Table of Contents

# 1 Technical Specification Document

## 1.1 Blockchain Fraud Detection Dashboard System

**Document Version:** 1.0  
**Date:** July 20, 2025  
**Project:** Blockchain Fraud Detection Dashboard  
**Status:** Successfully Deployed on Streamlit Cloud

## 1.2 Table of Contents

1. [Executive Summary](#executive-summary)
2. [System Overview](#system-overview)
3. [Architecture Design](#architecture-design)
4. [Technical Requirements](#technical-requirements)
5. [Blockchain Implementation](#blockchain-implementation)
6. [Fraud Detection Algorithm](#fraud-detection-algorithm)
7. [User Interface Design](#user-interface-design)
8. [API Specifications](#api-specifications)
9. [Security Considerations](#security-considerations)
10. [Deployment Architecture](#deployment-architecture)
11. [Performance Metrics](#performance-metrics)
12. [Testing Strategy](#testing-strategy)
13. [Maintenance and Support](#maintenance-and-support)
14. [Future Enhancements](#future-enhancements)

## 1.3 1. Executive Summary

### 1.3.1 1.1 Project Overview

The Blockchain Fraud Detection Dashboard is a comprehensive system that combines blockchain technology with advanced fraud detection algorithms to provide real-time transaction monitoring and risk assessment capabilities. The system has been successfully deployed on Streamlit Cloud and demonstrates the practical application of blockchain technology in cybersecurity.

### 1.3.2 1.2 Key Achievements

* ✅ **Successful Cloud Deployment** - Live on Streamlit Cloud
* ✅ **Blockchain Implementation** - Proof-of-work consensus mechanism
* ✅ **Real-time Fraud Detection** - Multi-level risk assessment
* ✅ **Interactive Dashboard** - User-friendly interface
* ✅ **Zero Dependencies** - Pure Python implementation

### 1.3.3 1.3 Business Value

* **Enhanced Security** - Blockchain-immutable transaction records
* **Real-time Monitoring** - Instant fraud detection and alerts
* **Scalable Architecture** - Cloud-based deployment
* **Cost-effective Solution** - Open-source implementation

## 1.4 2. System Overview

### 1.4.1 2.1 System Purpose

The system provides a comprehensive platform for: - Real-time transaction monitoring and validation - Blockchain-based transaction immutability - Multi-level fraud risk assessment - Interactive visualization of blockchain data - Smart contract-based rule enforcement

### 1.4.2 2.2 Core Components

1. **Blockchain Core** - Transaction processing and block mining
2. **Fraud Detection Engine** - Risk assessment algorithms
3. **Smart Contracts** - Rule-based validation system
4. **Web Dashboard** - User interface and visualization
5. **API Layer** - RESTful service endpoints

### 1.4.3 2.3 System Architecture

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐  
│ Web Dashboard │ │ Blockchain API │ │ Smart Contracts│  
│ (Streamlit) │◄──►│ (Flask) │◄──►│ (Python) │  
└─────────────────┘ └─────────────────┘ └─────────────────┘  
 │ │ │  
 ▼ ▼ ▼  
┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐  
│ Session State │ │ Blockchain │ │ Fraud Detection│  
│ Management │ │ Core Engine │ │ Algorithms │  
└─────────────────┘ └─────────────────┘ └─────────────────┘

## 1.5 3. Architecture Design

### 1.5.1 3.1 High-Level Architecture

The system follows a modular, microservices-based architecture with the following layers:

#### 1.5.1.1 3.1.1 Presentation Layer

* **Technology:** Streamlit
* **Purpose:** User interface and data visualization
* **Features:** Real-time updates, interactive controls, responsive design

#### 1.5.1.2 3.1.2 Application Layer

* **Technology:** Python Flask API
* **Purpose:** Business logic and data processing
* **Features:** RESTful endpoints, transaction validation, blockchain operations

#### 1.5.1.3 3.1.3 Data Layer

* **Technology:** In-memory session state
* **Purpose:** Transaction storage and blockchain state management
* **Features:** Persistent session data, real-time updates

### 1.5.2 3.2 Component Interactions

User Action → Streamlit UI → Flask API → Blockchain Engine → Smart Contracts  
 ↑ ↓  
 └─────────────── Session State ←──────────────────────────────┘

## 1.6 4. Technical Requirements

### 1.6.1 4.1 System Requirements

#### 1.6.1.1 4.1.1 Hardware Requirements

* **CPU:** Minimum 1 core, Recommended 2+ cores
* **RAM:** Minimum 512MB, Recommended 1GB+
* **Storage:** Minimum 100MB, Recommended 500MB+
* **Network:** Internet connection for cloud deployment

#### 1.6.1.2 4.1.2 Software Requirements

* **Operating System:** Cross-platform (Windows, macOS, Linux)
* **Python Version:** 3.9 or higher
* **Web Browser:** Modern browser with JavaScript support

### 1.6.2 4.2 Dependencies

#### 1.6.2.1 4.2.1 Core Dependencies

# Minimal dependencies for cloud deployment  
streamlit>=1.28.0  
flask>=2.3.0  
flask-cors>=4.0.0

#### 1.6.2.2 4.2.2 Optional Dependencies

# Enhanced features (when needed)  
pandas>=1.5.0  
numpy>=1.24.0  
plotly>=5.15.0  
requests>=2.31.0

### 1.6.3 4.3 Development Environment

* **IDE:** VS Code, PyCharm, or any Python IDE
* **Version Control:** Git
* **Package Manager:** pip
* **Virtual Environment:** venv or conda

## 1.7 5. Blockchain Implementation

### 1.7.1 5.1 Blockchain Structure

#### 1.7.1.1 5.1.1 Block Definition

@dataclass  
class Block:  
 index: int # Block number in chain  
 timestamp: float # Creation timestamp  
 transactions: List[Transaction] # List of transactions  
 previous\_hash: str # Hash of previous block  
 nonce: int # Proof-of-work nonce  
 merkle\_root: str # Merkle tree root  
 block\_hash: str # Current block hash

#### 1.7.1.2 5.1.2 Transaction Structure

@dataclass  
class Transaction:  
 transaction\_id: str # Unique transaction ID  
 customer\_id: str # Customer identifier  
 amount: float # Transaction amount  
 merchant\_id: str # Merchant identifier  
 timestamp: float # Transaction timestamp  
 location: str # Transaction location  
 payment\_method: str # Payment method used  
 risk\_score: float # Calculated risk score  
 risk\_level: RiskLevel # Risk level classification  
 fraud\_probability: float # Fraud probability  
 metadata: Dict[str, Any] # Additional transaction data

### 1.7.2 5.2 Consensus Mechanism

#### 1.7.2.1 5.2.1 Proof of Work

* **Algorithm:** SHA-256 hashing
* **Difficulty:** Configurable (default: 4 leading zeros)
* **Target:** Dynamic difficulty adjustment
* **Mining Process:** Nonce increment until target hash found

#### 1.7.2.2 5.2.2 Block Validation

def validate\_block(block: Block, previous\_block: Block) -> bool:  
 # Verify block hash  
 if block.calculate\_hash() != block.block\_hash:  
 return False  
   
 # Verify previous hash link  
 if block.previous\_hash != previous\_block.block\_hash:  
 return False  
   
 # Verify proof of work  
 target = "0" \* block.difficulty  
 if block.block\_hash[:block.difficulty] != target:  
 return False  
   
 return True

### 1.7.3 5.3 Merkle Tree Implementation

* **Purpose:** Efficient transaction verification
* **Algorithm:** SHA-256 hashing of transaction pairs
* **Benefits:** O(log n) verification complexity
* **Implementation:** Recursive tree construction

## 1.8 6. Fraud Detection Algorithm

### 1.8.1 6.1 Risk Assessment Framework

#### 1.8.1.1 6.1.1 Risk Levels

class RiskLevel(Enum):  
 SAFE = "safe" # 0-30% risk  
 LOW\_RISK = "low\_risk" # 30-50% risk  
 MEDIUM\_RISK = "medium\_risk" # 50-70% risk  
 HIGH\_RISK = "high\_risk" # 70%+ risk

#### 1.8.1.2 6.1.2 Risk Factors

1. **Transaction Amount** - High-value transactions
2. **Velocity** - Transaction frequency
3. **Location Mismatch** - Geographic anomalies
4. **Time Patterns** - Unusual transaction times
5. **Merchant History** - New merchant relationships
6. **Payment Method** - Suspicious payment methods

### 1.8.2 6.2 Smart Contract Rules

#### 1.8.2.1 6.2.1 Rule Configuration

smart\_contracts = {  
 'high\_amount\_threshold': 10000.0, # $10,000 threshold  
 'velocity\_threshold': 5, # 5 transactions per hour  
 'location\_mismatch\_penalty': 0.3, # 30% risk increase  
 'night\_transaction\_penalty': 0.2, # 20% risk increase  
 'new\_merchant\_penalty': 0.15, # 15% risk increase  
 'international\_penalty': 0.25 # 25% risk increase  
}

#### 1.8.2.2 6.2.2 Rule Application

def apply\_smart\_contract\_rules(transaction, customer\_history):  
 risk\_adjustment = 0.0  
 risk\_factors = []  
   
 # High amount check  
 if transaction.amount > high\_amount\_threshold:  
 risk\_adjustment += 0.3  
 risk\_factors.append('High Amount')  
   
 # Velocity check  
 recent\_transactions = get\_recent\_transactions(customer\_history, 3600)  
 if len(recent\_transactions) > velocity\_threshold:  
 risk\_adjustment += 0.4  
 risk\_factors.append('High Velocity')  
   
 return risk\_adjustment, risk\_factors

### 1.8.3 6.3 Machine Learning Integration (Future)

* **Algorithm:** Random Forest, Logistic Regression
* **Features:** Transaction patterns, customer behavior
* **Training:** Historical fraud data
* **Deployment:** Real-time prediction API

## 1.9 7. User Interface Design

### 1.9.1 7.1 Dashboard Layout

#### 1.9.1.1 7.1.1 Main Components

1. **Header Section** - Title and status indicators
2. **Control Panel** - Transaction generation and mining controls
3. **Metrics Dashboard** - Key performance indicators
4. **Transaction Feed** - Real-time transaction display
5. **Block Explorer** - Blockchain visualization
6. **Analytics Section** - Risk distribution and statistics

#### 1.9.1.2 7.1.2 Responsive Design

* **Desktop:** Full-featured interface with sidebars
* **Tablet:** Optimized layout with collapsible sections
* **Mobile:** Streamlined interface with touch controls

### 1.9.2 7.2 Interactive Features

#### 1.9.2.1 7.2.1 Real-time Updates

* **Auto-refresh:** 3-second intervals
* **Live Metrics:** Dynamic KPI updates
* **Transaction Feed:** Real-time transaction display
* **Block Mining:** Live mining progress

#### 1.9.2.2 7.2.2 User Controls

* **Transaction Generation** - Create test transactions
* **Block Mining** - Mine pending transactions
* **Smart Contract Rules** - Adjustable thresholds
* **System Reset** - Clear all data

### 1.9.3 7.3 Visualization Components

#### 1.9.3.1 7.3.1 Charts and Graphs

* **Risk Distribution** - Pie chart of risk levels
* **Transaction Volume** - Time-series analysis
* **Block Chain** - Visual blockchain representation
* **Fraud Alerts** - Real-time alert dashboard

## 1.10 8. API Specifications

### 1.10.1 8.1 RESTful API Endpoints

#### 1.10.1.1 8.1.1 Health Check

GET /health  
Response: {"status": "healthy", "timestamp": "2025-07-20T10:00:00Z"}

#### 1.10.1.2 8.1.2 Blockchain Status

GET /api/blockchain/status  
Response: {  
 "chain\_length": 5,  
 "pending\_transactions": 3,  
 "total\_transactions": 25,  
 "last\_block\_hash": "0000abc123..."  
}

#### 1.10.1.3 8.1.3 Transaction Management

POST /api/blockchain/transaction  
Request: {  
 "customer\_id": "CUST1234",  
 "amount": 1500.00,  
 "merchant\_id": "amazon",  
 "location": "New York",  
 "payment\_method": "credit\_card"  
}  
Response: {  
 "transaction\_id": "TX56789",  
 "risk\_score": 0.45,  
 "risk\_level": "low\_risk",  
 "status": "pending"  
}

#### 1.10.1.4 8.1.4 Block Mining

POST /api/blockchain/mine  
Response: {  
 "block\_index": 6,  
 "transactions\_count": 5,  
 "hash": "0000def456...",  
 "mining\_time": 2.3  
}

### 1.10.2 8.2 Data Models

#### 1.10.2.1 8.2.1 Request/Response Formats

* **Content-Type:** application/json
* **Encoding:** UTF-8
* **Date Format:** ISO 8601
* **Number Format:** IEEE 754 double precision

#### 1.10.2.2 8.2.2 Error Handling

{  
 "error": {  
 "code": "VALIDATION\_ERROR",  
 "message": "Invalid transaction data",  
 "details": ["amount must be positive"]  
 }  
}

## 1.11 9. Security Considerations

### 1.11.1 9.1 Data Security

#### 1.11.1.1 9.1.1 Transaction Data

* **Encryption:** HTTPS/TLS for data in transit
* **Validation:** Input sanitization and validation
* **Access Control:** Session-based authentication
* **Audit Trail:** Complete transaction logging

#### 1.11.1.2 9.1.2 Blockchain Security

* **Immutability:** Cryptographic hash verification
* **Integrity:** Merkle tree validation
* **Consensus:** Proof-of-work protection
* **Tamper Detection:** Chain validation algorithms

### 1.11.2 9.2 Application Security

#### 1.11.2.1 9.2.1 Input Validation

* **SQL Injection:** Parameterized queries
* **XSS Protection:** Output encoding
* **CSRF Protection:** Token-based validation
* **Rate Limiting:** Request throttling

#### 1.11.2.2 9.2.2 Authentication & Authorization

* **Session Management:** Secure session handling
* **Access Control:** Role-based permissions
* **Password Security:** Strong password policies
* **Multi-factor Authentication:** Future enhancement

### 1.11.3 9.3 Infrastructure Security

#### 1.11.3.1 9.3.1 Cloud Security

* **Network Security:** VPC and firewall rules
* **Data Encryption:** At-rest encryption
* **Backup Security:** Encrypted backups
* **Monitoring:** Security event logging

## 1.12 10. Deployment Architecture

### 1.12.1 10.1 Cloud Deployment

#### 1.12.1.1 10.1.1 Streamlit Cloud

* **Platform:** Streamlit Cloud
* **Region:** Global deployment
* **Scaling:** Automatic scaling
* **Monitoring:** Built-in analytics

#### 1.12.1.2 10.1.2 Configuration

# .streamlit/config.toml  
[server]  
headless = true  
port = 8501  
enableCORS = true  
enableXsrfProtection = true  
  
[theme]  
primaryColor = "#FF6B6B"  
backgroundColor = "#FFFFFF"  
secondaryBackgroundColor = "#F0F2F6"  
textColor = "#262730"

### 1.12.2 10.2 Container Deployment (Alternative)

#### 1.12.2.1 10.2.1 Docker Configuration

FROM python:3.9-slim  
  
WORKDIR /app  
COPY requirements.txt .  
RUN pip install -r requirements.txt  
  
COPY . .  
EXPOSE 8501  
  
CMD ["streamlit", "run", "ultra\_minimal\_app.py", "--server.port=8501"]

#### 1.12.2.2 10.2.2 Kubernetes Deployment

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: blockchain-fraud-dashboard  
spec:  
 replicas: 3  
 selector:  
 matchLabels:  
 app: blockchain-fraud-dashboard  
 template:  
 metadata:  
 labels:  
 app: blockchain-fraud-dashboard  
 spec:  
 containers:  
 - name: dashboard  
 image: blockchain-fraud-dashboard:latest  
 ports:  
 - containerPort: 8501

### 1.12.3 10.3 CI/CD Pipeline

#### 1.12.3.1 10.3.1 GitHub Actions

name: Deploy to Streamlit Cloud  
on:  
 push:  
 branches: [main]  
jobs:  
 deploy:  
 runs-on: ubuntu-latest  
 steps:  
 - uses: actions/checkout@v2  
 - name: Deploy to Streamlit Cloud  
 run: |  
 # Deployment steps

## 1.13 11. Performance Metrics

### 1.13.1 11.1 System Performance

#### 1.13.1.1 11.1.1 Response Times

* **Page Load:** < 2 seconds
* **Transaction Processing:** < 500ms
* **Block Mining:** < 5 seconds (4 difficulty)
* **API Response:** < 200ms

#### 1.13.1.2 11.1.2 Throughput

* **Concurrent Users:** 100+ users
* **Transactions/Second:** 50+ TPS
* **Block Mining Rate:** 1 block per 5 seconds
* **Data Processing:** 1000+ transactions per minute

### 1.13.2 11.2 Scalability Metrics

#### 1.13.2.1 11.2.1 Horizontal Scaling

* **Auto-scaling:** Based on CPU/memory usage
* **Load Balancing:** Multiple instance support
* **Database Scaling:** Read replicas and sharding
* **Cache Strategy:** Redis for session data

#### 1.13.2.2 11.2.2 Resource Utilization

* **CPU Usage:** < 70% under normal load
* **Memory Usage:** < 80% of allocated memory
* **Network I/O:** Optimized for cloud deployment
* **Storage:** Efficient data compression

## 1.14 12. Testing Strategy

### 1.14.1 12.1 Testing Levels

#### 1.14.1.1 12.1.1 Unit Testing

def test\_transaction\_creation():  
 tx = create\_transaction("CUST123", 100.0, "amazon")  
 assert tx.customer\_id == "CUST123"  
 assert tx.amount == 100.0  
 assert tx.risk\_score >= 0.0  
 assert tx.risk\_score <= 1.0

#### 1.14.1.2 12.1.2 Integration Testing

* **API Testing:** End-to-end API validation
* **Blockchain Testing:** Chain integrity verification
* **UI Testing:** User interface functionality
* **Performance Testing:** Load and stress testing

#### 1.14.1.3 12.1.3 System Testing

* **End-to-End Testing:** Complete workflow validation
* **Security Testing:** Vulnerability assessment
* **Compatibility Testing:** Cross-browser validation
* **Usability Testing:** User experience evaluation

### 1.14.2 12.2 Test Automation

#### 1.14.2.1 12.2.1 Automated Testing

* **Framework:** pytest
* **Coverage:** > 80% code coverage
* **CI/CD Integration:** Automated test execution
* **Reporting:** Detailed test reports

#### 1.14.2.2 12.2.2 Manual Testing

* **User Acceptance Testing:** Stakeholder validation
* **Exploratory Testing:** Ad-hoc testing scenarios
* **Regression Testing:** Feature validation
* **Performance Testing:** Load testing scenarios

## 1.15 13. Maintenance and Support

### 1.15.1 13.1 System Maintenance

#### 1.15.1.1 13.1.1 Regular Maintenance

* **Security Updates:** Monthly security patches
* **Performance Monitoring:** Continuous monitoring
* **Backup Management:** Daily automated backups
* **Log Management:** Log rotation and analysis

#### 1.15.1.2 13.1.2 Preventive Maintenance

* **Health Checks:** Automated system health monitoring
* **Capacity Planning:** Resource usage forecasting
* **Dependency Updates:** Regular package updates
* **Documentation Updates:** Continuous documentation maintenance

### 1.15.2 13.2 Support Procedures

#### 1.15.2.1 13.2.1 Incident Management

* **Issue Tracking:** GitHub Issues integration
* **Escalation Matrix:** Defined support levels
* **Response Times:** SLA-based response commitments
* **Resolution Procedures:** Standardized troubleshooting

#### 1.15.2.2 13.2.2 Change Management

* **Version Control:** Git-based version management
* **Release Planning:** Structured release process
* **Rollback Procedures:** Emergency rollback capabilities
* **Change Documentation:** Comprehensive change logs

## 1.16 14. Future Enhancements

### 1.16.1 14.1 Short-term Enhancements (3-6 months)

#### 1.16.1.1 14.1.1 Feature Additions

* **Machine Learning Integration:** AI-powered fraud detection
* **Real-time Notifications:** Email/SMS alerts
* **Advanced Analytics:** Predictive analytics dashboard
* **Mobile Application:** Native mobile app development

#### 1.16.1.2 14.1.2 Technical Improvements

* **Database Integration:** PostgreSQL/MongoDB integration
* **Caching Layer:** Redis for performance optimization
* **API Rate Limiting:** Enhanced API security
* **Monitoring Dashboard:** Comprehensive system monitoring

### 1.16.2 14.2 Medium-term Enhancements (6-12 months)

#### 1.16.2.1 14.2.1 Advanced Features

* **Multi-blockchain Support:** Ethereum, Hyperledger integration
* **Smart Contract Marketplace:** Deployable fraud detection rules
* **Advanced Visualization:** 3D blockchain visualization
* **API Marketplace:** Third-party integrations

#### 1.16.2.2 14.2.2 Enterprise Features

* **Multi-tenancy:** Multi-organization support
* **Advanced Security:** Zero-trust architecture
* **Compliance Framework:** GDPR, SOX, PCI DSS compliance
* **Audit Trail:** Comprehensive audit logging

### 1.16.3 14.3 Long-term Vision (1-2 years)

#### 1.16.3.1 14.3.1 Platform Evolution

* **Decentralized Architecture:** Distributed ledger technology
* **AI/ML Platform:** Comprehensive ML model marketplace
* **Global Scale:** Multi-region deployment
* **Industry Solutions:** Domain-specific fraud detection

#### 1.16.3.2 14.3.2 Innovation Areas

* **Quantum-resistant Cryptography:** Future-proof security
* **Edge Computing:** IoT device integration
* **Federated Learning:** Privacy-preserving ML
* **Blockchain Interoperability:** Cross-chain transactions

## 1.17 Appendices

### 1.17.1 Appendix A: Code Repository Structure

fraud\_modelling\_project/  
├── ultra\_minimal\_app.py # Basic working version  
├── enhanced\_app.py # Advanced features version  
├── simple\_streamlit\_app.py # Intermediate version  
├── src/  
│ ├── blockchain\_core.py # Core blockchain implementation  
│ ├── blockchain\_api.py # Flask API server  
│ └── blockchain\_dashboard.py # Dashboard components  
├── .streamlit/  
│ └── config.toml # Streamlit configuration  
├── requirements\_empty.txt # Minimal dependencies  
├── DEPLOYMENT\_SUCCESS.md # Deployment guide  
└── TECHNICAL\_SPECIFICATION.md # This document

### 1.17.2 Appendix B: API Response Examples

{  
 "blockchain\_status": {  
 "chain\_length": 5,  
 "pending\_transactions": 3,  
 "total\_transactions": 25,  
 "last\_block\_hash": "0000abc123def456",  
 "difficulty": 4,  
 "mining\_rate": "0.2 blocks/second"  
 }  
}

### 1.17.3 Appendix C: Configuration Parameters

# System Configuration  
SYSTEM\_CONFIG = {  
 "blockchain": {  
 "difficulty": 4,  
 "block\_time": 5,  
 "max\_transactions\_per\_block": 100  
 },  
 "fraud\_detection": {  
 "high\_amount\_threshold": 10000.0,  
 "velocity\_threshold": 5,  
 "location\_mismatch\_penalty": 0.3  
 },  
 "performance": {  
 "auto\_refresh\_interval": 3,  
 "max\_session\_duration": 3600,  
 "cache\_size": 1000  
 }  
}

**Document Control**

| Version | Date | Author | Changes |
| --- | --- | --- | --- |
| 1.0 | 2025-07-20 | Development Team | Initial Technical Specification |

**Approval**

| Role | Name | Date | Signature |
| --- | --- | --- | --- |
| Technical Lead | Development Team | 2025-07-20 | Approved |
| Project Manager | Development Team | 2025-07-20 | Approved |
| Stakeholder | Development Team | 2025-07-20 | Approved |

*This document provides a comprehensive technical specification for the Blockchain Fraud Detection Dashboard system. For questions or clarifications, please contact the development team.*