial/incorrect payments (mark as exception and require manual handling)
- late payment after expiry (configurable: accept and mark late, or ignore and flag)
- double payment attempts (first success locks; later events flagged)