BUSINESS ANALYSIS DOCUMENT

Payment API — Refund Functionality

Version 1.0 | August 2026

FIELD	VALUE
Project	Payment API
Feature	**Refund Functionality**
Document Type	**Business Analysis**
Author	Senior Business Analyst: **Tinatini Papashvili**
Status	Draft for Review
Date	August 2026
Version	**1.0**
Audience	Product Owner · Developers · QA · Architecture · Finance · Compliance

01 Executive Summary

Business Problem

The existing Payment API enables merchants to capture payments from customers but provides no programmatic mechanism for reversing those payments. When a refund is required — due to a product return, order cancellation, billing error, or customer dispute — merchants must rely on manual, error-prone processes: raising support tickets, contacting operations teams, or logging directly into a payment provider portal. This creates unacceptable delays, increases customer dissatisfaction, elevates chargeback risk, and imposes significant operational overhead on internal teams.

Proposed Solution

Introduce a Refund API as an extension of the existing Payment API that allows authorized merchants to programmatically initiate both full and partial refunds against captured payment transactions. The solution includes refund validation, idempotency protection, real-time status tracking, provider integration, webhook notifications, and a complete audit trail.

Business Value

- Merchant self-service: eliminates manual support-channel dependency for routine refunds
- Reduced chargebacks: fast refund processing reduces the likelihood of customers disputing charges through their bank
- Operational efficiency: Finance and ops teams no longer manage individual refund requests manually
- Competitive parity: standard capability expected by any serious merchant integration
- Revenue protection: proper refund management reduces reconciliation discrepancies and financial risk

Primary Users & Stakeholders

- Merchants — primary API consumers initiating refund requests
- End customers — recipients of refunded funds
- Finance / Reconciliation teams — consume refund data for accounting
- Payment Operations — monitor refund flows and handle exceptions
- Payment Provider / Acquirer — external party that processes the actual fund reversal

Expected Outcome

Merchants can initiate a full or partial refund against any eligible captured payment through a single authenticated API call. The system validates the request, communicates with the payment provider, updates transaction state, and notifies the merchant of the outcome via webhook — all within seconds for synchronous outcomes, with reliable async handling for delayed provider responses. The feature is expected to reduce refund processing time from hours/days to under 60 seconds for the majority of cases.

02 Assumptions

> **Note:** The following are working assumptions made in the absence of confirmed requirements. Each must be validated with relevant stakeholders before development begins.

ID	ASSUMPTION	RATIONALE	OWNER
A-01	The existing Payment API has a concept of a CAPTURED payment status representing a successfully completed, settled payment eligible for refund.	Standard payment lifecycle; refunds require a captured/settled base transaction.	Dev Lead
A-02	Each payment has a unique, stable paymentId that persists for its entire lifecycle.	Required for refund reference and idempotency.	Dev Lead
A-03	At least one payment provider or acquirer integration is already in place and supports refund operations via API.	Without provider refund capability, the feature cannot be implemented.	Architecture / Payment Ops
A-04	Refunds are returned to the original payment method only. Cross-method refunds are not in scope.	Industry standard for v1; alternative refund destinations add regulatory and operational complexity.	Product / Compliance
A-05	The company operates under at least PCI DSS compliance obligations due to handling of card payment data.	Reasonable assumption for any card payment business.	Compliance / Security
A-06	Merchants authenticate to the Payment API using API keys or OAuth 2.0 bearer tokens.	Common API authentication patterns; assumed consistent with existing auth.	Dev Lead / Security
A-07	The payment provider may return refund results asynchronously; the API must handle PENDING refund states.	Many acquirers do not confirm refunds synchronously.	Architecture
A-08	A refund window policy exists (e.g., refunds only permitted within 180 days of original payment). Exact window is an open question.	Most payment providers and processors impose time limits on refunds.	Payment Ops / Legal
A-09	The payment currency and refund currency must match (single-currency refunds only for v1).	Multi-currency refunds add FX complexity out of scope for MVP.	Finance / Product
A-10	The existing system has a relational database (e.g., PostgreSQL) that can be extended with new tables for refund data.	Standard assumption for a payment backend.	Dev Lead / DBA
A-11	A webhook delivery infrastructure either already exists or can be built as part of this project.	Real-time notifications are required for merchant integration.	Dev Lead
A-12	Merchant-level refund permissions can be configured (some merchants may be restricted from initiating refunds above a threshold).	Risk management requirement; assumed configurable per merchant.	Product / Risk

03 Feasibility Analysis

Business Feasibility

Customer/Merchant Need: Refund capability is a baseline expectation for any payment integration. Merchants cannot operate e-commerce, subscription, or service businesses without a reliable refund mechanism. Its absence creates direct customer service failures and competitive disadvantage.

Business Value: The feature directly reduces chargeback rates (chargebacks typically carry $15–$100 fees per occurrence plus chargeback ratio penalties), reduces ops team labor, and improves merchant retention. ROI is measurable within the first quarter of operation.

Operational Impact: Payment Ops and Finance teams will require updated workflows, monitoring dashboards, and reconciliation processes. A moderate one-time training and process-update effort is expected, with long-term operational savings.

Technical Feasibility

AREA	ASSESSMENT	RISK
Existing API architecture	Adding a refund endpoint follows the same patterns as the existing Payment API. Feasible with standard REST extension.	Low
Payment transaction identification	Assumes paymentId already exists and is indexed. No architectural change required.	Low
Provider/acquirer integration	Requires confirmed provider refund API support. Most major processors support it. Must verify provider-specific behavior.	Medium
Full & partial refunds	Standard implementation; requires tracking of refunded amount against original amount.	Low–Medium
Multiple partial refunds	Requires a refunds table linked to payments; sum of refunds must not exceed original amount. Concurrency control required.	Medium
Transaction state management	Payment status transitions (CAPTURED → PARTIALLY_REFUNDED → FULLY_REFUNDED) must be implemented cleanly.	Medium
Idempotency	Standard pattern using Idempotency-Key header and database deduplication. Well-understood implementation.	Low
Webhooks	If webhook infrastructure exists, extension is straightforward. If not, moderate build effort required.	Medium
Database changes	New refunds table + payment status columns. Backward-compatible migrations.	Low
Security	Existing auth can be extended. Refund-specific authorization rules require additional logic.	Low–Medium

Operational Feasibility

- Customer Support: ticket volume for manual refund requests expected to drop significantly
- Finance: reconciliation processes must be updated to consume refund events
- Merchant Operations: merchants gain self-service capability; documentation and sandbox testing required
- Reporting: refund metrics should be added to operational dashboards

Risks and Constraints

RISK	CATEGORY	SEVERITY
Payment provider does not support partial refunds via API	Technical	High
Race conditions when multiple partial refunds are submitted simultaneously	Technical	High
Provider processes refund but API receives timeout — double refund risk	Technical / Business	Critical
Unauthorized merchants exploiting refund endpoint	Security	High
Reconciliation mismatches between API state and provider state	Business	High
No confirmed refund window policy — regulatory risk	Compliance	Medium

> **Recommendation: FEASIBLE WITH CONDITIONS** The Refund Functionality is technically and commercially feasible. Implementation should proceed after confirming: (1) payment provider refund API capabilities, (2) the refund window policy, (3) merchant authorization model, and (4) idempotency and concurrency strategy. A phased approach starting with full refunds (MVP) before partial refunds is strongly recommended.

04 Project Scope

In Scope

FEATURE	DESCRIPTION	MVP?
Full Refund	Refund 100% of the original captured payment amount	✓ Yes
Partial Refund	Refund a portion of the original payment amount	✓ Yes
Multiple Partial Refunds	Allow successive partial refunds up to the captured amount	✓ Yes
POST /refunds endpoint	API endpoint to create a refund against a payment	✓ Yes
GET /refunds/{refundId}	Retrieve a specific refund by ID	✓ Yes
GET /payments/{id}/refunds	List all refunds for a payment	✓ Yes
Refund validation	Validate eligibility, amount, currency, status	✓ Yes
Refund status lifecycle	CREATED → PENDING → PROCESSING → SUCCEEDED / FAILED	✓ Yes
Idempotency	Idempotency-Key header support to prevent duplicate refunds	✓ Yes
Refund history	Persistent record of all refund attempts per payment	✓ Yes
Webhook notification	Push refund status events to merchant webhook URL	✓ Yes
Error catalogue	Standardized error codes and messages for all failure scenarios	✓ Yes
Audit trail	Log all refund actions with actor, timestamp, and before/after state	✓ Yes
Provider integration	Integration with payment provider refund API	✓ Yes
Refundable balance tracking	Track how much of a payment remains refundable	✓ Yes

Out of Scope (v1)

- Chargeback and dispute management
- Refunds to a different payment method than the original (e.g., card → bank transfer)
- Manual / offline bank transfer refunds
- Multi-currency refunds (FX conversion)
- Advanced merchant analytics dashboard for refund trends
- Automated refund policy engine (rule-based auto-approval)
- Merchant-facing UI (scope limited to API; portal UI is a separate project)
- Bulk refund operations (refunding multiple payments in one call)
- Void / pre-auth reversal (separate from post-capture refund)

Dependencies

DEPENDENCY	TYPE	RISK IF UNAVAILABLE
Payment Provider refund API	External / Critical	Feature blocked entirely
Internal Transaction Service	Internal	Cannot validate original payment
Auth & API Key Management Service	Internal	Cannot authenticate/authorize refund requests
Database (relational)	Infrastructure	Cannot persist refund records
Webhook / Notification Service	Internal (build if absent)	No async merchant notification

DEPENDENCY	TYPE	RISK IF UNAVAILABLE
Reconciliation System	Internal	Financial reporting gaps
Audit Log Service	Internal	Compliance and traceability gaps

05 Stakeholder Analysis

STAKEHOLDER	ROLE	INTEREST	INFLUENCE	ENGAGEMENT STRATEGY
Merchant	Primary API consumer	Self-service refunds, fast processing, clear errors	High	API documentation, sandbox, integration guide
End Customer	Refund recipient	Fast, reliable refund to original method	Low (indirect)	Ensure SLA meets customer expectations
Product Manager	Feature owner	Scope, timeline, business metrics	High	Weekly status; scope decisions
Payment Operations	Exception handler, monitor	Visibility into refund flows, minimal manual intervention	Medium	Dashboard requirements, escalation flows
Finance / Reconciliation	Financial integrity	Accurate refund data in settlement reports	Medium	Data model review, reporting requirements
Development Team	Implementers	Clear requirements, API contracts, edge case coverage	High	Refinement sessions, API design review
QA Engineering	Quality assurance	Testable acceptance criteria, sandbox provider	Medium	Test scenario review, provider sandbox access
Security / InfoSec	Risk management	Auth controls, data protection, audit logs	High	Security review of API design and auth model
Compliance / Legal	Regulatory oversight	Refund window policy, PCI DSS, data retention	Medium–High	Policy sign-off before development starts
Payment Provider	External processor	Correct API usage, provider SLA adherence	High (external)	Technical integration documentation, sandbox testing

RACI Matrix — Key Refund Activities

ACTIVITY	PRODUCT	DEV	QA	SECURITY	FINANCE	PAY OPS	COMPLIANCE
Define refund business rules	A	C	I	C	C	C	R
API contract design	C	R/A	C	C	I	I	I
Provider integration	I	R/A	C	C	I	C	I
Security review	I	C	C	R/A	I	I	C
Reconciliation design	C	C	I	I	R/A	C	I
UAT sign-off	A	C	R	C	C	C	C
Production release	A	R	C	C	I	I	I

R = Responsible, A = Accountable, C = Consulted, I = Informed

06 AS-IS State

Current Payment Lifecycle

The existing system supports the following payment states: INITIATED → PROCESSING → CAPTURED (success) or FAILED / DECLINED. Once a payment reaches CAPTURED status, it is considered final from the API's perspective. No subsequent state transitions are defined in the current system.

Current Refund Handling — Manual Process

1. Customer contacts merchant's customer service to request a refund
2. Merchant logs a support ticket or emails the company's Payment Operations team
3. Payment Operations reviews the request manually, verifies the original transaction in the payment provider portal
4. If approved, Payment Ops manually initiates a refund in the payment provider's web portal
5. Provider processes the refund (1–5 business days)
6. Payment Ops manually updates internal records and notifies the merchant by email
7. Finance reconciles the refund manually in accounting systems

Current Pain Points

- Average refund turnaround: 2–5 business days (assumption)
- High manual effort: estimated 20–30 minutes of ops time per refund
- No API-level record of refunds — reconciliation relies on portal exports
- No idempotency controls — duplicate refunds possible through human error
- No programmatic merchant notification — email-based, inconsistent
- Refund status is opaque to merchants; requires support follow-up
- Partial refunds are particularly difficult to track manually
- High chargeback risk due to slow processing

07 TO-BE State

Future Refund Flow Overview

After implementation, the refund lifecycle is fully automated and API-driven. The merchant initiates a refund via a single authenticated API call. The system performs synchronous validation, communicates with the payment provider, updates state, and delivers the result both synchronously (API response) and asynchronously (webhook). The entire happy-path flow completes in under 10 seconds for synchronous provider responses.

Process: Merchant → Payment API → Validation → Provider → DB → Webhook → Merchant

Refund Initiation

Merchant sends POST /v1/payments/{paymentId}/refunds with amount, currency, reason, and an Idempotency-Key header. The request is authenticated via API key / bearer token.

Validation

The system synchronously validates: (1) payment exists and belongs to merchant, (2) payment status is CAPTURED or PARTIALLY_REFUNDED, (3) refund amount ≤ remaining refundable balance, (4) currency matches original, (5) refund window not expired, (6) idempotency key is not a duplicate.

Processing

If validation passes, a refund record is created with status PENDING and a unique refundId. The system forwards the refund request to the payment provider. If the provider responds synchronously, status transitions to SUCCEEDED or FAILED. If the provider responds asynchronously, status remains PENDING until a provider webhook is received.

Status Outcomes

- SUCCEEDED: Provider confirms refund. Payment status updated to PARTIALLY_REFUNDED or FULLY_REFUNDED. Merchant webhook sent.
- FAILED: Provider rejects refund. Refund status set to FAILED with reason. Merchant webhook sent.
- PENDING: Provider has accepted but not yet confirmed. System waits for async callback. Merchant notified via webhook upon resolution.

Partial & Multiple Partial Refunds

Merchant may call the endpoint multiple times with partial amounts. Each creates a new refund record. The system tracks cumulative refunded amount and enforces that no further refunds exceed the original payment amount. The payment moves from CAPTURED → PARTIALLY_REFUNDED after the first partial success, and → FULLY_REFUNDED once the sum of succeeded refunds equals the original amount.

08 Gap Analysis

AREA	AS-IS	TO-BE	GAP	REQUIRED CHANGE
API	No refund endpoint exists	POST + GET refund endpoints	Complete absence of refund API surface	Build POST /v1/payments/{id}/refunds, GET /v1/refunds/{id}, GET /v1/payments/{id}/refunds
Business Logic	No programmatic refund logic	Full validation, amount tracking, status transitions	Entire refund engine missing	Build refund service: eligibility check, amount guard, status machine, partial refund accumulator
Data Model	No refunds table; no refund fields on payments	Dedicated refunds table + refundable_amount and status extensions	Schema does not support refunds	Add refunds table; add refunded_amount, refundable_amount to payments; add new statuses
Transaction Statuses	INITIATED, PROCESSING, CAPTURED, FAILED, DECLINED	Add PARTIALLY_REFUNDED, FULLY_REFUNDED + full refund status lifecycle	Missing 2 payment statuses + 6 refund statuses	Extend payment status enum; create refund status state machine
Security	Auth covers payment creation only	Refund authorization (merchant owns payment + has refund permission)	No refund-specific authorization layer	Add merchant ownership validation; add refund permission flag per merchant
Idempotency	Unknown if implemented for payments	Idempotency-Key required for refunds	No idempotency control on refund path	Implement idempotency key storage and lookup on refund creation
Provider Integration	Capture integration exists; no refund integration	Provider refund API call with async callback handling	Refund leg of provider integration missing	Implement provider refund API call; implement provider webhook handler
Merchant Experience	Manual, email-based, 2–5 days	API-driven, seconds for sync, minutes–hours for async	Entire merchant refund experience	API, webhooks, error codes, integration documentation
Notifications	Manual email from ops team	Programmatic webhook on refund status change	No webhook event for refund outcomes	Add refund.succeeded, refund.failed, refund.pending webhook event types
Reporting	Manual portal exports; no structured refund data	Queryable refund records with API + reporting integration	No refund data in internal reporting systems	Expose refund events to data warehouse / reconciliation system
Reconciliation	Manual matching of portal data	Automated refund event feed to reconciliation system	No programmatic reconciliation for refunds	Emit refund events to reconciliation pipeline
Error Handling	No standardized refund error codes	Structured error catalogue with machine-readable codes	No error standards for refund domain	Define and implement refund error catalogue
Audit Logging	No audit log for refund actions	Immutable audit log: who requested, when, what changed	Zero audit trail for refunds	Integrate refund events into audit log service; log all state transitions

09 Functional Requirements

FR-001 Create Full Refund | Must Have

The system shall allow an authorized merchant to create a full refund for an eligible CAPTURED payment by submitting a refund request equal to the full original payment amount. The system shall create a refund record, initiate the refund with the payment provider, and return a refund object including a unique refundId and current status.

FR-002 Create Partial Refund | Must Have

The system shall allow an authorized merchant to create a partial refund for an eligible payment by specifying an amount less than the total refundable balance. The refund amount must be greater than zero and must not exceed the remaining refundable amount (original amount minus all previously succeeded refunds).

FR-003 Support Multiple Partial Refunds | Must Have

The system shall allow multiple successive partial refund requests against the same payment, provided the cumulative refunded amount does not exceed the original captured amount. Each partial refund shall be a separate, independent refund record with its own refundId and status lifecycle.

FR-004 Validate Original Transaction Existence | Must Have

Before creating a refund, the system shall verify that a payment record exists for the given paymentId. If no payment is found, the system shall return a 404 Not Found error with error code PAYMENT_NOT_FOUND.

FR-005 Validate Payment Eligibility for Refund | Must Have

The system shall verify that the payment status is CAPTURED or PARTIALLY_REFUNDED before permitting a refund. Payments in status INITIATED, PROCESSING, FAILED, DECLINED, or FULLY_REFUNDED shall be rejected with error code PAYMENT_NOT_REFUNDABLE.

FR-006 Validate Refund Amount | Must Have

The system shall validate that: (a) the refund amount is a positive number greater than zero; (b) the refund amount does not exceed the remaining refundable balance; (c) the refund amount uses the same currency as the original payment. Violations shall return error code INVALID_REFUND_AMOUNT or REFUND_EXCEEDS_BALANCE.

FR-007 Prevent Refund Above Refundable Amount | Must Have

The system shall maintain an accurate refundable_amount field on the payment record. Any refund request that would result in a negative refundable_amount shall be rejected. This check shall be performed atomically to prevent race conditions from concurrent requests.

FR-008 Generate Unique Refund ID | Must Have

The system shall generate a globally unique, non-sequential refund identifier (e.g., ref_<uuid>) for each new refund record. This ID shall be returned in the API response and used in all subsequent operations, webhooks, and audit logs.

FR-009 Maintain Payment–Refund Relationship | Must Have

The system shall persist a foreign-key relationship between each refund record and its originating payment record. This relationship shall be queryable via the GET /v1/payments/{paymentId}/refunds endpoint.

FR-010 Refund Status Lifecycle | Must Have

The system shall support the following refund statuses and transitions: CREATED → PENDING → PROCESSING → SUCCEEDED or FAILED. Only valid transitions shall be permitted; invalid transitions shall raise an internal error and be logged.

FR-011 Update Payment Status on Refund Success | Must Have

When a refund reaches SUCCEEDED status, the system shall automatically update the parent payment status: if refundable_amount reaches zero, the payment status shall be updated to FULLY_REFUNDED; otherwise it shall be set to PARTIALLY_REFUNDED.

FR-012 Idempotency Support | Must Have

The system shall accept an optional Idempotency-Key header on refund creation requests. If a request is received with an Idempotency-Key that matches a previously processed request for the same merchant, the system shall return the original

response (HTTP 200 with the existing refund object) without creating a duplicate refund. Idempotency keys shall be stored for a minimum of 24 hours.

FR-013 Authentication | Must Have

All refund API endpoints shall require authentication via the existing API key or OAuth 2.0 bearer token mechanism. Unauthenticated requests shall return HTTP 401 with error code UNAUTHORIZED.

FR-014 Authorization — Merchant Ownership | Must Have

The system shall verify that the authenticated merchant is the owner of the payment referenced in the refund request. A merchant shall not be permitted to refund payments belonging to another merchant. Violations shall return HTTP 403 with error code FORBIDDEN.

FR-015 Webhook Notification on Refund Status Change | Must Have

The system shall send a webhook event to the merchant's configured webhook endpoint upon refund status transitions to SUCCEEDED, FAILED, or PENDING. The webhook payload shall include: refundId, paymentId, status, amount, currency, reason, and timestamp. Failed webhook deliveries shall be retried with exponential backoff for a minimum of 72 hours.

FR-016 Standardized Error Responses | Must Have

The system shall return machine-readable error responses for all failure cases. Each error response shall include: HTTP status code, error code (string), human-readable message, and request_id for tracing.

FR-017 Retrieve Refund by ID | Must Have

The system shall expose a GET /v1/refunds/{refundId} endpoint that returns the full refund object for a valid refundId belonging to the authenticated merchant.

FR-018 List Refunds for a Payment | Must Have

The system shall expose a GET /v1/payments/{paymentId}/refunds endpoint that returns a paginated list of all refund records associated with a given payment.

FR-019 Audit Logging | Must Have

The system shall record an immutable audit log entry for every significant refund event including: refund creation, status changes, provider responses, webhook delivery attempts. Each entry shall include: actor, timestamp (UTC), action type, refund ID, previous state, new state, and IP address.

FR-020 Refund Window Enforcement | Should Have

[Assumption A-08] The system shall enforce a configurable refund eligibility window (e.g., 180 days from payment capture date). Refund requests against payments older than the configured window shall be rejected with error code REFUND_WINDOW_EXPIRED. The window value shall be configurable without code changes.

FR-021 Handle Asynchronous Provider Responses | Must Have

The system shall handle payment providers that do not confirm refund outcomes synchronously. In such cases, the refund shall be stored in PENDING status and the API shall return 202 Accepted. The system shall process the provider's callback when received, update the refund status accordingly, and deliver the result to the merchant via webhook.

10 Non-Functional Requirements

Performance

ID	REQUIREMENT	TARGET
NFR-P01	Refund creation API response time (excl. provider time)	p95 ≤ 500ms; p99 ≤ 1,000ms
NFR-P02	Refund retrieval API response time	p95 ≤ 200ms
NFR-P03	Concurrent refund request throughput	≥ 500 refund requests/second at peak
NFR-P04	Idempotency key lookup latency	≤ 50ms (in-memory cache layer)

Availability

ID	REQUIREMENT	TARGET
NFR-A01	Refund API availability (monthly uptime SLA)	≥ 99.9% (≤ 43.8 min downtime/month)
NFR-A02	Read endpoints always available during planned maintenance	Read endpoints always available
NFR-A03	Provider unavailability shall not cause data loss	Zero refund records lost during provider outage

Security

ID	REQUIREMENT	TARGET
NFR-SEC01	All API communication shall use TLS 1.2 or higher	TLS 1.2 minimum; TLS 1.3 preferred
NFR-SEC02	API keys and tokens must be validated on every request	Stateless auth per request
NFR-SEC03	Sensitive payment data (card numbers) shall never be logged	Zero instances of PAN in logs (validated by security scan)
NFR-SEC04	Refund amount fields shall be validated server-side only	Server-side validation only
NFR-SEC05	Audit logs shall be immutable and tamper-evident	Append-only audit log with integrity checks
NFR-SEC06	Rate limiting applied per merchant	Default ≤ 100 refund creation requests/minute/merchant

Reliability & Idempotency

ID	REQUIREMENT	TARGET
NFR-R01	Idempotency keys shall guarantee exactly-once refund creation semantics	No duplicate refunds for same Idempotency-Key within 24 hours
NFR-R02	Provider timeout shall not cause phantom refunds	Zero phantom refunds from unconfirmed provider responses
NFR-R03	Failed webhook deliveries shall be retried with exponential backoff	Retry for ≥ 72 hours; final failure logged and alertable
NFR-R04	Refund creation and payment status update shall be atomic	No partial state — refund record exists ↔ payment reflects it

Observability

ID	REQUIREMENT	TARGET
NFR-O01	Every API request shall generate a unique trace ID	100% of requests traceable
NFR-O02	Metrics shall be emitted for refund success/failure/pending rates, provider latency, webhook delivery	Metrics available in monitoring system within 60s of occurrence
NFR-O03	Alerts shall fire when refund failure rate exceeds 5% over 5-minute window	Alert latency ≤ 2 minutes

Maintainability & Compliance

ID	REQUIREMENT
NFR-M01	Refund API shall be versioned (/v1/) to allow non-breaking evolution
NFR-M02	OpenAPI specification (OAS 3.x) shall be maintained and always in sync with implementation
NFR-C01	System should operate consistently with PCI DSS requirements — no storage of full PANs
NFR-C02	Audit logs retained for minimum period consistent with financial record-keeping requirements (assumed ≥ 5 years)
NFR-C03	PII in refund reason fields handled consistent with applicable data protection obligations (e.g., GDPR)

11 Business Rules

ID	RULE	CATEGORY
BR-001	A refund may only be initiated against a payment with status CAPTURED or PARTIALLY_REFUNDED.	Eligibility
BR-002	The total refunded amount across all SUCCEEDED refunds for a payment must never exceed the original captured amount.	Refund Limit
BR-003	Each individual refund amount must be greater than zero and expressed in the smallest currency unit (e.g., cents).	Amount Validation
BR-004	Refunds must be returned to the original payment method. Cross-method refunds are not permitted in v1.	Payment Method
BR-005	The currency of a refund must match the currency of the original payment exactly.	Currency
BR-006	A merchant may only refund payments that belong to their own merchant account. Cross-merchant refunds are forbidden.	Authorization
BR-007	[Assumption A-08] Refunds may not be initiated against payments captured more than 180 days prior (configurable).	Refund Window
BR-008	Multiple partial refunds are permitted against the same payment provided BR-002 is satisfied at all times.	Partial Refunds
BR-009	A PENDING refund does not immediately reduce the refundable balance. Only SUCCEEDED refunds reduce the refundable balance.	Balance Tracking
BR-010	Each refund request with a unique Idempotency-Key must produce at most one refund record.	Idempotency
BR-011	A FULLY_REFUNDED payment cannot accept any further refund requests.	Terminal State
BR-012	A refund that has reached a terminal status (SUCCEEDED, FAILED, CANCELLED) cannot be modified or reversed through the API.	Refund Status
BR-013	The refund reason field, if provided, shall be limited to: customer_request, duplicate, fraudulent, product_not_received, other.	Reason Code
BR-014	Behavior when a new refund is submitted while one is PENDING must be explicitly defined — see Open Question OQ-05.	Concurrent Refunds
BR-015	Refund amounts are always processed in the same currency as the original payment. No FX conversion is performed.	Currency

12 User Stories & Gherkin Acceptance Criteria

US-01 — Full Refund

As a merchant, I want to refund the full amount of a completed payment via the API, so that I can return money to my customer without manual intervention from the operations team.

FIELD	DETAIL
Business Value	Eliminates manual refund processing; improves customer satisfaction; reduces chargeback risk
Priority	Must Have
Preconditions	Merchant authenticated | Payment CAPTURED and owned by merchant | Not previously fully refunded | Refund window active
Postconditions	Refund record exists | Payment FULLY_REFUNDED | Merchant receives refund.succeeded webhook

Acceptance Criteria — Gherkin

```
Feature: Full payment refund

Scenario: Successful full refund
  Given a payment exists with ID 'pay_abc123' and status 'CAPTURED'
  And the payment amount is 10000 cents (EUR 100.00)
  And the payment has not been previously refunded
  And I am authenticated as the merchant who owns this payment
  When I send POST /v1/payments/pay_abc123/refunds with amount 10000, currency 'EUR'
  Then the response status should be 201 Created
  And the response body should contain a unique refundId
  And the refund status should be 'SUCCEEDED' or 'PENDING'
  And the payment pay_abc123 status should eventually be 'FULLY_REFUNDED'
  And a webhook event 'refund.succeeded' should be delivered

Scenario: Refund on already FULLY_REFUNDED payment
  Given a payment exists with status 'FULLY_REFUNDED'
  When I send POST /v1/payments/pay_abc123/refunds with amount 10000
  Then the response status should be 422 Unprocessable Entity
  And the error code should be 'PAYMENT_NOT_REFUNDABLE'

Scenario: Refund attempt by unauthorized merchant
  Given payment 'pay_abc123' belongs to merchant_A
  And I am authenticated as merchant_B
  When I send POST /v1/payments/pay_abc123/refunds with amount 10000
  Then the response status should be 403 Forbidden
  And the error code should be 'FORBIDDEN'
```

US-02 — Partial Refund

As a merchant, I want to refund a portion of a payment and later refund additional portions if needed, so that I can handle partial returns flexibly without over-refunding the customer.

FIELD	DETAIL
Business Value	Supports partial order cancellation, item return from multi-item order
Priority	Must Have
Preconditions	Payment CAPTURED or PARTIALLY_REFUNDED | refundable_amount > 0 | Merchant authenticated and owns payment

FIELD	DETAIL
Postconditions	New partial refund record created | Payment refundable_amount reduced | Status PARTIALLY_REFUNDED (or FULLY_REFUNDED if balance zeroed)

Acceptance Criteria — Gherkin

```
Feature: Partial payment refund

Scenario: Successful first partial refund
  Given payment 'pay_xyz' has amount 20000 cents, status 'CAPTURED', refundable
balance 20000
  When I submit a refund for 8000 cents
  Then a refund record is created with amount 8000
  And the payment status becomes 'PARTIALLY_REFUNDED'
  And the payment's refundable_amount becomes 12000

Scenario: Partial refund exceeds remaining balance
  Given payment 'pay_xyz' has refundable_amount 7000
  When I submit a refund for 9000 cents
  Then the response status should be 422
  And the error code should be 'REFUND_EXCEEDS_BALANCE'
  And no refund record should be created
```

US-03 — View Refund Status & Receive Webhook

As a merchant, I want to query the status of a refund and receive an automatic webhook notification when the status changes, so that I can update my system without polling repeatedly.

Acceptance Criteria — Gherkin

```
Feature: Refund status retrieval and webhook notification

Scenario: Merchant polls refund status
  Given a refund 'ref_001' exists with status 'PENDING'
  And I am authenticated as the merchant who owns this refund
  When I send GET /v1/refunds/ref_001
  Then the response status is 200 OK
  And the response contains refundId, paymentId, amount, currency, status

Scenario: Merchant receives webhook on refund success
  Given refund 'ref_001' transitions from 'PENDING' to 'SUCCEEDED'
  When the provider confirms the refund outcome
  Then a POST request is sent to my configured webhook URL
  And the payload contains event type 'refund.succeeded'
```

13 BPMN Refund Process

Process Description

The BPMN-style process spans five swim-lane participants: Merchant, Payment API / Refund Service, Payment Provider / Acquirer, Database, and Webhook / Notification Service. The flow below describes all tasks, gateways, and events.

Swim Lanes & Flow

STEP	PARTICIPANT	TYPE	DESCRIPTION
1	Merchant	Start Event	Merchant submits POST /v1/payments/{paymentId}/refunds with auth headers, amount, currency, and Idempotency-Key
2	Payment API	Exclusive Gateway	Authenticate & Authorize: validates API key/token. Failure → 401 error (End Event — Unauthorized)
3	Payment API / DB	Task	Fetch payment record and validate merchant ownership. Failure → 404 or 403 (End Event — Validation Error)
4	Payment API	Exclusive Gateway	Refund Eligibility: check payment status = CAPTURED or PARTIALLY_REFUNDED. Failure → 422 (End Event — Not Eligible)
5	Payment API	Task	Refund Amount Validation: amount > 0, currency matches, amount ≤ refundable_balance. Failure → 422
6	Payment API / DB	Exclusive Gateway	Idempotency Check: look up Idempotency-Key. If found → return original response (End Event — Duplicate Handled)
7	Database	Task	Create Refund Record with status PENDING; store idempotency key. Atomic operation.
8	Payment API	Task	Return 201/202 response to Merchant (synchronous response)
9	Refund Service	Task	Submit Refund Request to Payment Provider API
10	Refund Service	Exclusive Gateway	Provider Response: Sync Success / Sync Failure / Timeout-Async
11a	Database	Task	[Success] Update Refund → SUCCEEDED; Update Payment → PARTIALLY/FULLY_REFUNDED
11b	Database	Task	[Failure] Update Refund → FAILED; record failure reason
11c	Refund Service	Intermediate Timer Event	[Async] Keep PENDING; await provider webhook callback
12	Database	Task	[After async callback] Update Refund and Payment status
13	Webhook Service	Task	Dispatch refund.succeeded / refund.failed / refund.pending event to Merchant webhook URL
14	Webhook Service	Intermediate Catch Event	Retry on delivery failure with exponential backoff (up to 72h)
15	Multiple	End Events	Refund Succeeded | Refund Failed | Refund Pending (resolved async)

Note: A full BPMN 2.0 diagram can be rendered from the Mermaid sequence diagram included in the HTML version of this document. The table above is the textual BPMN description.

1. ***Start Event:*** *Merchant submits POST /v1/payments/{paymentId}/refunds with authentication headers, amount, currency, and Idempotency-Key.*

2. ***Authentication Gateway:*** *API validates API key/token. Failure → 401 error response (End Event – Unauthorized).*
3. ***Payment Validation Task:*** *System fetches payment by paymentId and validates merchant ownership. Failure → 404 or 403 error (End Event – Validation Error).*
4. ***Refund Eligibility Gateway:*** *System checks payment status is CAPTURED or PARTIALLY_REFUNDED. Failure → 422 error (End Event – Not Eligible).*
5. ***Refund Amount Validation Task:*** *System checks amount > 0, currency matches, amount ≤ refundable_balance. Failure → 422 error (End Event – Invalid Amount).*
6. ***Idempotency Check Gateway:*** *System looks up Idempotency-Key. If found → returns original response (End Event – Duplicate Handled).*
7. ***Create Refund Record Task (Database):*** *System creates refund record with status PENDING and stores idempotency key. Atomic operation.*
8. ***Submit to Provider Task:*** *Refund Service calls Payment Provider refund API with amount, currency, and provider payment reference.*
9. ***Provider Response Gateway:***
    - *Synchronous Success → Update refund to SUCCEEDED (Task), update payment status (Task)*
    - *Synchronous Failure → Update refund to FAILED (Task)*
    - *Timeout / Async → Keep PENDING; start async wait (Intermediate Timer Event)*
10. ***Intermediate Event – Provider Callback:*** *On receiving provider webhook, update refund and payment status accordingly.*
11. ***Status Update Task (Database):*** *Update refund status and payment refundable_amount atomically.*
12. ***Webhook Dispatch Task:*** *Notification Service dispatches event to merchant webhook URL. Retry on failure (up to 72h).*
13. ***Return API Response:*** *API returns synchronous response to original request (201 for new refund, 202 for async PENDING).*
14. ***End Events:*** *Refund Succeeded / Refund Failed / Refund Pending (awaiting async resolution)*

14 API-Level Analysis

Endpoints Summary

METHOD	ENDPOINT	DESCRIPTION	AUTH
POST	/v1/payments/{paymentId}/refunds	Create a refund (full or partial)	Required
GET	/v1/refunds/{refundId}	Retrieve a refund by ID	Required
GET	/v1/payments/{paymentId}/refunds	List all refunds for a payment	Required

POST /v1/payments/{paymentId}/refunds — Create Refund

Request Headers

```
Authorization: Bearer {token}           // Required
Content-Type: application/json          // Required
Idempotency-Key: {uuid}                 // Strongly Recommended
X-Request-ID: {uuid}                    // Optional — for tracing
```

Request Body

```
{
  "amount": 5000,               // integer, smallest currency unit (e.g. cents).
Required.
  "currency": "EUR",            // ISO 4217 code. Required. Must match original
payment.
  "reason": "customer_request", // enum: customer_request | duplicate | fraudulent |
                                //       product_not_received | other. Optional.
  "metadata": {                 // merchant-defined key-value map. Optional.
    "order_id": "ord_99887"
  }
}
```

Success Response — 201 Created

```
{
  "refundId": "ref_01H9K3X2M7Q4VBN8WPDTJ6YC5",
  "paymentId": "pay_01H9K2X1M6P3UAN7VOCTJ5XB4",
  "merchantId": "mer_01H8J1W0L5N2TAM6UNCSI4WA3",
  "amount": 5000,
  "currency": "EUR",
  "reason": "customer_request",
  "status": "PENDING",
  "providerRefundId": null,
  "metadata": { "order_id": "ord_99887" },
  "createdAt": "2026-08-12T10:34:22Z",
  "updatedAt": "2026-08-12T10:34:22Z"
}
```

HTTP Status Code Reference

CODE	MEANING	WHEN USED
200	OK	Successful retrieval; idempotent replay
201	Created	Refund created (sync outcome known)
202	Accepted	Refund accepted; outcome pending (async)
400	Bad Request	Malformed request body; invalid field types

CODE	MEANING	WHEN USED
401	Unauthorized	Missing or invalid authentication
403	Forbidden	Authenticated but not authorized for this payment
404	Not Found	Payment or refund not found
409	Conflict	Concurrent modification conflict
422	Unprocessable Entity	Business rule violation (ineligible, exceeds balance, etc.)
429	Too Many Requests	Rate limit exceeded
500	Internal Server Error	Unexpected system error
502	Bad Gateway	Provider unreachable
503	Service Unavailable	Refund service temporarily unavailable

Error Catalogue

ERROR CODE	HTTP	DESCRIPTION
UNAUTHORIZED	401	API key or token is missing, expired, or invalid
FORBIDDEN	403	Merchant does not own the payment referenced
PAYMENT_NOT_FOUND	404	No payment found for the given paymentId
REFUND_NOT_FOUND	404	No refund found for the given refundId
PAYMENT_NOT_REFUNDABLE	422	Payment status does not allow refunds (FAILED, FULLY_REFUNDED, etc.)
INVALID_REFUND_AMOUNT	422	Amount is zero, negative, or non-numeric
REFUND_EXCEEDS_BALANCE	422	Requested amount exceeds the remaining refundable balance
CURRENCY_MISMATCH	422	Refund currency does not match original payment currency
REFUND_WINDOW_EXPIRED	422	Payment is older than the allowed refund window
INVALID_REASON	400	Provided reason is not one of the accepted enum values
PROVIDER_UNAVAILABLE	502	Payment provider API is unreachable; retry later
PROVIDER_REFUND_REJECTED	422	Provider rejected the refund (provider-level rules)
RATE_LIMIT_EXCEEDED	429	Too many refund requests from this merchant in a short window
INTERNAL_ERROR	500	Unexpected internal error; contact support with request_id

15 Refund Status Model

Refund Status Definitions

STATUS	MEANING	TERMINAL?
CREATED	Refund record created. Validation passed. Provider call not yet initiated. Typically very brief transitional state.	No
PENDING	Refund submitted to payment provider but no confirmation received yet. System awaiting async callback.	No
PROCESSING	Provider has acknowledged and is actively processing the refund. Used when provider distinguishes received vs. processing states.	No
SUCCEEDED	Provider confirmed refund is complete. Funds being returned to customer. Terminal success state.	Yes ✓
FAILED	Refund rejected by provider or failed internally. Reason recorded. No funds moved. Terminal state.	Yes ✓
CANCELLED	Refund cancelled before provider submission. [Post-MVP feature] Terminal state.	Yes ✓

Payment Status Extensions

PAYMENT STATUS	MEANING
CAPTURED	Existing. Payment fully captured; full amount refundable. No succeeded refunds yet.
PARTIALLY_REFUNDED	NEW. One or more partial refunds have SUCCEEDED; refundable_amount > 0.
FULLY_REFUNDED	NEW. Total refunded amount equals original captured amount; refundable_amount = 0.

State Transition Rules

CREATED → PENDING (async provider) | CREATED → SUCCEEDED (sync success) | CREATED → FAILED (sync failure)

PENDING → PROCESSING → SUCCEEDED | PENDING → PROCESSING → FAILED | PENDING → SUCCEEDED | PENDING → FAILED

SUCCEEDED, FAILED, CANCELLED are terminal — no further transitions permitted.

16 Data Requirements

Refunds Table

FIELD	TYPE	DESCRIPTION	REQUIRED
refund_id	VARCHAR(36) PK	Unique refund identifier (ULID or UUID prefixed ref_)	Yes
payment_id	VARCHAR(36) FK	References payments.payment_id	Yes
merchant_id	VARCHAR(36)	Denormalized for query efficiency	Yes
amount	BIGINT	Refund amount in smallest currency unit. Never DECIMAL for monetary values.	Yes
currency	CHAR(3)	ISO 4217 currency code. Must match payment currency.	Yes
reason	VARCHAR(50)	Enum: customer_request, duplicate, fraudulent, product_not_received, other	No
status	VARCHAR(20)	CREATED, PENDING, PROCESSING, SUCCEEDED, FAILED, CANCELLED	Yes
provider_refund_id	VARCHAR(100)	Reference ID returned by payment provider. Null until provider responds.	Conditional
provider_response	JSONB	Raw provider response stored for audit and debugging	No
failure_reason	VARCHAR(255)	Human-readable reason for FAILED status	Conditional
idempotency_key	VARCHAR(255)	Unique per merchant; enforced by composite UNIQUE(merchant_id, idempotency_key)	Yes
metadata	JSONB	Merchant-defined key-value pairs. Max 10 keys, 500 chars each.	No
created_at	TIMESTAMP WITH TZ	UTC timestamp of refund creation	Yes
updated_at	TIMESTAMP WITH TZ	UTC timestamp of last status change	Yes
succeeded_at	TIMESTAMP WITH TZ	UTC timestamp when status transitioned to SUCCEEDED	Conditional

Payments Table — Required Additions

FIELD	TYPE	DESCRIPTION	CHANGE
refunded_amount	BIGINT DEFAULT 0	Sum of all SUCCEEDED refund amounts. Updated atomically.	New column
refundable_amount	BIGINT COMPUTED	original_amount − refunded_amount.	New column / computed
status	VARCHAR(30)	Extend enum to include: PARTIALLY_REFUNDED, FULLY_REFUNDED	Extend existing enum

Entity Relationships

ENTITY	RELATIONSHIP	DETAILS
Merchant → Payment	One-to-Many	A merchant owns many payments
Payment → Refund	One-to-Many	A payment can have many refund records (one per refund attempt, including partials)
Refund → Audit Log	One-to-Many	Each refund event generates one or more audit log entries

Multiple Partial Refunds — Representation

Each partial refund is a separate row in the refunds table with its own refund_id, amount, and independent status lifecycle. The relationship is one PAYMENT to many REFUNDs. The payment's refunded_amount field is updated atomically (using a database transaction with row-level locking on the payment record) each time a refund reaches SUCCEEDED status. This prevents concurrent over-refunding.

17 Use Cases

UC-01 — Process Full Refund

FIELD	DETAIL
Use Case ID	UC-01
Name	Process Full Refund
Primary Actor	Merchant (via API client)
Secondary Actors	Payment Provider / Acquirer; Webhook / Notification Service; Database
Trigger	Merchant sends POST /v1/payments/{paymentId}/refunds with amount equal to original captured amount
Preconditions	(1) Merchant authenticated | (2) Payment exists and belongs to merchant | (3) Payment status = CAPTURED | (4) No previous full refund | (5) Payment within refund window
Postconditions (Success)	(1) Refund record exists with status SUCCEEDED | (2) Payment status = FULLY_REFUNDED | (3) refunded_amount = original amount | (4) Audit log complete | (5) Merchant received refund.succeeded webhook

Basic Flow (Happy Path)

8. Merchant constructs refund request with amount = original captured amount, currency, and Idempotency-Key
9. API authenticates the merchant's credentials
10. System retrieves payment record and confirms merchant ownership
11. System validates payment status = CAPTURED and refund amount ≤ refundable_amount
12. System checks Idempotency-Key — no duplicate found
13. System creates refund record (status: PENDING) in an atomic database transaction
14. System returns 201/202 response with refund object to merchant
15. Refund Service submits refund to Payment Provider API
16. Provider returns success confirmation
17. System updates refund status → SUCCEEDED; payment status → FULLY_REFUNDED; refunded_amount updated
18. Webhook Service sends refund.succeeded event to merchant webhook URL
19. Audit log records all state transitions

Exception Flows

EXCEPTION	TRIGGER	SYSTEM RESPONSE
E1 — Auth failure	Invalid API key	401 UNAUTHORIZED; no record created
E2 — Payment not found	paymentId doesn't exist	404 PAYMENT_NOT_FOUND
E3 — Wrong merchant	Payment belongs to different merchant	403 FORBIDDEN
E4 — Ineligible status	Payment status is FAILED or FULLY_REFUNDED	422 PAYMENT_NOT_REFUNDABLE
E5 — Provider timeout	No response from provider within timeout	Refund stays PENDING; async resolution
E6 — Provider rejection	Provider returns error	Refund set to FAILED; refund.failed webhook
E7 — Duplicate idempotency key	Same Idempotency-Key resubmitted	200 with original refund object; no duplicate created

Business Rules Applied

BR-001, BR-002, BR-004, BR-005, BR-006, BR-007, BR-010, BR-011

UC-02 — Process Partial Refund

FIELD	DETAIL
Use Case ID	UC-02
Name	Process Partial Refund
Primary Actor	Merchant (via API client)
Secondary Actors	Payment Provider; Webhook Service; Database
Trigger	Merchant sends POST /v1/payments/{paymentId}/refunds with amount less than remaining refundable balance
Preconditions	Payment exists, belongs to merchant, status CAPTURED or PARTIALLY_REFUNDED | refundable_amount > 0 | Requested amount < refundable_amount
Postconditions (Success)	New refund record created with SUCCEEDED status | Payment refunded_amount increased | Payment status = PARTIALLY_REFUNDED (or FULLY_REFUNDED) | Merchant notified via webhook

Exception Flows

EXCEPTION	TRIGGER	SYSTEM RESPONSE
E1 — Exceeds balance	Amount > refundable_amount	422 REFUND_EXCEEDS_BALANCE; no record created
E2 — Zero amount	amount = 0 in request body	422 INVALID_REFUND_AMOUNT
E3 — Currency mismatch	Refund currency ≠ payment currency	422 CURRENCY_MISMATCH
E4 — Race condition	Two simultaneous partials that together exceed balance	One succeeds; second fails at DB-level optimistic lock check: 409 CONFLICT or 422 REFUND_EXCEEDS_BALANCE

18 Use Case Diagram

> **Diagram Note:** The UML Use Case Diagram is rendered as an interactive Mermaid diagram in the HTML version of this document. The table below presents the same information in structured form.

ACTOR	USE CASE	RELATIONSHIP
Merchant	Create Full Refund	Direct — initiates via API
Merchant	Create Partial Refund	Direct — initiates via API
Merchant	View Refund Status	Direct — GET endpoint
Merchant	List Refunds for Payment	Direct — GET endpoint
Merchant	Receive Refund Webhook	Indirect — notified asynchronously
Payment API	Authenticate Merchant	Included by Create Full/Partial Refund
Payment API	Validate Payment Eligibility	Included by Create Full/Partial Refund
Payment API	Enforce Idempotency	Included by Create Full/Partial Refund
Payment API / Refund Service	Process Provider Response	Extends Create Refund when provider is involved
Payment Provider	Process Provider Response	Participates bidirectionally — receives request, returns response
Admin / Payment Ops	View Refund Status	Direct — for investigation and support
Admin / Payment Ops	List Refunds for Payment	Direct — for investigation and support

19 Risks and Mitigations

RISK	CATEGORY	IMPACT	PROBABILITY	MITIGATION
Duplicate refund (double refund to customer)	Financial	Critical	Medium	Enforce Idempotency-Key; row-level locking; provider refund ID deduplication in reconciliation job; monitor refunded_amount > original_amount
Financial inconsistency between API state and provider state	Financial / Operational	High	Medium	Daily reconciliation job comparing internal records vs. provider settlement reports; alert on mismatches
Provider timeout leaving refund permanently PENDING	Technical	High	Medium	Background job polls provider for long-running PENDING refunds; escalate after 24h PENDING; alert ops on stale PENDING
Race condition — concurrent partial refunds exceeding balance	Technical	High	Low–Medium	Database row-level locking on payment record; load test with concurrent requests before release
Incorrect refundable balance calculation	Financial	High	Low	Unit test all balance scenarios; use only SUCCEEDED refunds in computation; DB constraint CHECK(refunded_amount <= original_amount)
Reconciliation mismatch — provider settles on different date	Finance / Compliance	Medium	High	Store provider_refund_id and settlement timestamps; match on provider reference; daily automated reconciliation report with exception alerts
Webhook delivery failure — merchant not informed of outcome	Operational	Medium	Medium	Retry with exponential backoff 72h; alert ops after persistent failure; merchant can poll GET endpoint
Unauthorized refund — merchant refunds other merchant's payment	Security	Critical	Low	Always verify merchant_id on payment record; integration test for cross-merchant access; security review before release
Fraudulent refunds by compromised merchant credentials	Security / Fraud	High	Low–Medium	Rate limiting; configurable refund thresholds; anomaly detection alerting; audit logs; consider 2FA for large refunds
Payment provider does not support partial refunds	Technical	High	Low (if confirmed)	Confirm provider capability before development starts; design abstraction layer so full refunds ship independently
API downtime during refund processing	Availability	Medium	Low	Durable PENDING records survive restarts; idempotency prevents duplicate on retry; async processing decoupled from synchronous API response

20 Feature-Level Development Roadmap

MVP Scope: Phase 1 (Discovery) + Phase 2 (Core — full refunds) + Phase 3 (Partial refunds) + Phase 4 (Reliability & Notifications). Phases 5–6 complete the production-ready feature.

Phase 1 — Discovery & Analysis [MVP Prerequisite]

FEATURE	BUSINESS OBJECTIVE	MAIN DELIVERABLES	DEPENDENCIES
Business requirements finalization	Confirm scope and rules	Approved BR document, refund window policy	Product, Legal, Finance
Provider capability analysis	Confirm provider supports full + partial refunds	Provider API docs review; sandbox credentials	Payment Ops, Provider
API contract design	Align on interface before build	OpenAPI 3.x spec (draft); error catalogue	Dev, Product, QA
Data model design	Schema approved before migrations	ERD; migration scripts; schema review	Dev, DBA
Security design review	Identify auth and data risks early	Security design doc; threat model	Security, Dev

Phase 2 — Core Refund Engine [MVP]

FEATURE	BUSINESS OBJECTIVE	MAIN DELIVERABLES	DEPENDENCIES
POST /refunds endpoint	Enable full refund creation via API	Working endpoint; auth; request validation	Phase 1 complete
Full refund logic	Refund 100% of captured payment	Refund service; payment status update	Database migrations
Provider integration (full refund)	Submit refund to payment provider	Provider client; sync + async response handling	Provider sandbox
Refund status lifecycle	Accurate status tracking	Status machine; DB update logic	Refund service
GET refund endpoints	Allow merchants to query refund status	GET /refunds/{id}; GET /payments/{id}/refunds	Core refund engine
Audit logging	Compliance and traceability	Audit log for all refund events	Audit log service

Phase 3 — Partial Refunds [MVP]

FEATURE	BUSINESS OBJECTIVE	MAIN DELIVERABLES	DEPENDENCIES
Partial refund logic	Support refunds < original amount	Amount validation; balance deduction logic	Phase 2
Multiple partial refunds	Enable successive partial refunds	Cumulative balance tracking; PARTIALLY_REFUNDED status	Partial refund logic
Remaining refundable balance API	Merchants can check available balance	refundable_amount in GET payment response	Balance tracking
Concurrency protection	Prevent over-refunding under load	Row-level locking; optimistic concurrency tests	DB transaction design

Phase 4 — Reliability & Notifications [MVP]

FEATURE	BUSINESS OBJECTIVE	MAIN DELIVERABLES	DEPENDENCIES
Idempotency	Prevent duplicate refunds from retries	Idempotency-Key header; key storage; deduplication	Phase 3
Webhooks	Real-time merchant notification	refund.succeeded/failed/pending events; delivery service	Webhook infrastructure
Retry mechanism	Recover from transient provider failures	Retry with backoff; DLQ for permanent failures	Async processing queue
Provider timeout handling	Handle async provider responses safely	PENDING state management; background resolver job	Phase 2 provider integration

Phase 5 — Operations & Observability

FEATURE	BUSINESS OBJECTIVE	MAIN DELIVERABLES	DEPENDENCIES
Structured logging & tracing	Diagnostics and debugging	Trace IDs; structured log format; log aggregation	Observability platform
Monitoring & alerting	Detect failure conditions early	Dashboards; alerts (failure rate, pending rate, provider latency)	Metrics platform
Reconciliation integration	Financial accuracy	Refund event feed to reconciliation system	Finance systems
Merchant support tooling	Ops team can investigate issues	Internal refund lookup; manual status override (admin API)	Internal tooling team

Phase 6 — Testing & Production Release

ACTIVITY	OBJECTIVE	DELIVERABLE
Integration testing	End-to-end refund flow verification	Integration test suite (all flows)
API contract testing	Verify API matches OpenAPI spec	Contract test suite (Pact or similar)
Security testing	Pen test auth, authorization, data exposure	Security test report; no P0/P1 open findings
Performance testing	Validate NFR-P01 through NFR-P03	Load test results; benchmark report
UAT	Merchant acceptance of API behavior	UAT sign-off from Product Owner
Documentation	Developer and merchant integration guide	API reference; integration guide; error catalogue
Production rollout	Controlled release to merchants	Feature flag per merchant; canary → full rollout

21 Requirements Traceability Matrix

BUSINESS NEED	REQUIREMENT	USER STORY / USE CASE	TEST SCENARIO
Merchants can initiate full refunds via API	FR-001, BR-001, BR-002	US-01, UC-01	TS-001, TS-002, TS-010
Merchants can initiate partial refunds	FR-002, FR-003, BR-008	US-02, UC-02	TS-003, TS-004, TS-005
Prevent over-refunding	FR-007, BR-002, NFR-R04	US-02, UC-02	TS-006, TS-015, TS-016
Prevent duplicate refunds	FR-012, BR-010, NFR-R01	US-01, US-02	TS-013, TS-014
Merchants receive refund status updates	FR-015, FR-017, FR-018	US-03	TS-017, TS-018, TS-019
Only authorized merchants can create refunds	FR-013, FR-014, BR-006	US-01, US-02, UC-01	TS-007, TS-008, TS-009
Refund history is fully auditable	FR-019, NFR-SEC05	UC-01, UC-02	TS-020
Handle provider timeouts and async responses	FR-021, NFR-R02	UC-01	TS-021, TS-022
Financial accuracy and reconciliation	FR-011, NFR-R04	UC-01, UC-02	TS-023, TS-024
API is performant under load	NFR-P01, NFR-P03	—	TS-025 (load test)

22 Recommended Test Scenarios

Happy Path Tests

ID	SCENARIO	EXPECTED RESULT
TS-001	Create full refund for CAPTURED payment	201/202; refund created; FULLY_REFUNDED on success
TS-002	Verify payment status becomes FULLY_REFUNDED	Payment.status = FULLY_REFUNDED; refundable_amount = 0
TS-003	Create first partial refund on CAPTURED payment	201; payment status = PARTIALLY_REFUNDED; correct balance
TS-004	Create second partial refund on PARTIALLY_REFUNDED payment	201; cumulative balance updated correctly
TS-005	Final partial refund that zeros the balance	Payment status becomes FULLY_REFUNDED
TS-017	GET /refunds/{refundId} for own refund	200 with full refund object
TS-018	GET /payments/{paymentId}/refunds lists all refunds	200 with array of refund objects, correct summary
TS-019	Webhook delivered on refund.succeeded	Merchant endpoint receives correct payload within SLA

Negative / Validation Tests

ID	SCENARIO	EXPECTED RESULT
TS-006	Refund amount > refundable balance	422 REFUND_EXCEEDS_BALANCE; no record created
TS-007	Unauthenticated refund request	401 UNAUTHORIZED
TS-008	Merchant B attempts refund on Merchant A's payment	403 FORBIDDEN
TS-009	Refund with invalid/non-existent paymentId	404 PAYMENT_NOT_FOUND
TS-010	Refund on FAILED payment	422 PAYMENT_NOT_REFUNDABLE
TS-011	Refund with amount = 0	422 INVALID_REFUND_AMOUNT
TS-012	Refund with mismatched currency	422 CURRENCY_MISMATCH
TS-020	Refund against expired window payment	422 REFUND_WINDOW_EXPIRED

Provider Failure Tests

ID	SCENARIO	EXPECTED RESULT
TS-021	Provider returns timeout; merchant retries with same Idempotency-Key	Single refund record; PENDING resolves via callback; no duplicate
TS-022	Provider is unavailable (connection refused)	502 PROVIDER_UNAVAILABLE; no refund record created
TS-023	Provider processes refund but API never receives confirmation (orphaned)	Reconciliation job detects orphan; ops alerted; manual resolution available

Webhook Tests

ID	SCENARIO	EXPECTED RESULT
TS-019a	Webhook endpoint returns 500 on delivery	System retries; subsequent attempts succeed

ID	SCENARIO	EXPECTED RESULT
TS-019b	Webhook endpoint permanently unreachable for 72h	Event marked as permanently failed; alert raised; refund status correctly recorded
TS-019c	Async refund: PENDING → SUCCEEDED via provider callback	Merchant receives refund.pending then refund.succeeded in order

23 Open Questions

ID	QUESTION	IMPACT IF UNRESOLVED	OWNER
OQ-01	Does the payment provider support partial refunds via their API? What are the provider-specific constraints (minimum refund amount, maximum number of partial refunds, time limits)?	Partial refund feature may need to be scoped out or redesigned	Dev Lead / Payment Ops
OQ-02	What is the official refund eligibility window (e.g., 180 days)? Is this a legal/regulatory requirement or a business policy?	Cannot implement FR-020 without this; compliance risk	Legal / Compliance
OQ-03	Should the Idempotency-Key be mandatory (required) or strongly recommended (optional)? Making it optional creates duplicate refund risk.	Affects API contract and risk exposure	Product Manager / Architect
OQ-04	Does an existing webhook delivery infrastructure exist, or does it need to be built? If it exists, does it support retry and dead-letter queuing?	Significant build effort if webhook system doesn't exist	Dev Lead / Architecture
OQ-05	When a refund is PENDING, should subsequent refund requests be blocked until the pending one resolves, or evaluated against the current confirmed balance only?	Affects BR-014, FR-007, and over-refunding risk model	Product Manager / Finance
OQ-06	What is the provider's typical response time for refund processing? Do they support synchronous confirmation or is async the norm?	Affects API response design (201 vs. 202) and user experience	Payment Ops / Provider
OQ-07	Should merchants be able to refund a payment that is in PENDING status with a provider (i.e., captured but not yet fully settled at the acquirer level)?	Affects FR-005 and eligibility validation logic	Product / Payment Ops
OQ-08	What merchant-level refund limits or permissions should exist? Who manages these configurations?	Affects authorization model and admin tooling scope	Product / Risk / Finance
OQ-09	What data retention policy applies to refund records and audit logs? (Assumed ≥ 5 years — must be confirmed.)	Database storage planning; compliance risk	Legal / DBA
OQ-10	Is a sandbox/test environment available with the payment provider for QA testing without processing real money?	QA and integration testing blocked without sandbox	Dev Lead / Payment Ops

24 Definition of Ready

A user story or feature is ready to enter development when all of the following are true:

- All relevant open questions for that feature have been answered and documented
- Business rules and acceptance criteria are written, reviewed, and approved by Product Owner
- Payment provider refund capability confirmed (OQ-01) — no development should begin until provider integration is validated in a sandbox
- Refund eligibility window policy confirmed and documented (OQ-02)
- Data model reviewed and approved by the DBA / Architecture team
- API contract (OpenAPI spec) reviewed by at least one developer and one QA engineer
- Security design reviewed and approved by InfoSec
- Idempotency strategy agreed upon (OQ-03)
- Development environment available with provider sandbox access (OQ-10)
- Story is estimated and fits within one sprint, or is broken down to do so

25 Definition of Done

The Refund Functionality is considered production-ready when all of the following are true:

Functional

- All FR-001 through FR-021 are implemented and verified by QA
- All Business Rules BR-001 through BR-015 are enforced and covered by test scenarios
- All acceptance criteria in US-01, US-02, US-03 pass in staging environment
- Happy path, negative, idempotency, concurrency, and provider failure test scenarios all pass

Non-Functional

- API response time NFR-P01 (p95 ≤ 500ms) verified under load test
- Idempotency tested and confirmed — no duplicate refunds under concurrent retries
- Concurrency test — no over-refunding under simultaneous partial refund load
- All endpoints return correct HTTP status codes and error bodies per the error catalogue

Security

- Security review completed with no P0 or P1 open findings
- No sensitive payment data (PAN, CVV) logged — confirmed by log inspection
- Cross-merchant access forbidden — confirmed by integration test TS-008
- Rate limiting configured and tested

Operations

- Audit logs emitted for all refund events and persisted to audit store
- Monitoring dashboards published: refund success rate, failure rate, PENDING backlog, webhook delivery rate
- Alerts configured and tested for failure rate threshold breaches
- Reconciliation integration implemented and tested (refund events reach reconciliation system)
- Runbook documented for common failure scenarios (stale PENDING, webhook failure, provider outage)

Documentation & Release

- OpenAPI spec updated and published in developer portal
- Integration guide and error catalogue documentation complete
- UAT sign-off received from Product Owner
- Deployment plan reviewed and approved (feature flag or phased rollout)
- Rollback plan documented and tested