REQUIREMENT ANALYSIS DOCUMENT

1. Project Overview

1.1 Project Title

Smart Informal Business Credit & Record App

1.2 Project Description

This mobile application is designed to empower informal traders and small business owners in African markets by providing tools to record financial transactions, build a verifiable financial history, generate creditworthiness scores, and facilitate access to credit from lenders or cooperatives. The app addresses key challenges such as lack of formal financial records, poor internet connectivity, reliance on mobile money, and the need for a simple user interface for non-technical users.

The app will have two primary sides:

- **User Side**: For business owners to log transactions, view summaries, and share reports.
- **Admin/Partner Side**: For verification, scoring management, and analytics accessible by admins, lenders, or cooperatives.

The solution will be offline-first, ensuring functionality in areas with unreliable internet, with automatic synchronization when connectivity is available.

1.3 Objectives

- Enable easy recording of daily sales and expenses to build a financial history.
- Automatically compute a creditworthiness score based on transaction data.
- Integrate with mobile money services (e.g., MTN MoMo, Orange Money) for seamless transactions.
- Provide actionable insights and growth suggestions to users.
- Allow secure sharing of verified financial reports.
- Support administrative oversight for verification, risk analysis, and partner access.
- Ensure the app is accessible, secure, and scalable for African markets.

1.4 Scope

In Scope:

- Mobile app for Android and iOS using React Native (Expo).
- Backend server with Node.js and PostgreSQL database.
- Offline-first architecture with auto-sync.
- Core features for user logging, summaries, credit scoring, and report sharing.
- Integration with mobile money APIs (focusing on MTN and Orange styles).
- Admin portal for verification, scoring rules, and analytics.
- Basic insights and growth suggestions based on data analysis.

Out of Scope:

- Full banking integration (e.g., direct loan disbursement).
- Advanced AI/ML for predictions beyond basic scoring.
- Multi-language support beyond English (unless specified later).
- Hardware integrations (e.g., receipt printers).
- Compliance with specific regional regulations (to be handled post-analysis if needed).

1.5 Assumptions and Constraints

Assumptions:

- Users have basic smartphones capable of running React Native apps.
- Mobile money APIs from providers like MTN and Orange are accessible and documented.
- Internet is intermittently available for sync; app must handle prolonged offline periods.
- Data privacy complies with general standards (e.g., GDPR-like principles for African contexts).
- Credit scoring algorithm will be rule-based initially, with potential for enhancements.

Constraints:

- Poor internet: Prioritize offline functionality.
- Simple UX: Design for low literacy and non-tech-savvy users (e.g., 1-tap actions).
- Security: Handle sensitive financial data with encryption and role-based access.
- Tech Stack: Must use React Native (Expo), Node.js, PostgreSQL.
- Budget/Time: Not specified; assume iterative development.

2. Stakeholders

- **Client (Business Owner)**: Oversees the network of informal traders; primary beneficiary for admin features.
- **End Users (Informal Business Owners)**: Market sellers, mobile vendors, service providers; use the app daily for logging and insights.
- **Lenders/Cooperatives**: Access verified reports and analytics for credit decisions.
- **Admins/Partners**: Manage verifications, scoring rules, and risk analytics.
- **Developers**: Full stack developer and me (AI assistant) for implementation.

3. Functional Requirements

3.1 User Side (Mobile App)

Transaction Logging:

- 1-tap entry for sales and expenses (e.g., categorize as cash, mobile money).
- Support offline entry; store locally and sync when online.

Financial Summaries:

- Generate weekly/monthly profit overviews (e.g., charts, totals).
- Display historical trends.

Credit Score Dashboard:

- Real-time score based on transaction history, consistency, and other factors.

- Visual indicators (e.g., gauge or color-coded).

Mobile Money Integration:

- Record transactions via MTN/Orange-style APIs (e.g., send/receive confirmations).
- Auto-log payments/receipts.

Insights & Suggestions:

- Basic analytics: e.g., "Increase sales by X% to improve score" or expense reduction tips.

Report Export/Share:

- Generate PDF reports of financial history.
- Share securely with lenders (e.g., via email, link, or in-app portal).

3.2 Admin/Partner Side (Web/Admin Portal)

Business Verification:

- Admin approval workflow for user accounts (e.g., ID upload, manual review).

Credit Score Rules Engine:

- Configurable rules for scoring (e.g., weights for sales volume, expense ratios).
- Secure algorithm to prevent tampering.

Access Portal:

- Role-based login for lenders/cooperatives to view user reports (with consent).

Analytics:

- Risk assessment (e.g., repayment probability based on history).
- Aggregate data for network-wide insights (anonymized).

3.3 Non-Functional Requirements

- **Performance**: Offline-first; sync within 5-10 seconds when online.

- **Usability**: Simple, intuitive UI/UX; large buttons, minimal text, voice input if feasible.
- **Security**:
  - Data encryption at rest/transit.
  - Role-based access control (RBAC).
  - Secure scoring to avoid reverse-engineering.
- **Reliability**: Handle crashes gracefully; auto-backup local data.
- **Scalability**: Cloud-ready (e.g., AWS/Heroku); support 1,000+ users initially.
- **Accessibility**: Support for low-bandwidth; dark mode; large fonts.
- **Compatibility**: Android 8+ and iOS 12+.

4. Technical Architecture

4.1 High-Level Design

- **Frontend**: React Native (Expo) for cross-platform mobile app.
- **Backend**: Node.js server with RESTful APIs or GraphQL for data handling.
- **Database**: PostgreSQL for structured data (users, transactions, scores).
- **Offline Sync**: Use libraries like Realm or SQLite for local storage; sync engine (e.g., WatermelonDB or custom with Node.js).
- **Integrations**: Mobile money APIs (e.g., via SDKs); PDF generation (e.g., react-native-pdf-lib).
- **Deployment**: Cloud infrastructure (e.g., AWS EC2 for Node.js, RDS for PostgreSQL).

4.2 Data Flow

1. User logs transaction offline → Stored locally.
2. When online → Sync to backend → Update financial history and score.
3. Score computed on backend (securely) → Pushed back to app.
4. Reports generated on-demand → Shared via secure links.

5. Admin accesses via web portal → Views/Edits rules and analytics.

5. Risks and Mitigations

- **Risk**: Unreliable mobile money APIs → Mitigation: Use fallback manual entry; test integrations early.
- **Risk**: Data privacy breaches → Mitigation: Implement encryption and consent mechanisms.
- **Risk**: Poor adoption due to UX → Mitigation: User testing with target audience.
- **Risk**: Sync conflicts → Mitigation: Use conflict resolution strategies (e.g., last-write-wins).
- **Risk**: Scoring bias → Mitigation: Transparent rules; periodic reviews.