

TECHNICAL SPECIFICATION

Version 1.0 | Provider-neutral | Sandbox-first

Offer -> Order -> Payment intent -> Provider response -> Verification ->
Idempotent event -> Ledger -> Receipt -> Fulfilment -> Reconciliation

1. Purpose and scope

The Revenue Rail is a reference architecture and training capability. It is not a bank, wallet, payment provider, remittance service or regulatory approval.

2. Functional requirements

- Create a persistent internal order before requesting payment.
- Generate a provider payment request or checkout session.
- Receive a webhook/callback or poll the provider status.
- Verify event authenticity using the provider-approved method.
- Process each provider event idempotently.
- Map provider status to an internal state.
- Write an immutable transaction event linked to the order.
- Issue a receipt only after confirmed status.
- Record refund requests and outcomes.
- Export a reconciliation dataset.

3. Canonical states

Object	Canonical states
Order	CREATED PAYMENT_PENDING PAID FULFILMENT_PENDING FULFILLED CANCELLED REFUND_PENDING REFUNDED
Payment	INITIATED PENDING SUCCEEDED FAILED CANCELLED REVERSED
Fulfilment	NOT_STARTED IN_PROGRESS DELIVERED FAILED

4. Minimum data model

Offer	offer_id, name, unit, currency, unit_price, active
Order	order_id, offer_id, customer_alias, expected_amount, currency, status, created_at
Payment attempt	attempt_id, order_id, provider, provider_reference, confirmed_amount, status, initiated_at, confirmed_at
Ledger event	event_id, order_id, attempt_id, event_type, amount, currency, occurred_at, source
Receipt	receipt_id, order_id, provider_reference, amount, currency, issued_at
Refund	refund_id, order_id, reason, amount, status, authorised_by, created_at

5. Minimum API contract

- POST /api/orders - create an order and return order_id.
- POST /api/orders/{order_id}/payment-intents - initialise provider request.
- POST /api/payments/webhook - receive and verify provider event.
- GET /api/orders/{order_id} - return order, payment and fulfilment status.
- POST /api/refunds - create a human-authorised refund request.
- GET /api/reconciliation?date=YYYY-MM-DD - export settlement data.

6. Webhook and event-processing rules

- Read the raw request body where signature verification requires it.
- Reject unverifiable events.
- Store the provider event identifier before business processing.
- Acknowledge duplicates without duplicating ledger entries or fulfilment.
- Verify internal order reference, amount and currency.
- Update status transactionally.
- Create exactly one ledger event for each accepted financial event.
- Trigger fulfilment only from a confirmed internal state.
- Log failures without logging secrets or sensitive payment data.

7. Security baseline

- Protected environment variables.
- Separate sandbox and production credentials.
- Least-privilege access.
- HTTPS transport.
- Restricted administration and reconciliation endpoints.
- Redacted logs and screenshots.
- Immediate credential rotation after exposure.
- Recorded dependency and deployment versions.

8. Mandatory failure tests

- Successful, failed and pending payment.
- Duplicate and delayed callback.
- Wrong amount, wrong currency and unknown order.
- User cancellation and refund request.
- Provider unavailable and ledger mismatch.
- Secret accidentally committed to a test repository.

9. Reconciliation export

- order_id and provider_reference
- expected_amount and confirmed_amount
- currency
- payment_status and provider_status
- ledger_total and variance
- test_or_live classification

10. Definition of done

- One offer creates one persistent order.
- One payment attempt traces to one order.
- A verified provider event changes state once.
- A duplicate event cannot duplicate fulfilment.
- A receipt is generated only after confirmation.
- The daily export matches provider evidence.
- Failure states are visible to users.
- No secrets are present in source control.

Customer selects an offer -> system creates an order -> payment is requested -> provider status is verified -> ledger records once -> receipt is issued -> team reconciles.

TECHNICAL WORKSTREAM

This page authorises reuse of the Revenue Rail architecture, security baseline, failure tests and definition of done.

Decision required: APPROVE, APPROVE WITH CHANGES, or HOLD before operational release.

Department and workstream reviewers

- Lolwe AI Technical Lead
- Security / DevOps Lead
- Finance / Revenue Operations Lead
- GKHack26 Kenyan Node Lead
- Payment-provider technical reviewer, where available

What this is and how to use it

- Use sections 1-5 to build the minimum integration.
- Run every failure test before any live-value test.
- Use the reconciliation export as the daily close input.
- Document all provider-specific deviations.

Approval checks

- Sandbox and production are separated.
- Authenticity verification follows provider documentation.
- Idempotency prevents duplicate processing.
- No unsupported regulatory or provider claims are made.
- Data minimisation and redaction are enforced.
- A named human owns refunds and incident escalation.

Lead	Decision	Date
Name: _____	Approve / Amend / Hold	____ / ____ / 2026
Signature: _____	Conditions: _____	Version 1.0