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	Engineering Team
Related Documents	BRD.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

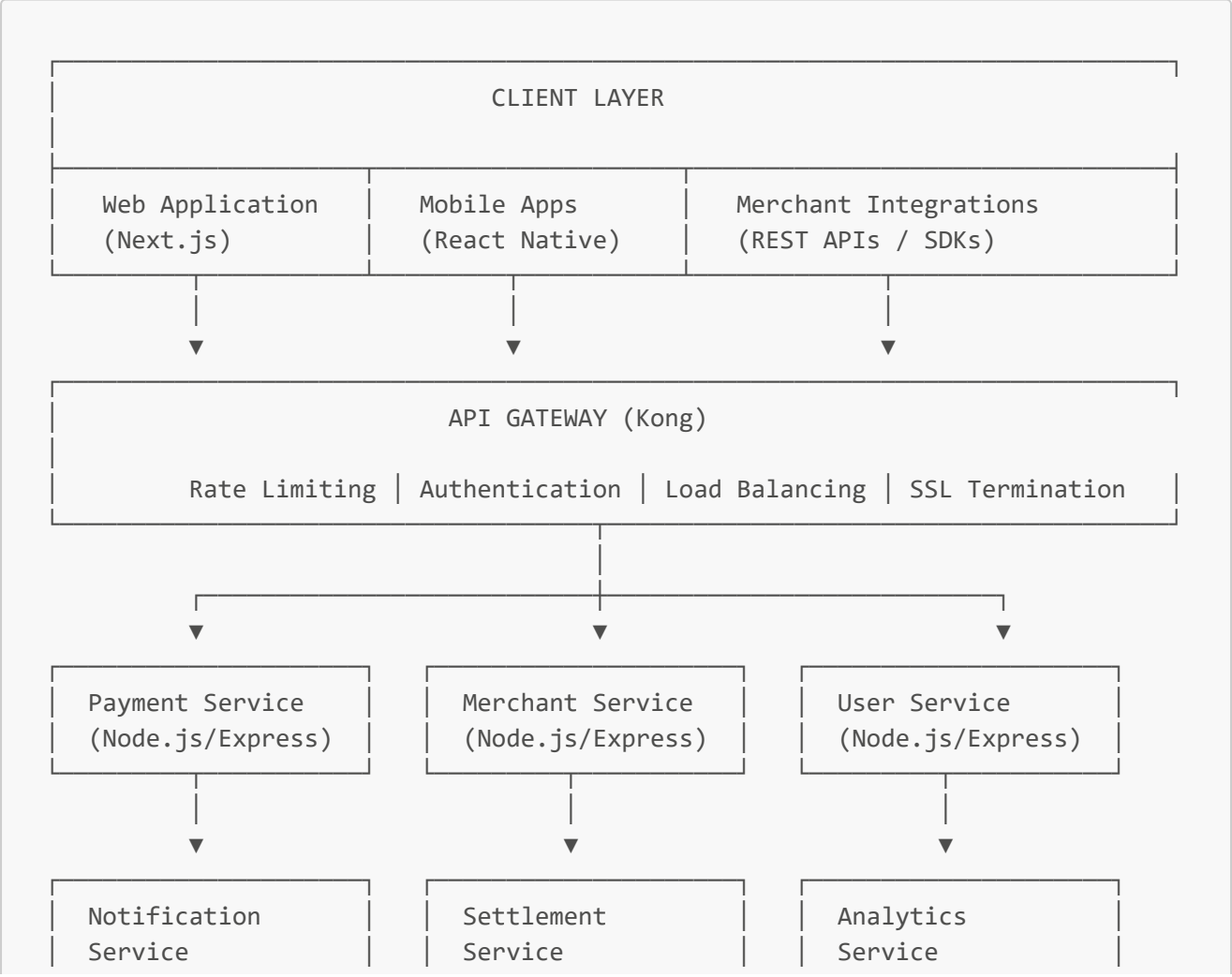
1.3 Goals

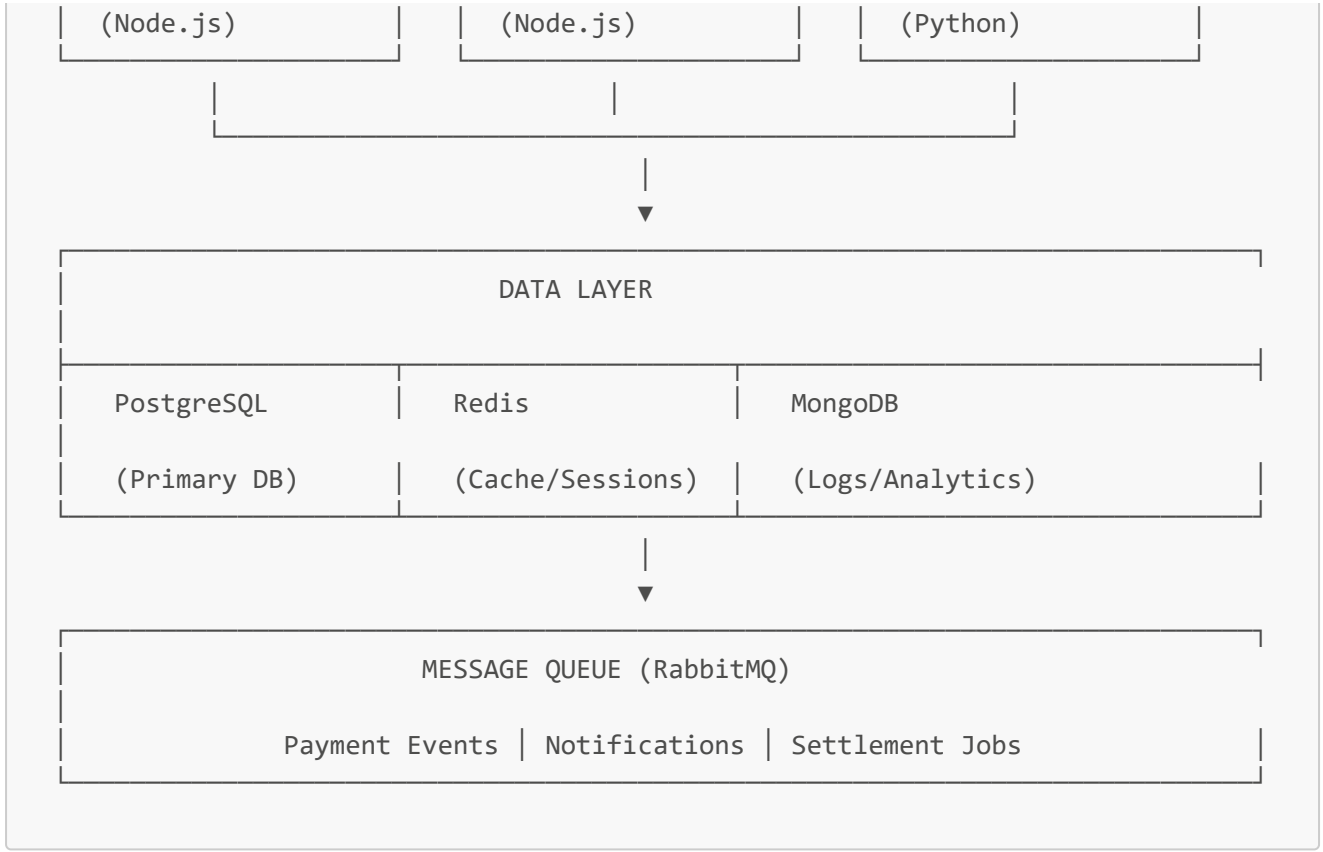
- Build a scalable, secure payment gateway for Nigerian businesses
- Achieve 99.9% uptime for payment processing
- Process transactions within 3 seconds
- Support 10,000+ concurrent transactions
- PCI DSS Level 1 compliance

---

2. System Architecture

2.1 High-Level Architecture





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

Layer	Technology	Version	Justification
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

Component	Technology	Justification
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
Monitoring	Laravel Telescope + Sentry	APM, error tracking
Logging	Laravel Log + CloudWatch	Centralized logging
Uptime Monitoring	UptimeRobot / Pingdom	Availability alerts

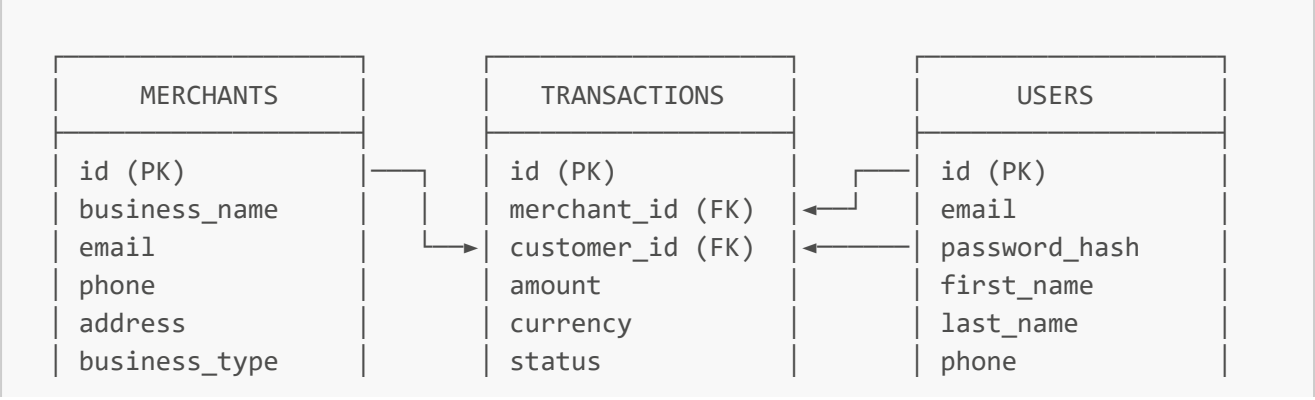
3.6 Mobile (Future Phase)

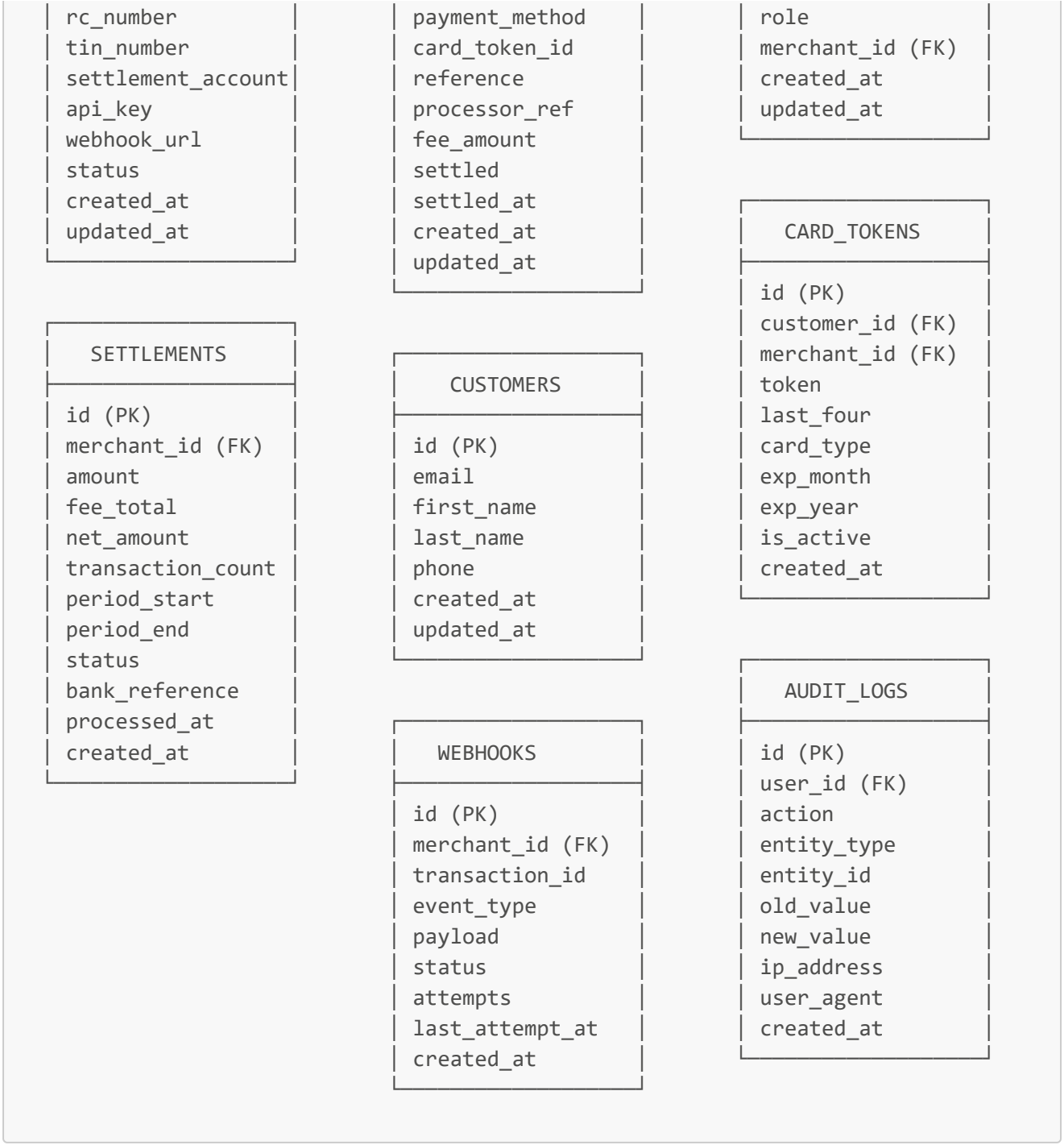
Component	Technology	Justification
Framework	React Native	Code sharing with web
Navigation	React Navigation	Standard solution
State	Zustand	Consistency with web

---

4. Database Design

4.1 Entity Relationship Diagram





4.2 Key Tables Schema

merchants

```
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

```
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);
```

---

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
POST	`/api/v1/payments/{reference}/verify`	Verify payment
POST	`/api/v1/payments/charge`	Direct card charge
GET	`/api/v1/payments`	List merchant payments

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

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"
}
```

---

6. Security Architecture

6.1 PCI DSS Compliance Requirements

Requirement	Implementation
Secure Network	VPC, security groups, WAF
Cardholder Data Protection	Tokenization, encryption at rest
Vulnerability Management	Regular scans, patching
Access Control	RBAC, MFA, least privilege
Network Monitoring	IDS/IPS, logging
Security Policies	Documented procedures

6.2 Data Security

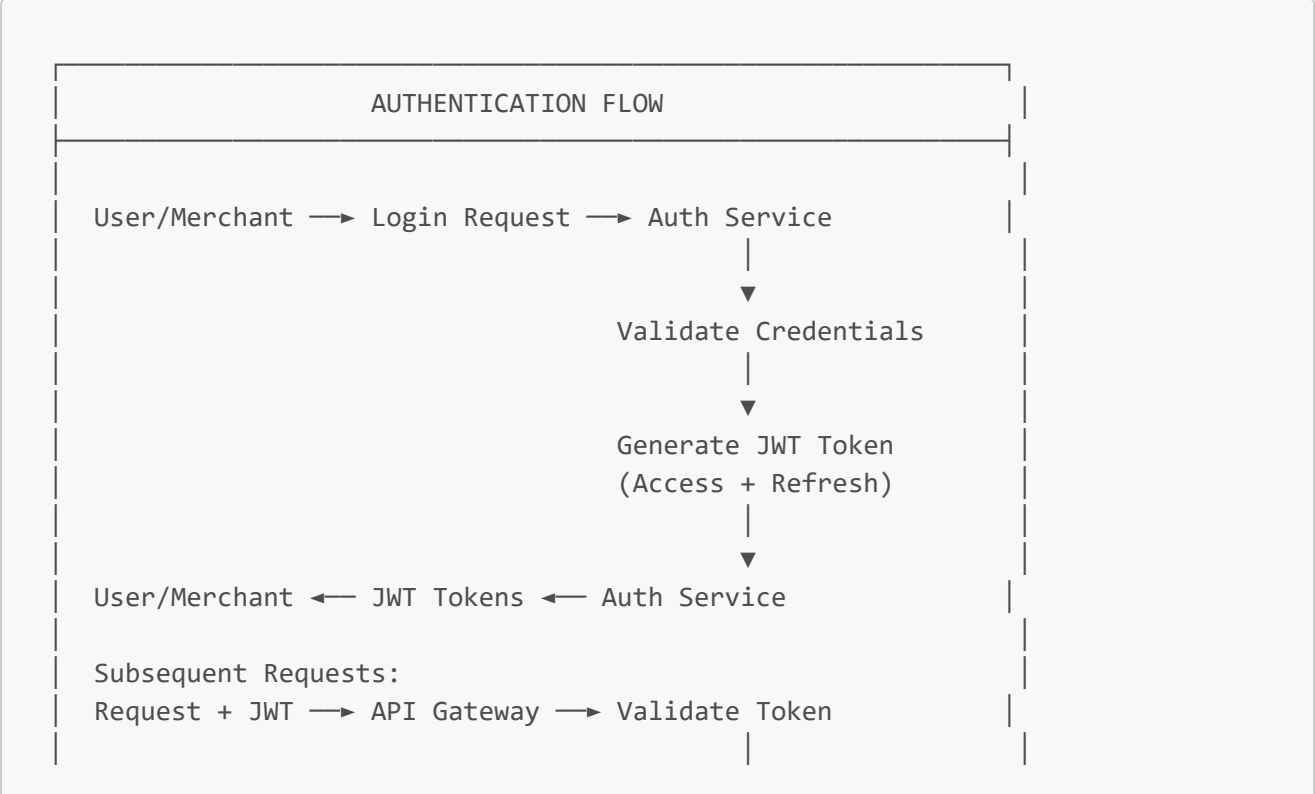
Encryption

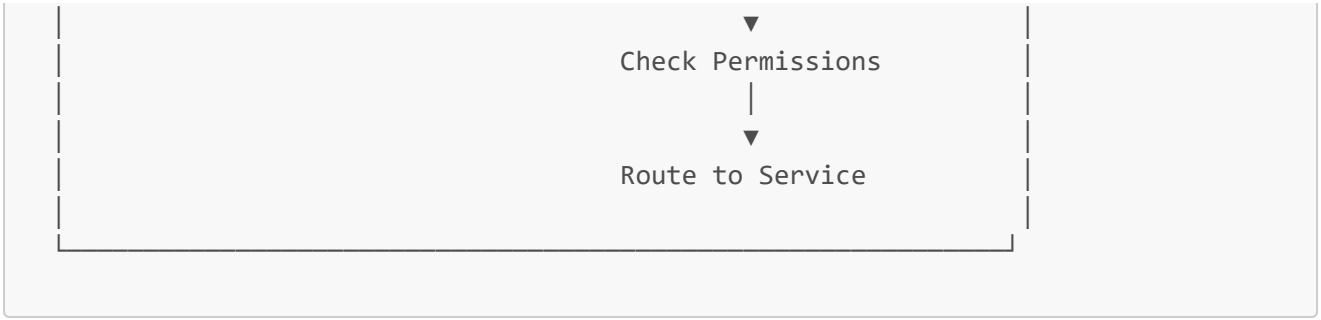
- **In Transit**: TLS 1.3 for all communications
- **At Rest**: AES-256 encryption for sensitive data
- **Card Data**: Never stored; tokenized via processor

Key Management

- AWS KMS for encryption keys
- Key rotation every 90 days
- Separate keys per environment

6.3 Authentication & Authorization





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
```

---

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
Push	Firebase FCM	Mobile notifications

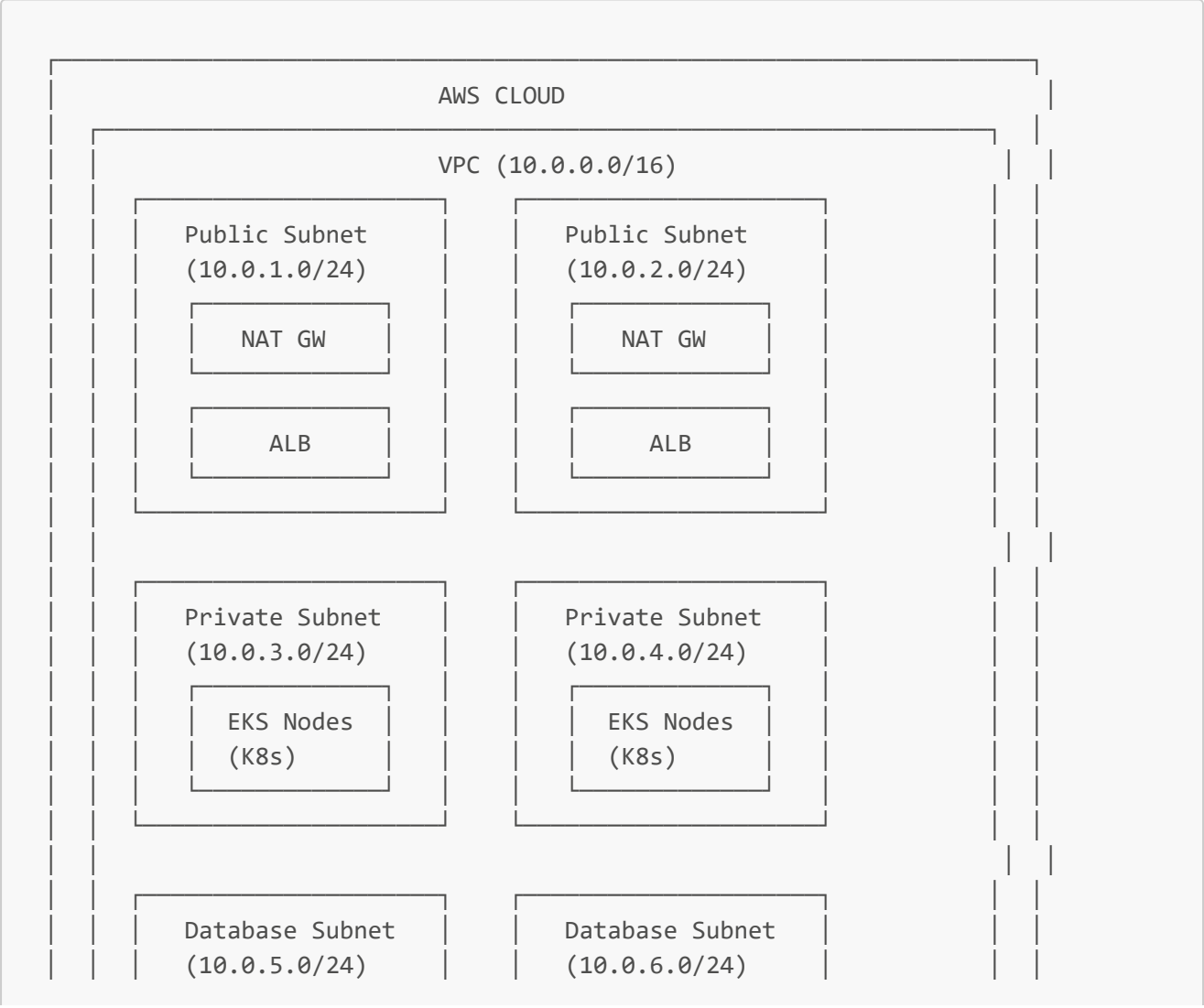
7.4 Identity & KYC

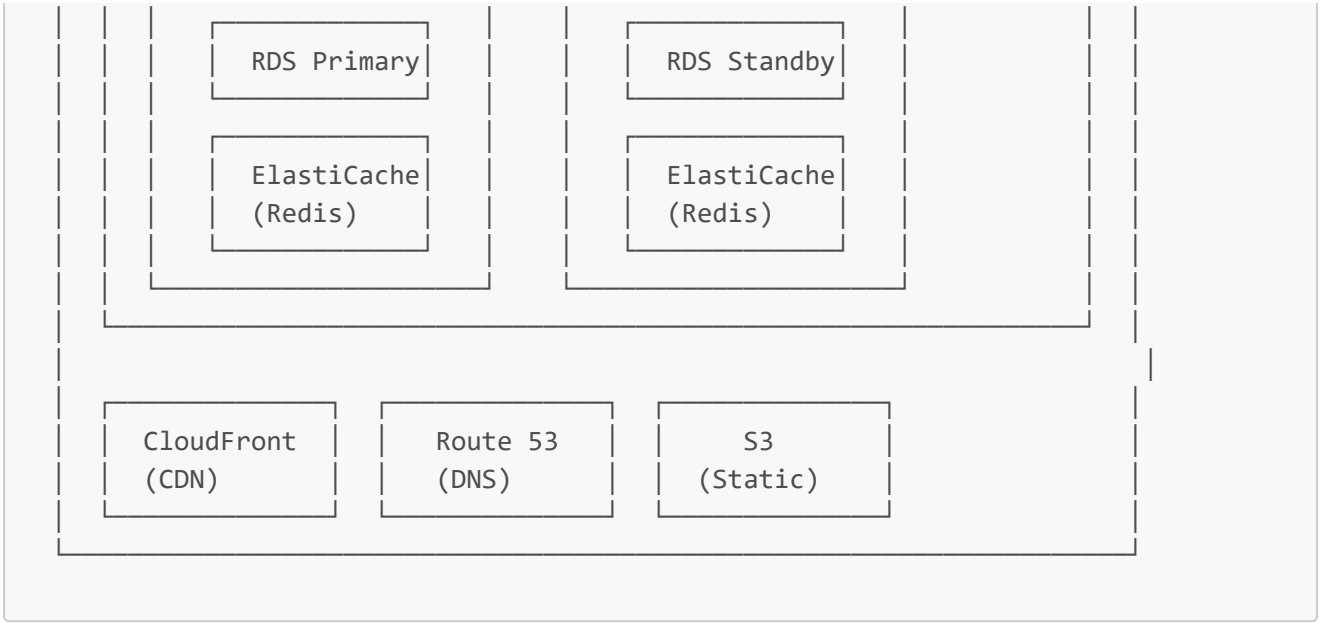
Service	Purpose
Smile Identity	Identity verification
Youverify	Business verification
CAC API	Company registration lookup

---

8. Infrastructure & DevOps

8.1 AWS Architecture





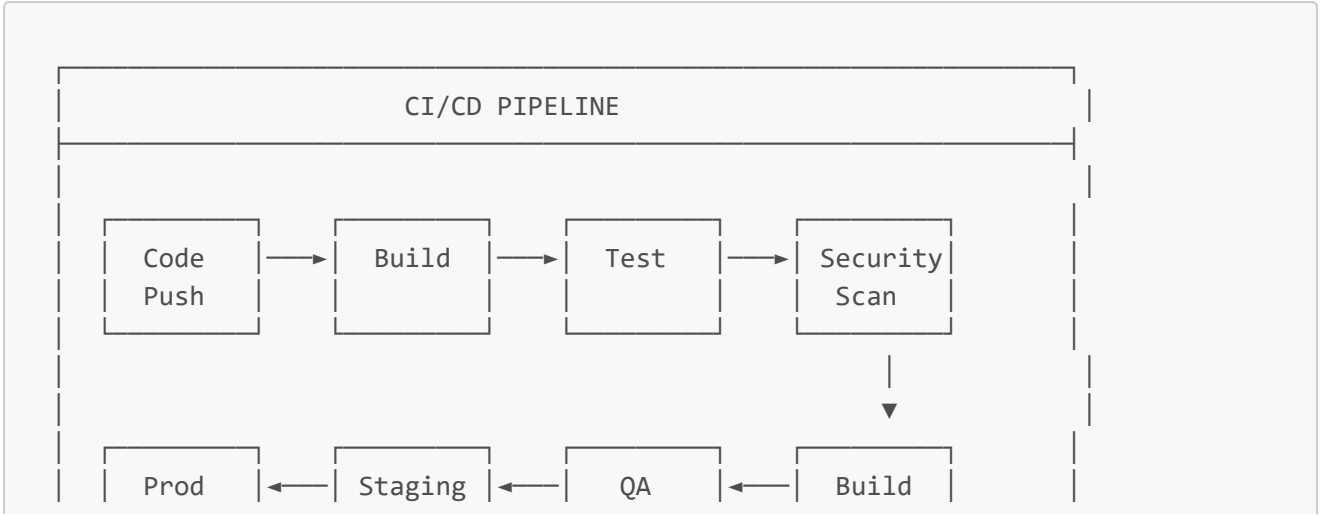
8.2 Simplified Infrastructure (Small Team)

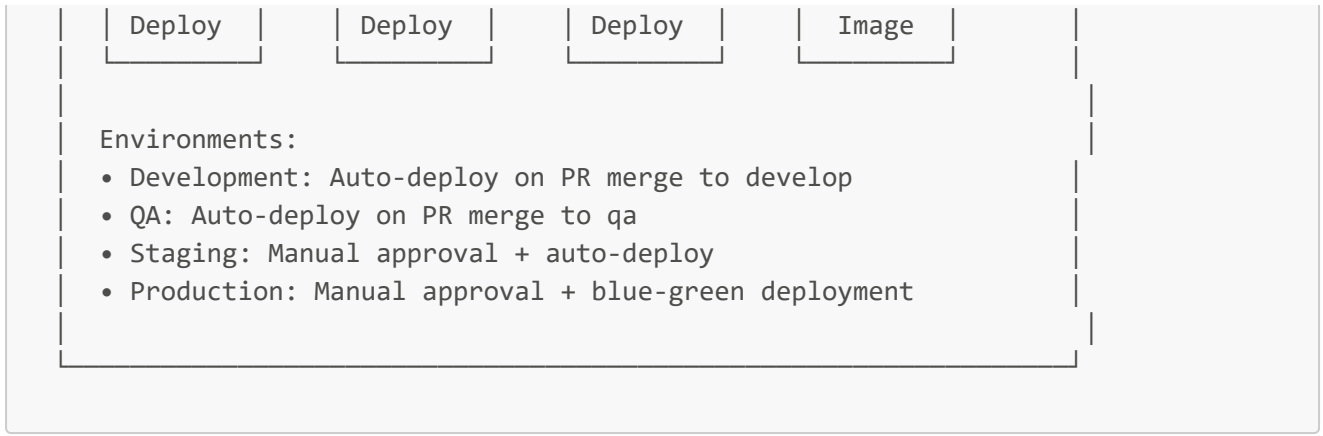
For a 5-person team, we recommend a simpler infrastructure using managed services:

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

Note: Kubernetes (EKS) can be adopted in Phase 2 when team scales beyond 10 engineers.

8.3 CI/CD Pipeline



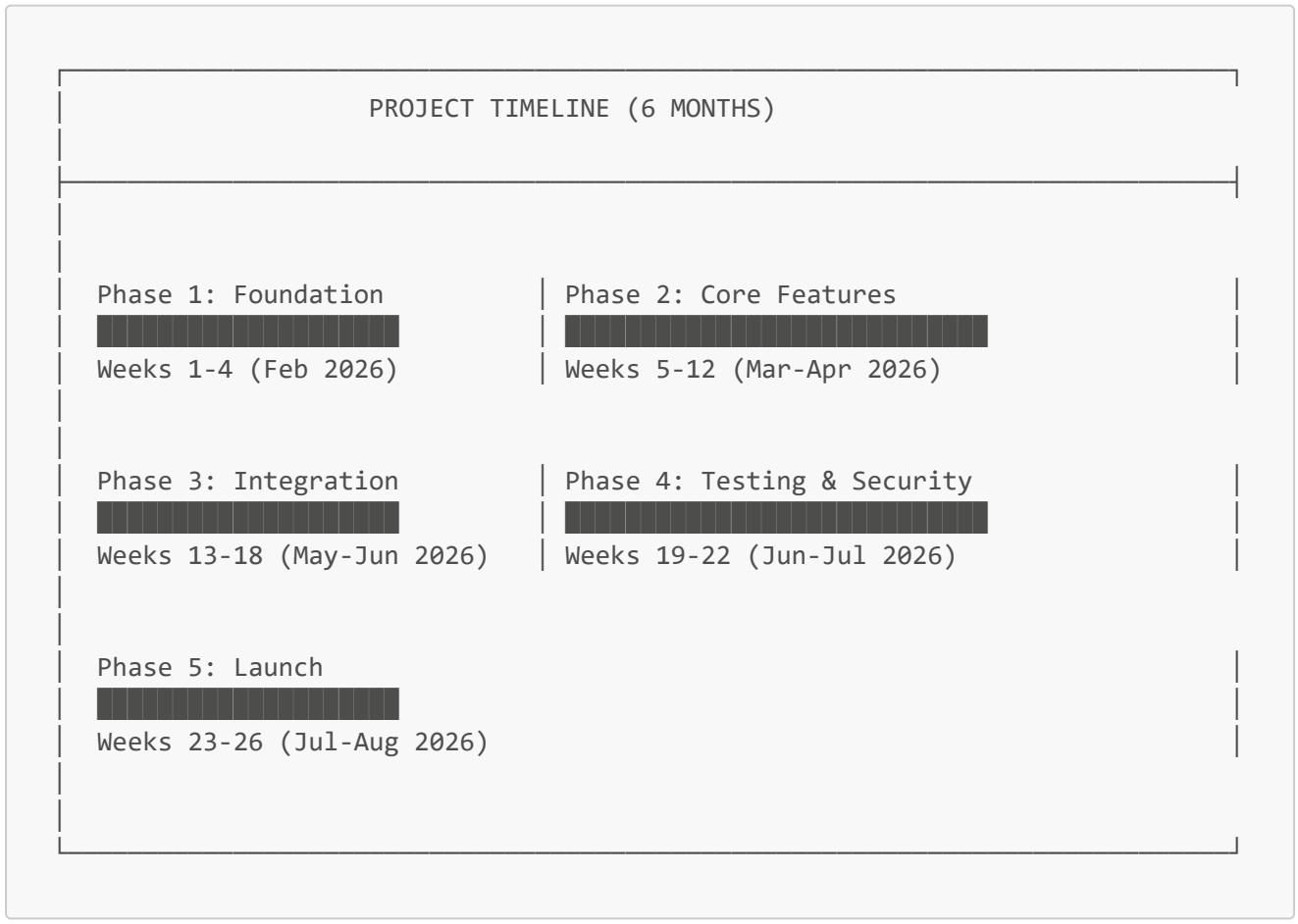


8.4 Environment Configuration

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



9.2 Detailed Phase Breakdown

Phase 1: Foundation (Weeks 1-4)

Timeline: February 1 - February 28, 2026

Week	Tasks	Deliverables
Week 1	Project setup, repo structure, CI/CD foundation	Git repo, basic pipeline
Week 2	Infrastructure setup (Terraform), database setup	AWS resources, DB schemas
Week 3	Authentication service, API gateway setup	Auth endpoints, Kong config
Week 4	Base UI components, design system	Component library, theme

Milestone: Development environment ready, team can start coding

---

Phase 2: Core Features (Weeks 5-12)

Timeline: March 1 - April 25, 2026

Week	Tasks	Deliverables
Week 5-6	Merchant onboarding, dashboard structure	Registration flow, dashboard UI
Week 7-8	Payment service core, transaction processing	Payment APIs, checkout page
Week 9-10	Card integration, tokenization	Card payment flow
Week 11-12	Settlement service, reconciliation	Settlement processing

Milestone: Core payment flow working end-to-end

Phase 3: Integration & Features (Weeks 13-18)

Timeline: April 26 - June 6, 2026

Week	Tasks	Deliverables
Week 13-14	Payment processor integration (Interswitch)	Live card processing
Week 15-16	Notification service, webhooks	Email/SMS, webhook delivery
Week 17-18	Analytics dashboard, reporting	Charts, reports, exports

Milestone: Feature complete for MVP

Phase 4: Testing & Security (Weeks 19-22)

Timeline: June 7 - July 4, 2026

Week	Tasks	Deliverables
Week 19	Security audit, penetration testing	Security report
Week 20	Performance testing, load testing	Performance benchmarks
Week 21	Bug fixes, optimization	Stable build
Week 22	PCI DSS compliance review	Compliance documentation

Milestone: System ready for production

Phase 5: Launch (Weeks 23-26)

Timeline: July 5 - August 1, 2026

Week	Tasks	Deliverables
Week 23	Staging deployment, UAT	Staging environment
Week 24	Production deployment, monitoring setup	Production environment
Week 25	Soft launch (beta merchants)	Beta testing
Week 26	Public launch, documentation	Go-live

Milestone: altpay live in production 🚀

9.3 Sprint Schedule

Sprint	Dates	Focus
Sprint 1	Feb 1-14	Foundation
Sprint 2	Feb 15-28	Auth & Infrastructure
Sprint 3	Mar 1-14	Merchant Module
Sprint 4	Mar 15-28	Payment Core
Sprint 5	Mar 29 - Apr 11	Payment Flow
Sprint 6	Apr 12-25	Settlement
Sprint 7	Apr 26 - May 9	Integrations
Sprint 8	May 10-23	Notifications
Sprint 9	May 24 - Jun 6	Analytics
Sprint 10	Jun 7-20	Testing
Sprint 11	Jun 21 - Jul 4	Security

Sprint	Dates	Focus
Sprint 12	Jul 5-18	Deployment
Sprint 13	Jul 19 - Aug 1	Launch

10. Team Structure

10.1 Required Team

Role	Count	Responsibilities
Tech Lead	1	Architecture, code review, technical decisions, DevOps, security oversight
Backend Engineer	1	Laravel API development, database, integrations, payment processing
Frontend Engineers	2	Next.js dashboard, checkout UI, responsive design, API integration
QA Engineer	1	Manual testing, test automation, quality assurance, UAT coordination

Total: 5 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

Backend Engineer

- Laravel API development
- Database design and optimization
- Payment processor integrations (Interswitch, NIBSS)
- Queue jobs and background processing
- Settlement and reconciliation logic

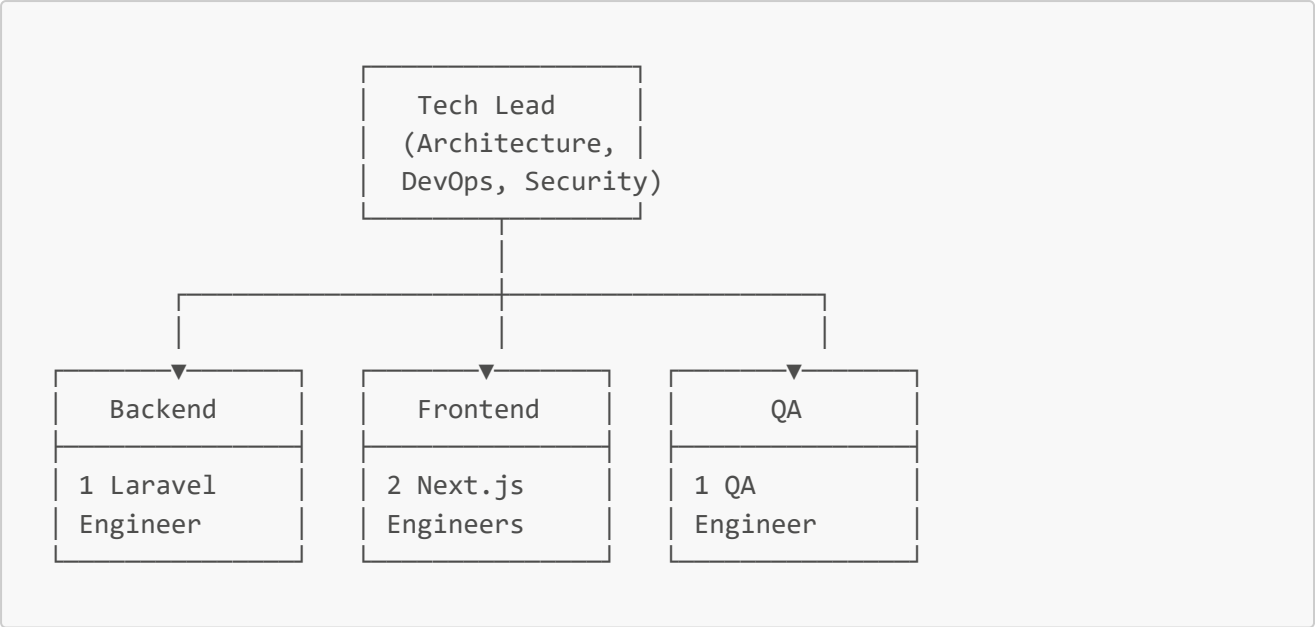
Frontend Engineers (2)

- **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



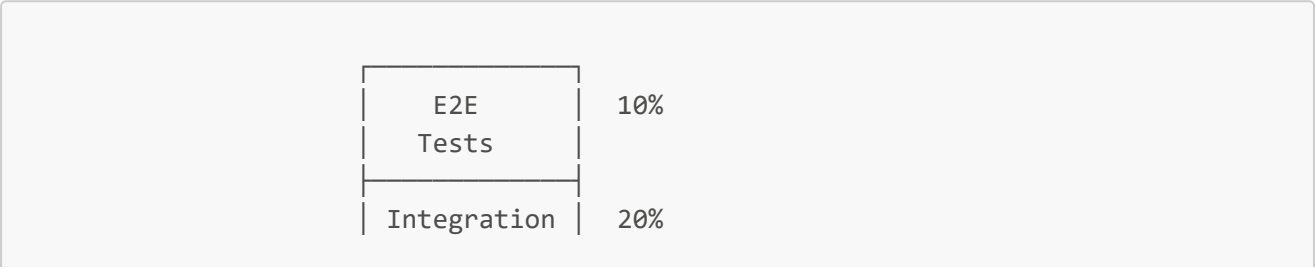
10.4 Communication & Workflow

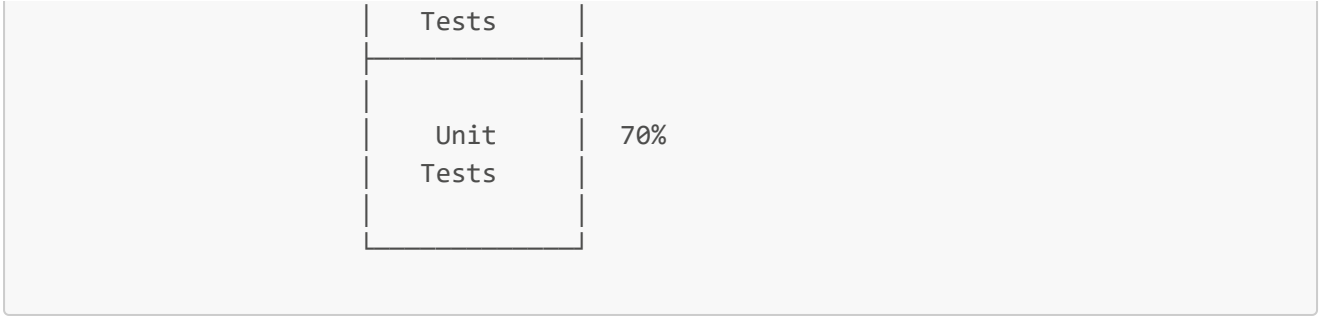
Activity	Frequency	Participants
Daily Standup	Daily (15 min)	All team
Sprint Planning	Bi-weekly	All team
Code Review	Ongoing	Tech Lead + relevant engineer
Sprint Retro	Bi-weekly	All team
Tech Sync	Weekly	Tech Lead + Engineers

---

11. Testing Strategy

11.1 Testing Pyramid





11.2 Testing Types

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

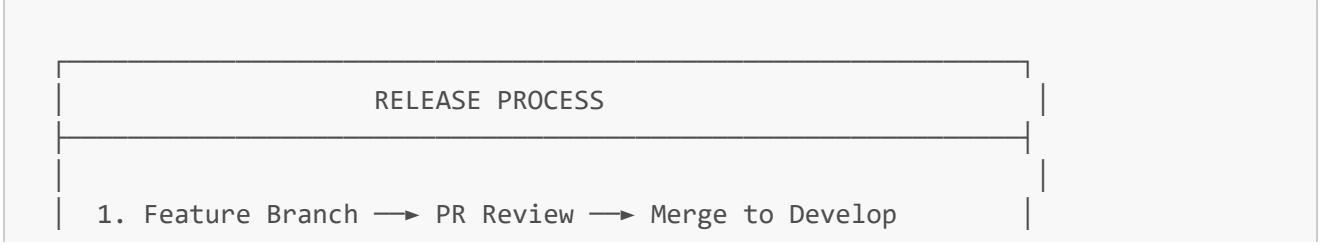
---

12. Deployment Strategy

12.1 Deployment Approach

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

12.2 Release Process



```
│                                                                 │
│  2. Develop ──► QA Deploy ──► QA Testing ──► Approval           │
│                                                                 │
│  3. Release Branch ──► Staging Deploy ──► UAT ──► Sign-off      │
│                                                                 │
│  4. Production Deploy ──► Smoke Tests ──► Monitor               │
│                                                                 │
│  5. If issues: Rollback ──► Hotfix ──► Re-deploy                │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

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
APM	Datadog / New Relic	Application performance
Logs	ELK Stack	Centralized logging
Metrics	Prometheus + Grafana	Custom metrics
Tracing	Jaeger	Distributed tracing
Alerts	PagerDuty	Incident management
Uptime	Pingdom	External monitoring

13.2 Key Metrics (SLIs/SLOs)

Metric	SLO	Alert Threshold
Availability	99.9%	< 99.5%
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
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