Technical Design Document (TDD)

altpay - Nigerian Payment Gateway

Document Information

Field	Details
Project Name	altpay Payment Gateway
Version	1.0
Date	January 30, 2026
Status	Draft
Author	Balogun Abdulsamad (Engineering Team Lead)
Related Documents	BRD.md, MVP_SPECIFICATION.md

Table of Contents

1. Introduction

1.1 Purpose

This Technical Design Document outlines the technical architecture, technology choices, implementation approach, and development timeline for the altpay payment gateway platform.

1.2 Scope

This document covers:

- System architecture and design patterns
- Technology stack selection with justifications
- Database schema design
- API specifications
- Security implementation
- Infrastructure setup
- Development phases and timeline
- Team requirements

1.3 Goals

- Build a scalable, secure payment gateway for Nigerian businesses
- Achieve 99.9% uptime for payment processing
- Process transactions within 3 seconds
- Support 10,000+ concurrent transactions
- PCI DSS Level 1 compliance

---

2. System Architecture

2.1 High-Level Architecture

```
flowchart TB
    subgraph CLIENT["CLIENT LAYER"]
        direction LR
        WEB["Web Dashboard & Checkout<br/>(Next.js 15 + TypeScript)"]
        SDK["Merchant Integrations<br/>(REST APIs / SDKs)"]
    end

    subgraph BACKEND["LARAVEL 12 BACKEND (Monolith)<br/>Powered by Laravel
Octane"]
        direction LR
        PAY["Payment Module<br/>• Transactions<br/>• Card Processing<br/>•
Tokenization"]
        MERCH["Merchant Module<br/>• Onboarding<br/>• KYC<br/>• Dashboard API"]
        SETTLE["Settlement Module<br/>• Daily Payouts<br/>•
Reconciliation<br/>• Reports"]
        NOTIFY["Notification Module<br/>• Email/SMS<br/>• Webhooks"]
    end

    subgraph DATA["DATA LAYER"]
        direction LR
        PG["PostgreSQL 16<br/>(Primary Database)"]
        REDIS["Redis 7<br/>(Cache/Queue/Sessions)"]
        MEILI["Meilisearch<br/>(Search)"]
    end

    subgraph QUEUE["LARAVEL QUEUE (Redis Driver)"]
```

```
        Q["Emails | SMS | Webhooks | Settlements | Reports"]
    end

    subgraph OBS["OBSERVABILITY LAYER"]
        direction LR
        SENTRY["Sentry<br/>(Error Tracking)"]
        GRAFANA["Grafana + Prometheus<br/>(Metrics/Dashboards)"]
        POSTHOG["PostHog<br/>(Product Analytics)"]
        ELK["ELK Stack<br/>(Centralized Logging)"]
        SEGMENT["Segment<br/>(Data Pipeline)"]
        CW["CloudWatch<br/>(AWS Logs)"]
    end

    CLIENT --> BACKEND
    BACKEND --> DATA
    DATA --> QUEUE
    QUEUE --> OBS
```

2.2 Modular Monolith Architecture

Given the 5-person team size, we adopt a **Modular Monolith** approach - organized like microservices but deployed as a single application.

Module	Responsibility	Future Extraction Priority
Payment	Transaction processing, card tokenization	High (PCI compliance)
Merchant	Merchant management, onboarding, KYC	Low
Settlement	Fund disbursement, reconciliation	Low
Notification	Email, SMS, webhooks	Medium
Analytics	Reporting, dashboards, metrics	Medium

Benefits:

- Faster development with small team
- Easier debugging and deployment
- Clear module boundaries for future extraction
- Lower infrastructure complexity

2.3 Design Patterns

- **Modular Monolith**: Single deployable with clear module boundaries
- **Repository Pattern**: Data access abstraction via Laravel Eloquent
- **Service Layer**: Business logic encapsulation
- **Queue-based Processing**: Async tasks via Laravel Queue
- **API Resources**: Consistent JSON response transformation
- **Form Requests**: Validation and authorization

3. Technology Stack

3.1 Frontend

Layer	Technology	Version	Justification
Framework	Next.js	15.x	SSR, SEO optimization, React 19 support, App Router
Language	TypeScript	5.7	Type safety, better DX, reduced bugs
Styling	Tailwind CSS	4.x	Rapid UI development, consistent design
UI Components	Shadcn/ui	Latest	Accessible, customizable components
State Management	Zustand	5.x	Lightweight, simple API
Form Handling	React Hook Form + Zod	Latest	Performance, validation
HTTP Client	Axios / TanStack Query	5.x	Caching, request management
Charts	Recharts	2.x	Dashboard visualizations

3.2 Backend

Layer	Technology	Version	Justification
Framework	Laravel	12.x	Batteries-included, excellent for APIs, large Nigerian community
Language	PHP	8.4	Modern features, improved performance
ORM	Eloquent	12.x	Elegant Active Record implementation
Validation	Laravel Validation	Built-in	Powerful, declarative rules
Authentication	Laravel Sanctum	4.x	API token authentication
API Documentation	Scramble / L5-Swagger	Latest	Auto-generated OpenAPI docs
Task Queue	Laravel Queue (Redis)	Built-in	Background job processing
Caching	Laravel Cache (Redis)	Built-in	Application caching
Performance	Laravel Octane (Swoole)	2.x	High-performance application server

3.3 Database

Type	Technology	Version	Use Case
Primary DB	PostgreSQL	16.x	Transactional data, ACID compliance
Cache	Redis	7.x	Sessions, rate limiting, caching, queues
Search	Meilisearch	1.x	Transaction search, fast indexing (Laravel Scout)

3.4 Infrastructure & Cloud

Component	Technology	Justification
Cloud Provider	AWS (Primary)	Reliability, Lagos region availability
Compute	AWS EC2 / Laravel Forge	Managed server provisioning
Container Orchestration	Kubernetes (EKS)	Scalability, auto-healing (Phase 2)
Container Runtime	Docker	Consistent environments
Load Balancer	AWS ALB	High availability
CDN	CloudFront	Static asset delivery, low latency
DNS	Route 53	DNS management, health checks
SSL/TLS	AWS ACM	Free SSL certificates
File Storage	AWS S3	Uploads, documents, exports

3.5 DevOps & CI/CD

Component	Technology	Purpose
Version Control	GitHub	Code repository
CI/CD	GitHub Actions	Automated pipelines
Deployment	Laravel Forge / Envoyer	Zero-downtime deployments
Configuration	Laravel .env + AWS Secrets Manager	Environment config
Error Tracking	Sentry	Real-time error monitoring and alerts
Metrics	Grafana + Prometheus	Dashboards and system metrics
APM	Laravel Telescope	Local debugging and query analysis

3.6 Analytics & Observability

Component	Technology	Purpose
Error Tracking	Sentry	Real-time error monitoring, stack traces, releases

Component	Technology	Purpose
Metrics & Dashboards	Grafana + Prometheus	Custom dashboards, system metrics, alerts
Product Analytics	PostHog	User behavior, funnels, feature flags, session replay
Data Pipeline	Segment	Customer data collection, event routing, integrations
Application Logs	ELK Stack (Elasticsearch, Logstash, Kibana)	Centralized logging, log analysis, search
Cloud Logs	AWS CloudWatch	Infrastructure logs, Lambda logs
Uptime Monitoring	UptimeRobot	Availability alerts

4. Database Design

4.1 Entity Relationship Diagram

```
erDiagram
    MERCHANTS {
        uuid id PK
        string business_name
        string email
        string phone
        text address
        string business_type
        string rc_number
        string tin_number
        string settlement_account
        string api_key
        string webhook_url
        string status
        timestamp created_at
        timestamp updated_at
    }

    TRANSACTIONS {
        uuid id PK
        uuid merchant_id FK
        uuid customer_id FK
        decimal amount
        string currency
        string status
        string payment_method
        uuid card_token_id FK
        string reference
        string processor_ref
        decimal fee_amount
        boolean settled
        timestamp settled_at
        timestamp created_at
        timestamp updated_at
    }

    USERS {
        uuid id PK
        string email
        string password_hash
        string first_name
        string last_name
        string phone
        string role
        uuid merchant_id FK
        timestamp created_at
        timestamp updated_at
    }

    CARD_TOKENS {
        uuid id PK
        uuid customer_id FK
        uuid merchant_id FK
        string token
        string last_four
        string card_type
        int exp_month
        int exp_year
        boolean is_active
        timestamp created_at
    }

    SETTLEMENTS {
        uuid id PK
        uuid merchant_id FK
        decimal amount
        decimal fee_total
        decimal net_amount
        int transaction_count
        timestamp period_start
        timestamp period_end
        string status
        string bank_reference
        timestamp processed_at
        timestamp created_at
    }

    CUSTOMERS {
        uuid id PK
        string email
        string first_name
        string last_name
        string phone
        timestamp created_at
        timestamp updated_at
    }

    WEBHOOKS {
        uuid id PK
        uuid merchant_id FK
        uuid transaction_id FK
        string event_type
        json payload
        string status
        int attempts
        timestamp last_attempt_at
        timestamp created_at
    }

    AUDIT_LOGS {
        uuid id PK
        uuid user_id FK
        string action
        string entity_type
        uuid entity_id
        json old_value
        json new_value
        string ip_address
        string user_agent
        timestamp created_at
    }

    MERCHANTS ||--o{ TRANSACTIONS : has
    MERCHANTS ||--o{ USERS : employs
    MERCHANTS ||--o{ SETTLEMENTS : receives
    MERCHANTS ||--o{ WEBHOOKS : triggers
    MERCHANTS ||--o{ CARD_TOKENS : stores
    CUSTOMERS ||--o{ TRANSACTIONS : makes
    CUSTOMERS ||--o{ CARD_TOKENS : owns
    TRANSACTIONS ||--o| CARD_TOKENS : uses
    TRANSACTIONS ||--o{ WEBHOOKS : generates
    USERS ||--o{ AUDIT_LOGS : creates
```

4.2 Key Tables Schema

merchants

```sql
CREATE TABLE merchants (
    id              UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    business_name   VARCHAR(255) NOT NULL,
    email           VARCHAR(255) UNIQUE NOT NULL,
    phone           VARCHAR(20) NOT NULL,
    address         TEXT,
    business_type   VARCHAR(50),
    rc_number       VARCHAR(50),
    tin_number      VARCHAR(50),
    settlement_bank VARCHAR(100),
    settlement_account VARCHAR(20),
    api_key         VARCHAR(255) UNIQUE,
    api_secret_hash VARCHAR(255),
    webhook_url     VARCHAR(500),
    status          VARCHAR(20) DEFAULT 'pending',
    kyc_verified    BOOLEAN DEFAULT false,
    created_at      TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at      TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

transactions

```sql
CREATE TABLE transactions (
    id              UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    merchant_id     UUID REFERENCES merchants(id),
    customer_id     UUID REFERENCES customers(id),
    amount          DECIMAL(15, 2) NOT NULL,
    currency        VARCHAR(3) DEFAULT 'NGN',
    status          VARCHAR(20) DEFAULT 'pending',
    payment_method  VARCHAR(50) NOT NULL,
    card_token_id   UUID REFERENCES card_tokens(id),
    reference       VARCHAR(100) UNIQUE NOT NULL,
    processor_ref   VARCHAR(100),
    fee_amount      DECIMAL(15, 2),
    fee_percentage  DECIMAL(5, 4) DEFAULT 0.015,
    fee_cap         DECIMAL(15, 2) DEFAULT 2000.00,
    settled         BOOLEAN DEFAULT false,
    settled_at      TIMESTAMP,
    metadata        JSONB,
    created_at      TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at      TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_transactions_merchant ON transactions(merchant_id);
CREATE INDEX idx_transactions_status ON transactions(status);
CREATE INDEX idx_transactions_created ON transactions(created_at);
CREATE INDEX idx_transactions_reference ON transactions(reference);
```

refunds

```sql
CREATE TABLE refunds (
    id                UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    transaction_id  UUID REFERENCES transactions(id) NOT NULL,
    merchant_id      UUID REFERENCES merchants(id) NOT NULL,
    amount            DECIMAL(15, 2) NOT NULL,
    reason            VARCHAR(255),
    status            VARCHAR(20) DEFAULT 'pending',
    -- Status: pending, processing, succeeded, failed
    processor_ref    VARCHAR(100),
    refunded_at      TIMESTAMP,
    metadata          JSONB,
    created_at       TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at       TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_refunds_transaction ON refunds(transaction_id);
CREATE INDEX idx_refunds_merchant ON refunds(merchant_id);
CREATE INDEX idx_refunds_status ON refunds(status);
```

disputes (Chargebacks)

```sql
CREATE TABLE disputes (
    id                UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    transaction_id  UUID REFERENCES transactions(id) NOT NULL,
    merchant_id      UUID REFERENCES merchants(id) NOT NULL,
    amount            DECIMAL(15, 2) NOT NULL,
    reason            VARCHAR(100) NOT NULL,
    -- Reasons: fraudulent, duplicate, product_not_received,
    --           product_unacceptable, subscription_canceled, other
    status            VARCHAR(20) DEFAULT 'needs_response',
    -- Status: needs_response, under_review, won, lost
    evidence_due_by TIMESTAMP,
    processor_ref    VARCHAR(100),
    resolved_at      TIMESTAMP,
    created_at       TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at       TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_disputes_merchant ON disputes(merchant_id);
CREATE INDEX idx_disputes_status ON disputes(status);
```

dispute_evidence

```sql
CREATE TABLE dispute_evidence (
    id                UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    dispute_id       UUID REFERENCES disputes(id) NOT NULL,
    evidence_type    VARCHAR(50) NOT NULL,
    -- Types: receipt, shipping_documentation, customer_communication,
    --        refund_policy, service_documentation, other
    file_url        VARCHAR(500),
    description     TEXT,
    submitted_at    TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

payment_links (Coming Soon)

```
CREATE TABLE payment_links (
    id              UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    merchant_id     UUID REFERENCES merchants(id) NOT NULL,
    name            VARCHAR(255) NOT NULL,
    description     TEXT,
    amount          DECIMAL(15, 2), -- NULL for variable amount
    currency        VARCHAR(3) DEFAULT 'NGN',
    slug            VARCHAR(100) UNIQUE NOT NULL,
    url             VARCHAR(500) NOT NULL,
    active          BOOLEAN DEFAULT true,
    expires_at      TIMESTAMP,
    success_url     VARCHAR(500),
    metadata        JSONB,
    created_at      TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at      TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_payment_links_merchant ON payment_links(merchant_id);
CREATE INDEX idx_payment_links_slug ON payment_links(slug);
```

fraud_rules

```
CREATE TABLE fraud_rules (
    id              UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    merchant_id     UUID REFERENCES merchants(id),
    -- NULL merchant_id = global rule
    rule_type       VARCHAR(50) NOT NULL,
    -- Types: velocity, geolocation, amount, card_country, email_domain
    condition       JSONB NOT NULL,
    action          VARCHAR(20) NOT NULL,
    -- Actions: block, challenge, review, allow
    priority        INTEGER DEFAULT 100,
    active          BOOLEAN DEFAULT true,
    created_at      TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

api_keys

```
CREATE TABLE api_keys (
    id                UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    merchant_id       UUID REFERENCES merchants(id) NOT NULL,
    key_type          VARCHAR(20) NOT NULL,
    -- Types: publishable, secret, restricted
    key_prefix        VARCHAR(20) NOT NULL,
    -- Prefixes: pk_live_, pk_test_, sk_live_, sk_test_, rk_live_, rk_test_
    key_hash          VARCHAR(255) NOT NULL,
    last_four         VARCHAR(4) NOT NULL,
    name              VARCHAR(100),
    permissions       JSONB, -- For restricted keys
    last_used_at      TIMESTAMP,
    expires_at        TIMESTAMP,
    revoked_at        TIMESTAMP,
    created_at        TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_api_keys_merchant ON api_keys(merchant_id);
CREATE INDEX idx_api_keys_prefix ON api_keys(key_prefix);
```

5. API Design

5.1 API Standards

- RESTful design principles
- JSON request/response format
- API versioning via URL path (e.g., `/api/v1/`)
- ISO 8601 date format
- Snake_case for JSON keys
- Pagination for list endpoints
- Consistent error response format

5.2 Authentication

- API Key + Secret for merchant APIs
- JWT tokens for dashboard access
- OAuth 2.0 for third-party integrations

5.3 Core API Endpoints

Payment APIs

Method	Endpoint	Description
POST	`/api/v1/payments/initialize`	Initialize a payment
GET	`/api/v1/payments/{reference}`	Get payment status

Method	Endpoint	Description
POST	`/api/v1/payments/{reference}/verify`	Verify payment
POST	`/api/v1/payments/charge`	Direct card charge
GET	`/api/v1/payments`	List merchant payments
POST	`/api/v1/payments/{id}/capture`	Capture authorized payment
POST	`/api/v1/payments/{id}/cancel`	Cancel pending payment

Refund APIs

Method	Endpoint	Description
POST	`/api/v1/refunds`	Create a refund
GET	`/api/v1/refunds/{id}`	Get refund details
GET	`/api/v1/refunds`	List all refunds

Customer APIs

Method	Endpoint	Description
POST	`/api/v1/customers`	Create customer
GET	`/api/v1/customers/{id}`	Get customer details
PUT	`/api/v1/customers/{id}`	Update customer
DELETE	`/api/v1/customers/{id}`	Delete customer
GET	`/api/v1/customers`	List customers

Card/Payment Method APIs

Method	Endpoint	Description
POST	`/api/v1/tokens`	Tokenize card (single-use)
POST	`/api/v1/payment_methods`	Create saved payment method
GET	`/api/v1/payment_methods/{id}`	Get payment method
DELETE	`/api/v1/payment_methods/{id}`	Delete payment method
POST	`/api/v1/payment_methods/{id}/attach`	Attach to customer
POST	`/api/v1/payment_methods/{id}/detach`	Detach from customer

Dispute/Chargeback APIs

Method	Endpoint	Description
GET	`/api/v1/disputes`	List disputes
GET	`/api/v1/disputes/{id}`	Get dispute details
POST	`/api/v1/disputes/{id}/accept`	Accept dispute
POST	`/api/v1/disputes/{id}/submit_evidence`	Submit evidence

Merchant APIs

Method	Endpoint	Description
POST	`/api/v1/merchants/register`	Register merchant
GET	`/api/v1/merchants/me`	Get merchant profile
PUT	`/api/v1/merchants/me`	Update merchant
GET	`/api/v1/merchants/balance`	Get merchant balance
GET	`/api/v1/merchants/transactions`	List transactions

Settlement APIs

Method	Endpoint	Description
GET	`/api/v1/settlements`	List settlements
GET	`/api/v1/settlements/{id}`	Get settlement details
GET	`/api/v1/settlements/{id}/transactions`	Get settlement transactions

Webhook APIs

Method	Endpoint	Description
POST	`/api/v1/webhook_endpoints`	Create webhook endpoint
GET	`/api/v1/webhook_endpoints`	List webhook endpoints
PUT	`/api/v1/webhook_endpoints/{id}`	Update webhook endpoint
DELETE	`/api/v1/webhook_endpoints/{id}`	Delete webhook endpoint

Payment Link APIs (Coming Soon)

Method	Endpoint	Description
POST	`/api/v1/payment_links`	Create payment link
GET	`/api/v1/payment_links/{id}`	Get payment link

Method	Endpoint	Description
PUT	`/api/v1/payment_links/{id}`	Update payment link
POST	`/api/v1/payment_links/{id}/deactivate`	Deactivate link

5.4 Sample API Request/Response

Initialize Payment

```
POST /api/v1/payments/initialize
Authorization: Bearer sk_live_xxxxxxxx
Content-Type: application/json

{
    "amount": 10000,
    "currency": "NGN",
    "email": "customer@example.com",
    "reference": "ALT-TXN-123456",
    "callback_url": "https://merchant.com/callback",
    "metadata": {
        "order_id": "ORD-789",
        "customer_name": "John Doe"
    }
}
```

Response

```
{
    "status": true,
    "message": "Payment initialized",
    "data": {
        "reference": "ALT-TXN-123456",
        "authorization_url": "https://checkout.altpay.ng/pay/xyz123",
        "access_code": "xyz123",
        "expires_at": "2026-01-30T12:30:00Z"
    }
}
```

5.5 Error Response Format

```
{
    "status": false,
    "message": "Validation error",
    "errors": [
        {
            "field": "amount",
            "message": "Amount must be greater than 0"
        }
    ],
    "error_code": "VALIDATION_ERROR"
}
```

5.6 Error Codes Reference

Error Code	HTTP Status	Description
`VALIDATION_ERROR`	400	Invalid request parameters
`INVALID_API_KEY`	401	API key is invalid or expired
`AUTHENTICATION_ERROR`	401	Authentication failed
`PERMISSION_DENIED`	403	Insufficient permissions
`RESOURCE_NOT_FOUND`	404	Requested resource not found
`IDEMPOTENCY_ERROR`	409	Idempotency key already used with different params
`RATE_LIMIT_ERROR`	429	Too many requests
`CARD_ERROR`	402	Card was declined
`PROCESSING_ERROR`	500	Payment processor error
`API_ERROR`	500	Internal server error

Card Decline Codes

Code	Description	Suggested Action
`insufficient_funds`	Card has insufficient funds	Try different card
`card_declined`	Card declined by issuer	Contact bank
`expired_card`	Card has expired	Use different card
`invalid_cvv`	CVV verification failed	Re-enter CVV
`invalid_expiry`	Invalid expiration date	Re-enter expiry
`card_not_supported`	Card type not supported	Use Visa/Mastercard/Verve
`fraud_suspected`	Suspected fraud	Contact support
`do_not_honor`	Bank refused transaction	Contact bank
`lost_card`	Card reported lost	Contact bank
`stolen_card`	Card reported stolen	Contact bank
`processing_error`	Processor error	Retry later

Code	Description	Suggested Action
3ds_required	3D Secure required	Complete 3DS
3ds_failed	3D Secure failed	Retry or use different card

5.7 Pagination

All list endpoints support cursor-based pagination:

```
GET /api/v1/payments?limit=25&starting_after=pay_abc123
```

Response includes pagination info:

```
{
    "object": "list",
    "data": [...],
    "has_more": true,
    "url": "/api/v1/payments"
}
```

5.8 Expanding Objects

Use the expand parameter to include related objects:

```
GET /api/v1/payments/pay_abc123?expand[]=customer&expand[]=refunds
```

5.9 Filtering & Search

```
GET /api/v1/payments?
status=succeeded&created[gte]=1704067200&email=customer@example.com
```

Filter	Type	Description
status	string	Filter by status
created[gte]	timestamp	Created after
created[lte]	timestamp	Created before
amount[gte]	integer	Amount greater than
email	string	Customer email

6. Security Architecture

6.1 PCI DSS Level 1 Compliance

As a payment service provider, altpay must achieve **PCI DSS Level 1 compliance** (highest level).

Requirement	Implementation	Status
Req 1-2: Network Security	VPC isolation, security groups, WAF, network segmentation	Required
Req 3: Protect Cardholder Data	Tokenization, never store CVV, mask PAN	Required
Req 4: Encrypt Transmission	TLS 1.3, certificate pinning	Required
Req 5-6: Vulnerability Mgmt	Daily scans, patch within 30 days, secure SDLC	Required
Req 7-9: Access Control	RBAC, MFA, physical security, unique IDs	Required
Req 10-11: Monitoring & Testing	Audit logs, IDS/IPS, quarterly pen tests	Required
Req 12: Security Policies	Documented policies, annual review	Required

Compliance Timeline

- **Month 1-2**: Self-Assessment Questionnaire (SAQ D)
- **Month 3-4**: Remediation of gaps
- **Month 5**: QSA audit engagement
- **Month 6**: ROC (Report on Compliance) submission

6.2 Data Security & Encryption

Encryption Standards

Data Type	At Rest	In Transit	Storage
Card Numbers (PAN)	AES-256-GCM	TLS 1.3	Tokenized only
CVV/CVC	Never stored	TLS 1.3	Never stored
API Keys	AES-256	TLS 1.3	Hashed (Argon2id)
Passwords	Argon2id	TLS 1.3	Hashed
PII (Name, Email)	AES-256	TLS 1.3	Encrypted
Transaction Data	AES-256	TLS 1.3	Encrypted

Key Management (AWS KMS)

```
flowchart TB
    subgraph KMS["KEY HIERARCHY"]
        CMK["AWS KMS Master Key (CMK)"]
        DEK1["DEK - Card Tokens"]
        DEK2["DEK - PII Data"]
        DEK3["DEK - API Secrets"]
        DEK4["DEK - Webhook Secrets"]

        CMK --> DEK1
        CMK --> DEK2
        CMK --> DEK3
        CMK --> DEK4
    end

    subgraph POLICY["Key Policies"]
        ROT["Key Rotation: Every 90 days (automatic)"]
        ACC["Key Access: IAM roles with least privilege"]
        AUD["Audit: All key usage logged to CloudTrail"]
    end
```

6.3 Card Tokenization (Card Vault)

altpay implements a **card vault** for secure tokenization:

```
sequenceDiagram
    participant Browser as Customer Browser
    participant SDK as altpay.js SDK<br/>(Secure iFrame)
    participant Vault as Card Vault Service<br/>(Isolated PCI Environment)
    participant App as Merchant App

    Browser->>SDK: Card: 4242***<br/>Exp: 12/28<br/>CVV: ***
    Note over Browser,SDK: HTTPS (TLS 1.3)
    SDK->>Vault: Submit card data<br/>(Hosted fields - PCI scope reduction)
    Vault->>Vault: Encrypt & Store<br/>Input: PAN
    Vault-->>SDK: Token: tok_abc123xyz789
    SDK-->>Browser: Return token
    Browser->>App: tok_abc123xyz789<br/>(Safe to store, log, transmit)
```

Token Types

Token Type	Prefix	Validity	Use Case
Single-use Token	`tok_`	15 minutes	One-time payment
Payment Method	`pm_`	Permanent	Saved card
Customer	`cus_`	Permanent	Customer reference
Card Fingerprint	`fp_`	Permanent	Duplicate detection

6.3 Authentication & Authorization

```
sequenceDiagram
    participant User as User/Merchant
    participant Auth as Auth Service
    participant GW as API Gateway
    participant Svc as Backend Service

    rect rgb(240, 248, 255)
        Note over User,Auth: Initial Login
        User->>Auth: Login Request
        Auth->>Auth: Validate Credentials
        Auth->>Auth: Generate JWT Token<br/>(Access + Refresh)
        Auth-->>User: JWT Tokens
    end

    rect rgb(255, 248, 240)
        Note over User,Svc: Subsequent Requests
        User->>GW: Request + JWT
        GW->>GW: Validate Token
        GW->>GW: Check Permissions
        GW->>Svc: Route to Service
        Svc-->>User: Response
    end
```

6.4 Rate Limiting

Endpoint Type	Rate Limit
Payment Initialize	100 req/min per merchant
Payment Verify	200 req/min per merchant
General API	1000 req/min per merchant
Checkout Page	60 req/min per IP

6.5 Security Headers

```
Strict-Transport-Security: max-age=31536000; includeSubDomains
Content-Security-Policy: default-src 'self'
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
```

6.11 Idempotency

All mutating API endpoints support **idempotency keys** to safely retry requests:

```
POST /api/v1/payments/charge
Authorization: Bearer sk_live_xxxx
Idempotency-Key: order_12345_charge_001
Content-Type: application/json

{
    "amount": 50000,
    "currency": "NGN",
    "source": "tok_xxx"
}
```

Behavior	Description
Key Storage	Redis with 24-hour TTL
Key Format	Any string up to 255 characters
Collision	Returns cached response (same key = same result)
Different Payload	Returns 409 Conflict error

7. Third-Party Integrations

7.1 Payment Processors

Processor	Purpose	Priority
Interswitch	Card processing, switching	Primary
NIBSS	Bank transfers, NIP	Primary
Flutterwave	Backup processor	Secondary
Paystack	Backup processor	Secondary

7.2 Banking Partners

Partner	Integration	Purpose
alt bank	Direct API	Settlement, corporate accounts
NIBSS	NIP, NUBAN	Account verification

7.3 Communication Services

Service	Provider	Purpose
Email	SendGrid / AWS SES	Transactional emails
SMS	Termii / Africa's Talking	OTP, notifications

7.4 Identity & KYC

Service	Purpose
Smile Identity	Identity verification
Youverify	Business verification
CAC API	Company registration lookup

7.5 Analytics & Data Platform

Service	Purpose	Integration
Segment	Customer data platform, event collection	Backend + Frontend
PostHog	Product analytics, funnels, session replay	Via Segment
Sentry	Error tracking, performance monitoring	Backend + Frontend
Grafana	Metrics dashboards, system monitoring	Backend metrics

Data Flow Architecture

```
flowchart TB
    subgraph SOURCES["Data Sources"]
        direction LR
        FE["Next.js Frontend<br/>• User Events<br/>• Page views<br/>•
Clicks<br/>• Forms"]
        BE["Laravel Backend<br/>• Server-side Events<br/>• Payments<br/>•
Signups"]
    end

    FE --> SEG
    BE --> SEG

    SEG["SEGMENT<br/>(Data Collection & Routing)"]

    SEG --> POSTHOG["PostHog<br/>(Product Analytics)"]
    SEG --> SENTRY["Sentry<br/>(Errors)"]
    SEG --> GRAFANA["Grafana<br/>(Metrics)"]
```

7.6 Webhook System

altpay implements a robust webhook system for real-time event notifications:

Webhook Events

Category	Event Type	Description

Category	Event Type	Description
Payment	`payment.created`	Payment intent created
Payment	`payment.succeeded`	Payment completed successfully
Payment	`payment.failed`	Payment attempt failed
Payment	`payment.pending`	Awaiting customer action (3DS)
Payment	`payment.refunded`	Full refund processed
Payment	`payment.partially_refunded`	Partial refund processed
Charge	`charge.succeeded`	Card charge successful
Charge	`charge.failed`	Card charge failed
Charge	`charge.dispute.created`	Chargeback initiated
Charge	`charge.dispute.closed`	Dispute resolved
Customer	`customer.created`	New customer record
Customer	`customer.updated`	Customer details changed
Card	`card.created`	Card tokenized and saved
Card	`card.expiring`	Card expires within 30 days
Settlement	`settlement.created`	Settlement batch created
Settlement	`settlement.completed`	Funds transferred to bank
Settlement	`settlement.failed`	Settlement transfer failed
Account	`account.updated`	Merchant account changed

Webhook Payload Structure

```json
{
    "id": "evt_1abc2def3ghi4jkl",
    "object": "event",
    "api_version": "2026-01-30",
    "created": 1738252800,
    "type": "payment.succeeded",
    "livemode": true,
    "pending_webhooks": 1,
    "request": {
        "id": "req_xyz123",
        "idempotency_key": "order_12345"
    },
    "data": {
        "object": {
            "id": "pay_abc123xyz789",
            "object": "payment",
            "amount": 50000,
            "currency": "ngn",
            "status": "succeeded",
            "customer": "cus_xxx",
            "metadata": {
                "order_id": "ORD-789"
            }
        },
        "previous_attributes": {
            "status": "pending"
        }
    }
}
```

Webhook Security

```
sequenceDiagram
    participant AP as altpay Server
    participant WH as Webhook Delivery
    participant MR as Merchant Server

    Note over AP: 1. Generate Signature
    AP->>AP: signature = HMAC-SHA256<br/>(webhook_secret, payload + timestamp)

    Note over AP,MR: 2. Send Webhook with Headers
    AP->>WH: POST /webhook
    WH->>MR: altpay-Signature: t=1738252800,v1=5257a869...

    Note over MR: 3. Merchant Verification
    MR->>MR: Check timestamp within 5 minutes<br/>(replay attack prevention)
    MR->>MR: Verify signature matches<br/>computed value

    MR-->>WH: HTTP 200 OK
```

PHP Verification Example:

```php
$payload = file_get_contents('php://input');
$signature = $_SERVER['HTTP_ALTPAY_SIGNATURE'];
$isValid = Altpay\Webhook::verify($payload, $signature, $secret);
```

Webhook Delivery & Retry

Attempt	Delay	Total Time
1	Immediate	0

Attempt	Delay	Total Time
2	5 minutes	5 min
3	30 minutes	35 min
4	2 hours	2h 35m
5	8 hours	10h 35m
6	24 hours	34h 35m
Give up	-	After 6 attempts

7.7 SDKs & Libraries

altpay provides official SDKs for easy integration:

Official SDKs

Language	Package	Installation
PHP	`altpay/altpay-php`	`composer require altpay/altpay-php`
JavaScript/Node	`@altpay/altpay-js`	`npm install @altpay/altpay-js`
Python	`altpay`	`pip install altpay`

JavaScript SDK (altpay.js)

```
<!-- Hosted checkout fields (PCI compliant) -->
<script src="https://js.altpay.ng/v1/altpay.js"></script>

<script>
const altpay = Altpay('pk_live_xxxxxxxxxx');

// Create payment element
const elements = altpay.elements();
const cardElement = elements.create('card', {
    style: {
        base: {
            fontSize: '16px',
            color: '#32325d',
        }
    }
});
cardElement.mount('#card-element');

// Handle payment
const form = document.getElementById('payment-form');
form.addEventListener('submit', async (e) => {
    e.preventDefault();
    const { token, error } = await altpay.createToken(cardElement);

    if (error) {
        console.error(error.message);
    } else {
        // Send token to your server
        submitPayment(token.id);
    }
});
</script>
```

PHP SDK Example

```php
<?php
use Altpay\AltpayClient;
use Altpay\Exception\ApiException;

$altpay = new AltpayClient('sk_live_xxxxxxxxxx');

try {
    // Initialize payment
    $payment = $altpay->payments->create([
        'amount' => 50000, // ₦500.00 in kobo
        'currency' => 'ngn',
        'email' => 'customer@example.com',
        'reference' => 'order_' . uniqid(),
        'callback_url' => 'https://yoursite.com/callback',
        'metadata' => [
            'order_id' => 'ORD-123',
        ],
    ]);

    // Redirect to checkout
    header('Location: ' . $payment->authorization_url);

} catch (ApiException $e) {
    // Handle error
    echo $e->getMessage();
}

// Verify payment
$verification = $altpay->payments->verify('reference_here');
if ($verification->status === 'success') {
    // Payment successful
}
```

7.8 Checkout Integration Options

Option	PCI Scope	Customization	Best For
Redirect Checkout	SAQ A	Low	Quick integration
Embedded Checkout	SAQ A	Medium	In-page experience
Custom Elements	SAQ A-EP	High	Full UI control
Direct API	SAQ D	Full	Existing PCI compliance

8. Infrastructure & DevOps

8.1 AWS Architecture

```
flowchart TB
    subgraph AWS["AWS CLOUD"]
        subgraph VPC["VPC (10.0.0.0/16)"]
            subgraph PUB1["Public Subnet (10.0.1.0/24)"]
                NAT1["NAT GW"]
                ALB1["ALB"]
            end

            subgraph PUB2["Public Subnet (10.0.2.0/24)"]
                NAT2["NAT GW"]
                ALB2["ALB"]
            end

            subgraph PRIV1["Private Subnet (10.0.3.0/24)"]
                EKS1["EKS Nodes<br/>(K8s)"]
            end

            subgraph PRIV2["Private Subnet (10.0.4.0/24)"]
                EKS2["EKS Nodes<br/>(K8s)"]
            end

            subgraph DB1["Database Subnet (10.0.5.0/24)"]
                RDS1["RDS Primary"]
                REDIS1["ElastiCache<br/>(Redis)"]
            end

            subgraph DB2["Database Subnet (10.0.6.0/24)"]
                RDS2["RDS Standby"]
                REDIS2["ElastiCache<br/>(Redis)"]
            end
        end

        CF["CloudFront<br/>(CDN)"]
        R53["Route 53<br/>(DNS)"]
        S3["S3<br/>(Static)"]
    end
    R53 --> CF
    CF --> ALB1 & ALB2
    ALB1 --> EKS1
    ALB2 --> EKS2
    EKS1 --> RDS1 & REDIS1
    EKS2 --> RDS2 & REDIS2
    RDS1 <-.-> RDS2
    REDIS1 <-.-> REDIS2
```

8.2 Infrastructure Phases

Phase 1: Simplified Infrastructure (MVP - Small Team)

For initial launch with 5-person team, we use managed services:

Component	Technology	Justification
Hosting	Laravel Forge / AWS Elastic Beanstalk	Simplified deployment, managed servers
Web Server	Nginx + Laravel Octane	High performance
Database	AWS RDS (PostgreSQL 16)	Managed, auto-backups
Cache/Queue	AWS ElastiCache (Redis)	Managed Redis
File Storage	AWS S3	Scalable storage
CDN	CloudFront	Static assets, low latency
SSL	AWS ACM / Let's Encrypt	Free SSL certificates
Logging	ELK Stack (Managed OpenSearch)	Centralized log management

Phase 2: Kubernetes Migration (Scale - Larger Team)

When team scales beyond 10 engineers or transaction volume exceeds 100K/day:

Component	Specification
Cluster	AWS EKS 1.29
Node Groups	2 (General + Compute-optimized)
Min Nodes	3
Max Nodes	20
Instance Type	t3.large (general), c5.xlarge (compute)
Autoscaling	Horizontal Pod Autoscaler
Service Mesh	Istio (optional)

8.3 CI/CD Pipeline

```
flowchart LR
    subgraph BUILD["Build Stage"]
        PUSH["Code Push"] --> BLD["Build"]
        BLD --> TEST["Test"]
        TEST --> SEC["Security Scan"]
    end

    subgraph DEPLOY["Deploy Stage"]
        SEC --> IMG["Build Image"]
        IMG --> QA["QA Deploy"]
        QA --> STG["Staging Deploy"]
        STG --> PROD["Prod Deploy"]
    end

    subgraph ENV["Environments"]
        direction TB
        E1["Development: Auto-deploy on PR merge to develop"]
        E2["QA: Auto-deploy on PR merge to qa"]
        E3["Staging: Manual approval + auto-deploy"]
        E4["Production: Manual approval + blue-green deployment"]
    end
```

8.4 Environment Configuration

Environment	Purpose	Database	Scaling
Development	Feature development	Shared DB	Minimal
QA	Testing	Isolated DB	Minimal
Staging	Pre-production	Production mirror	Medium
Production	Live system	HA cluster	Auto-scale

9. Development Timeline

9.1 Project Phases Overview (5 Months Total)

```
gantt
    title PROJECT TIMELINE (5 MONTHS)
    dateFormat  YYYY-MM-DD
    section MVP Phase
    MVP Phase (2 Months)     :mvp, 2026-02-03, 8w
    section Phase 2
    Enhanced Features (6 Weeks)    :phase2, 2026-03-31, 6w
    section Phase 3
    Scale & Advanced (6 Weeks)    :phase3, 2026-05-12, 6w
    section Milestones
    MVP Launch              :milestone, 2026-03-28, 0d
    Full Launch             :milestone, 2026-06-20, 0d
```

9.2 MVP Phase Breakdown (8 Weeks)

Sprint 1: Foundation (Week 1-2)

Timeline: February 3-14, 2026

Task	Owner	Deliverables
Project setup, repos, CI/CD	Tech Lead	Git repos, pipelines
AWS infrastructure (Forge, RDS)	Tech Lead	Cloud resources
Database schema & migrations	Backend 1	DB structure
Laravel project + Sanctum auth	Backend 1	Auth system
Merchant registration API	Backend 2	Registration endpoints
Next.js project setup	Frontend 1	Frontend foundation
Design system (Shadcn)	Frontend 2	Component library
Test planning	QA	Test strategy

Milestone: Development environment running, registration works

---

Sprint 2: Payment Core (Week 3-4)

Timeline: February 17-28, 2026

Task	Owner	Deliverables
Payment initialization API	Backend 1	Payment endpoints
Interswitch integration	Backend 1	Card processing
Card tokenization	Backend 2	Secure card handling
API key generation	Backend 2	Key management
Checkout page (card form)	Frontend 2	Payment UI
Merchant dashboard	Frontend 1	Dashboard UI
Payment flow testing	QA	Test coverage

Milestone: Card payments working in test mode

Sprint 3: 3DS & Webhooks (Week 5-6)

Timeline: March 3-14, 2026

Task	Owner	Deliverables
3D Secure integration	Backend 1	3DS authentication
Payment verification API	Backend 1	Verify endpoints
Webhook system	Backend 2	Event delivery
Transaction status updates	Backend 2	Status tracking
3DS challenge UI	Frontend 2	3DS pages
Transaction views	Frontend 1	History pages
Integration testing	QA	E2E tests

Milestone: End-to-end payment flow complete

Sprint 4: Settlement & Launch (Week 7-8)

Timeline: March 17-28, 2026

Task	Owner	Deliverables
Settlement engine	Backend 1	Daily settlements
Bank transfer integration	Backend 1	Payout system
Rate limiting & security	Backend 2	API protection
Audit logging	Backend 2	Transaction logs
Settlement pages	Frontend 1	Settlement UI
API documentation	Frontend 2	Docs site
UAT & security testing	QA	Final validation
Production deployment	Tech Lead	Live environment

Milestone: 🚀 MVP Launch - March 28, 2026

9.3 Phase 2: Enhanced Features (6 Weeks)

Timeline: March 31 - May 9, 2026

Week	Features	Description
Week 9-10	Refunds	Full/partial refund processing
Week 11-12	Payment Links	Shareable payment pages

Week	Features	Description
Week 13-14	Bank Transfers	Accept bank payments, Dispute management

Milestone: Phase 2 Complete - Enhanced payment gateway

9.4 Phase 3: Scale & Advanced (6 Weeks)

Timeline: May 12 - June 20, 2026

Week	Features	Description
Week 15-16	Recurring Payments	Subscription billing
Week 17-18	Multi-currency & Fraud	USD/GBP, ML fraud detection
Week 19-20	Advanced Features	Installments, polish, documentation

Milestone: 🎉 Full Launch - June 20, 2026

9.5 Sprint Schedule Summary

Sprint	Dates	Focus	Phase
Sprint 1	Feb 3-14	Foundation & Auth	MVP
Sprint 2	Feb 17-28	Payment Core	MVP
Sprint 3	Mar 3-14	3DS & Webhooks	MVP
Sprint 4	Mar 17-28	Settlement & Launch	MVP
Sprint 5	Mar 31 - Apr 11	Refunds	Phase 2
Sprint 6	Apr 14-25	Payment Links	Phase 2
Sprint 7	Apr 28 - May 9	Bank Transfers & Disputes	Phase 2
Sprint 8	May 12-23	Recurring Payments	Phase 3
Sprint 9	May 26 - Jun 6	Multi-currency & Fraud	Phase 3
Sprint 10	Jun 9-20	Final Polish & Launch	Phase 3

---

10. Team Structure

10.1 Required Team

Role	Count	Responsibilities
Tech Lead	1	Architecture, code review, technical decisions, DevOps, security oversight

Role	Count	Responsibilities
UI Designer	1	User experience design, interface mockups, design system, prototyping
Backend Engineers	2	Laravel API development, database, integrations, payment processing, settlements
Frontend Engineers	2	Next.js dashboard, checkout UI, responsive design, API integration
QA Engineer	1	Manual testing, test automation, quality assurance, UAT coordination

Total: 7 team members

10.2 Role Details

Tech Lead

- Full-stack expertise (Laravel + Next.js)
- Infrastructure and DevOps management
- Security and PCI compliance oversight
- Code reviews and architectural decisions
- Mentoring team members
- Stakeholder communication

UI Designer

- User experience (UX) research and design
- Interface mockups and prototypes (Figma)
- Design system and component library design
- Checkout flow and payment UI design
- Dashboard layouts and merchant experience
- Collaborate with frontend engineers on implementation

Backend Engineers (2)

- **Backend Engineer 1**: Payment processing, card tokenization, Interswitch integration, 3DS
- **Backend Engineer 2**: Settlement engine, webhooks, notifications, audit logging
- Both: Laravel API development, database design, queue jobs

Frontend Engineers (2)

- **Frontend Engineer 1**: Merchant dashboard, analytics, settings
- **Frontend Engineer 2**: Checkout flow, payment UI, public pages
- Both: Component library, responsive design, API integration

QA Engineer

- Test planning and execution

- Automated testing (Playwright, PHPUnit)
- API testing (Postman, Pest)
- Bug tracking and regression testing
- UAT coordination with stakeholders

10.3 Team Organization

```
flowchart TB
    TL["Tech Lead<br/>(Architecture, DevOps, Security)"]

    TL --> UI["UI Designer<br/>UX & Interface Design"]
    TL --> BE["Backend Team<br/>2 Laravel Engineers"]
    TL --> FE["Frontend Team<br/>2 Next.js Engineers"]
    TL --> QA["QA Team<br/>1 QA Engineer"]

    UI -.->|Design Specs| FE
```

10.4 Communication & Workflow

Activity	Frequency	Participants
Daily Standup	Daily (15 min)	All team
Sprint Planning	Bi-weekly	All team
Code Review	Ongoing	Tech Lead + relevant engineer
Sprint Retro	Bi-weekly	All team
Tech Sync	Weekly	Tech Lead + Engineers

11. Testing Strategy

11.1 Testing Pyramid

```
pie showData
    title Testing Distribution
    "Unit Tests (70%)" : 70
    "Integration Tests (20%)" : 20
    "E2E Tests (10%)" : 10
```

11.2 Testing Types

Type	Tools	Coverage Target
Unit Tests (Backend)	PHPUnit, Pest	80% code coverage

Type	Tools	Coverage Target
Unit Tests (Frontend)	Vitest, React Testing Library	70% code coverage
Integration Tests	Pest, Postman	All API endpoints
E2E Tests	Playwright	Critical user flows
Performance Tests	k6	Load testing
Security Tests	OWASP ZAP, Laravel Security Checker	Vulnerability scanning

11.3 Test Environments

Environment	Data	Purpose
Local	Mocked	Developer testing
CI	Seeded	Automated tests
QA	Test data	Manual QA
Staging	Sanitized prod	Pre-release validation

12. Deployment Strategy

12.1 Deployment Approach

- **Strategy**: Blue-Green Deployment
- **Rollback Time**: < 5 minutes
- **Zero Downtime**: Required

12.2 Release Process

```
flowchart LR
    subgraph DEV["Development"]
        FB["Feature Branch"] --> PR["PR Review"]
        PR --> DEV_MERGE["Merge to Develop"]
    end

    subgraph QA_ENV["QA"]
        DEV_MERGE --> QA_DEPLOY["QA Deploy"]
        QA_DEPLOY --> QA_TEST["QA Testing"]
        QA_TEST --> APPROVAL["Approval"]
    end

    subgraph STAGING["Staging"]
        APPROVAL --> REL["Release Branch"]
        REL --> STG_DEPLOY["Staging Deploy"]
        STG_DEPLOY --> UAT["UAT"]
        UAT --> SIGNOFF["Sign-off"]
    end
    subgraph PROD["Production"]
        SIGNOFF --> PROD_DEPLOY["Production Deploy"]
        PROD_DEPLOY --> SMOKE["Smoke Tests"]
        SMOKE --> MONITOR["Monitor"]
    end

    subgraph ROLLBACK["If Issues"]
        MONITOR -.-> ROLL["Rollback"]
        ROLL -.-> HOTFIX["Hotfix"]
        HOTFIX -.-> PROD_DEPLOY
    end
```

12.3 Database Migrations

- Forward-only migrations
- Backward compatible changes
- Migration testing in staging first
- Rollback scripts for emergency

---

13. Monitoring & Observability

13.1 Monitoring Stack

Component	Tool	Purpose
Error Tracking	Sentry	Exception monitoring, stack traces, release tracking
Metrics & Dashboards	Grafana + Prometheus	System metrics, custom dashboards, alerting
Product Analytics	PostHog	User behavior, conversion funnels, feature flags
Data Pipeline	Segment	Event collection, data routing to analytics tools
Logs	ELK Stack (OpenSearch)	Centralized logging, log search, analysis
Cloud Logs	AWS CloudWatch	Infrastructure and Lambda logs
APM (Dev)	Laravel Telescope	Query debugging, request inspection
Uptime	UptimeRobot	External availability monitoring

13.2 Key Metrics (SLIs/SLOs)

Metric	SLO	Alert Threshold
Availability	99.9%	< 99.5%

Metric	SLO	Alert Threshold
API Latency (p95)	< 500ms	> 800ms
Payment Success Rate	> 98%	< 95%
Error Rate	< 0.1%	> 0.5%
Settlement Time	< 24 hours	> 20 hours

13.3 Alerting Rules

Severity	Response Time	Examples
Critical	5 minutes	Payment service down, DB failure
High	15 minutes	High error rate, slow responses
Medium	1 hour	Elevated latency, disk space
Low	Next business day	Non-critical warnings

14. Risk Assessment

14.1 Technical Risks

Risk	Probability	Impact	Mitigation
Payment processor downtime	Medium	High	Multi-processor failover
Database failure	Low	Critical	Multi-AZ, automated backups
Security breach	Low	Critical	PCI compliance, WAF, monitoring
Scaling issues	Medium	High	Auto-scaling, load testing
Third-party API changes	Medium	Medium	API versioning, abstraction layer

14.2 Project Risks

Risk	Probability	Impact	Mitigation
Scope creep	High	Medium	Clear requirements, change control
Resource availability	Medium	High	Cross-training, documentation
Integration delays	Medium	High	Early integration, mock services
Compliance delays	Medium	High	Early engagement with compliance

14.3 Contingency Plans

Scenario	Response

Scenario	Response
Primary processor down	Automatic failover to secondary
Data center outage	Multi-region deployment
Key personnel unavailable	Documentation, pair programming
Security incident	Incident response plan, communication

15. Appendix

15.1 Glossary

Term	Definition
PCI DSS	Payment Card Industry Data Security Standard
JWT	JSON Web Token
APM	Application Performance Monitoring
SLI	Service Level Indicator
SLO	Service Level Objective
E2E	End-to-End
UAT	User Acceptance Testing
WAF	Web Application Firewall

15.2 References

- PCI DSS Requirements
- AWS Well-Architected Framework
- OWASP Security Guidelines
- Twelve-Factor App Methodology

15.3 Document History

Version	Date	Author	Changes
1.0	Jan 30, 2026	Engineering Team	Initial draft

End of Technical Design Document