# **CHAPTER FOUR: SYSTEM DESIGN AND IMPLEMENTATION**

### **4.1 Objective of Design**

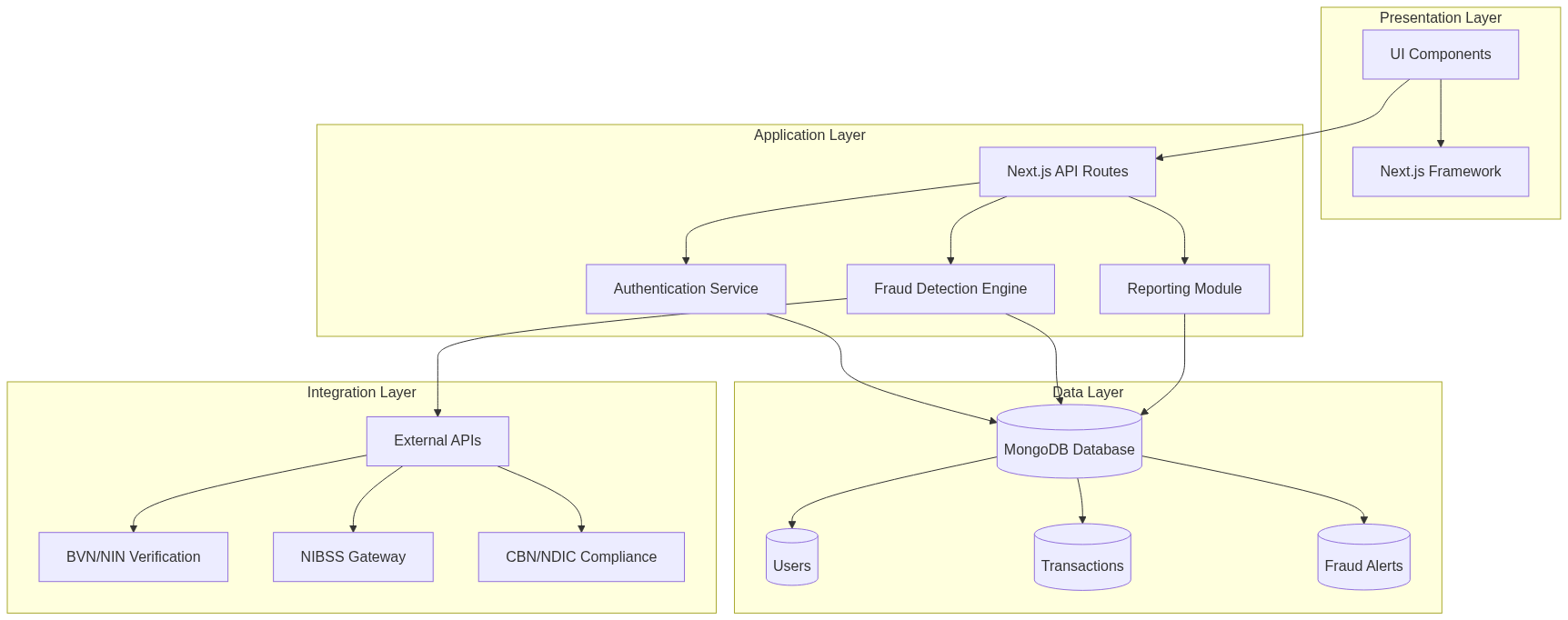
The design of the Fraud Control System for Cross River University of Technology (CRUTECH) Microfinance Bank aims to transform the identified requirements into a structured system that ensures **fraud prevention, real-time monitoring, and secure banking operations**. Specifically, the design seeks to:

* Provide **multi-factor authentication** to strengthen system security.
* Create a **modular and scalable architecture** to integrate fraud detection with banking services.
* Automate **real-time monitoring and alerts** for suspicious transactions.
* Ensure **regulatory compliance** with CBN and NDIC reporting standards.
* Deliver a **user-friendly interface** for both customers and staff.

### **4.2 System Architecture in Terms of Tiers**

The system adopts a **four-tier architecture** for scalability and robustness:

1. **Presentation Layer (UI/UX)**
   * Provides access portals for customers, staff, and administrators.
   * Designed with **HTML5, CSS3, JavaScript, and Next.js** for responsive web interfaces.
2. **Application Layer (Business Logic)**
   * Hosts the **fraud detection engine**, authentication services, and reporting module.
   * Implements **rule-based and AI-driven anomaly detection algorithms**.
3. **Data Layer**
   * Centralized **PostgreSQL/MongoDB database** for transactions, logs, audit trails, and fraud alerts.
   * Ensures referential integrity and secure data storage.
4. **Integration Layer**
   * Connects external services such as **BVN/NIN APIs, NIBSS Gateway, CBN/NDIC compliance portals**.



*Figure 4.1: Fraud Control System Architecture Diagram*

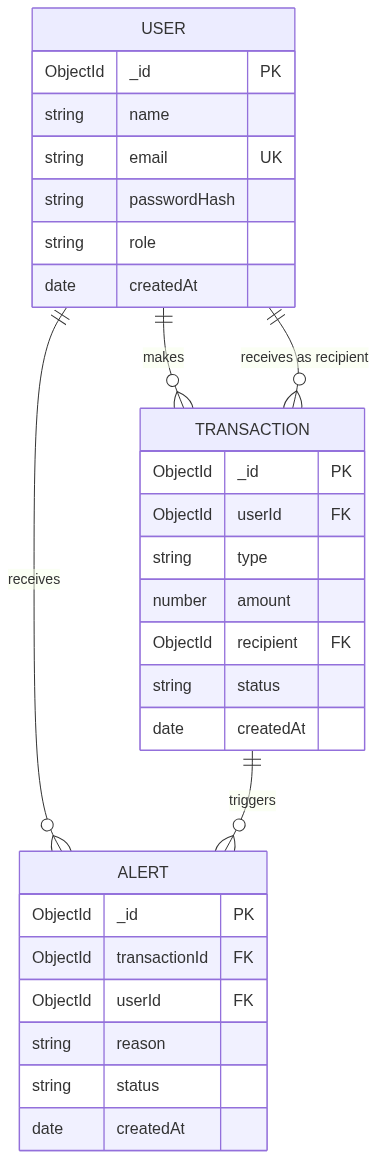
### **4.3 Choice of Programming Environment**

* **Frontend:** Next.js, React, Tailwind CSS
* **Backend:** Node.js with Next.js API routes
* **Database:** MongoDB for users, transactions, and fraud alerts
* **Authentication:** NextAuth.js with JWT
* **State Management:** React hooks and context API
* **Deployment:** Vercel / AWS with GitHub version control

### **4.4 Database Structure**

The database maintains strong relationships among **users, transactions, and fraud alerts**. Major entities include:

* **Users**: Customers, staff, administrators with roles-based access
* **Transactions**: Banking operations (withdrawals, deposits, transfers) with status tracking
* **Fraud Alerts**: Suspicious activities flagged by the detection engine with specific reasons and status



*Figure 4.2: Entity-Relationship Diagram – ERD*

### **4.5 Database Table Definition**

**User Table**

| **Field** | **Type** | **Attributes** |
| --- | --- | --- |
| \_id | ObjectId | Primary Key |
| name | String | Required, Max length: 60 |
| email | String | Unique, Required, Validated |
| passwordHash | String | Required |
| role | String | Enum: [customer, staff, admin], Default: customer |
| createdAt | Date | Default: current timestamp |

**Transaction Table**

| **Field** | **Type** | **Attributes** |
| --- | --- | --- |
| \_id | ObjectId | Primary Key |
| userId | ObjectId | Foreign Key → User, Required |
| type | String | Enum: [withdrawal, deposit, transfer], Required |
| amount | Number | Required, Min: 1 |
| recipient | ObjectId | Foreign Key → User, Required for transfers |
| status | String | Enum: [pending, approved, rejected], Default: pending |
| createdAt | Date | Default: current timestamp |

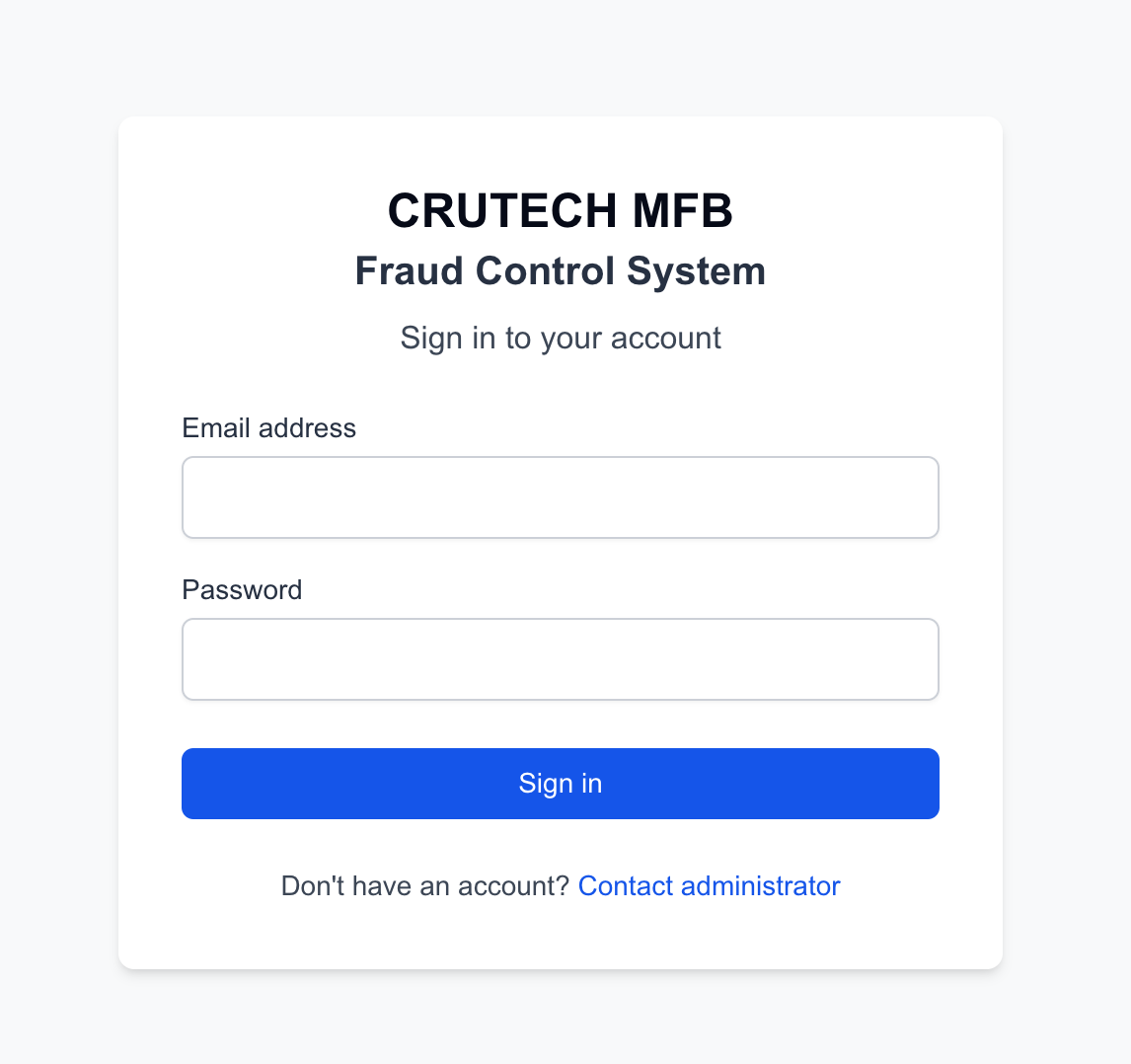
**Fraud Alert Table**

| **Field** | **Type** | **Attributes** |
| --- | --- | --- |
| \_id | ObjectId | Primary Key |
| transactionId | ObjectId | Foreign Key → Transaction, Required |
| userId | ObjectId | Foreign Key → User, Required |
| reason | String | Required |
| status | String | Enum: [open, resolved], Default: open |
| createdAt | Date | Default: current timestamp |

### **4.6 Input and Output Screen Formats**

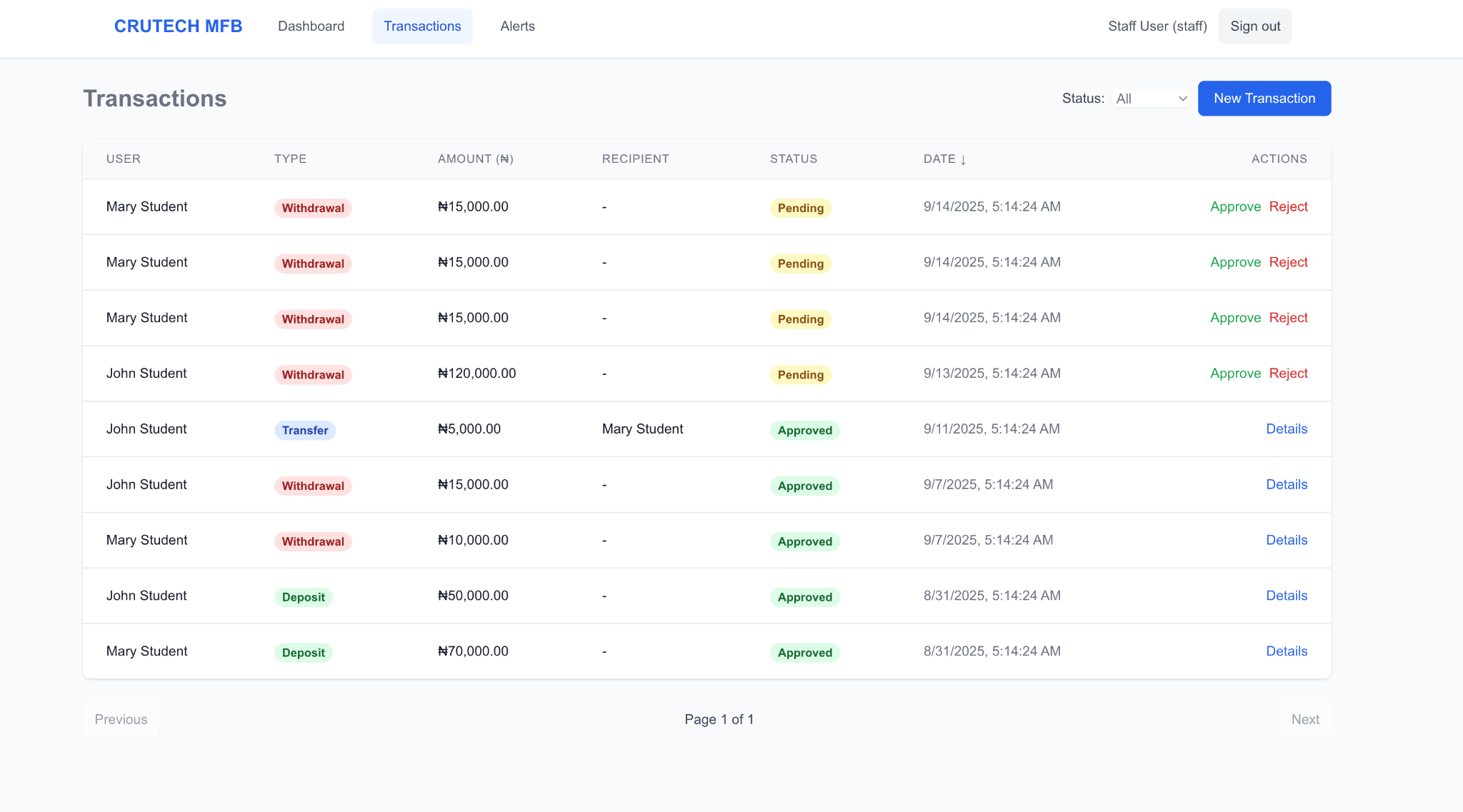
Input Screens

* Login Screen: User authentication screen with email and password fields



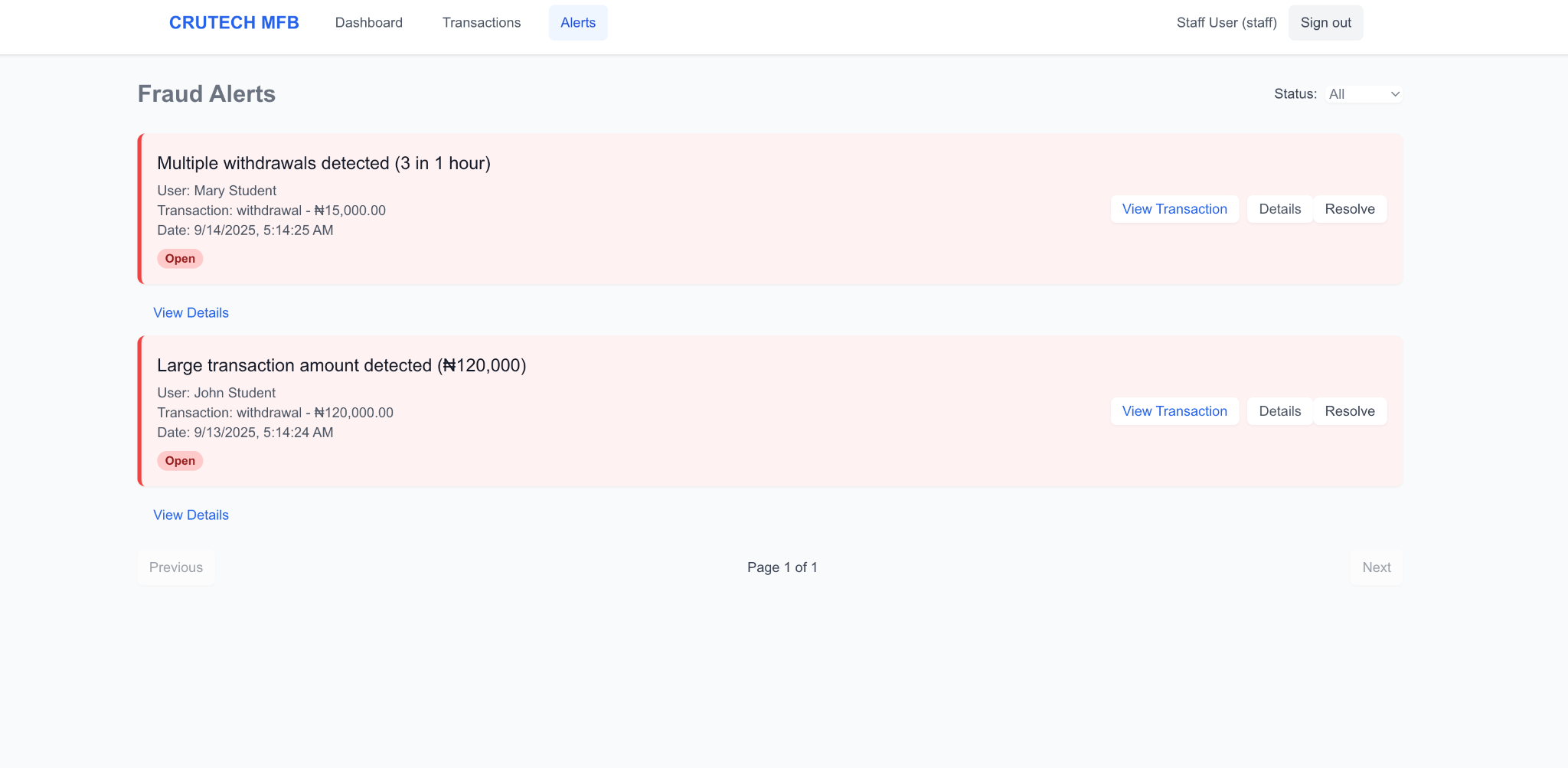
*Figures 4.3 Login Screens*

* Transaction Management: Interface for creating and managing deposits, withdrawals, and transfers with validation



*Figures 4.4 Transaction* *Management*

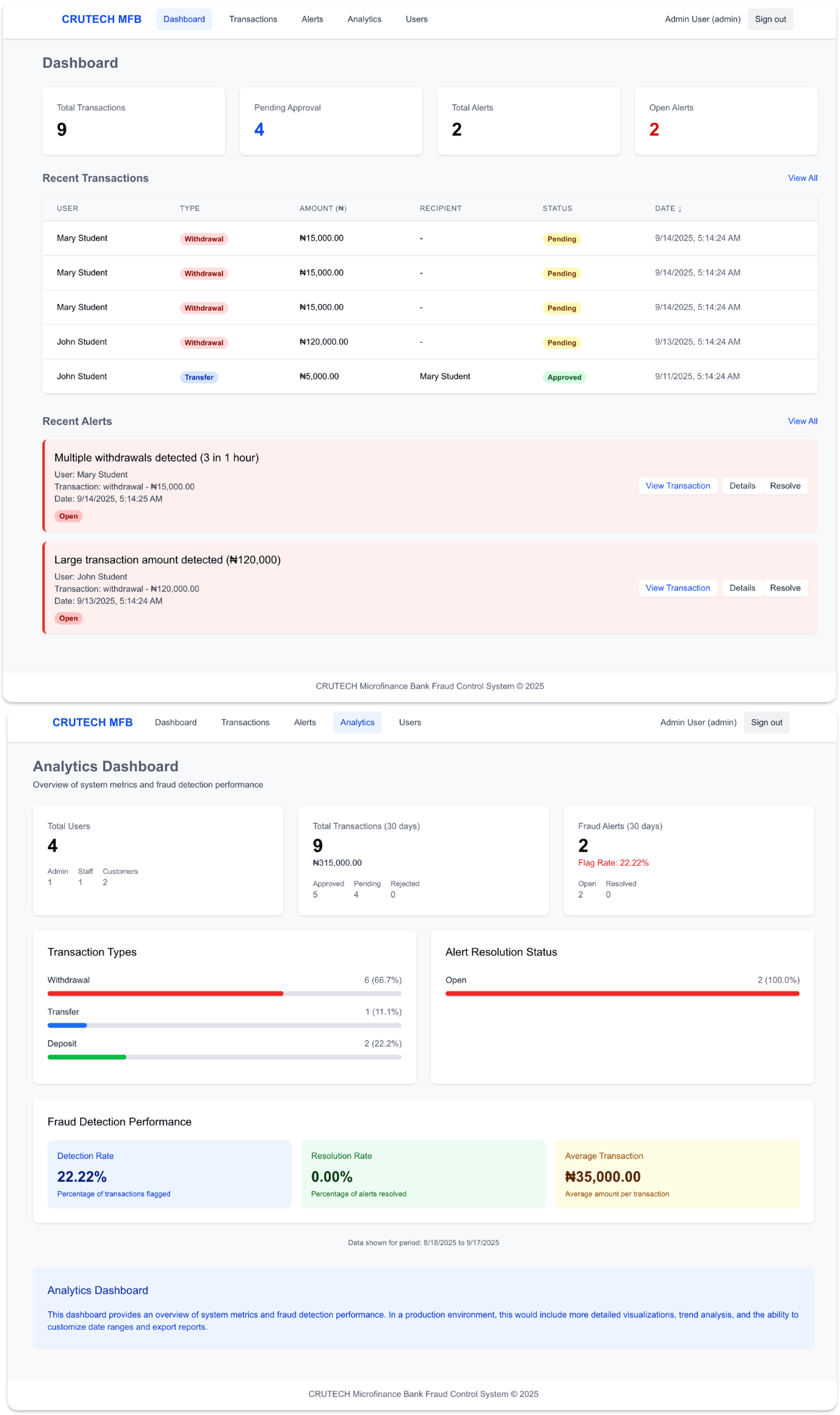
* Alert Resolution Interface: Form for staff to review and resolve flagged fraud alerts



*Figures 4.5* Alert Resolution Interface

Output Screens

* Role-Based Dashboard: Dashboard showing transaction statistics and recent activities customized by user role
* Transactions List: Comprehensive list of transactions with filtering by status and pagination as seen in *Figures 4.3*
* Fraud Alerts Dashboard: Displays flagged transactions with details, allowing staff to review and resolve alerts as seen in *Figures 4.3*
* Analytics Dashboard: Admin-only screen with detailed metrics, transaction patterns, and fraud statistics



*Figures 4.6* Role-Based Dashboard / Analytics Dashboard

### **4.7 Program Algorithm**

**Login Algorithm:**

* Input email and password → Validate credentials → Check user role → Grant role-based access

**Transaction Algorithm:**

* Input transaction details → Validate input → Process transaction → Run fraud detection rules → Create alert if fraudulent → Update transaction status

**Fraud Detection Algorithm:**

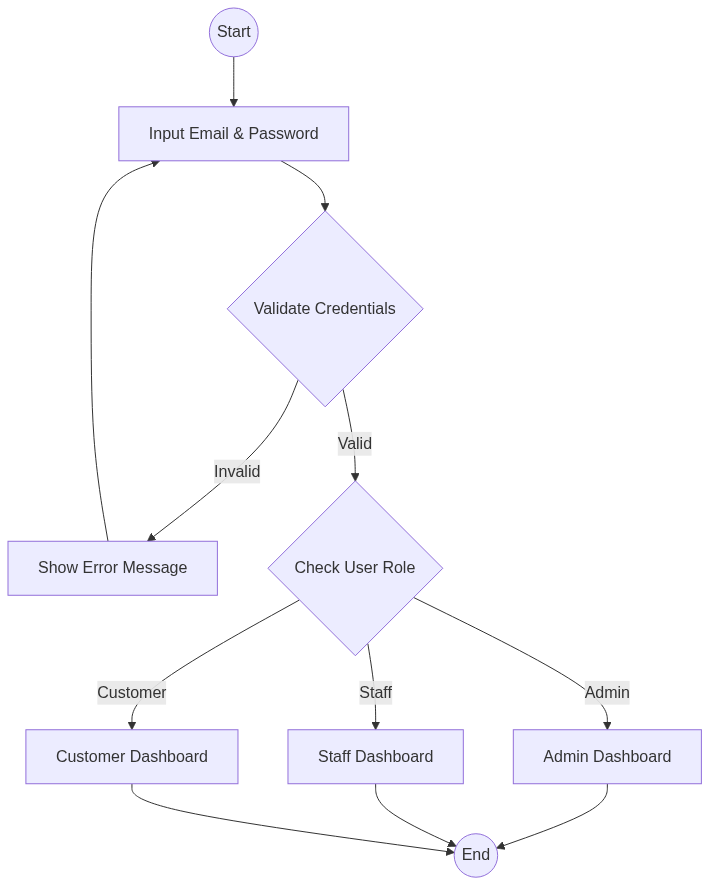
* The system uses a rule-based detection engine that evaluates each transaction against multiple fraud rules:
  + **Large Amount Detection**: Flags transactions above ₦100,000
  + **Multiple Transactions**: Detects when a user makes 5 or more transactions within 24 hours
  + **Unusual Transaction Time**: Identifies transactions occurring between 11 PM and 5 AM
  + **Rapid Sequential Withdrawals**: Alerts when 3 or more withdrawals occur within 1 hour

**Reporting Algorithm:**

* Fetch transaction and alert data → Process and aggregate metrics → Calculate statistics (flag rate, resolution rate) → Generate visualizations → Present analytics dashboard

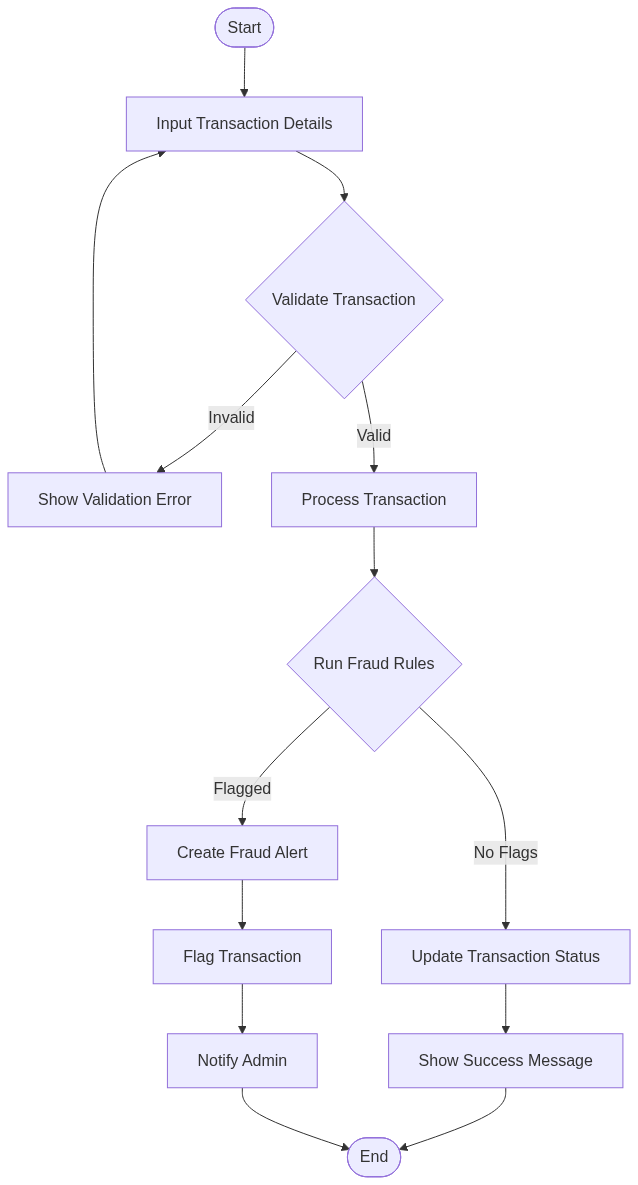
### **4.8 Program Flowcharts**

#### **4.8.1 User Login Flowchart**



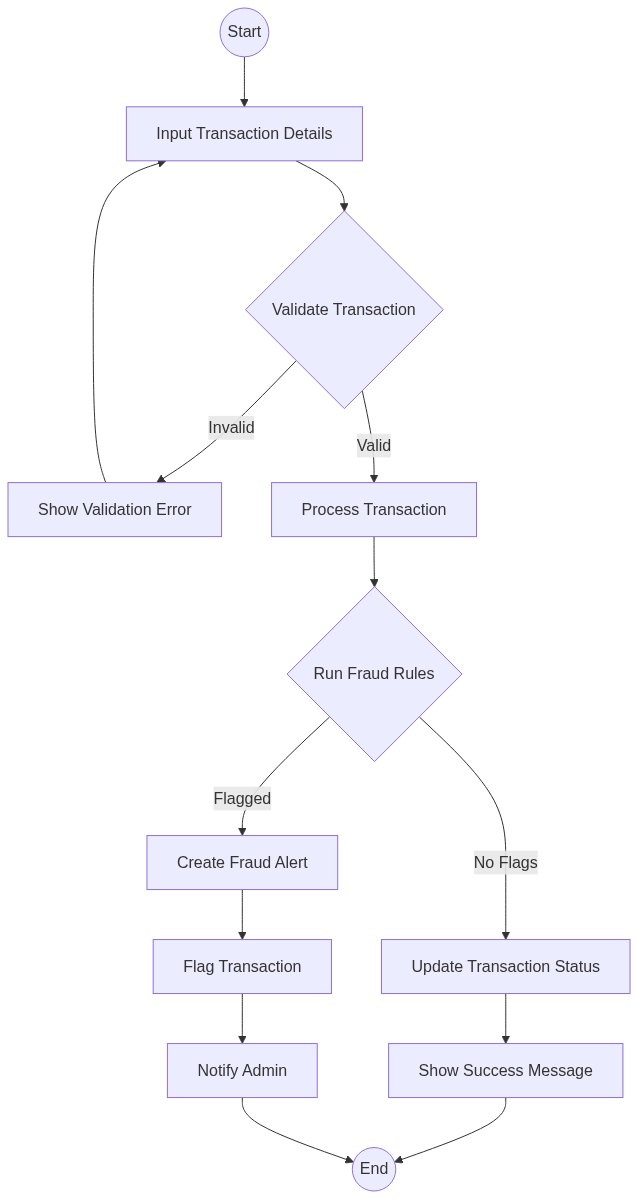
*Figure 4.7: User Login Flowchart*

#### **4.8.2 Transaction & Fraud Detection Flowchart**



*Figure 4.8: Transaction & Fraud Detection Flowchart*

#### **4.8.3 Reporting Flowchart**



*Figure 4.9: Reporting Flowchart*

### **4.9 Hardware Requirements**

* **Server**: Minimum 8GB RAM, Quad-Core CPU, 100GB SSD
* **Client Devices**: Any smartphone or PC with modern browser
* **Network**: Stable broadband internet connection

### **4.10 Software Requirements**

* **OS**: Any OS supporting Node.js (Windows/macOS/Linux)
* **Runtime Environment**: Node.js
* **Database**: MongoDB
* **Frontend Framework**: Next.js with React
* **Styling**: Tailwind CSS
* **Authentication**: NextAuth.js, JWT
* **Version Control**: Git/GitHub
* **Deployment**: Vercel/AWS

### **4.11 Documentation**

Documentation for the system includes:

* **User Manual** - Separate guides for customers, staff, and administrators
* **API Documentation** - Documentation of all API routes and authentication requirements
* **Database Schema** - Complete MongoDB schema with relationships
* **Installation Guide** - Step-by-step setup instructions for development and production
* **Fraud Rules Documentation** - Details on implemented fraud detection rules and thresholds

# **CHAPTER FIVE: SUMMARY, CONCLUSION, AND RECOMMENDATION**

### **5.1 Summary**

This project designed and implemented a **Fraud Control System for CRUTECH Microfinance Bank** to address the inefficiencies and risks of manual fraud detection.

* **Chapter One** introduced the study and problem statement.
* **Chapter Two** reviewed fraud concepts, causes, and control systems.
* **Chapter Three** analyzed the existing manual system and proposed an automated solution.
* **Chapter Four** presented the system design, architecture, database structure, algorithms, and implementation details.

The system integrates **real-time monitoring, multi-factor authentication, and intelligent fraud detection** to improve operational efficiency and compliance.

### **5.2 Conclusion**

The implemented system provides:

* A **secure platform** that minimizes fraud risk.
* **Automated fraud detection** using rule-based algorithms and anomaly detection.
* **Centralized database and reporting tools** that ensure regulatory compliance.
* **Improved trust** between customers and the Microfinance Bank through real-time alerts and transparent reporting.

The project demonstrates that fraud in microfinance banks can be significantly reduced with the deployment of an intelligent, automated control system.

### **5.3 Recommendations**

For future improvements to the fraud control system:

1. **Enhanced Fraud Detection Rules** - Expand the existing rule-based engine with more sophisticated patterns and adaptive thresholds based on user transaction history.
2. **Machine Learning Integration** - Implement supervised learning algorithms to detect anomalies beyond the current rule-based approach, enabling the system to evolve with new fraud patterns.
3. **Real-time Notification System** - Add SMS and email notifications for both customers and staff when suspicious activities are detected.
4. **Two-Factor Authentication Enhancement** - Strengthen the authentication process by adding additional verification methods like OTP via mobile app.
5. **Mobile Application** - Develop dedicated Android/iOS apps for customers to enhance user experience and provide real-time fraud alerts.
6. **Transaction Analytics Dashboard** - Create more detailed visualization tools for staff to identify patterns and trends in transaction data.
7. **API Integration** - Connect with external financial systems like BVN/NIN verification services to validate customer identity.

With these improvements, CRUTECH Microfinance Bank will be better positioned to safeguard assets, ensure compliance, and maintain customer confidence.