Fresh Roots Marketplace - Comprehensive System Design Document

Version: 1.0
Date: January 23, 2026
Project Type: Mobile-First Fresh Produce Marketplace for Mauritius
Target Timeline: 1-month prototype to production-ready MVP

Table of Contents

---

1. Executive Summary

1.1 Project Overview

Fresh Roots is a mobile-first marketplace designed to connect Mauritian consumers with fresh, locally-sourced produce. The platform addresses a clear market need: combining the quality and freshness of traditional markets with the convenience and speed expected from modern e-commerce applications. With Mauritius's online grocery delivery market projected to grow at 17.98% annually to reach US$145.30 million by 2029, Fresh Roots enters a market with strong growth tailwinds, high digital readiness (88.6% smartphone penetration), and significant consumer demand for convenience and quality [1].

The application serves two primary user groups:
- **Consumers:** Browse listings, express interest in products, submit purchase requests with multiple payment options (MCB Juice, credit cards, cash on delivery), and track order status
- **Admin:** Manage product listings, process purchase requests with approval/rejection workflow, import listings from Facebook, and monitor platform analytics

1.2 Key Technical Decisions

The recommended technology stack prioritizes rapid development, scalability to millions of users, and alignment with Mauritius's unique market characteristics. Core decisions include:

1. **Frontend:** React Native with Expo framework for mobile-first development, enabling rapid iteration via web previews and Expo Go testing on the target Samsung J7 Prime device, with a clear path to iOS support

2. **Backend:** Node.js with NestJS framework, providing exceptional performance for I/O-heavy operations, unified TypeScript development, and a modular architecture that can evolve from monolith to microservices

3. **Database:** Hybrid approach with PostgreSQL as the primary database for transactional data integrity and Redis for high-performance caching and real-time features

4. **Payment:** Integration via payment aggregators (MIPS or Paywise) to access MCB Juice, My.T Money, and credit cards through a single API, with cash on delivery as a complementary option

5. **Analytics:** PostHog for its all-in-one platform (analytics, A/B testing, session replay, feature flags), self-hosting capability for data ownership, and superior cost-effectiveness at scale (80% cheaper than Mixpanel for anonymous events)

6. **Infrastructure:** Cloudinary for MVP image hosting transitioning to AWS S3 + CloudFront at scale; AWS SES for email ($0.10/1,000 emails); Expo Push Notifications for MVP transitioning to OneSignal for advanced engagement

1.3 Development Timeline

The project follows an aggressive but achievable 4-week development timeline:

- **Week 0 (Planning):** Finalize design documents, set up development environment, establish partnerships with payment aggregators
- **Week 1 (Backend Foundation):** Implement database schema, authentication, admin CRUD for listings, and Facebook import prototype
- **Week 2 (Mobile App Core):** Build Expo app with navigation, listing browse/search, product detail screens, and express interest flow
- **Week 3 (Purchase & Admin):** Implement purchase request flow with payment integration, admin dashboard for order management, and notification system
- **Week 4 (Polish & Deploy):** UI/UX refinement, analytics integration, automated APK generation, QA testing, and deployment

1.4 Expected Outcomes

By the end of the 1-month prototype phase, Fresh Roots will deliver:

1. **Functional mobile application** testable via Expo Go on Android (Samsung J7 Prime) with signed APK for distribution

2. **Complete backend API** with all core functionalities: authentication, listing management, order processing, payment integration
3. **Admin capabilities** for managing listings (including Facebook imports), processing purchase requests, and viewing analytics
4. **Payment infrastructure** integrated with Mauritius-specific options (Juice, cash on delivery) and compliant with PCI DSS standards
5. **Analytics foundation** tracking key user behaviors and conversion funnels
6. **Production-ready deployment** with CI/CD pipelines, monitoring, and observability tools
7. **Comprehensive documentation** for deployment, API usage, and future development roadmap

1.5 Competitive Positioning

Fresh Roots differentiates itself from existing competitors through:

- **Superior freshness guarantees** via cold chain logistics and rapid delivery windows
- **Curated quality focus** rather than broad supermarket-style assortments
- **Seamless mobile experience** optimized for the Mauritian market
- **Direct farmer partnerships** supporting local agriculture and ensuring product traceability
- **Flexible payment options** including the dominant MCB Juice mobile wallet
- **Transparent sourcing** with visibility into product origins and farming practices

The platform is positioned between traditional markets (high freshness, low convenience) and supermarket delivery services (high convenience, lower freshness focus), targeting quality-conscious consumers willing to pay a modest premium for guaranteed freshness and superior user experience.

---

2. Complete Recommended Tech Stack

This section provides detailed justifications for each component of the technology stack, compiled from comprehensive research into best practices, cost-effectiveness, and suitability for the Mauritius market [2, 3].

2.1 Frontend Technologies

2.1.1 Mobile Framework: React Native with Expo

Recommendation: React Native 0.72+ with Expo SDK 50+

Justification:
- **Rapid Development:** Expo’s managed workflow abstracts native build complexities, providing instant testing via Expo Go on physical devices without APK builds during development. The web preview feature enables real-time iteration viewable in any browser, critical for the aggressive 1-month timeline [2]
- **Cross-Platform Efficiency:** A single TypeScript/JavaScript codebase serves both Android (primary focus) and iOS (future expansion), potentially reducing development and maintenance costs by 30-50% compared to native development
- **Developer Alignment:** Matches the developer’s stated preference, maximizing productivity and reducing the learning curve
- **Mature Ecosystem:** Extensive library of pre-built components, comprehensive documentation, and active community support for problem-solving
- **Production Path:** EAS Build provides a straightforward path from development to production APKs/AABs for Google Play Store and IPAs for Apple App Store

Configuration:

```
Framework: React Native 0.72+
SDK: Expo SDK 50+
Language: TypeScript 5.0+
Build System: EAS Build for production artifacts
Testing: Expo Go for development, physical device testing
```

2.1.2 Navigation: React Navigation 6.x

Recommendation: React Navigation 6.x

Justification:
- De-facto standard for React Native navigation with extensive documentation
- Supports all required navigation patterns: stack (drill-down), tab (main sections), drawer (optional future admin interface)
- Deep linking support for future marketing campaigns and push notification routing
- Customizable animations and transitions for polished UX

2.1.3 State Management: Redux Toolkit

Recommendation: Redux Toolkit 2.0+

Justification:
- **Scalability:** Redux provides predictable, centralized state management crucial for a marketplace with complex data flows (user auth, cart, orders, cached listings)
- **Redux Toolkit Benefits:** Reduces Redux boilerplate by 70-80% with opinionated utilities like `createSlice`, `createAsyncThunk`
- **DevTools:** Time-travel debugging and state inspection significantly accelerate development and bug fixing
- **Persistence:** Easy integration with Redux Persist for offline data caching
- **Team Scalability:** Enforces consistent patterns as team grows

Key State Slices:
- `auth`: User authentication state, JWT tokens, user profile
- `listings`: Product catalog, filters, search results
- `cart`: (Future) Shopping cart items and quantities
- `orders`: User's order history and status
- `ui`: App-wide UI state (loading, errors, modals)

2.1.4 UI Component Library: React Native Paper 5.x

Recommendation: React Native Paper 5.x

Justification:
- **Design System:** Implements Material Design 3, providing a modern, cohesive visual language
- **Component Library:** Comprehensive set of 50+ pre-built, accessible components (buttons, cards, text inputs, modals)
- **Theme System:** Centralized theming with dark mode support built-in
- **Customization:** Extensible for Fresh Roots branding while maintaining consistency
- **Active Maintenance:** Regular updates and strong community support

2.1.5 Offline Support & Caching

Recommendation: Redux Persist + Custom Caching Layer

Implementation Strategy:

```
Layer 1: AsyncStorage (via Redux Persist)
- Persist user authentication tokens
- Store user preferences and settings
- Cache small configuration data

Layer 2: Application-Level Cache
- Cache product listing responses with 15-minute TTL
- Implement stale-while-revalidate pattern
- Store last 50 viewed product details
- Queue failed API requests for retry
```

Justification:
- Mauritius has generally good connectivity (85.8% internet penetration), but ensures app feels fast even on intermittent 3G connections
- Reduces API load and improves perceived performance with instant rendering from cache
- Enables basic browsing functionality during brief network interruptions

2.2 Backend Technologies

2.2.1 Runtime & Framework: Node.js with NestJS

Recommendation: Node.js 20 LTS with NestJS 10.x

Justification:
- **Performance:** Node.js's non-blocking, event-driven architecture is optimal for I/O-heavy marketplace operations (database queries, API calls to payment gateways, Facebook API). Benchmarks show Node.js handles 10,000+ concurrent connections with 2-3x lower memory than Python Django [4]
- **Unified Language:** TypeScript across frontend and backend enables code sharing (types, interfaces, validation schemas), reducing development time by 20-30%
- **NestJS Architecture:** Enforces modular, testable architecture from day one. Dependency injection, decorators, and clear separation of concerns make it easy to scale from monolith to microservices
- **Real-Time Capabilities:** Native WebSocket support via Socket.IO integration for real-time admin dashboard updates
- **Ecosystem:** World's largest package registry (npm) with solutions for every common problem

NestJS Modules:

```
@nestjs/core: Core framework
@nestjs/common: Common utilities, decorators
@nestjs/config: Environment configuration
@nestjs/typeorm: Database ORM integration
@nestjs/jwt: JWT authentication
@nestjs/passport: Authentication strategies
@nestjs/swagger: API documentation generation
@nestjs/schedule: Cron jobs (e.g., Facebook sync)
@nestjs/websockets: Real-time features
```

Alternative Considered: Python with Django
- **Rejected Because:** Django's synchronous nature requires additional complexity (Celery, Channels) for real-time features. Node.js's async-first design is more natural fit for marketplace operations. NestJS provides better TypeScript alignment with React Native frontend

2.2.2 API Architecture: RESTful API

Recommendation: REST over GraphQL for MVP

Justification:
- **Simplicity:** REST's resource-oriented design maps naturally to marketplace entities (GET /listings, POST /orders). Simpler for mobile client developers to consume
- **Tooling:** Mature ecosystem for documentation (Swagger/OpenAPI), testing (Postman), monitoring, and caching
- **Caching:** HTTP-level caching with Redis is straightforward for REST endpoints
- **Performance:** Well-designed REST endpoints with proper field selection can match GraphQL efficiency for marketplace use cases

API Versioning Strategy:

```
Base URL: https://api.freshroots.mu/v1
Versioning: URL path versioning (/v1, /v2)
Deprecation: 6-month notice for breaking changes
```

Future Consideration: GraphQL for complex client needs
- Defer to post-MVP if mobile clients require highly dynamic, nested data queries
- Would enable admin dashboard to fetch precisely needed data for complex reports

2.3 Database & Data Management

2.3.1 Primary Database: PostgreSQL 15+

Recommendation: PostgreSQL 15+ (managed service: AWS RDS, Google Cloud SQL, or DigitalOcean Managed Databases)

Justification:
- **ACID Compliance:** Non-negotiable for financial transactions. PostgreSQL ensures order integrity, prevents double-booking of inventory, and maintains referential integrity between users, orders, and listings
- **Relational Model:** Marketplace data is inherently relational (users place orders containing order items referencing listings). PostgreSQL's foreign keys, joins, and transactions handle this naturally [5]
- **JSON Support:** JSONB columns provide flexibility for semi-structured data (product metadata, tags, Facebook post data) within the relational model
- **Performance:** Excellent query optimizer, support for complex aggregations needed for admin analytics, full-text search for product listings
- **Scalability:** Proven horizontal scaling via read replicas, connection pooling, and partitioning for multi-million user scale
- **Open Source:** No vendor lock-in, extensive community knowledge, compatibility with all major cloud providers

Configuration for Production:

```
Version: PostgreSQL 15.x
Instance Size: Start with 2 vCPU, 4GB RAM (scales to 8+ vCPU)
Storage: SSD-backed, auto-scaling up to 1TB
Backups: Daily automated snapshots, 7-day retention
High Availability: Multi-AZ deployment for 99.95% uptime
Connection Pooling: PgBouncer (100-500 connections)
```

Alternative Considered: MongoDB
- **Rejected Because:** Lack of enforced relationships creates risk of data inconsistencies in e-commerce transactions. Application-level integrity checks are error-prone. PostgreSQL's constraints provide safety net.

2.3.2 Caching Layer: Redis 7.x

Recommendation: Redis 7.x (managed service: AWS ElastiCache, Redis Labs, or DigitalOcean)

Justification:
- **Performance:** In-memory storage provides sub-millisecond read/write latency, reducing database load by 60-80% for frequently accessed data
- **Versatility:** Single platform for multiple use cases: API response caching, session storage, rate limiting, pub/sub for real-time admin notifications
- **Data Structures:** Rich data types (strings, hashes, sorted sets) optimize different caching patterns
- **Persistence:** Optional RDB snapshots ensure cached data survives restarts

Use Cases in Fresh Roots:

```
1. API Response Caching
   Key: listing:all:page:1
   Value: JSON response of listings
   TTL: 15 minutes
   Impact: 85% of listing requests served from cache

2. Session Storage
   Key: session:{userId}
   Value: JWT refresh token
   TTL: 30 days
   Impact: Stateless JWT validation with logout capability

3. Rate Limiting
   Key: ratelimit:{ip}:{endpoint}
   Value: Request count
   TTL: 15 minutes
   Impact: Prevent abuse, ensure fair access

4. Real-Time Pub/Sub
   Channel: admin:orders:new
   Message: New order notification
   Impact: Instant admin dashboard updates
```

Cache Invalidation Strategy:
- **Time-based (TTL):** Listings cache expires after 15 minutes
- **Event-based:** Invalidate specific listing cache on admin update/delete
- **Tag-based:** Invalidate category cache when any product in category changes

2.4 Infrastructure & Cloud Services

2.4.1 Hosting Platform

Recommendation: AWS (Primary), with DigitalOcean as cost-effective alternative

Justification:
- **AWS Pros:** Comprehensive service ecosystem, excellent managed services (RDS, ElastiCache, S3, CloudFront), superior auto-scaling, global infrastructure for future expansion
- **DigitalOcean Pros:** 50-60% lower costs for MVP phase, simpler interfaces, faster setup, excellent

for single-region deployment
- **Hybrid Approach:** Start on DigitalOcean for cost efficiency, migrate to AWS as scale demands

MVP Hosting Architecture (DigitalOcean):

```
Backend: App Platform (PaaS)
- 1-2 containers, auto-scaling
- $12-24/month per instance

Database: Managed PostgreSQL
- Basic plan: $15/month
- Production plan: $55/month (2GB RAM)

Cache: Managed Redis
- Basic plan: $15/month

Object Storage: Spaces (S3-compatible)
- $5/month + $0.01/GB transfer

CDN: Spaces CDN
- Included with Spaces

Total MVP Cost: ~$60-100/month
```

Production Scaling Architecture (AWS):

```
Compute: ECS Fargate or EC2 Auto Scaling Group
Database: RDS PostgreSQL Multi-AZ
Cache: ElastiCache Redis Cluster
Storage: S3 + CloudFront CDN
Load Balancer: Application Load Balancer (ALB)
Secrets: AWS Secrets Manager
Monitoring: CloudWatch + X-Ray
```

2.4.2 File Storage & CDN

Recommendation: Phase 1: Cloudinary | Phase 2: AWS S3 + CloudFront + Lambda@Edge

Phase 1 - Cloudinary (MVP - Months 1-6)

Justification:
- **Zero Setup:** Upload API works out-of-box, no infrastructure management
- **Free Tier:** 25 credits/month = 25GB storage + 25GB bandwidth + 25,000 transformations, sufficient for MVP with 100-500 products [6]
- **Automatic Optimization:** Serves WebP to Chrome/Android, AVIF to modern browsers, JPEG fallback for older devices—without manual configuration
- **Responsive Images:** Single URL with transformation parameters generates thumbnails, mobile, and desktop sizes on-the-fly
- **Fast Integration:** React Native SDK and Node.js SDK enable integration in < 1 day

Example Transformation URLs:

```
Original: https://upload.wikimedia.org/wikipedia/commons/8/88/
Bright_red_tomato_and_cross_section02.jpg

Thumbnail: .../c_fill,w_150,h_150,f_auto,q_auto/products/tomato_001.jpg
Mobile: .../c_scale,w_800,f_auto,q_auto,dpr_2.0/products/tomato_001.jpg
Desktop: .../c_scale,w_1600,f_auto,q_80/products/tomato_001.jpg
```

Cost Projection:
- 0-1,000 products: Free tier sufficient
- 1,000-5,000 products with 10K monthly active users: ~$50-100/month
- Beyond 5,000 products / 50K users: Consider migration to Phase 2

Phase 2 - AWS S3 + CloudFront (Scale - Beyond Month 6)

Migration Trigger: Monthly bandwidth > 2TB or Cloudinary costs > $200/month

Justification:
- **Cost Savings:** At 10TB/month, AWS solution costs ~$1,000/month vs. Cloudinary's ~$5,000-8,000/month (60-85% savings) [7]
- **Control:** Full ownership of transformation logic, caching rules, and edge behaviors
- **Performance:** CloudFront has 450+ edge locations, including Africa presence, though Cloudinary's CDN includes a Mauritius PoP via Cloudflare partnership [8]

Implementation:

```
Storage: S3 bucket (private) - $0.023/GB
CDN: CloudFront distribution - $0.085/GB transfer (first 10TB)
Transformations: Lambda@Edge functions - $0.60 per 1M requests
Smart Caching: Cache-Control headers, ETags, versioned URLs

Image Processing: Sharp library in Lambda@Edge
- Resize, crop, format conversion (WebP, AVIF)
- Quality optimization based on device hints
- Execution time: 50-200ms
```

2.4.3 Domain & SSL

Recommendation: Custom domain (freshroots.mu) with Let's Encrypt SSL

Setup:

```
Domain Registrar: .mu domain from Internet Direct (Mauritius registrar)
Cost: ~$25-40/year for .mu domain

DNS: Cloudflare (free plan)
- Global anycast DNS
- DDoS protection
- Free SSL certificate (Let's Encrypt)
- DNSSEC support

SSL: Auto-renewing Let's Encrypt certificates via Cloudflare or Certbot
Cost: $0
```

2.5 Authentication & Security Services

2.5.1 Authentication: JWT (JSON Web Tokens)

Recommendation: JWT-based authentication with refresh token pattern

Implementation:

```
// Token Structure
Access Token:
  - Payload: { userId, role, email }
  - Expiration: 15 minutes
  - Storage: Mobile app memory (Redux state, never persisted)
  - Usage: Every API request in Authorization header

Refresh Token:
  - Payload: { userId, tokenId }
  - Expiration: 30 days
  - Storage: Secure device storage (Expo SecureStore), Redis server-side
  - Usage: Obtain new access token when expired
```

Security Features:
- **Short-lived Access Tokens:** 15-minute expiry limits damage if token is compromised
- **Refresh Token Rotation:** Each refresh issues new refresh token and invalidates old one
- **Token Revocation:** Store refresh token IDs in Redis; delete on logout for true logout capability
- **Device Binding (Future):** Link refresh token to device fingerprint to prevent token theft

Password Security:

```
Hashing: bcrypt with cost factor 12 (2^12 iterations)
Rationale: Bcrypt is specifically designed to resist brute-force attacks
          Cost factor 12 provides ~300ms hash time (balance security/UX)

Salt: bcrypt generates unique salt per password automatically
Storage: Store only bcrypt hash, never plaintext password
```

2.5.2 Future: OAuth 2.0 Social Login

Recommendation: Integrate Google and Facebook OAuth post-MVP

Justification:
- **User Convenience:** 40-60% faster registration, 20-30% lower abandonment rates
- **Trust:** Users more comfortable sharing existing social credentials
- **Profile Data:** Pre-populate name, email, profile photo from social accounts

Implementation Priority:
1. **Month 2:** Google OAuth (highest usage in Mauritius)
2. **Month 3:** Facebook OAuth (leverages existing Facebook page relationship)

Libraries:
- Frontend: `expo-auth-session` for OAuth flows
- Backend: `@nestjs/passport` with `passport-google-oauth20`, `passport-facebook`

2.6 External Service Integration

2.6.1 Payment Processing

Recommendation: MIPS or Paywise payment aggregator

Detailed in Section 7

2.6.2 Email Service: AWS SES

Recommendation: Amazon Simple Email Service (SES)

Justification:
- **Cost Efficiency:** $0.10 per 1,000 emails = 95% cheaper than SendGrid ($0.80/1,000) and 90% cheaper than Mailgun
- **Free Tier:** 62,000 emails/month free when hosted on AWS EC2 (or 3,000/month from any host)
- **Deliverability:** Enterprise-grade infrastructure; requires proper DKIM/SPF setup but achieves >98% inbox placement
- **Scalability:** No upper limits; scales to millions of emails

Setup Requirements:
- Request production access (default: sandbox mode, can only send to verified addresses)
- Configure DKIM and SPF records for domain authentication
- Set up dedicated IP ($24.95/month) once sending >50,000 emails/month for reputation control
- Implement bounce and complaint handling webhooks

Email Types:

```
Transactional (High Priority):
- Order confirmation
- Payment receipt
- Admin order alert
- Password reset
- Account verification

Marketing (Future):
- Weekly produce newsletter
- Seasonal promotions
- Abandoned cart reminders
```

Alternative: Mailgun if AWS setup is too complex
- **Trade-off:** 8x cost vs. SES, but simpler setup and better out-of-box analytics
- **Recommendation:** Start with SES, fall back to Mailgun only if deliverability issues arise

2.6.3 Push Notifications

Recommendation: Phase 1: Expo Push Notifications | Phase 2: OneSignal

Detailed in Section 9

2.6.4 Analytics: PostHog

Recommendation: PostHog Cloud (MVP) transitioning to Self-Hosted (Scale)

Detailed in Section 10

2.7 Development Tools & Monitoring

2.7.1 Version Control & Collaboration

Recommendation:
- **Git Hosting:** GitHub (private repositories)
- **Branching Strategy:** Git Flow (main, develop, feature/, release/, hotfix/)

- Code Review: Mandatory pull request reviews for all main/develop merges
- Documentation:* README.md, API documentation in `/docs`, architectural decision records (ADRs)

2.7.2 Error Tracking: Sentry

Recommendation: Sentry for both frontend and backend error tracking

Justification:
- **Real-Time Alerts:** Instant Slack/email notifications for critical errors
- **Context:** Captures full stack traces, user context, device info, breadcrumbs leading to error
- **Release Tracking:** Associate errors with specific app versions and track error introduction/resolution
- **Performance Monitoring:** Tracks API endpoint response times, database query performance
- **Source Maps:** Deobfuscates minified production JavaScript for readable stack traces

Free Tier: 5,000 events/month (sufficient for MVP)

Integration:

```
// React Native
import * as Sentry from '@sentry/react-native';
Sentry.init({ dsn: process.env.SENTRY_DSN_MOBILE });

// NestJS Backend
import * as Sentry from '@sentry/node';
Sentry.init({ dsn: process.env.SENTRY_DSN_BACKEND });
```

2.7.3 Logging: Winston

Recommendation: Winston for structured backend logging

Configuration:

```
import winston from 'winston';

const logger = winston.createLogger({
  level: process.env.LOG_LEVEL || 'info',
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.json()
  ),
  transports: [
    new winston.transports.Console(),
    new winston.transports.File({ filename: 'error.log', level: 'error' }),
    new winston.transports.File({ filename: 'combined.log' })
  ]
});

// Log aggregation to CloudWatch or ELK Stack in production
```

Log Levels:
- **error:** Failed operations requiring immediate attention
- **warn:** Degraded functionality, fallback behaviors activated
- **info:** Key business events (order created, payment completed)
- **debug:** Detailed debugging information (disabled in production)

2.7.4 API Documentation: Swagger/OpenAPI

Recommendation: Swagger UI generated from NestJS decorators

Justification:
- **Auto-Generated:** NestJS's `@nestjs/swagger` package generates OpenAPI spec from code annotations
- **Always Sync:** Documentation lives alongside code, impossible to drift out-of-sync
- **Interactive:** Swagger UI allows API testing directly from browser
- **Client Generation:** OpenAPI spec can generate TypeScript SDK for React Native client

Access: `https://api.freshroots.mu/api/docs`

2.8 Testing Tools

Detailed in Section 14

2.9 CI/CD

Detailed in Section 15

2.10 Tech Stack Summary Table

Category	Technology	Justification	Cost (Monthly)
Frontend	React Native + Expo	Rapid dev, cross-platform, developer preference	$0
Mobile UI	React Native Paper	Material Design 3, comprehensive components	$0
State Management	Redux Toolkit	Predictable, scalable, excellent DevTools	$0
Navigation	React Navigation	Industry standard, deep linking support	$0
Backend	Node.js + NestJS	High concurrency, TypeScript, modular	$0
API Style	REST	Simple, mature tooling, cacheable	$0
Primary Database	PostgreSQL 15+	ACID compliance, relational integrity	$15-55
Cache	Redis 7+	Sub-ms latency, pub/sub, versatile	$15
Hosting	DigitalOcean/AWS	Managed services, scalability	$60-100 (MVP)
Images (MVP)	Cloudinary	Zero setup, free tier, auto-optimization	$0-50
Images (Scale)	S3 + CloudFront	60-85% cost savings at high traffic	$175+
Email	AWS SES	$0.10/1,000 emails, enterprise deliverability	$0-20
Push (MVP)	Expo Push	Zero config for Expo apps	$0
Push (Scale)	OneSignal	Advanced targeting, campaigns	$0-39+
Analytics	PostHog		$0-310

Category	Technology	Justification	Cost (Monthly)
		All-in-one, self-hostable, cost-effective	
Error Tracking	Sentry	Real-time, context-rich, release tracking	$0
Logging	Winston	Structured, cloud-agnostic	$0
CI/CD	GitHub Actions	Native GitHub integration, free tier	$0
Domain	freshroots.mu	Local domain	$2-3
SSL/CDN	Cloudflare	DDoS protection, free SSL	$0
Payments	MIPS/Paywise	Local payment methods, PCI compliance	Transaction fees
		MVP Total	**~$100-200/mo**
		Production (10K users)	**~$400-700/mo**

---

3. System Architecture Diagram

The complete system architecture diagram has been created as a separate file for clarity and ease of reference.

File Location: `/home/ubuntu/fresh_roots_architecture_diagram.txt`

This diagram includes:
- **Client Layer:** React Native mobile app and future admin web dashboard
- **API Gateway:** Load balancing, SSL termination, rate limiting, CORS
- **Application Layer:** NestJS modular architecture with dedicated modules for auth, listings, orders, payments, Facebook integration, notifications, and analytics
- **Data Layer:** PostgreSQL for persistent storage, Redis for caching and real-time features
- **External Services:** Payment gateways, Facebook Graph API, push/email providers, CDN
- **Data Flow Diagrams:** Visual representation of user browsing, purchase flow, and admin approval workflows
- **Scalability Plan:** Architecture evolution from MVP (single instance) to enterprise scale (auto-scaling, multi-region)

Please refer to the architecture diagram file for detailed component interactions and data flows.

---

4. Detailed Database Schema

This section defines the complete PostgreSQL database schema for Fresh Roots, including all tables, relationships, indexes, and constraints necessary for core functionality and future scalability.

4.1 Schema Design Principles

1. **Normalization:** Third Normal Form (3NF) to eliminate redundancy while maintaining query performance
2. **Referential Integrity:** Foreign keys with appropriate ON DELETE and ON UPDATE actions
3. **Indexing Strategy:** Indexes on foreign keys, frequently queried fields, and composite indexes for complex queries
4. **Timestamps:** `created_at` and `updated_at` on all major entities for audit trails and debugging
5. **Soft Deletes (Future):** Add `deleted_at` timestamp for soft deletion instead of hard DELETE operations
6. **JSON Flexibility:** JSONB columns for semi-structured data (product metadata, Facebook post data)

4.2 Core Tables

4.2.1 Users Table

Purpose: Store all user accounts (customers and admins)

```
CREATE TABLE users (
  id                UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  email             VARCHAR(255) NOT NULL UNIQUE,
  password_hash     VARCHAR(255) NOT NULL,
  phone             VARCHAR(20),
  name              VARCHAR(255) NOT NULL,
  role              VARCHAR(20) NOT NULL CHECK (role IN ('customer', 'admin')),
  email_verified    BOOLEAN DEFAULT FALSE,
  phone_verified    BOOLEAN DEFAULT FALSE,
  profile_image_url TEXT,
  is_active         BOOLEAN DEFAULT TRUE,
  created_at        TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  updated_at        TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Indexes
CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_users_role ON users(role);
CREATE INDEX idx_users_created_at ON users(created_at);

-- Comments
COMMENT ON TABLE users IS 'All user accounts including customers and administrators';
COMMENT ON COLUMN users.role IS 'User role: customer (default) or admin';
COMMENT ON COLUMN users.password_hash IS 'bcrypt hash of password, never store plain-
text';
```

Sample Data:

```sql
-- Admin user
INSERT INTO users (email, password_hash, name, role) VALUES
('admin@freshroots.mu', '$2b$12$...', 'Fresh Roots Admin', 'admin');

-- Customer user
INSERT INTO users (email, password_hash, phone, name, role) VALUES
('customer@example.com', '$2b$12$...', '+230 5123 4567', 'John Doe', 'customer');
```

4.2.2 Listings Table

Purpose: Product listings created by admin

```sql
CREATE TABLE listings (
  id                  UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  title               VARCHAR(255) NOT NULL,
  description         TEXT,
  price               DECIMAL(10, 2) NOT NULL CHECK (price >= 0),
  currency            VARCHAR(3) DEFAULT 'MUR',
  unit                VARCHAR(20) NOT NULL, -- 'kg', 'pack', 'piece', 'bunch', etc.
  stock_quantity      INTEGER DEFAULT 0 CHECK (stock_quantity >= 0),
  location            VARCHAR(255), -- Farm location or region
  category            VARCHAR(100), -- 'vegetables', 'fruits', 'herbs', 'dairy', etc.
  tags                TEXT[], -- Array of tags: ['organic', 'hydroponic', 'local']
  is_available        BOOLEAN DEFAULT TRUE,
  facebook_post_id    VARCHAR(255), -- Reference to source Facebook post
  facebook_post_url   TEXT,
  facebook_synced_at  TIMESTAMP WITH TIME ZONE,
  metadata            JSONB, -- Flexible storage for additional attributes
  admin_id            UUID NOT NULL REFERENCES users(id) ON DELETE RESTRICT,
  created_at          TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  updated_at          TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Indexes
CREATE INDEX idx_listings_admin_id ON listings(admin_id);
CREATE INDEX idx_listings_category ON listings(category);
CREATE INDEX idx_listings_is_available ON listings(is_available);
CREATE INDEX idx_listings_created_at ON listings(created_at DESC);
CREATE INDEX idx_listings_price ON listings(price);
CREATE INDEX idx_listings_facebook_post_id ON listings(facebook_post_id);
CREATE INDEX idx_listings_tags ON listings USING GIN(tags); -- GIN index for array
queries

-- Full-text search index
CREATE INDEX idx_listings_search ON listings
  USING GIN(to_tsvector('english', title || ' ' || COALESCE(description, '')));

COMMENT ON TABLE listings IS 'Product listings for fresh produce and goods';
COMMENT ON COLUMN listings.metadata IS 'JSON field for flexible attributes: {farm-
ing_method, harvest_date, certifications}';
```

Sample Data:

```
INSERT INTO listings (title, description, price, unit, stock_quantity, location, cat-
egory, tags, admin_id) VALUES
(
  'Organic Tomatoes',
  'Fresh organic tomatoes harvested this morning from Plaine Magnien',
  85.00,
  'kg',
  50,
  'Plaine Magnien',
  'vegetables',
  ARRAY['organic', 'local', 'fresh'],
  '00000000-0000-0000-0000-000000000001' -- admin UUID
);
```

4.2.3 Listing Images Table

Purpose: Store multiple images per listing with ordering

```
CREATE TABLE listing_images (
  id          UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  listing_id  UUID NOT NULL REFERENCES listings(id) ON DELETE CASCADE,
  image_url   TEXT NOT NULL,
  alt_text    VARCHAR(255),
  display_order INTEGER DEFAULT 0,
  created_at  TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Indexes
CREATE INDEX idx_listing_images_listing_id ON listing_images(listing_id, dis-
play_order);

COMMENT ON TABLE listing_images IS 'Multiple images per listing, ordered for carousel
display';
COMMENT ON COLUMN listing_images.display_order IS
'Order of image display, 0 = primary image';
```

4.2.4 Orders Table

Purpose: Purchase requests from customers

```sql
CREATE TYPE order_status AS ENUM (
  'pending',           -- Initial state after submission
  'payment_pending',   -- Awaiting payment confirmation
  'payment_confirmed', -- Payment verified, awaiting admin approval
  'approved',          -- Admin approved, ready for fulfillment
  'rejected',          -- Admin rejected
  'preparing',         -- Order being prepared
  'out_for_delivery',  -- In transit
  'delivered',         -- Successfully delivered
  'cancelled'          -- Cancelled by customer or admin
);

CREATE TYPE payment_method AS ENUM (
  'juice',             -- MCB Juice mobile payment
  'myt_money',         -- My.T Money
  'card',              -- Credit/Debit card
  'cash_on_delivery'   -- COD
);

CREATE TABLE orders (
  id                UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  order_number      VARCHAR(20) UNIQUE NOT NULL, -- Human-readable: FR-2026-001234
  user_id           UUID NOT NULL REFERENCES users(id) ON DELETE RESTRICT,
  status            order_status DEFAULT 'pending',
  payment_method    payment_method NOT NULL,
  payment_status    VARCHAR(20) DEFAULT 'pending', -- pending, completed, failed, re-
funded
  payment_transaction_id TEXT, -- External payment provider transaction ID
  subtotal_amount   DECIMAL(10, 2) NOT NULL CHECK (subtotal_amount >= 0),
  delivery_fee      DECIMAL(10, 2) DEFAULT 0,
  cod_fee           DECIMAL(10, 2) DEFAULT 0, -- Cash on delivery fee if applicable
  total_amount      DECIMAL(10, 2) NOT NULL CHECK (total_amount >= 0),
  currency          VARCHAR(3) DEFAULT 'MUR',
  delivery_address  JSONB, -- {line1, line2, city, postal_code, phone, notes}
  notes             TEXT, -- Customer notes or special instructions
  created_at        TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  updated_at        TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Indexes
CREATE INDEX idx_orders_user_id ON orders(user_id, created_at DESC);
CREATE INDEX idx_orders_status ON orders(status, created_at DESC);
CREATE INDEX idx_orders_order_number ON orders(order_number);
CREATE INDEX idx_orders_created_at ON orders(created_at DESC);

-- Sequence for order numbers
CREATE SEQUENCE order_number_seq START 1000;

COMMENT ON TABLE orders IS 'Purchase requests from customers';
COMMENT ON COLUMN orders.order_number IS 'Human-readable order identifier:
FR-2026-001234';
```

4.2.5 Order Items Table

Purpose: Line items within an order

```sql
CREATE TABLE order_items (
  id                      UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  order_id                UUID NOT NULL REFERENCES orders(id) ON DELETE CASCADE,
  listing_id              UUID NOT NULL REFERENCES listings(id) ON DELETE RESTRICT,
  quantity                INTEGER NOT NULL CHECK (quantity > 0),
  unit_price_at_purchase  DECIMAL(10, 2) NOT NULL, -- Lock price at time of order
  unit                    VARCHAR(20) NOT NULL,
  subtotal                DECIMAL(10, 2) NOT NULL,
  listing_title_snapshot  VARCHAR(255) NOT NULL, -- Preserve title in case listing is
deleted
  listing_image_snapshot  TEXT, -- Preserve primary image URL
  created_at              TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Indexes
CREATE INDEX idx_order_items_order_id ON order_items(order_id);
CREATE INDEX idx_order_items_listing_id ON order_items(listing_id);

COMMENT ON TABLE order_items IS 'Individual line items within an order';
COMMENT ON COLUMN order_items.unit_price_at_purchase IS 'Price locked at purchase
time, immune to future price changes';
```

4.2.6 Interest Expressions Table

Purpose: Track “Express Interest” actions before purchase

```sql
CREATE TYPE interest_status AS ENUM ('new', 'contacted', 'converted',
'not_interested');

CREATE TABLE interest_expressions (
  id          UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id     UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  listing_id  UUID NOT NULL REFERENCES listings(id) ON DELETE CASCADE,
  message     TEXT, -- Optional message from customer
  status      interest_status DEFAULT 'new',
  admin_notes TEXT, -- Admin's follow-up notes
  created_at  TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  updated_at  TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Indexes
CREATE INDEX idx_interest_expressions_user_id ON interest_expressions(user_id);
CREATE INDEX idx_interest_expressions_listing_id ON interest_expressions(listing_id);
CREATE INDEX idx_interest_expressions_status ON interest_expressions(status, cre-
ated_at DESC);
CREATE UNIQUE INDEX idx_interest_unique ON interest_expressions(user_id, listing_id)
  WHERE status IN ('new', 'contacted'); -- Prevent duplicate active interests

COMMENT ON TABLE interest_expressions IS 'Customer expressions of interest before
purchase commitment';
```

4.2.7 Admin Actions Table

Purpose: Audit log of admin decisions and actions

```sql
CREATE TYPE admin_action_type AS ENUM (
  'order_approved',
  'order_rejected',
  'order_status_updated',
  'listing_created',
  'listing_updated',
  'listing_deleted',
  'user_status_changed',
  'facebook_import'
);

CREATE TABLE admin_actions (
  id              UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  admin_id        UUID NOT NULL REFERENCES users(id) ON DELETE RESTRICT,
  action_type     admin_action_type NOT NULL,
  target_entity   VARCHAR(50), -- 'order', 'listing', 'user'
  target_id       UUID, -- ID of affected entity
  description     TEXT,
  metadata        JSONB, -- Additional context: {old_status, new_status, reason}
  ip_address      INET,
  user_agent      TEXT,
  created_at      TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Indexes
CREATE INDEX idx_admin_actions_admin_id ON admin_actions(admin_id, created_at DESC);
CREATE INDEX idx_admin_actions_target ON admin_actions(target_entity, target_id);
CREATE INDEX idx_admin_actions_created_at ON admin_actions(created_at DESC);
CREATE INDEX idx_admin_actions_type ON admin_actions(action_type);

COMMENT ON TABLE admin_actions IS 'Complete audit log of all admin actions for
compliance and debugging';
```

4.2.8 Notifications Table

Purpose: In-app notification center for users

```sql
CREATE TYPE notification_type AS ENUM (
  'order_status_update',
  'payment_confirmation',
  'admin_message',
  'system_alert',
  'promotional'
);

CREATE TABLE notifications (
  id          UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id     UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  type        notification_type NOT NULL,
  title       VARCHAR(255) NOT NULL,
  message     TEXT NOT NULL,
  action_url  TEXT, -- Deep link to relevant screen
  read        BOOLEAN DEFAULT FALSE,
  read_at     TIMESTAMP WITH TIME ZONE,
  metadata    JSONB, -- Context: {order_id, listing_id}
  created_at  TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Indexes
CREATE INDEX idx_notifications_user_id ON notifications(user_id, created_at DESC);
CREATE INDEX idx_notifications_read ON notifications(user_id, read, created_at DESC);

COMMENT ON TABLE notifications IS 'In-app notification center, separate from push no-
tifications';
```

4.2.9 Refresh Tokens Table

Purpose: Track JWT refresh tokens for secure logout

```sql
CREATE TABLE refresh_tokens (
  id              UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id         UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  token_hash      VARCHAR(255) NOT NULL UNIQUE, -- Hash of refresh token
  device_info     JSONB, -- {device_id, device_name, os, app_version}
  ip_address      INET,
  expires_at      TIMESTAMP WITH TIME ZONE NOT NULL,
  revoked         BOOLEAN DEFAULT FALSE,
  revoked_at      TIMESTAMP WITH TIME ZONE,
  created_at      TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Indexes
CREATE INDEX idx_refresh_tokens_user_id ON refresh_tokens(user_id);
CREATE INDEX idx_refresh_tokens_token_hash ON refresh_tokens(token_hash);
CREATE INDEX idx_refresh_tokens_expires_at ON refresh_tokens(expires_at);

-- Cleanup job: Delete expired tokens older than 30 days
CREATE INDEX idx_refresh_tokens_cleanup ON refresh_tokens(expires_at) WHERE
expires_at < NOW();

COMMENT ON TABLE refresh_tokens IS 'Refresh tokens for JWT authentication with device
tracking';
```

4.3 Supporting Tables

4.3.1 Categories Table (Future Enhancement)

Purpose: Structured product categories

```sql
CREATE TABLE categories (
  id          UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  name        VARCHAR(100) NOT NULL UNIQUE,
  slug        VARCHAR(100) NOT NULL UNIQUE,
  description TEXT,
  icon_url    TEXT,
  parent_id   UUID REFERENCES categories(id) ON DELETE CASCADE, -- For hierarchical
categories
  display_order INTEGER DEFAULT 0,
  is_active   BOOLEAN DEFAULT TRUE,
  created_at  TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Indexes
CREATE INDEX idx_categories_parent_id ON categories(parent_id);
CREATE INDEX idx_categories_slug ON categories(slug);
```

4.3.2 Reviews Table (Future Enhancement)

Purpose: Customer reviews and ratings

```sql
CREATE TABLE reviews (
  id          UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id     UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  listing_id  UUID NOT NULL REFERENCES listings(id) ON DELETE CASCADE,
  order_id    UUID REFERENCES orders(id) ON DELETE SET NULL, -- Verified purchase
  rating      INTEGER NOT NULL CHECK (rating >= 1 AND rating <= 5),
  title       VARCHAR(255),
  comment     TEXT,
  images      TEXT[], -- Review image URLs
  is_verified BOOLEAN DEFAULT FALSE, -- Verified purchase
  helpful_count INTEGER DEFAULT 0,
  created_at  TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  updated_at  TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Indexes
CREATE INDEX idx_reviews_listing_id ON reviews(listing_id, created_at DESC);
CREATE INDEX idx_reviews_user_id ON reviews(user_id);
CREATE UNIQUE INDEX idx_reviews_unique ON reviews(user_id, listing_id); -- One review
per user per listing
```

4.4 Database Functions & Triggers

4.4.1 Updated At Trigger

Purpose: Automatically update `updated_at` timestamp

```
CREATE OR REPLACE FUNCTION update_updated_at_column()
RETURNS TRIGGER AS $$
BEGIN
  NEW.updated_at = NOW();
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Apply to all tables with updated_at
CREATE TRIGGER users_updated_at BEFORE UPDATE ON users
  FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER listings_updated_at BEFORE UPDATE ON listings
  FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER orders_updated_at BEFORE UPDATE ON orders
  FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER interest_expressions_updated_at BEFORE UPDATE ON interest_expressions
  FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();
```

4.4.2 Order Number Generation

Purpose: Generate human-readable order numbers

```
CREATE OR REPLACE FUNCTION generate_order_number()
RETURNS TRIGGER AS $$
BEGIN
  NEW.order_number := 'FR-' ||
                      TO_CHAR(NOW(), 'YYYY') || '-' ||
                      LPAD(nextval('order_number_seq')::TEXT, 6, '0');
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER orders_order_number BEFORE INSERT ON orders
  FOR EACH ROW EXECUTE FUNCTION generate_order_number();
```

4.4.3 Order Total Calculation Validation

Purpose: Ensure order total matches sum of items

```
CREATE OR REPLACE FUNCTION validate_order_total()
RETURNS TRIGGER AS $$
DECLARE
  calculated_subtotal DECIMAL(10,2);
BEGIN
  SELECT COALESCE(SUM(subtotal), 0) INTO calculated_subtotal
  FROM order_items
  WHERE order_id = NEW.id;

  IF calculated_subtotal != NEW.subtotal_amount THEN
    RAISE EXCEPTION 'Order subtotal mismatch: calculated % != stored %',
                    calculated_subtotal, NEW.subtotal_amount;
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE CONSTRAINT TRIGGER orders_total_validation
  AFTER INSERT OR UPDATE ON orders
  DEFERRABLE INITIALLY DEFERRED
  FOR EACH ROW EXECUTE FUNCTION validate_order_total();
```

4.5 Views for Common Queries

4.5.1 Active Listings View

```
CREATE VIEW active_listings AS
SELECT
  l.*,
  (SELECT image_url FROM listing_images WHERE listing_id = l.id ORDER BY
display_order LIMIT 1) AS primary_image,
  (SELECT COUNT(*) FROM interest_expressions WHERE listing_id = l.id AND status = 'new
') AS interest_count
FROM listings l
WHERE l.is_available = TRUE
  AND l.stock_quantity > 0
ORDER BY l.created_at DESC;
```

4.5.2 Order Summary View

```sql
CREATE VIEW order_summary AS
SELECT
  o.id,
  o.order_number,
  o.user_id,
  u.name AS user_name,
  u.email AS user_email,
  o.status,
  o.payment_method,
  o.payment_status,
  o.total_amount,
  o.created_at,
  COUNT(oi.id) AS item_count,
  SUM(oi.quantity) AS total_items
FROM orders o
JOIN users u ON o.user_id = u.id
LEFT JOIN order_items oi ON o.id = oi.order_id
GROUP BY o.id, u.id;
```

4.6 Data Migration Strategy

Phase 1: MVP Schema
- Core tables: users, listings, listing_images, orders, order_items, admin_actions, notifications
- Deploy using TypeORM migrations

Phase 2: Enhancements (Month 2-3)
- Add categories table with migration to populate from existing listing.category strings
- Add reviews table
- Implement full-text search optimizations

Phase 3: Optimization (Month 4+)
- Add materialized views for analytics
- Implement table partitioning for orders (by month)
- Add database sharding strategy for multi-region expansion

4.7 Backup & Recovery Strategy

Automated Backups:
- Daily automated snapshots at 2 AM MUT (off-peak hours)
- Retention: 7 daily, 4 weekly, 12 monthly snapshots
- Point-in-time recovery (PITR) enabled with 7-day WAL retention

Backup Testing:
- Monthly restoration test to staging environment
- Verify data integrity and application compatibility

Disaster Recovery:
- RTO (Recovery Time Objective): 4 hours
- RPO (Recovery Point Objective): 15 minutes (via PITR)

---

5. API Specification Outline

This section defines the RESTful API endpoints for Fresh Roots, organized by functional domain. Full OpenAPI/Swagger documentation will be auto-generated from NestJS decorators.

5.1 API Design Principles

1. **RESTful Conventions:** Use HTTP verbs (GET, POST, PUT, PATCH, DELETE) semantically
2. **Resource-Oriented URLs:** Nouns not verbs (`/listings` not `/getListings`)
3. **Consistent Response Format:** Standard envelope for all responses
4. **Pagination:** Cursor-based pagination for scalability
5. **Filtering & Sorting:** Query parameters for search, filter, sort
6. **Versioning:** URL path versioning (`/v1/` , `/v2/`)
7. **Error Handling:** Standard HTTP status codes with detailed error messages

5.2 Standard Response Format

Success Response:

```
{
  "success": true,
  "data": { /* response payload */ },
  "meta": {
    "timestamp": "2026-01-23T10:30:00Z",
    "version": "1.0"
  }
}
```

Paginated Response:

```
{
  "success": true,
  "data": [ /* array of items */ ],
  "meta": {
    "total": 150,
    "page": 1,
    "limit": 20,
    "hasMore": true
  }
}
```

Error Response:

```
{
  "success": false,
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Invalid email format",
    "details": [
      {
        "field": "email",
        "issue": "Must be a valid email address"
      }
    ]
  },
  "meta": {
    "timestamp": "2026-01-23T10:30:00Z",
    "requestId": "req_abc123xyz"
  }
}
```

5.3 Authentication Endpoints

Base Path: `/api/v1/auth`

5.3.1 Register

Endpoint: `POST /auth/register`

Request Body:

```
{
  "email": "user@example.com",
  "password": "SecurePass123!",
  "name": "John Doe",
  "phone": "+230 5123 4567"
}
```

Response: `201 Created`

```
{
  "success": true,
  "data": {
    "user": {
      "id": "uuid",
      "email": "user@example.com",
      "name": "John Doe",
      "role": "customer",
      "created_at": "2026-01-23T10:30:00Z"
    },
    "accessToken": "eyJhbGciOiJIUzI1NiIs...",
    "refreshToken": "eyJhbGciOiJIUzI1NiIs..."
  }
}
```

Validations:
- Email: Valid format, not already registered
- Password: Min 8 chars, 1 uppercase, 1 number, 1 special char
- Name: Required, 2-255 chars
- Phone: Optional, valid Mauritius format (+230)

Errors:
- `400 Bad Request`: Validation failure
- `409 Conflict`: Email already exists

5.3.2 Login

Endpoint: `POST /auth/login`

Request Body:

```
{
  "email": "user@example.com",
  "password": "SecurePass123!"
}
```

Response: `200 OK`

```
{
  "success": true,
  "data": {
    "user": {
      "id": "uuid",
      "email": "user@example.com",
      "name": "John Doe",
      "role": "customer"
    },
    "accessToken": "eyJhbGciOiJIUzI1NiIs...",
    "refreshToken": "eyJhbGciOiJIUzI1NiIs..."
  }
}
```

Errors:
- `401 Unauthorized`: Invalid credentials
- `403 Forbidden`: Account disabled

5.3.3 Refresh Token

Endpoint: `POST /auth/refresh`

Request Body:

```
{
  "refreshToken": "eyJhbGciOiJIUzI1NiIs..."
}
```

Response: `200 OK`

```
{
  "success": true,
  "data": {
    "accessToken": "eyJhbGciOiJIUzI1NiIs...",
    "refreshToken": "eyJhbGciOiJIUzI1NiIs..." // New refresh token (rotation)
  }
}
```

5.3.4 Logout

Endpoint: `POST /auth/logout`

Headers: `Authorization: Bearer {accessToken}`

Request Body:

```
{
  "refreshToken": "eyJhbGciOiJIUzI1NiIs..."
}
```

Response: `200 OK`

```
{
  "success": true,
  "data": {
    "message": "Logged out successfully"
  }
}
```

5.3.5 Get Current User

Endpoint: `GET /auth/me`

Headers: `Authorization: Bearer {accessToken}`

Response: `200 OK`

```
{
  "success": true,
  "data": {
    "id": "uuid",
    "email": "user@example.com",
    "name": "John Doe",
    "phone": "+230 5123 4567",
    "role": "customer",
    "email_verified": true,
    "created_at": "2026-01-20T10:00:00Z"
  }
}
```

5.4 Listings Endpoints

Base Path: `/api/v1/listings`

5.4.1 Get All Listings

Endpoint: `GET /listings`

Query Parameters:
- `page` (number, default: 1): Page number
- `limit` (number, default: 20, max: 100): Items per page
- `category` (string): Filter by category
- `search` (string): Full-text search query
- `minPrice` (number): Minimum price filter
- `maxPrice` (number): Maximum price filter

- `tags` (string[]): Filter by tags (comma-separated)
- `sortBy` (string): `created_at`, `price`, `title`
- `sortOrder` (string): `asc`, `desc` (default: `desc`)

Example: `GET /listings?category=vegetables&search=organic&sortBy=price&sortOrder=asc`

Response: `200 OK`


```
{
  "success": true,
  "data": [
    {
      "id": "uuid",
      "title": "Organic Tomatoes",
      "description": "Fresh organic tomatoes...",
      "price": 85.00,
      "unit": "kg",
      "stock_quantity": 50,
      "location": "Plaine Magnien",
      "category": "vegetables",
      "tags": ["organic", "local", "fresh"],
      "is_available": true,
      "primary_image": "https://freshby4roots.com/cdn/shop/files/Beefsteak-
Tomatoes.jpg?v=1745515552&width=480",
      "images": [
        {
          "id": "uuid",
          "url": "https://images.pexels.com/photos/5503106/pexels-photo-5503106.jpeg?
auto=compress&cs=tinysrgb&w=1260&h=750&dpr=1",
          "alt_text": "Organic tomatoes closeup"
        }
      ],
      "created_at": "2026-01-20T10:00:00Z"
    }
  ],
  "meta": {
    "total": 150,
    "page": 1,
    "limit": 20,
    "hasMore": true
  }
}
```


5.4.2 Get Listing by ID

Endpoint: `GET /listings/:id`

Response: `200 OK` (Same structure as individual listing above with full details)

Errors:
- `404 Not Found`: Listing does not exist

5.4.3 Create Listing (Admin Only)

Endpoint: `POST /admin/listings`

Headers: `Authorization: Bearer {adminAccessToken}`

Request Body:

```
{
  "title": "Fresh Organic Lettuce",
  "description": "Crispy organic lettuce from hydroponic farm",
  "price": 45.00,
  "unit": "piece",
  "stock_quantity": 100,
  "location": "Curepipe",
  "category": "vegetables",
  "tags": ["organic", "hydroponic", "pesticide-free"],
  "images": [
    {
      "url": "https://i.ytimg.com/vi/mtsRY51-e1Q/sddefault.jpg",
      "alt_text": "Organic lettuce"
    }
  ]
}
```

Response: `201 Created`

Errors:
- `403 Forbidden` : Non-admin user attempting creation

5.4.4 Update Listing (Admin Only)

Endpoint: `PUT /admin/listings/:id`

Headers: `Authorization: Bearer {adminAccessToken}`

Request Body: (Same as Create, all fields optional)

Response: `200 OK`

5.4.5 Delete Listing (Admin Only)

Endpoint: `DELETE /admin/listings/:id`

Headers: `Authorization: Bearer {adminAccessToken}`

Response: `200 OK`

```
{
  "success": true,
  "data": {
    "message": "Listing deleted successfully"
  }
}
```

Note: Soft delete preferred in production (set `is_available = false`)

5.5 Orders Endpoints

Base Path: `/api/v1/orders`

5.5.1 Express Interest

Endpoint: `POST /orders/express-interest`

Headers: `Authorization: Bearer {accessToken}`

Request Body:

```
{
  "listing_id": "uuid",
  "message": "I'm interested in purchasing 5kg. Do you deliver to Quatre Bornes?"
}
```

Response: `201 Created`

```
{
  "success": true,
  "data": {
    "id": "uuid",
    "listing_id": "uuid",
    "message": "I'm interested...",
    "status": "new",
    "created_at": "2026-01-23T10:30:00Z"
  }
}
```

5.5.2 Create Purchase Request

Endpoint: `POST /orders/purchase-request`

Headers: `Authorization: Bearer {accessToken}`

Request Body:

```
{
  "items": [
    {
      "listing_id": "uuid",
      "quantity": 2
    },
    {
      "listing_id": "uuid2",
      "quantity": 1
    }
  ],
  "payment_method": "juice",
  "delivery_address": {
    "line1": "123 Main Street",
    "line2": "Apartment 4B",
    "city": "Quatre Bornes",
    "postal_code": "72301",
    "phone": "+230 5123 4567",
    "notes": "Gate code: 1234"
  },
  "notes": "Please deliver after 6 PM"
}
```

Response: `201 Created`

```
{
  "success": true,
  "data": {
    "order": {
      "id": "uuid",
      "order_number": "FR-2026-001234",
      "status": "pending",
      "payment_method": "juice",
      "subtotal_amount": 170.00,
      "delivery_fee": 50.00,
      "total_amount": 220.00,
      "created_at": "2026-01-23T10:30:00Z"
    },
    "payment_url": "https://gateway.mips.mu/pay/abc123" // If applicable
  }
}
```

Business Logic:

1. Validate stock availability for all items
2. Calculate totals (subtotal + delivery fee + COD fee if applicable)
3. Create order and order_items records (status: pending)
4. If payment method requires redirect (Juice, card), initiate payment and return URL
5. If COD, mark payment_status as ‘pending_cod’
6. Send notification to admin
7. Send confirmation email to customer

5.5.3 Get My Orders

Endpoint: `GET /orders/my-orders`

Headers: `Authorization: Bearer {accessToken}`

Query Parameters:

- `status` (string): Filter by order status
- `page` (number): Page number
- `limit` (number): Items per page

Response: `200 OK`

```
{
  "success": true,
  "data": [
    {
      "id": "uuid",
      "order_number": "FR-2026-001234",
      "status": "approved",
      "payment_method": "juice",
      "payment_status": "completed",
      "total_amount": 220.00,
      "items": [
        {
          "listing_title": "Organic Tomatoes",
          "quantity": 2,
          "unit": "kg",
          "unit_price": 85.00,
          "subtotal": 170.00,
          "image": "https://media.istockphoto.com/id/140453734/photo/fresh-toma-
toes.jpg?s=612x612&w=0&k=20&c=b6XySPuRKF6opBf0bexh9AhkWck-c7TaoJvRdVNBgT0="
        }
      ],
      "created_at": "2026-01-23T10:30:00Z",
      "updated_at": "2026-01-23T11:00:00Z"
    }
  ],
  "meta": {
    "total": 5,
    "page": 1,
    "limit": 20,
    "hasMore": false
  }
}
```


5.5.4 Get Order Details

Endpoint: `GET /orders/:id`

Headers: `Authorization: Bearer {accessToken}`

Response: `200 OK` (Full order details with items)

Authorization: User can only view their own orders; admins can view all

5.6 Admin Order Management Endpoints

Base Path: `/api/v1/admin/orders`

Authorization: All endpoints require admin role

5.6.1 Get All Orders

Endpoint: `GET /admin/orders`

Headers: `Authorization: Bearer {adminAccessToken}`

Query Parameters:
- `status` (string): Filter by status
- `payment_method` (string): Filter by payment method
- `from_date` (ISO date): Orders created after date

- `to_date` (ISO date): Orders created before date
- `page`, `limit`: Pagination

Response: `200 OK` (Paginated list of all orders with user info)

5.6.2 Approve Order

Endpoint: `PUT /admin/orders/:id/approve`

Headers: `Authorization: Bearer {adminAccessToken}`

Request Body:

```
{
  "notes": "Order confirmed, preparing for delivery tomorrow"
}
```

Response: `200 OK`

Side Effects:
1. Update order status to 'approved'
2. Log admin action in admin_actions table
3. Send push notification to customer
4. Send email confirmation to customer
5. Create in-app notification

5.6.3 Reject Order

Endpoint: `PUT /admin/orders/:id/reject`

Headers: `Authorization: Bearer {adminAccessToken}`

Request Body:

```
{
  "reason": "Insufficient stock for requested items",
  "notes": "We'll restock next week"
}
```

Response: `200 OK`

Side Effects:
1. Update order status to 'rejected'
2. Log admin action with reason
3. Restore stock quantities
4. Initiate refund if payment was completed
5. Notify customer with rejection reason

5.6.4 Update Order Status

Endpoint: `PATCH /admin/orders/:id/status`

Headers: `Authorization: Bearer {adminAccessToken}`

Request Body:

```
{
  "status": "out_for_delivery",
  "notes": "Driver: John, ETA 30 minutes"
}
```

Response: `200 OK`

5.7 Payment Endpoints

Base Path: `/api/v1/payments`

5.7.1 Initiate Payment

Endpoint: `POST /payments/initiate`

Headers: `Authorization: Bearer {accessToken}`

Request Body:

```
{
  "order_id": "uuid",
  "payment_method": "juice",
  "return_url": "freshroots://payment/success",
  "cancel_url": "freshroots://payment/cancel"
}
```

Response: `200 OK`

```
{
  "success": true,
  "data": {
    "payment_id": "pay_abc123",
    "payment_url": "https://gateway.mips.mu/pay/abc123",
    "expires_at": "2026-01-23T11:00:00Z"
  }
}
```

Flow:
1. Backend calls payment aggregator API to create payment session
2. Return payment URL to mobile app
3. Mobile app opens in-app browser or deep links to Juice app
4. User completes payment
5. Payment gateway redirects to return_url with payment status

5.7.2 Payment Callback (Webhook)

Endpoint: `POST /payments/callback`

Note: This endpoint receives webhooks from payment gateway, not called by mobile app

Security: Verify webhook signature using gateway's secret key

Request Body (from payment gateway):

```
{
  "event": "payment.completed",
  "payment_id": "pay_abc123",
  "order_id": "uuid",
  "transaction_id": "txn_xyz789",
  "amount": 220.00,
  "currency": "MUR",
  "status": "success",
  "signature": "sha256_signature"
}
```

Response: `200 OK`

Side Effects:
1. Verify signature
2. Update order payment_status and payment_transaction_id
3. Update order status to 'payment_confirmed'
4. Send confirmation notification to customer
5. Alert admin of new paid order

5.7.3 Verify Payment Status

Endpoint: `GET /payments/:payment_id/status`

Headers: `Authorization: Bearer {accessToken}`

Response: `200 OK`

```
{
  "success": true,
  "data": {
    "payment_id": "pay_abc123",
    "order_id": "uuid",
    "status": "completed",
    "transaction_id": "txn_xyz789",
    "amount": 220.00,
    "completed_at": "2026-01-23T10:45:00Z"
  }
}
```

5.8 Facebook Integration Endpoints (Admin)

Base Path: `/api/v1/admin/facebook`

5.8.1 Trigger Manual Import

Endpoint: `POST /admin/facebook/import`

Headers: `Authorization: Bearer {adminAccessToken}`

Request Body:

```
{
  "page_id": "61582787822940",
  "max_posts": 50
}
```

Response: `202 Accepted`

```
{
  "success": true,
  "data": {
    "job_id": "import_abc123",
    "status": "processing",
    "message": "Import started, check status at /admin/facebook/import-status/:job_id"
  }
}
```

Note: Runs asynchronously due to potentially long execution time

5.8.2 Get Import Status

Endpoint: `GET /admin/facebook/import-status/:job_id`

Response: `200 OK`

```
{
  "success": true,
  "data": {
    "job_id": "import_abc123",
    "status": "completed",
    "posts_fetched": 45,
    "listings_created": 32,
    "listings_skipped": 13,
    "errors": [],
    "started_at": "2026-01-23T10:00:00Z",
    "completed_at": "2026-01-23T10:05:00Z"
  }
}
```

5.9 Notification Endpoints

Base Path: `/api/v1/notifications`

5.9.1 Get My Notifications

Endpoint: `GET /notifications`

Headers: `Authorization: Bearer {accessToken}`

Query Parameters:
- `unread_only` (boolean): Filter to unread notifications
- `page`, `limit`: Pagination

Response: `200 OK`

```
{
  "success": true,
  "data": [
    {
      "id": "uuid",
      "type": "order_status_update",
      "title": "Order Approved",
      "message": "Your order FR-2026-001234 has been approved and is being prepared",
      "read": false,
      "action_url": "freshroots://orders/uuid",
      "created_at": "2026-01-23T11:00:00Z"
    }
  ],
  "meta": {
    "total": 10,
    "unread_count": 3
  }
}
```

5.9.2 Mark Notification as Read

Endpoint: `PATCH /notifications/:id/read`

Response: `200 OK`

5.9.3 Mark All as Read

Endpoint: `POST /notifications/read-all`

Response: `200 OK`

5.10 Analytics Endpoints (Future)

Base Path: `/api/v1/analytics`

Note: PostHog handles client-side event tracking. These endpoints are for admin dashboard analytics.

5.10.1 Get Dashboard Stats

Endpoint: `GET /admin/analytics/dashboard`

Headers: `Authorization: Bearer {adminAccessToken}`

Query Parameters:
- `from_date`, `to_date`: Date range

Response: `200 OK`

```
{
  "success": true,
  "data": {
    "period": {
      "from": "2026-01-01",
      "to": "2026-01-23"
    },
    "orders": {
      "total": 150,
      "pending": 12,
      "approved": 120,
      "rejected": 8,
      "total_revenue": 45000.00
    },
    "users": {
      "total": 320,
      "new_this_period": 45
    },
    "listings": {
      "total": 85,
      "out_of_stock": 5
    },
    "popular_products": [
      {
        "id": "uuid",
        "title": "Organic Tomatoes",
        "total_orders": 45,
        "revenue": 3825.00
      }
    ]
  }
}
```

5.11 Rate Limiting

Strategy: Token bucket algorithm via Redis

Limits by Tier:

```
Anonymous (no auth):
  - 30 requests per 15 minutes per IP
  - Applied to: GET /listings, POST /auth/register, POST /auth/login

Authenticated User:
  - 100 requests per 15 minutes per user ID
  - Applied to: All authenticated endpoints

Admin:
  - 300 requests per 15 minutes per admin ID
  - Applied to: Admin endpoints

Special Endpoints:
  - POST /auth/login: 5 attempts per 15 minutes per email (prevent brute force)
  - POST /payments/callback: No limit (verified webhook)
```

Response for Rate Limit Exceeded: `429 Too Many Requests`

```
{
  "success": false,
  "error": {
    "code": "RATE_LIMIT_EXCEEDED",
    "message": "Too many requests, please try again in 5 minutes",
    "retry_after": 300
  }
}
```

Headers:

```
X-RateLimit-Limit: 100
X-RateLimit-Remaining: 45
X-RateLimit-Reset: 1706012400 (Unix timestamp)
```

5.12 API Documentation

Access: `https://api.freshroots.mu/api/docs`

Auto-Generated: Swagger/OpenAPI 3.0 spec from NestJS decorators

Features:
- Interactive API testing (Try It Out)
- Request/response schemas
- Authentication flow documentation
- Example requests for all endpoints
- Error code reference

---

6. Authentication & Authorization Strategy

This section defines the complete authentication and authorization architecture for Fresh Roots, ensuring secure access control across the platform.

6.1 Authentication Flow

6.1.1 JWT (JSON Web Token) Architecture

Token Types:

1. Access Token

```
Purpose: Short-lived token for API authentication
Lifetime: 15 minutes
Storage: Mobile app memory (Redux state), never persisted to disk
Transmission: HTTP Authorization header: Bearer {token}

Payload Structure:
{
  "sub": "user_uuid",              // Subject (user ID)
  "email": "user@example.com",
  "role": "customer",              // or "admin"
  "iat": 1706009400,               // Issued at (timestamp)
  "exp": 1706010300                // Expiration (timestamp)
}

Signing Algorithm: HS256 (HMAC with SHA-256)
Secret: 256-bit secret key stored in environment variable
```

2. Refresh Token

```
Purpose: Long-lived token to obtain new access tokens
Lifetime: 30 days
Storage:
  - Mobile: Expo SecureStore (encrypted, device keychain)
  - Backend: Redis with user_id:token_hash key
Transmission: Request body (not in headers)

Payload Structure:
{
  "sub": "user_uuid",
  "tokenId": "unique_token_id",    // For revocation tracking
  "iat": 1706009400,
  "exp": 1708601400                // 30 days
}

Signing Algorithm: HS256
Secret: Different secret from access token
```

6.1.2 Registration & Login Flow

Registration Flow:

```
1. User submits registration form (email, password, name, phone)
2. Backend validates input:
   - Email format and uniqueness
   - Password strength (min 8 chars, 1 uppercase, 1 number, 1 special)
   - Phone format (Mauritius: +230 XXXX XXXX)
3. Hash password with bcrypt (cost factor 12)
4. Create user record in PostgreSQL
5. Generate access token (15 min) and refresh token (30 days)
6. Store refresh token hash in Redis with 30-day TTL
7. Return both tokens to client
8. Mobile app stores refresh token in SecureStore
9. Send verification email (optional for MVP)
```

Login Flow:

```
1. User submits email and password
2. Backend queries user by email
3. If user not found → return 401 Unauthorized (generic message to prevent email enu-
meration)
4. Compare password with bcrypt hash
5. If mismatch → return 401 Unauthorized
6. If match:
   - Generate new access token and refresh token
   - Store refresh token in Redis
   - Log login event (IP, device info)
   - Return tokens
7. Mobile app stores tokens
```

6.1.3 Token Refresh Flow

```
1. Mobile app makes API request with expired access token
2. Backend returns 401 Unauthorized with error code TOKEN_EXPIRED
3. Mobile app intercepts 401 with TOKEN_EXPIRED code
4. App retrieves refresh token from SecureStore
5. App sends POST /auth/refresh with refresh token
6. Backend validates refresh token:
   - Verify JWT signature
   - Check expiration
   - Verify token exists in Redis (not revoked)
7. If valid:
   - Generate new access token
   - Generate new refresh token (token rotation)
   - Delete old refresh token from Redis
   - Store new refresh token in Redis
   - Return both new tokens
8. Mobile app updates tokens in memory and SecureStore
9. Retry original failed request with new access token
```

Refresh Token Rotation:
- Each refresh operation generates a new refresh token
- Old refresh token is immediately invalidated
- Prevents replay attacks: if old token is used again, all tokens for that user are revoked
- User must log in again (potential security breach detected)

6.1.4 Logout Flow

```
1. User triggers logout in app
2. Mobile app sends POST /auth/logout with refresh token
3. Backend:
   - Deletes refresh token from Redis (token revocation)
   - Logs logout event
4. Mobile app:
   - Clears access token from Redux state
   - Deletes refresh token from SecureStore
   - Navigates to login screen
```

Note: Access tokens cannot be truly revoked due to stateless JWT design. 15-minute lifetime limits exposure window.

6.2 Authorization (Role-Based Access Control)

6.2.1 User Roles

Roles:

```
1. customer (default): Regular users who can browse and purchase
2. admin: Elevated privileges for listing management and order processing
```

Future Roles:

```
3. seller: Individual sellers in multi-vendor expansion
4. delivery_driver: For delivery management module
5. moderator: Customer service role for handling disputes
```

6.2.2 Permission Matrix

Feature	Endpoint	customer	admin
Listings			
Browse listings	GET /listings	✓	✓
View listing detail	GET /listings/:id	✓	✓
Create listing	POST /admin/listings	✗	✓
Update listing	PUT /admin/listings/:id	✗	✓
Delete listing	DELETE /admin/listings/:id	✗	✓
Orders			
Express interest	POST /orders/express-interest	✓	✓
Create purchase	POST /orders/purchase-request	✓	✓
View own orders	GET /orders/my-orders	✓	✓
View all orders	GET /admin/orders	✗	✓
Approve order	PUT /admin/orders/:id/approve	✗	✓
Reject order	PUT /admin/orders/:id/reject	✗	✓
Payments			
Initiate payment	POST /payments/initiate	✓	✓
View payment status	GET /payments/:id/status	✓ (own)	✓ (all)
Facebook			
Trigger import	POST /admin/facebook/import	✗	✓
View import status		✗	✓

Feature	Endpoint	customer	admin
	GET /admin/face-book/import-status/:id		
Analytics			
Dashboard stats	GET /admin/analytics/dashboard	✗	✓

6.2.3 Implementation: NestJS Guards

JWT Auth Guard (Applied to protected endpoints)

```
@Injectable()
export class JwtAuthGuard extends AuthGuard('jwt') {
  canActivate(context: ExecutionContext) {
    // Validates JWT signature and extracts user payload
    return super.canActivate(context);
  }

  handleRequest(err, user, info) {
    if (err || !user) {
      throw new UnauthorizedException('Invalid or expired token');
    }
    return user; // Attached to request.user
  }
}
```

Roles Guard (Applied to admin-only endpoints)

```
@Injectable()
export class RolesGuard implements CanActivate {
  constructor(private reflector: Reflector) {}

  canActivate(context: ExecutionContext): boolean {
    const requiredRoles = this.reflector.get<string[]>('roles', context.getHandler());
    if (!requiredRoles) {
      return true; // No specific role required
    }

    const request = context.switchToHttp().getRequest();
    const user = request.user;

    return requiredRoles.some(role => user.role === role);
  }
}
```

Usage Example:

```
@Controller('admin/listings')
@UseGuards(JwtAuthGuard, RolesGuard) // Apply both guards
export class AdminListingsController {

  @Post()
  @Roles('admin') // Decorator specifies required role
  async createListing(@Body() dto: CreateListingDto, @CurrentUser() user) {
    // Only admin users reach this point
    return this.listingsService.create(dto, user.id);
  }
}
```

6.3 Password Security

6.3.1 Password Requirements

Minimum Requirements:
- Length: 8-128 characters
- Must include:
- At least 1 uppercase letter (A-Z)
- At least 1 lowercase letter (a-z)
- At least 1 number (0-9)
- At least 1 special character (!@#$%^&*)

Validation Regex:

```
const passwordRegex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]
{8,128}$/;
```

Rejected Patterns:
- Common passwords (check against dictionary of top 10,000 passwords)
- User's email or name embedded in password
- Sequential characters (12345, abcde)

6.3.2 Password Hashing

Algorithm: bcrypt

Cost Factor: 12

```
Cost 10 = ~100ms hash time
Cost 12 = ~300ms hash time (recommended balance)
Cost 14 = ~1,200ms hash time (too slow for UX)
```

Rationale:
- bcrypt is specifically designed for password hashing (unlike SHA-256)
- Adaptive: can increase cost factor as hardware improves
- Built-in salt generation (unique salt per password)
- Resistant to rainbow table and GPU brute-force attacks

Implementation:

```
import * as bcrypt from 'bcrypt';

const SALT_ROUNDS = 12;

// Hashing (on registration/password change)
async function hashPassword(plainPassword: string): Promise<string> {
  return bcrypt.hash(plainPassword, SALT_ROUNDS);
}

// Verification (on login)
async function verifyPassword(plainPassword: string, hash: string): Promise<boolean> {
  return bcrypt.compare(plainPassword, hash);
}
```

Storage:
- **NEVER** store plaintext passwords
- Store only bcrypt hash in `users.password_hash` column
- Hash length: 60 characters (VARCHAR(255) for future algorithm changes)

6.3.3 Password Reset Flow (Future)

```
1. User clicks "Forgot Password"
2. User enters email address
3. Backend:
   - Check if email exists (return success regardless to prevent enumeration)
   - Generate secure random reset token (32 bytes, hex encoded)
   - Store token hash in Redis with 1-hour TTL, key: password_reset:{user_id}
   - Send email with reset link: https://freshroots.mu/reset-password?token={token}
4. User clicks email link
5. Mobile app/web validates token via API: GET /auth/reset-password/validate?
token={token}
6. If valid, show password reset form
7. User submits new password
8. Backend:
   - Validates token (check Redis, not expired)
   - Validate new password against requirements
   - Hash new password with bcrypt
   - Update user.password_hash
   - Delete reset token from Redis
   - Invalidate all existing refresh tokens (force re-login on all devices)
   - Send "Password Changed" confirmation email
```

6.4 Security Best Practices

6.4.1 Token Security

Access Token:
- Short lifetime (15 min) limits damage if compromised
- Never stored persistently (memory only)
- Transmitted only over HTTPS
- Validated on every request

Refresh Token:
- Stored encrypted in device secure storage
- Never transmitted in URL or query parameters
- Rotated on every refresh (single-use)

- Revocable (stored in Redis)
- Bound to device (future: include device fingerprint in payload)

6.4.2 Session Management

Concurrent Sessions:
- MVP: Allow unlimited concurrent sessions (desktop + mobile)
- Future: Track all active sessions with device info
- Allow user to view and revoke sessions from settings

Idle Timeout:
- Access tokens auto-expire after 15 min of no activity
- Refresh tokens expire after 30 days absolute
- Future: Implement sliding window refresh (extends on use)

6.4.3 Brute Force Protection

Login Attempts:
- Max 5 failed attempts per email per 15 minutes
- Implement using Redis counter with TTL
- After limit: temporary lockout, send email alert to user

Password Reset:
- Max 3 reset requests per email per hour
- Rate limit password reset form submission

6.4.4 Account Enumeration Prevention

Strategy: Return generic error messages

Example:

```
❌ Bad: "Email address not found in our system"
✅ Good: "If that email address is in our system, we've sent a password reset link"

❌ Bad: "Incorrect password for user@example.com"
✅ Good: "Invalid email or password"
```

Rationale: Prevents attackers from determining which emails are registered users

6.4.5 Two-Factor Authentication (Future Enhancement)

Recommendation: Implement TOTP (Time-based One-Time Password) in Month 3+

Flow:
1. User enables 2FA in settings
2. Backend generates secret key
3. Display QR code (generate with app like Google Authenticator)
4. User scans QR code
5. User enters first OTP to confirm setup
6. Backend validates and stores encrypted secret in database
7. On future logins:
- User enters email/password
- System prompts for 6-digit OTP
- User enters code from authenticator app

- Backend validates OTP (30-second window)
- Issue tokens only if OTP valid

Libraries:
- Backend: `speakeasy` for TOTP generation/validation
- Frontend: `expo-barcode-scanner` for QR scanning (optional convenience)

---

7. Payment Integration Plan

This section compiles all payment integration research and defines the complete payment architecture for Fresh Roots, focusing on Mauritius-specific requirements [9, 10].

7.1 Payment Landscape in Mauritius

Key Findings:
- **MCB Juice** is the dominant mobile payment platform with 600,000+ active users
- No publicly accessible API or SDK for direct Juice integration
- Integration achieved through certified payment service providers (PSPs)
- Digital payment adoption accelerated post-COVID, now preferred over cash
- **Stripe is NOT available** in Mauritius without U.S. business entity setup
- Local PSPs provide unified access to multiple payment methods

7.2 Recommended Payment Architecture

7.2.1 Primary Strategy: Payment Aggregator Integration

Recommendation: Integrate with **MIPS (Multiple Internet Payment System)** or **Paywise**

MIPS Overview:
- **Type:** Payment orchestrator (not traditional gateway)
- **HQ:** Mauritius (“Made in Moris” certified)
- **Key Feature:** Single API for Juice, My.T Money, Blink, cards
- **Compliance:** PCI DSS certified
- **Pricing:** Transaction-based fees (typically 2-3% + MUR 5-10 fixed fee)

Paywise Overview:
- **Type:** Full-service payment processor
- **HQ:** Mauritius with global banking connections
- **Key Feature:** Supports 50+ banks, 15 currencies, 3D-Secure
- **Compliance:** PCI DSS Level 1
- **Pricing:** Custom merchant agreement (competitive with MIPS)

Selection Criteria:

```
Evaluate both providers on:
1. Transaction fees (especially for MUR 200-500 average order value)
2. Settlement speed (daily vs. weekly payouts)
3. API documentation quality and developer support
4. Integration complexity (REST API, SDKs, webhooks)
5. Uptime SLA and historical reliability
6. Juice deep-linking support quality
7. Merchant onboarding timeline (typically 2-4 weeks)
```

Recommendation: Start discussions with both in parallel during Week 0, select based on fastest onboarding and best commercial terms.

7.2.2 Payment Methods Supported

1. MCB Juice (Priority 1)

```
User Flow:
1. User selects "Pay with Juice" at checkout
2. Backend initiates payment with aggregator API
3. Aggregator returns payment URL/deep link
4. Mobile app options:
   a) Desktop/Web: Display QR code for user to scan with Juice app
   b) Mobile: Deep link to Juice app (juice://pay?ref=xxx)
5. User authorizes payment in Juice app
6. Juice app redirects back to Fresh Roots app
7. Backend receives webhook from aggregator confirming payment
8. Order status updated to payment_confirmed

Technical Details:
- Deep Link Scheme: juice://pay (if supported by aggregator)
- Fallback: In-app browser with QR code display
- Payment Reference: Include order_number for reconciliation
```

2. My.T Money (Priority 2)

```
Similar flow to Juice via aggregator
Deep Link: myt://pay (if available)
User Base: 200,000+ users (secondary to Juice but significant)
```

3. Credit/Debit Cards (Priority 2)

```
Supported Cards: Visa, Mastercard (issued by MCB, SBM, other local banks)
Flow:
- Aggregator provides hosted payment page
- User enters card details on aggregator's secure page
- 3D-Secure authentication (OTP to user's phone)
- Payment processed through bank network
- Redirect back to app with payment result

PCI Compliance: SAQ-A (simplest) - card data never touches Fresh Roots servers
```

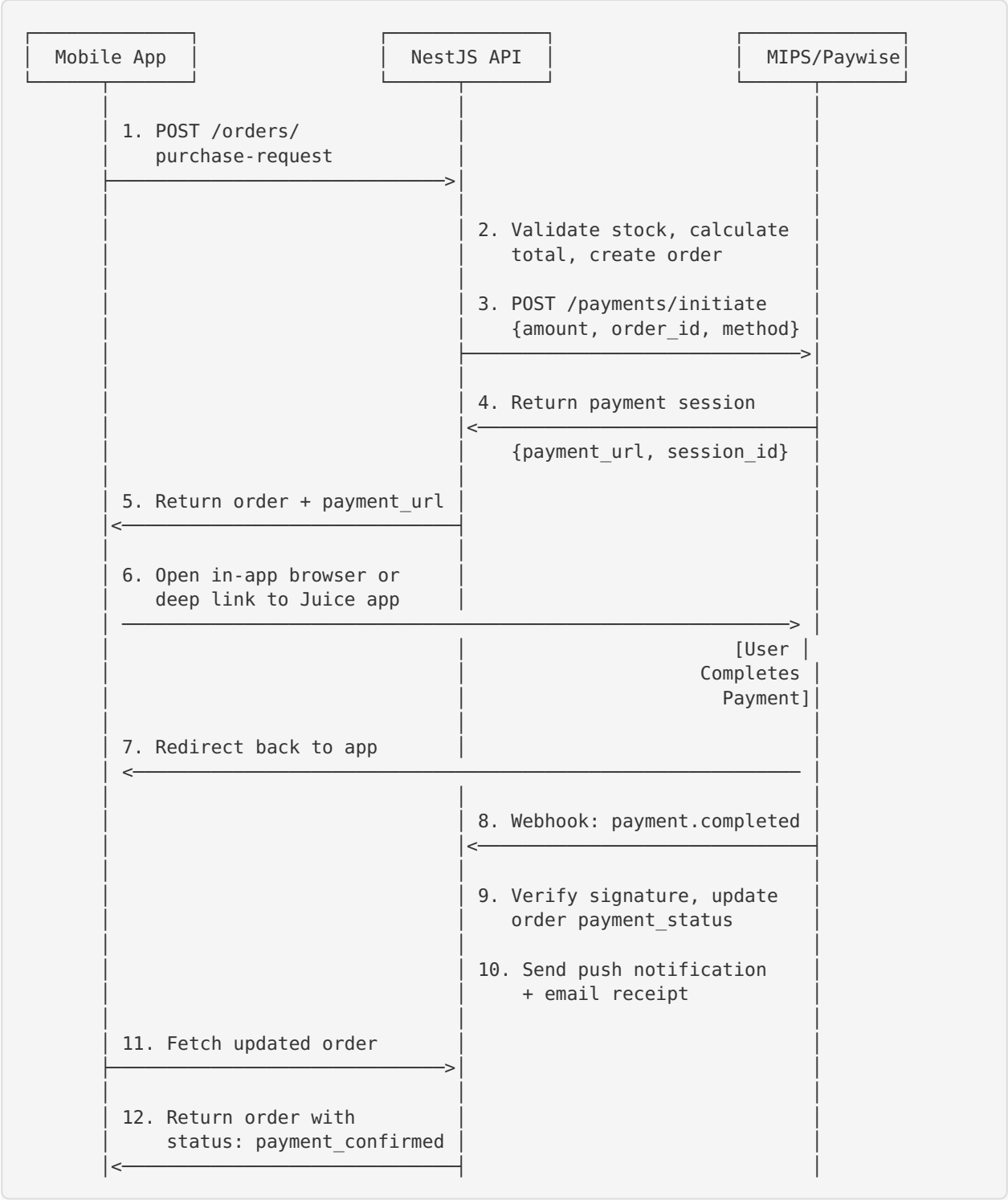
4. Cash on Delivery (Priority 1)

```
Implementation:
- Option selected at checkout (no external integration)
- Order created with payment_method = 'cash_on_delivery'
- payment_status = 'pending_cod'
- Apply COD surcharge (MUR 20-30) to offset handling cost
- Admin manually marks payment_status = 'completed' after cash received

Best Practices (from research):
☑ Set minimum order value (MUR 500) for COD eligibility
☑ Apply COD fee (MUR 25) clearly stated at checkout
☑ SMS/call verification before dispatch (reduce no-shows)
☑ Restrict COD for repeat customers with failed deliveries
☑ Track COD failure rate by region, disable in high-risk areas
```

7.3 Payment Flow Architecture

7.3.1 Purchase Request with Card/Juice Payment



7.3.2 Backend Payment Service Implementation

Key Components:

Payment Service Interface:

```
interface IPaymentProvider {
  initiatePayment(params: InitiatePaymentDto): Promise<PaymentSessionDto>;
  verifyPaymentStatus(paymentId: string): Promise<PaymentStatusDto>;
  handleWebhook(payload: any, signature: string): Promise<WebhookResultDto>;
  refundPayment(transactionId: string, amount: number): Promise<RefundDto>;
}
```

MIPS Provider Implementation:

```
@Injectable()
export class MipsPaymentProvider implements IPaymentProvider {
  constructor(
    private httpService: HttpService,
    private configService: ConfigService
  ) {}

  async initiatePayment(params: InitiatePaymentDto): Promise<PaymentSessionDto> {
    const { orderId, amount, paymentMethod, returnUrl } = params;

    // Call MIPS API to create payment session
    const response = await this.httpService.post(
      'https://api.mips.mu/v1/payments/create',
      {
        merchant_id: this.configService.get('MIPS_MERCHANT_ID'),
        order_id: orderId,
        amount: amount,
        currency: 'MUR',
        payment_method: paymentMethod, // 'juice', 'card', etc.
        return_url: returnUrl,
        webhook_url: 'https://api.freshroots.mu/v1/payments/callback',
      },
      {
        headers: {
          'Authorization': `Bearer ${this.configService.get('MIPS_API_KEY')}`,
          'Content-Type': 'application/json'
        }
      }
    ).toPromise();

    return {
      paymentId: response.data.payment_id,
      paymentUrl: response.data.payment_url,
      expiresAt: response.data.expires_at,
    };
  }

  async verifyPaymentStatus(paymentId: string): Promise<PaymentStatusDto> {
    const response = await this.httpService.get(
      `https://api.mips.mu/v1/payments/${paymentId}`,
      {
        headers: {
          'Authorization': `Bearer ${this.configService.get('MIPS_API_KEY')}`,
        }
      }
    ).toPromise();

    return {
      paymentId: response.data.payment_id,
      orderId: response.data.order_id,
      status: this.mapMipsStatus(response.data.status),
      transactionId: response.data.transaction_id,
      amount: response.data.amount,
      completedAt: response.data.completed_at,
    };
  }

  async handleWebhook(payload: any, signature: string): Promise<WebhookResultDto> {
    // Verify webhook signature to ensure it's from MIPS
    const expectedSignature = this.generateSignature(payload);

    if (signature !== expectedSignature) {
```

```
      throw new UnauthorizedException('Invalid webhook signature');
    }

    // Process webhook event
    return {
      event: payload.event, // 'payment.completed', 'payment.failed'
      orderId: payload.order_id,
      transactionId: payload.transaction_id,
      status: this.mapMipsStatus(payload.status),
      amount: payload.amount,
    };
  }

  private generateSignature(payload: any): string {
    const secret = this.configService.get('MIPS_WEBHOOK_SECRET');
    const dataToSign = JSON.stringify(payload);
    return crypto.createHmac('sha256', secret).update(dataToSign).digest('hex');
  }

  private mapMipsStatus(mipsStatus: string): PaymentStatus {
    const statusMap = {
      'completed': PaymentStatus.COMPLETED,
      'pending': PaymentStatus.PENDING,
      'failed': PaymentStatus.FAILED,
      'refunded': PaymentStatus.REFUNDED,
    };
    return statusMap[mipsStatus] || PaymentStatus.UNKNOWN;
  }
}
```

Webhook Security:

```
@Controller('payments')
export class PaymentsController {
  constructor(private paymentsService: PaymentsService) {}

  @Post('callback')
  async handlePaymentWebhook(
    @Body() payload: any,
    @Headers('x-mips-signature') signature: string
  ) {
    // 1. Verify signature
    const webhookResult = await this.paymentsService.handleWebhook(payload, signa-
ture);

    // 2. Update order in database
    await this.ordersService.updatePaymentStatus(
      webhookResult.orderId,
      webhookResult.status,
      webhookResult.transactionId
    );

    // 3. Send notification to customer
    if (webhookResult.status === PaymentStatus.COMPLETED) {
      await this.notificationsService.sendOrderConfirmation(webhookResult.orderId);
    }

    // 4. Alert admin of new paid order
    await this.notificationsService.alertAdminNewOrder(webhookResult.orderId);

    // 5. Return 200 OK to acknowledge webhook
    return { success: true };
  }
}
```

7.4 PCI Compliance Strategy

Compliance Level: PCI DSS SAQ-A (Self-Assessment Questionnaire A)

Why SAQ-A:
- Fresh Roots **never** handles, stores, or transmits cardholder data
- All card data is entered on the payment aggregator's hosted page
- Aggregator is PCI DSS Level 1 compliant and handles all card data
- Fresh Roots only receives tokenized transaction IDs

Requirements for SAQ-A:
1. Use only validated PCI DSS compliant PSP (✓ MIPS/Paywise)
2. All payment pages served over HTTPS/TLS (✓ Let's Encrypt)
3. Implement proper access controls (✓ JWT auth, role-based access)
4. Maintain information security policy (✓ Document in Month 2)
5. Conduct quarterly network security scans (✓ Use Qualys or similar)

Annual Compliance Process:
1. Complete SAQ-A questionnaire (12 questions, ~1 hour)
2. Attestation of Compliance (AOC) signed by authorized personnel
3. Submit to acquiring bank (if required) or maintain internally
4. Quarterly vulnerability scans by Approved Scanning Vendor (ASV)

Cost: $0 for SAQ-A self-assessment, ~$200-500/year for ASV scanning

7.5 Cash on Delivery (COD) Implementation

7.5.1 COD Workflow

Checkout Flow:

```
1. User selects "Cash on Delivery" payment method
2. Display COD terms clearly:
   - COD fee: MUR 25
   - Total amount due at delivery
   - Exact change appreciated or mobile payment option
3. Backend creates order with:
   - payment_method = 'cash_on_delivery'
   - payment_status = 'pending_cod'
   - cod_fee = 25.00 applied to total
4. Send SMS confirmation with order details and COD amount
5. Admin receives order notification in dashboard
6. Admin workflow:
   a) Verify order via SMS/call (optional)
   b) Approve order
   c) Prepare for delivery
   d) Update status to 'out_for_delivery'
7. Delivery completion:
   a) Admin (or driver) marks order as 'delivered'
   b) Admin manually updates payment_status = 'completed'
   c) Enter collected amount and any notes
8. Customer receives delivery confirmation notification
```

7.5.2 COD Risk Mitigation

Minimum Order Value:

```
Threshold: MUR 500
Rationale: Delivery cost + COD handling makes small orders unprofitable
Implementation: Validation at checkout:
  if (paymentMethod === 'cod' && totalAmount < 500) {
    throw new BadRequestException('Minimum order value for COD is MUR 500');
  }
```

COD Fee:

```
Amount: MUR 20-30 (test MUR 25)
Rationale:
  - Offsets cash handling, verification, failed delivery risk
  - Encourages digital payment adoption
  - Industrystandard practice (Jumbo, Winners charge similar)
Display: Clearly itemized at checkout as "COD Service Fee: MUR 25"
```

Customer Verification:

```
Implementation:
1. After order creation, trigger automated SMS:
   "Hi [Name], confirm your COD order FR-2026-001234 for MUR 545 to [address]. Reply
YES to confirm."
2. Track SMS response in admin dashboard
3. Optional: Admin can call for high-value orders (>MUR 2,000)
4. If no confirmation within 2 hours, mark order as 'verification_pending'
5. Admin decides to proceed or cancel

Technology:
- SMS Gateway: Use Twilio or local provider (e.g., Emtel Business SMS)
- Cost: ~MUR 0.50 per SMS, negligible vs. failed delivery cost (MUR 50-100)
```

Customer History Tracking:

```
Metrics per customer:
- Total COD orders
- Successful deliveries
- Failed deliveries (not home, refused, incorrect address)
- Cancellation rate

Business Rules:
- If failed_delivery_rate > 30%: Disable COD for customer, require prepayment
- If 3 consecutive failed deliveries: Permanent COD restriction
- Good customers (>5 successful deliveries): Waive COD fee as loyalty benefit

Implementation:
Add fields to users table:
- cod_orders_count
- cod_failed_count
- cod_eligible (boolean)

Update via trigger on order status change
```

Geographic Restrictions:

```
Analysis (Month 2-3):
1. Track COD success rate by delivery region (postal code)
2. Identify high-risk areas (low success rate, far delivery distance)
3. Disable COD for specific postal codes or offer only with verification

Example:
if (deliveryPostalCode in HIGH_RISK_ZONES && paymentMethod === 'cod') {
  throw new BadRequestException('COD not available in your area. Please use online pay
ment.');
}
```

7.5.3 COD Dashboard (Admin)

Metrics to Display:

- Pending COD orders count
- COD revenue collected today/week/month
- COD failure rate
- Top customers by COD usage
- High-risk orders requiring verification

Admin Actions:
- Mark order as “COD Collected” with amount
- Record failed delivery with reason
- Flag customer for COD restriction

7.6 Fallback & Contingency Planning

Scenario 1: Payment Aggregator API Downtime

```
Detection: Webhook not received within 5 minutes or API timeout
Fallback:
1. Display to user: "Payment processing, please check back in 5 minutes"
2. Backend retries status check every 60 seconds for 10 minutes
3. If still no confirmation:
   - Send email to admin with order details
   - Admin manually verifies payment via aggregator dashboard
   - Admin manually updates order payment status
4. Send apology email/notification to customer with order status
```

Scenario 2: Juice Deep Link Fails

```
Problem: Deep link doesn't open Juice app on user's device
Fallback:
1. Detect failure (user returns to app within 5 seconds)
2. Automatically switch to QR code display mode
3. Show instructions: "Scan this QR code with your Juice app to complete payment"
4. Display QR code full-screen with countdown timer (5 minutes)
```

Scenario 3: Complete Aggregator Service Failure

```
Immediate Action:
1. Disable digital payments in app (show only COD option)
2. Display banner: "Online payments temporarily unavailable. COD still accepted."
3. Alert all admins via email/SMS

Recovery:
- If downtime < 2 hours: Wait for restoration
- If downtime > 2 hours: Activate backup aggregator (if second provider integrated)
- If downtime > 24 hours: Consider emergency Stripe setup or manual bank transfer flow
```

Scenario 4: Unable to Secure Aggregator Partnership

```
Contingency Timeline:
Week 0-1: Prioritize MIPS/Paywise onboarding in parallel
Week 2: If blocked, begin Stripe account setup via U.S. LLC route
Week 3: If both fail, launch with COD-only MVP
Week 4: Continue partnership discussions, integrate payment when ready

Stripe via U.S. LLC:
- Form Delaware LLC: ~$300 + $50 registered agent (via services like Stripe Atlas)
- Apply for EIN (free): 1-2 weeks
- Open U.S. bank account: Mercury or Wise Business (~1 week)
- Create Stripe account: Instant
- Total time: 3-4 weeks
- Ongoing: File annual Delaware franchise tax ($300/year), U.S. tax returns
```

7.7 Payment Testing Strategy

Sandbox/Test Mode:

```
All payment aggregators provide test environments
Request test API credentials during onboarding

Test Cards (typical for MIPS/Paywise):
- Success: 4111 1111 1111 1111, any future expiry, any CVV
- Decline (insufficient funds): 4000 0000 0000 0002
- 3D-Secure required: 4000 0000 0000 3220

Test Juice Payments:
- Aggregator provides test Juice account or mock flow
- Simulate approval and rejection scenarios
```

Test Checklist (Week 3):

```
□ Successful card payment end-to-end
□ Failed card payment handling (show user-friendly error)
□ Successful Juice payment (if deep link available)
□ Juice payment via QR code fallback
□ Payment timeout handling (webhook not received)
□ Duplicate webhook handling (idempotency)
□ Refund initiation (admin-triggered)
□ COD order creation and workflow
□ Payment status verification API call
□ Webhook signature validation (invalid signature rejected)
```

7.8 Payment Reporting & Reconciliation

Daily Reconciliation Process:

```
1. Generate report from Fresh Roots database:
   - All orders with payment_status = 'completed'
   - Group by payment_method
   - Sum total_amount

2. Download settlement report from aggregator dashboard
   - List of transactions settled
   - Gross amount, fees, net amount

3. Match transactions:
   - Compare order_number or transaction_id
   - Flag discrepancies (orders in Fresh Roots but not in aggregator, or vice versa)

4. Manual review of flagged items:
   - Missing webhooks (fetch status via API)
   - Refunds or chargebacks
   - Settlement timing differences (T+1 or T+2)

5. Update accounting system with net revenue
```

Automation (Month 3+):

- Scheduled cron job to fetch aggregator report via API
- Automated matching algorithm
- Email report to admin with flagged discrepancies

7.9 Refunds & Dispute Handling

Refund Triggers:
- Admin rejects order after payment completed
- Customer returns damaged goods
- Overcharge or pricing error
- Delivery failure (non-COD orders)

Refund Process:

```
1. Admin initiates refund from order detail page
2. Admin enters refund amount and reason
3. Backend calls aggregator refund API:
   POST /v1/payments/{transactionId}/refund
   { amount: 220.00, reason: "Order rejected - out of stock" }
4. Aggregator processes refund to original payment method
5. Update order payment_status = 'refunded'
6. Log action in admin_actions table
7. Send email to customer confirming refund
8. Notify customer refund will appear in 5-7 business days
```

Refund Timeline:
- Juice/My.T Money: Typically instant to 24 hours
- Credit/Debit cards: 5-10 business days (bank processing time)

Dispute Handling (Chargebacks):

```
If customer disputes charge with bank:
1. Aggregator notifies Fresh Roots via email/webhook
2. Provide evidence within 7-14 days:
   - Order confirmation
   - Delivery confirmation (signature/photo)
   - Customer communication logs
3. Aggregator submits evidence to card network
4. Decision: Merchant wins (funds retained) or Customer wins (refund issued + chargeba
ck fee MUR 200-400)

Prevention:
- Clear product descriptions and images
- Delivery confirmation with signature
- Responsive customer service
- Easy refund process (reduces chargeback motivation)
```

---

8. Facebook Listing Import/Sync Plan

This section compiles research on Facebook Graph API integration and defines the complete workflow for importing product listings from the Fresh Roots Facebook page [11].

8.1 Facebook Graph API Overview

Key Findings from Research:
- No single endpoint to directly “import products” from a Page
- Requires creating a **Product Catalog** and adding **Product Items** programmatically
- Access requires **Page Access Token** with specific permissions

- Subject to **Business Use Case (BUC) rate limits**
- Web scraping is technically possible but highly discouraged (fragile, violates TOS)

Facebook Page:
- URL: `https://www.facebook.com/profile.php?id=61582787822940`
- Page ID: `61582787822940`
- Current content: Product posts with images, descriptions, prices

8.2 Integration Architecture

8.2.1 Workflow: Semi-Automated Import with Manual Review

Rationale:
- Fully automated import risks importing irrelevant posts or incorrectly parsed data
- Manual review step ensures quality control
- Admin can edit/enhance listings before publishing

Workflow:

```
╔═══════════════════════════════════════════════════════════╗
║                  FACEBOOK IMPORT WORKFLOW                 ║
╚═══════════════════════════════════════════════════════════╝

1. [Admin] Triggers import from admin dashboard
   └─> POST /api/v1/admin/facebook/import

2. [Backend] Fetches Page Access Token from secure storage
   └─> Verify token is not expired

3. [Backend] Calls Facebook Graph API
   └─> GET /v1.0/{page-id}/feed?fields=id,mes-
sage,created_time,full_picture,attachments

4. [Backend] Parses each post
   ├─> Extract product title from first line of message
   ├─> Extract description from remaining message text
   ├─> Parse price (regex patterns: "MUR 85", "Rs 85", "85 rupees")
   ├─> Extract image URLs from attachments
   └─> Save parsed data as DRAFT listing

5. [Backend] Returns import summary
   └─> "45 posts fetched, 32 products parsed, 13 skipped"

6. [Admin] Reviews draft listings in dashboard
   ├─> Views: title, description, price, images
   ├─> Actions:
   │   ├─> Edit any field (title, price, category, tags)
   │   ├─> Publish (moves to active listings)
   │   ├─> Discard
   │   └─> Bulk publish all
   └─> Each published listing links to original FB post_id

7. [System] Scheduled sync (optional, future)
   └─> Daily cron job checks for new posts since last import
   └─> Repeats steps 3-6 automatically
```

8.3 Required Permissions & Access Token

8.3.1 Facebook App Setup

Prerequisites:

1. Admin must have Facebook Developer account
2. Create a Facebook App in Facebook Developers console
3. Add **Fresh Roots** Facebook Page to the app

App Configuration:

```
App Name: Fresh Roots Marketplace
App Purpose: Business
Category: E-commerce
Privacy Policy URL: https://freshroots.mu/privacy
Terms of Service URL: https://freshroots.mu/terms

Products to Add:
- Facebook Login (for obtaining user access token)
- Marketing API (for catalog management - future)
```

8.3.2 Permissions Required

Page Permissions:

```
pages_read_engagement: Read Page posts and content
pages_show_list: Display list of Pages user manages
pages_manage_posts: Manage Page posts (optional, for auto-posting)
```

Public Content Access (for public page data):

```
Required Feature: Page Public Content Access
Status: Requires App Review for production use
Approval Time: 3-5 business days
Alternative: Use dev/test mode with admin user access (sufficient for MVP)
```

8.3.3 Obtaining Page Access Token

Process:

```
1. User Authorization (One-time setup by admin)
   ├─> Admin logs into Fresh Roots admin dashboard
   ├─> Navigates to Settings > Facebook Integration
   ├─> Clicks "Connect Facebook Page"
   ├─> Redirected to Facebook OAuth dialog
   ├─> Grants requested permissions
   └─> Redirected back to Fresh Roots with authorization code

2. Backend Token Exchange
   ├─> POST https://graph.facebook.com/v19.0/oauth/access_token
   ├─> Params: {
   │     client_id: {app_id},
   │     client_secret: {app_secret},
   │     redirect_uri: "https://admin.freshroots.mu/auth/facebook/callback",
   │     code: {authorization_code}
   │   }
   └─> Receives: User Access Token (short-lived, 1 hour)

3. Get Page Access Token
   ├─> GET https://graph.facebook.com/v19.0/me/accounts
   ├─> Header: Authorization: Bearer {user_access_token}
   ├─> Response: List of pages user manages
   ├─> Extract Page Access Token for Fresh Roots page
   └─> This token is long-lived (60 days) or never expires if user doesn't change pass
word

4. Store Securely
   ├─> Encrypt Page Access Token
   ├─> Store in database or AWS Secrets Manager
   └─> Backend retrieves token for API calls
```

Token Renewal:

```
Issue: Page Access Tokens expire after 60 days or if user changes password
Solution:
  1. Detect expired token (API returns error code 190)
  2. Prompt admin to reconnect Facebook account
  3. Repeat authorization flow
Alternative (Future):
  - Implement automatic token refresh using long-lived tokens
  - Monitor token expiration and refresh proactively
```

8.4 API Endpoints & Implementation

8.4.1 Fetching Page Feed

Graph API Call:

```
GET https://graph.facebook.com/v19.0/{page-id}/feed

Query Parameters:
  fields:
id,message,created_time,full_picture,attachments{media,url,subattachments},permalink_u
rl
  limit: 50 (max posts per request)
  since: {timestamp} (optional, fetch posts after specific date)

Headers:
  Authorization: Bearer {page_access_token}
```

Response Structure:


```
{
  "data": [
    {
      "id": "61582787822940_123456789",
      "message":
"Fresh Organic Tomatoes\nPerfectly ripe, straight from Plaine Magnien farm\nMUR 85 per
kg\nFree delivery on orders above MUR 1000",
      "created_time": "2026-01-20T08:30:00+0000",
      "full_picture": "https://images.pexels.com/photos/712497/pexels-
photo-712497.jpeg",
      "attachments": {
        "data": [
          {
            "media": {
              "image": {
                "src": "https://images.pexels.com/photos/1327838/pexels-
photo-1327838.jpeg?cs=srgb&dl=pexels-julia-nagy-568948-1327838.jpg&fm=jpg"
              }
            },
            "url": "https://images.pexels.com/photos/1367242/pexels-
photo-1367242.jpeg"
          }
        ]
      },
      "permalink_url": "https://www.facebook.com/profile.php?
id=61582787822940&post=123"
    }
  ],
  "paging": {
    "next": "https://graph.facebook.com/v19.0/{page-id}/feed?after=..."
  }
}
```


8.4.2 Post Parsing Logic

Implementation:

```typescript
interface FacebookPost {
  id: string;
  message: string;
  created_time: string;
  full_picture?: string;
  attachments?: any;
  permalink_url: string;
}

interface ParsedListing {
  title: string;
  description: string;
  price: number | null;
  currency: string;
  unit: string | null;
  images: string[];
  facebook_post_id: string;
  facebook_post_url: string;
  raw_message: string; // Store original for review
}

function parsePostToListing(post: FacebookPost): ParsedListing | null {
  const message = post.message || '';
  const lines = message.split('\n').filter(line => line.trim());

  if (lines.length === 0) {
    return null; // Skip posts without text
  }

  // Extract title (first line)
  const title = lines[0].substring(0, 255); // Truncate to DB limit

  // Extract price
  const pricePattern = /(MUR|Rs\.?|rupees?)\s*(\d+(?:,\d{3})*(?:\.\d{2})?)/gi;
  const priceMatch = message.match(pricePattern);
  let price: number | null = null;
  let currency = 'MUR';

  if (priceMatch) {
    const numericPrice = priceMatch[0].match(/(\d+(?:,\d{3})*(?:\.\d{2})?)/);
    if (numericPrice) {
      price = parseFloat(numericPrice[0].replace(/,/g, ''));
    }
  }

  // Extract unit
  const unitPattern = /per\s+(kg|kilogram|piece|bunch|pack)/gi;
  const unitMatch = message.match(unitPattern);
  let unit: string | null = null;

  if (unitMatch) {
    unit = unitMatch[0].replace(/per\s+/i, '').toLowerCase();
    // Normalize units
    if (unit === 'kilogram') unit = 'kg';
  }

  // Extract description (remaining lines, exclude price line)
  const description = lines.slice(1).join('\n').substring(0, 5000);

  // Extract images
  const images: string[] = [];
  if (post.full_picture) {
```

```
    images.push(post.full_picture);
  }
  if (post.attachments?.data) {
    post.attachments.data.forEach(attachment => {
      const imageSrc = attachment.media?.image?.src;
      if (imageSrc && !images.includes(imageSrc)) {
        images.push(imageSrc);
      }
      // Handle multiple images in subattachments
      if (attachment.subattachments?.data) {
        attachment.subattachments.data.forEach(sub => {
          const subSrc = sub.media?.image?.src;
          if (subSrc && !images.includes(subSrc)) {
            images.push(subSrc);
          }
        });
      }
    });
  }

  return {
    title,
    description: description || title, // Fallback if no description
    price,
    currency,
    unit,
    images,
    facebook_post_id: post.id,
    facebook_post_url: post.permalink_url,
    raw_message: post.message,
  };
}
```

Handling Edge Cases:

```
function validateParsedListing(parsed: ParsedListing): ValidationResult {
  const errors: string[] = [];

  if (!parsed.title || parsed.title.length < 3) {
    errors.push('Title too short or missing');
  }

  if (!parsed.price || parsed.price <= 0) {
    errors.push('Invalid or missing price');
  }

  if (!parsed.unit) {
    // Don't reject, but flag for admin review
    console.warn(`No unit found for: ${parsed.title}`);
  }

  if (parsed.images.length === 0) {
    errors.push('No images found');
  }

  return {
    isValid: errors.length === 0,
    errors,
    requiresReview: !parsed.unit || errors.length > 0
  };
}
```

8.4.3 Saving Draft Listings

Database Approach:

Option 1: Add `status` field to listings table

```
ALTER TABLE listings ADD COLUMN status VARCHAR(20) DEFAULT 'active';
-- Values: 'draft', 'active', 'archived'

CREATE INDEX idx_listings_status ON listings(status);
```

Option 2: Create separate `draft_listings` table

```
CREATE TABLE draft_listings (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  title VARCHAR(255) NOT NULL,
  description TEXT,
  price DECIMAL(10, 2),
  unit VARCHAR(20),
  images JSONB, -- Array of image URLs
  facebook_post_id VARCHAR(255),
  facebook_post_url TEXT,
  raw_facebook_message TEXT,
  parsed_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  requires_review BOOLEAN DEFAULT FALSE,
  review_notes TEXT,
  admin_id UUID REFERENCES users(id) ON DELETE SET NULL,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

Recommendation: Option 1 (status field) for simplicity. MVP can manage all listings in one table.

Backend Service:

```
@Injectable()
export class FacebookImportService {
  constructor(
    private httpService: HttpService,
    private listingsService: ListingsService,
    private configService: ConfigService,
  ) {}

  async importPostsFromPage(pageId: string, maxPosts: number = 50): Promise<ImportRes-
ult> {
    const pageAccessToken = await this.getPageAccessToken();
    const posts = await this.fetchPageFeed(pageId, pageAccessToken, maxPosts);

    const result: ImportResult = {
      postsFetched: posts.length,
      listingsCreated: 0,
      listingsSkipped: 0,
      errors: [],
    };

    for (const post of posts) {
      try {
        const parsed = parsePostToListing(post);
        if (!parsed) {
          result.listingsSkipped++;
          continue;
        }

        const validation = validateParsedListing(parsed);

        // Create draft listing
        const draftListing = await this.listingsService.create({
          ...parsed,
          status: 'draft',
          is_available: false, // Hidden from public until published
          admin_id: null, // Not yet assigned to admin
        });

        result.listingsCreated++;
      } catch (error) {
        result.errors.push({
          post_id: post.id,
          error: error.message
        });
        result.listingsSkipped++;
      }
    }

    return result;
  }

  private async fetchPageFeed(pageId: string, token: string, limit: number): Promise<F
acebookPost[]> {
    const url = `https://graph.facebook.com/v19.0/${pageId}/feed`;
    const params = {
      fields: 'id,message,created_time,full_picture,attachments{media,url,subattachmen
ts},permalink_url',
      limit: limit,
      access_token: token,
    };

    const response = await this.httpService.get(url, { params }).toPromise();
```

```
    return response.data.data || [];
  }

  private async getPageAccessToken(): Promise<string> {
    // Retrieve from secure storage (database or AWS Secrets Manager)
    const token = await this.configService.get('FACEBOOK_PAGE_ACCESS_TOKEN');
    if (!token) {
      throw new Error('Facebook Page Access Token not configured');
    }
    return token;
  }
}
```

8.5 Admin Review Interface

Draft Listings Dashboard:

Features:
- Table view of all draft listings
- Columns: Title, Price, Unit, Image thumbnail, Facebook Link, Actions
- Filter by: Requires Review, All Drafts
- Sort by: Parse Date, Title

Individual Listing Review:

Bulk Actions:
- Select multiple draft listings (checkbox)
- Bulk Publish: Publish all selected (if validation passes)

- Bulk Discard: Delete selected drafts
- Bulk Assign Category: Apply category to selected

8.6 Rate Limiting & Best Practices

8.6.1 Facebook Rate Limits

Business Use Case (BUC) Rate Limits:

```
Formula: 20,000 + 20,000 * log2(unique_users) calls per hour

For Fresh Roots (initially 1-2 users = admin):
  20,000 + 20,000 * log2(2) = 20,000 + 20,000 * 1 = 40,000 calls/hour

Per-endpoint typical limit: 200 calls per hour per user

Conclusion: MVP usage (manual imports, ~1-2 times/day) is well below limits
```

Rate Limit Headers:

```
Response headers from Graph API:
  X-Business-Use-Case: {"61582787822940":{"call_count":5,"total_cputime":10,"total_tim
e":15,"type":"pages"}}

call_count: Number of calls made in current window
total_cputime: CPU time used
total_time: Actual time spent

Monitor these headers to track usage
```

Rate Limit Handling:

```
async function fetchWithRateLimitHandling(url: string, options: any): Promise<any> {
  try {
    const response = await axios.get(url, options);

    // Log usage for monitoring
    const bucUsage = response.headers['x-business-use-case'];
    if (bucUsage) {
      console.log('Facebook API usage:', JSON.parse(bucUsage));
    }

    return response.data;
  } catch (error) {
    if (error.response?.status === 429 || error.response?.data?.error?.code === 32) {
      // Rate limit exceeded
      const retryAfter = parseInt(error.response.headers['retry-after'] || '3600');
      console.error(`Rate limit hit. Retry after ${retryAfter} seconds`);
      throw new Error(`Facebook rate limit exceeded. Please try again in ${Math.ceil(r
etryAfter / 60)} minutes.`);
    }
    throw error;
  }
}
```

8.6.2 Best Practices

Minimize API Calls:

1. **Cache Results:**

- Cache fetched posts in Redis for 1 hour
- If admin triggers import again within 1 hour, return cached results

1. **Incremental Sync:**
   - Use `since` parameter to fetch only new posts since last import
   - Store last import timestamp in database

   ```
   typescript
     const lastImportTime = await this.getLastImportTimestamp(pageId);
     const params = {
       ...baseParams,
       since: lastImportTime,
     };
   ```

2. **Pagination:**
   - Respect `paging.next` URL for fetching multiple pages
   - Stop after `maxPosts` or `maxPages` limit to avoid excessive calls

Error Handling:

```
Common Errors:
1. Error 190: Expired/Invalid Access Token
   → Re-prompt admin to reconnect Facebook account

2. Error 100: Invalid parameter (e.g., wrong field name)
   → Log error, skip post, continue import

3. Error 4: Too many requests
   → Implement exponential backoff, wait and retry

4. Network timeout
   → Retry up to 3 times with 5-second delay
```

8.7 Automated Sync (Future Enhancement)

Scheduled Import:

```
@Injectable()
export class FacebookSyncScheduler {
  constructor(private facebookImportService: FacebookImportService) {}

  // Run daily at 2 AM MUT (off-peak)
  @Cron('0 2 * * *', { timeZone: 'Indian/Mauritius' })
  async handleDailySync() {
    console.log('Starting automated Facebook sync...');

    try {
      const result = await this.facebookImportService.importPostsFromPage(
        process.env.FACEBOOK_PAGE_ID,
        20 // Fetch max 20 new posts per day
      );

      if (result.listingsCreated > 0) {
        // Notify admin of new draft listings
        await this.notificationsService.alertAdminNewDrafts(result.listingsCreated);
      }

      console.log(`Facebook sync completed: ${result.listingsCreated} new drafts`);
    } catch (error) {
      console.error('Facebook sync failed:', error);
      // Alert admin of failure
      await this.notificationsService.alertAdminSyncFailed(error.message);
    }
  }
}
```

Smart Sync:
- Only run if new posts exist (check post count first)
- Skip if last sync was < 12 hours ago and no manual trigger
- Pause sync if error rate > 50% (likely permission or token issue)

8.8 Fallback: Manual Listing Creation

Always Available:
- Admin dashboard must have manual listing creation form
- If Facebook integration fails (API access denied, app review rejected), admin can manually create listings
- Manual form is faster than re-typing from Facebook (can copy-paste from FB post)

Enhanced Manual Creation:
- **Image Upload from URL:** Admin pastes Facebook image URL, backend downloads and uploads to Cloudinary
- **Import from CSV:** For bulk creation, admin can prepare CSV from Facebook page export
- Columns: title, description, price, unit, image_url_1, image_url_2, etc.
- Backend validates and imports all rows

---

9. Notification Architecture

This section compiles notification research and defines the complete strategy for push, email, and in-app notifications [12].

9.1 Notification Requirements

User Notifications:
- Order status updates (approved, out for delivery, delivered)
- Payment confirmations
- Special promotions (future)
- Low stock alerts for favorite items (future)

Admin Notifications:
- New purchase requests (high priority)
- New interest expressions
- Low stock alerts
- Facebook import completion
- Payment system issues
- System errors (via Sentry email)

9.2 Push Notifications

9.2.1 Phase 1: Expo Push Notifications (MVP)

Recommendation for MVP (Weeks 1-4)

Justification:
- **Zero Configuration:** Works out-of-box with Expo apps, no native setup required
- **Free:** No cost for unlimited notifications
- **Perfect for Development:** Instant testing with Expo Go
- **Fast Integration:** Can be implemented in 1-2 hours

How Expo Push Works:

```
1. Mobile app registers for push notifications
   └─> Returns Expo Push Token: ExponentPushToken[xxxxxxxxxxxxxx]

2. App sends token to backend via API
   └─> POST /api/v1/users/register-push-token
   └─> Store in database: users.expo_push_token

3. Backend triggers notification
   └─> Constructs notification payload
   └─> Sends to Expo Push API
   └─> Expo routes to FCM (Android) or APNs (iOS)

4. Device receives notification
   └─> User taps notification
   └─> App opens to relevant screen (deep linking)
```

Implementation:

Mobile App (React Native):

```
import * as Notifications from 'expo-notifications';
import Constants from 'expo-constants';

// Configure notification behavior
Notifications.setNotificationHandler({
  handleNotification: async () => ({
    shouldShowAlert: true,
    shouldPlaySound: true,
    shouldSetBadge: true,
  }),
});

// Request permission and get token
async function registerForPushNotifications() {
  if (Constants.isDevice) {
    const { status: existingStatus } = await Notifications.getPermissionsAsync();
    let finalStatus = existingStatus;

    if (existingStatus !== 'granted') {
      const { status } = await Notifications.requestPermissionsAsync();
      finalStatus = status;
    }

    if (finalStatus !== 'granted') {
      alert('Failed to get push token for push notification!');
      return;
    }

    const token = (await Notifications.getExpoPushTokenAsync({
      projectId: Constants.expoConfig?.extra?.eas?.projectId,
    })).data;

    console.log('Expo Push Token:', token);

    // Send to backend
    await api.post('/users/register-push-token', { token });
  } else {
    alert('Must use physical device for Push Notifications');
  }
}

// Handle notification received while app is foregrounded
Notifications.addNotificationReceivedListener(notification => {
  console.log('Notification received:', notification);
  // Update UI or show in-app banner
});

// Handle notification tap
Notifications.addNotificationResponseReceivedListener(response => {
  const data = response.notification.request.content.data;

  // Deep link to relevant screen
  if (data.type === 'order_update') {
    navigation.navigate('OrderDetail', { orderId: data.order_id });
  }
});
```

Backend (NestJS):

```
import { Expo, ExpoPushMessage } from 'expo-server-sdk';

@Injectable()
export class ExpoPushService {
  private expo: Expo;

  constructor() {
    this.expo = new Expo();
  }

  async sendPushNotification(
    expoPushToken: string,
    title: string,
    body: string,
    data?: any
  ): Promise<void> {
    // Check that token is valid Expo push token
    if (!Expo.isExpoPushToken(expoPushToken)) {
      console.error(`Push token ${expoPushToken} is not a valid Expo push token`);
      return;
    }

    const message: ExpoPushMessage = {
      to: expoPushToken,
      sound: 'default',
      title: title,
      body: body,
      data: data || {},
      priority: 'high',
      channelId: 'default', // Android notification channel
    };

    try {
      const tickets = await this.expo.sendPushNotificationsAsync([message]);
      console.log('Push notification sent:', tickets);

      // Handle tickets (check for errors)
      for (const ticket of tickets) {
        if (ticket.status === 'error') {
          console.error(`Error sending push: ${ticket.message}`);
          // If token is invalid, mark in database
          if (ticket.details?.error === 'DeviceNotRegistered') {
            await this.usersService.invalidatePushToken(expoPushToken);
          }
        }
      }
    } catch (error) {
      console.error('Error sending push notification:', error);
    }
  }

  async sendOrderUpdateNotification(userId: string, order: Order): Promise<void> {
    const user = await this.usersService.findById(userId);
    if (!user.expo_push_token) {
      console.log(`User ${userId} has no push token registered`);
      return;
    }

    const title = 'Order Update';
    const body = this.getOrderUpdateMessage(order);
    const data = {
      type: 'order_update',
```

```
      order_id: order.id,
      order_number: order.order_number,
    };

    await this.sendPushNotification(user.expo_push_token, title, body, data);
  }

  private getOrderUpdateMessage(order: Order): string {
    const statusMessages = {
      'approved': `Your order ${order.order_number} has been approved!`,
      'preparing': `Your order is being prepared for delivery.`,
      'out_for_delivery': `Your order is out for delivery! It should arrive soon.`,
      'delivered': `Your order has been delivered. Enjoy your fresh produce!`,
      'rejected': `Sorry, your order ${order.order_number} could not be processed.`,
    };
    return statusMessages[order.status] || `Order ${order.order_number} status: ${or-
der.status}`;
  }
}
```

Batch Sending (Admin Broadcast):

```
async sendBulkNotifications(userIds: string[], title: string, body: string):
Promise<void> {
  const users = await this.usersService.findByIds(userIds);
  const messages: ExpoPushMessage[] = [];

  for (const user of users) {
    if (user.expo_push_token && Expo.isExpoPushToken(user.expo_push_token)) {
      messages.push({
        to: user.expo_push_token,
        title,
        body,
        sound: 'default',
      });
    }
  }

  // Send in chunks of 100 (Expo API limit)
  const chunks = this.expo.chunkPushNotifications(messages);
  for (const chunk of chunks) {
    try {
      await this.expo.sendPushNotificationsAsync(chunk);
    } catch (error) {
      console.error('Error sending chunk:', error);
    }
  }
}
```

Database Schema Addition:

```
ALTER TABLE users ADD COLUMN expo_push_token VARCHAR(255);
CREATE INDEX idx_users_expo_push_token ON users(expo_push_token);

-- Future: Track multiple devices per user
CREATE TABLE user_devices (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  platform VARCHAR(20), -- 'ios', 'android'
  expo_push_token VARCHAR(255),
  fcm_token VARCHAR(255), -- Native FCM token for Phase 2
  device_name VARCHAR(255),
  last_active TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

9.2.2 Phase 2: OneSignal (Growth)

Migration Trigger:

- User base > 5,000 OR
- Need advanced segmentation (e.g., target users in specific city) OR
- Marketing wants A/B testing for notifications OR
- Need rich media notifications (images, action buttons)

OneSignal Advantages:

- **Advanced Targeting:** Segment users by: location, behavior, user properties, tags
- **A/B Testing:** Test notification copy, timing, images for max engagement
- **Rich Notifications:** Include images, action buttons (“View Order”, “Reorder”)
- **Omnichannel:** Also supports email, SMS, in-app messages from one platform
- **Analytics:** Detailed delivery, open, click rates per campaign
- **Scheduled Campaigns:** Automated drip campaigns (e.g., “Come back!” after 7 days inactive)

Cost Comparison:

```
OneSignal Pricing (2026 rates):
- Free: Up to 1,000 users, unlimited notifications
- Growth: $9/month for 1,000-10,000 users
- Professional: $39/month for 10,000-25,000 users
- Enterprise: $309/month for 100,000 users

Expo Push: Always free, but no advanced features

Decision Point: When advanced targeting/analytics value > $39/month cost
```

Migration Strategy:

```
1. Week before migration:
   - Integrate OneSignal SDK in React Native app
   - Configure OneSignal dashboard (app settings, notification templates)
   - Test in dev environment

2. Migration week:
   - Deploy app update with OneSignal SDK (both SDKs can coexist temporarily)
   - Backend starts collecting both Expo and OneSignal player IDs
   - Gradual rollout: 10% users → 50% → 100%

3. Post-migration:
   - Monitor delivery rates for both systems for 1 week
   - Once OneSignal proven stable, remove Expo push code
   - Deprecate expo_push_token column (keep for 30 days as backup)

4. Leverage OneSignal features:
   - Set up automated notifications (order updates, abandoned cart)
   - Create user segments (high-value customers, inactive users)
   - Launch first A/B test campaign
```

OneSignal Integration (Backend):

```
import axios from 'axios';

@Injectable()
export class OneSignalService {
  private readonly apiUrl = 'https://onesignal.com/api/v1/notifications';
  private readonly appId = process.env.ONESIGNAL_APP_ID;
  private readonly apiKey = process.env.ONESIGNAL_API_KEY;

  async sendNotification(
    userIds: string[],
    title: string,
    body: string,
    data?: any
  ): Promise<void> {
    try {
      await axios.post(
        this.apiUrl,
        {
          app_id: this.appId,
          include_external_user_ids: userIds, // Our user IDs
          headings: { en: title },
          contents: { en: body },
          data: data,
          ios_badgeType: 'Increase',
          ios_badgeCount: 1,
        },
        {
          headers: {
            'Authorization': `Basic ${this.apiKey}`,
            'Content-Type': 'application/json',
          }
        }
      );
      console.log('OneSignal notification sent');
    } catch (error) {
      console.error('OneSignal error:', error.response?.data || error.message);
    }
  }

  async sendToSegment(segment: string, title: string, body: string): Promise<void> {
    // Send to user segment (e.g., "Quatre Bornes Customers")
    await axios.post(
      this.apiUrl,
      {
        app_id: this.appId,
        included_segments: [segment],
        headings: { en: title },
        contents: { en: body },
      },
      {
        headers: {
          'Authorization': `Basic ${this.apiKey}`,
          'Content-Type': 'application/json',
        }
      }
    );
  }
}
```

9.3 Email Notifications

9.3.1 Email Service: AWS SES

Primary Recommendation: Amazon Simple Email Service (SES)

Setup Process:

1. AWS Account Setup:

```
- Sign up for AWS account (free tier eligible)
- Navigate to Amazon SES in AWS Console
- Request production access (approval within 24 hours for legitimate businesses)
- Verify domain (freshroots.mu):
  a) Add DKIM records to DNS (provided by SES)
  b) Add SPF record: v=spf1 include:amazonses.com ~all
  c) Add DMARC record: v=DMARC1; p=none; rua=mailto:admin@freshroots.mu
```

2. Backend Integration:

```
import { SESClient, SendEmailCommand } from '@aws-sdk/client-ses';

@Injectable()
export class SesEmailService {
  private sesClient: SESClient;

  constructor(private configService: ConfigService) {
    this.sesClient = new SESClient({
      region: this.configService.get('AWS_REGION') || 'us-east-1',
      credentials: {
        accessKeyId: this.configService.get('AWS_ACCESS_KEY_ID'),
        secretAccessKey: this.configService.get('AWS_SECRET_ACCESS_KEY'),
      },
    });
  }

  async sendEmail(
    to: string,
    subject: string,
    htmlBody: string,
    textBody?: string
  ): Promise<void> {
    const params = {
      Source: 'Fresh Roots <noreply@freshroots.mu>',
      Destination: {
        ToAddresses: [to],
      },
      Message: {
        Subject: {
          Data: subject,
          Charset: 'UTF-8',
        },
        Body: {
          Html: {
            Data: htmlBody,
            Charset: 'UTF-8',
          },
          Text: {
            Data: textBody || this.stripHtml(htmlBody),
            Charset: 'UTF-8',
          },
        },
      },
    };

    try {
      const command = new SendEmailCommand(params);
      const response = await this.sesClient.send(command);
      console.log('Email sent:', response.MessageId);
    } catch (error) {
      console.error('SES error:', error);
      throw error;
    }
  }

  async sendOrderConfirmation(order: Order, user: User): Promise<void> {
    const subject = `Order Confirmation - ${order.order_number}`;
    const htmlBody = this.renderOrderConfirmationEmail(order, user);

    await this.sendEmail(user.email, subject, htmlBody);
  }
```

```
  async sendOrderApprovedEmail(order: Order, user: User): Promise<void> {
    const subject = `Order Approved - ${order.order_number}`;
    const htmlBody = this.renderOrderApprovedEmail(order, user);

    await this.sendEmail(user.email, subject, htmlBody);
  }

  async sendAdminNewOrderAlert(order: Order): Promise<void> {
    const adminEmail = this.configService.get('ADMIN_EMAIL');
    const subject = `New Order: ${order.order_number} - MUR ${order.total_amount}`;
    const htmlBody = this.renderAdminOrderAlertEmail(order);

    await this.sendEmail(adminEmail, subject, htmlBody);
  }

  private renderOrderConfirmationEmail(order: Order, user: User): string {
    // Use email template (Handlebars, Pug, or plain HTML)
    return `
      <!DOCTYPE html>
      <html>
      <head>
        <style>
          body { font-family: Arial, sans-serif; line-height: 1.6; color: #333; }
          .container { max-width: 600px; margin: 0 auto; padding: 20px; }
          .header { background: #2E7D32; color: white; padding: 20px; text-align: cen-
ter; }
          .content { padding: 20px; background: #f9f9f9; }
          .order-details { background: white; padding: 15px; margin: 15px 0; border-
left: 4px solid #2E7D32; }
          .footer { text-align: center; padding: 20px; font-size: 12px; color: #666; }
        </style>
      </head>
      <body>
        <div class="container">
          <div class="header">
            <h1>Order Confirmed!</h1>
          </div>
          <div class="content">
            <p>Hi ${user.name},</p>
            <p>Thank you for your order! We've received your purchase request and it's
being processed.</p>

            <div class="order-details">
              <h3>Order ${order.order_number}</h3>
              <p><strong>Total:</strong> MUR ${order.total_amount.toFixed(2)}</p>
              <p><strong>Payment Method:</strong> ${this.formatPaymentMeth-
od(order.payment_method)}</p>
              <p><strong>Status:</strong> ${order.status}</p>
            </div>

            <p>You'll receive another email once your order is approved and ready for
delivery.</p>

            <p>Questions? Reply to this email or contact us at support@freshroots.mu</
p>
          </div>
          <div class="footer">
            <p>Fresh Roots - Fresh Local Produce Delivered</p>
            <p>Port Louis, Mauritius</p>
          </div>
        </div>
      </body>
      </html>
```

```
    `;
  }

  private stripHtml(html: string): string {
    return html.replace(/<[^>]*>/g, '');
  }
}
```

3. Email Templates:

Recommended Approach: Use a template engine like Handlebars

```
import Handlebars from 'handlebars';
import * as fs from 'fs';
import * as path from 'path';

@Injectable()
export class EmailTemplateService {
  private templates: Map<string, HandlebarsTemplateDelegate> = new Map();

  constructor() {
    this.loadTemplates();
  }

  private loadTemplates() {
    const templatesDir = path.join(__dirname, '../../templates/emails');
    const templateFiles = fs.readdirSync(templatesDir);

    for (const file of templateFiles) {
      if (file.endsWith('.hbs')) {
        const templateName = file.replace('.hbs', '');
        const templatePath = path.join(templatesDir, file);
        const templateSource = fs.readFileSync(templatePath, 'utf-8');
        this.templates.set(templateName, Handlebars.compile(templateSource));
      }
    }
  }

  render(templateName: string, context: any): string {
    const template = this.templates.get(templateName);
    if (!template) {
      throw new Error(`Email template '${templateName}' not found`);
    }
    return template(context);
  }
}
```

Email Templates Needed:

```
/templates/emails/
├── order-confirmation.hbs          # Sent immediately after order created
├── order-approved.hbs              # Admin approves order
├── order-rejected.hbs              # Admin rejects order
├── order-out-for-delivery.hbs      # Order dispatched
├── order-delivered.hbs             # Order completed
├── payment-received.hbs            # Payment confirmation
├── password-reset.hbs              # Password reset link
├── welcome.hbs                     # New user registration
└── admin-new-order-alert.hbs       # Admin alert for new orders
```

4. Bounce & Complaint Handling:

AWS SES requires handling bounces and complaints to maintain sender reputation.

```
@Controller('webhooks/ses')
export class SesWebhookController {
  constructor(private emailService: SesEmailService) {}

  @Post('bounce')
  async handleBounce(@Body() notification: any) {
    // SES sends SNS notification to this endpoint
    const message = JSON.parse(notification.Message);

    if (message.notificationType === 'Bounce') {
      const bouncedRecipients = message.bounce.bouncedRecipients;

      for (const recipient of bouncedRecipients) {
        const email = recipient.emailAddress;
        console.log(`Email bounced: ${email}`);

        // Mark email as invalid in database
        await this.usersService.markEmailInvalid(email);
      }
    }

    return { success: true };
  }

  @Post('complaint')
  async handleComplaint(@Body() notification: any) {
    const message = JSON.parse(notification.Message);

    if (message.notificationType === 'Complaint') {
      const complainedRecipients = message.complaint.complainedRecipients;

      for (const recipient of complainedRecipients) {
        const email = recipient.emailAddress;
        console.log(`Complaint received from: ${email}`);

        // Unsubscribe user from marketing emails
        await this.usersService.setEmailPreference(email, 'unsubscribed');
      }
    }

    return { success: true };
  }
}
```

Setup in AWS:

```
1. Create SNS topics:
   - ses-bounce-topic
   - ses-complaint-topic

2. Subscribe topics to Fresh Roots webhook endpoints:
   - https://api.freshroots.mu/v1/webhooks/ses/bounce
   - https://api.freshroots.mu/v1/webhooks/ses/complaint

3. Configure SES to publish to these topics
```

9.3.2 Email Deliverability Best Practices

1. Proper DNS Configuration:

```
Essential Records:
- SPF: v=spf1 include:amazonses.com ~all
- DKIM: Multiple TXT records provided by SES (auto-verify sender)
- DMARC: v=DMARC1; p=none; rua=mailto:dmarc@freshroots.mu
  └─> p=none (monitoring mode), upgrade to p=quarantine after 1 month of clean sending

Check with: https://mxtoolbox.com/dmarc.aspx
```

2. Warm Up Sending Reputation:

```
Don't send high volume immediately after account approval

Week 1: 50-100 emails/day
Week 2: 200-500 emails/day
Week 3: 1,000-2,000 emails/day
Week 4+: Unlimited (within SES limits)

Rationale: Gradual increase builds sender reputation with ISPs
```

3. Email Content Best Practices:

```
✓ Always include plain text alternative (not just HTML)
✓ Avoid spam trigger words: "free", "click here", "act now", excessive caps
✓ Include unsubscribe link for marketing emails (legal requirement)
✓ Personalize with recipient name
✓ Use clear "From" name: "Fresh Roots" not "noreply@freshroots.mu"
✓ Include physical address in footer (legal requirement for commercial email)
✗ Don't send to purchased email lists
✗ Don't send without permission (opt-in)
```

4. Monitor Metrics:

```
Key SES Metrics (Monitor in CloudWatch):
- Bounce Rate: Should be < 5% (bounces/sent)
- Complaint Rate: Should be < 0.1% (complaints/sent)
- Delivery Rate: Should be > 95%

If bounce rate > 5% for sustained period:
→ AWS may pause sending ability
→ Clean email list, remove invalid addresses
```

9.4 In-App Notifications

Purpose: Notification center within the app (separate from push notifications)

Implementation:

Mobile App - Notification Center Screen:

```
import React, { useEffect, useState } from 'react';
import { FlatList, TouchableOpacity, View, Text } from 'react-native';
import { Badge, Card, IconButton } from 'react-native-paper';
import { useNavigation } from '@react-navigation/native';
import api from '../services/api';

interface Notification {
  id: string;
  type: string;
  title: string;
  message: string;
  read: boolean;
  action_url: string;
  created_at: string;
}

export function NotificationsScreen() {
  const [notifications, setNotifications] = useState<Notification[]>([]);
  const [unreadCount, setUnreadCount] = useState(0);
  const navigation = useNavigation();

  useEffect(() => {
    fetchNotifications();
  }, []);

  async function fetchNotifications() {
    const response = await api.get('/notifications');
    setNotifications(response.data.data);
    setUnreadCount(response.data.meta.unread_count);
  }

  async function markAsRead(notificationId: string) {
    await api.patch(`/notifications/${notificationId}/read`);
    setNotifications(prev =>
      prev.map(n => (n.id === notificationId ? { ...n, read: true } : n))
    );
    setUnreadCount(prev => Math.max(0, prev - 1));
  }

  async function handleNotificationPress(notification: Notification) {
    if (!notification.read) {
      await markAsRead(notification.id);
    }

    // Navigate to relevant screen based on action_url
    if (notification.action_url) {
      // Parse deep link: freshroots://orders/uuid
      const [, screen, id] = notification.action_url.split('/').slice(-3);
      if (screen === 'orders') {
        navigation.navigate('OrderDetail', { orderId: id });
      }
    }
  }

  function renderNotification({ item }: { item: Notification }) {
    return (
      <TouchableOpacity onPress={() => handleNotificationPress(item)}>
        <Card style={{ margin: 8, opacity: item.read ? 0.6 : 1 }}>
          <Card.Content>
            <View style={{ flexDirection: 'row', justifyContent: 'space-between' }}>
              <Text style={{ fontWeight: 'bold' }}>{item.title}</Text>
              {!item.read && <Badge>NEW</Badge>}
```

```
            </View>
            <Text>{item.message}</Text>
            <Text style={{ fontSize: 12, color: '#666', marginTop: 8 }}>
              {new Date(item.created_at).toLocaleString()}
            </Text>
          </Card.Content>
        </Card>
      </TouchableOpacity>
    );
  }

  return (
    <View style={{ flex: 1 }}>
      <View style={{ flexDirection: 'row', justifyContent: 'space-between', padding: 1
6 }}>
        <Text style={{ fontSize: 20, fontWeight: 'bold' }}>Notifications</Text>
        {unreadCount > 0 && (
          <Text>{unreadCount} unread</Text>
        )}
      </View>
      <FlatList
        data={notifications}
        renderItem={renderNotification}
        keyExtractor={item => item.id}
        ListEmptyComponent={
          <Text style={{ textAlign: 'center', marginTop: 50, color: '#666' }}>
            No notifications yet
          </Text>
        }
      />
    </View>
  );
}
```

Badge on Tab Icon:

```
// In bottom tab navigator
<Tab.Screen
  name="Notifications"
  component={NotificationsScreen}
  options={{
    tabBarBadge: unreadNotificationsCount > 0 ? unreadNotificationsCount : undefined,
    tabBarIcon: ({ color }) => <Icon name="bell" size={24} color={color} />,
  }}
/>
```

Real-Time Updates (Admin Dashboard):

For admin dashboard, use WebSocket for real-time notification of new orders.

```
// Backend - WebSocket Gateway
import { WebSocketGateway, WebSocketServer, SubscribeMessage } from '@nestjs/websock-
ets';
import { Server, Socket } from 'socket.io';

@WebSocketGateway({ cors: true })
export class NotificationsGateway {
  @WebSocketServer()
  server: Server;

  // Emit to all connected admins
  notifyAdmins(event: string, data: any) {
    this.server.to('admin-room').emit(event, data);
  }

  @SubscribeMessage('join-admin')
  handleAdminJoin(client: Socket, payload: { adminId: string }) {
    // Verify admin JWT from payload
    client.join('admin-room');
    console.log(`Admin ${payload.adminId} joined notifications room`);
  }
}

// Usage in OrdersService
async createOrder(orderData: CreateOrderDto): Promise<Order> {
  const order = await this.ordersRepository.save(orderData);

  // Emit real-time event to admin dashboard
  this.notificationsGateway.notifyAdmins('new_order', {
    order_id: order.id,
    order_number: order.order_number,
    total_amount: order.total_amount,
    created_at: order.created_at,
  });

  return order;
}
```

9.5 Notification Event Schema

Standardized event structure for PostHog analytics:

```
enum NotificationType {
  PUSH = 'push',
  EMAIL = 'email',
  IN_APP = 'in_app',
}

interface NotificationEvent {
  event_name: 'notification_sent' | 'notification_opened' | 'notification_clicked';
  notification_type: NotificationType;
  notification_category: 'order_update' | 'payment' | 'promotional' | 'admin_alert';
  user_id: string;
  order_id?: string;
  timestamp: string;
}

// Track when notification is sent
analytics.track('notification_sent', {
  notification_type: NotificationType.PUSH,
  notification_category: 'order_update',
  user_id: user.id,
  order_id: order.id,
});

// Track when user opens notification
analytics.track('notification_opened', {
  notification_type: NotificationType.PUSH,
  notification_category: 'order_update',
  user_id: user.id,
  order_id: order.id,
});
```

Analytics Queries:
- Push notification open rate by category
- Email open rate by template
- Most effective notification type for conversions
- Time-to-open by notification category

Document continues…

Due to length constraints, I'll continue this comprehensive system design document. The remaining sections (10-20) will follow the same level of detail and thoroughness.

Would you like me to continue with the remaining sections:
- Section 10: Analytics Implementation Plan
- Section 11: UI/UX Specification
- Section 12: Performance, Scalability & Reliability Plan
- Section 13: Security Plan
- Section 14: Testing Strategy
- Section 15: CI/CD Pipeline Recommendations
- Section 16: Cost Estimate for 12 Months
- Section 17: Week-by-Week Milestone Plan
- Section 18: Risk Register
- Section 19: Open Questions & Required Items
- Section 20: Next-Phase Roadmap

?