# SafePayAI - Comprehensive Project Documentation

---

## 1. Executive Summary (Non-Technical)

**SafePayAI** is an intelligent fraud detection and prevention system designed to protect users during digital UPI (Unified Payments Interface) transactions. In an era where digital payments are becoming the norm, the risk of fraudulent activities is increasing. SafePayAI addresses this challenge by using advanced Artificial Intelligence to analyze transactions in real-time and detect potential fraud before it happens.

### What Does It Do?

Imagine you are sending money to someone via a payment app. Before the money leaves your account, SafePayAI acts as a "smart gatekeeper" that analyzes the transaction against a multitude of risk factors:

1. **Is the recipient trustworthy?** It checks the recipient's history—have they been reported for fraud before? Are they on a blacklist?
2. **Is the transaction normal?** It compares the amount and timing of your transaction to your usual behavior. A sudden, large payment at 3 AM might be flagged for review.
3. **Are there any red flags?** It looks for suspicious patterns like using a VPN, a new device, or a location you've never transacted from.

If the transaction is deemed safe, it goes through instantly. If it's flagged as potentially fraudulent, it is **blocked**, protecting you from potential financial loss. An alert is sent to an administrator for review.

### Key Benefits

- **Proactive Protection:** Fraud is stopped *before* money is lost.
- **Real-Time Analysis:** Decisions are made in milliseconds.
- **Intelligent Learning:** The system uses a Machine Learning model trained on thousands of transaction patterns.
- **Multi-Layered Security:** Combines AI predictions with explicit business rules for robust detection.
- **User-Friendly Interface:** Provides a clean, modern dashboard for users and administrators.

---

## 2. Technical Overview

SafePayAI is a full-stack web application with a sophisticated backend Machine Learning pipeline.

### 2.1 Technology Stack

| Layer | Technology |
|---|---|
| **Frontend** | React 18, Vite, Tailwind CSS, Framer Motion |
| **Backend** | Python, Flask, Flask-SocketIO |
| **Database** | SQLite (via SQLAlchemy ORM) |
| **Auth** | Firebase Authentication (Google Sign-In) |
| **ML Model** | Scikit-learn (Random Forest Classifier) |
| **Data Gen** | Pandas, NumPy |

### 2.2 Directory Structure

```
FraudDetectionUsingGAN/
├── AI_model_server_Flask/        # Backend Application
│   ├── app/                      # Flask application module
│   │   ├── __init__.py           # App factory & seeding logic
│   │   ├── api_routes.py         # Core API endpoints & fraud check logic
│   │   ├── auth_middleware.py    # JWT/Firebase auth decorators
│   │   ├── data_service.py       # CSV data loading & querying
│   │   ├── database.py           # SQLAlchemy initialization
│   │   ├── models.py             # ORM models (User, Transaction, etc.)
│   │   ├── routes.py             # Legacy/simple routes
│   │   └── services.py           # ML Model loading & prediction service
│   ├── app.py                    # Main entry point for Flask server
│   ├── generate_synthetic_data.py# Script to generate demo data
│   ├── best_rf_model (1).pkl     # Pre-trained Random Forest model
│   ├── upi_users.csv             # Synthetic user data
│   └── upi_transactions.csv      # Synthetic transaction history
│
├── fraudAI_Frontend_React/       # Frontend Application
│   ├── src/
│   │   ├── App.jsx               # Main routing component
│   │   ├── pages/                # Page-level components
│   │   │   ├── Dashboard.jsx
│   │   │   ├── SendTransaction.jsx
│   │   │   ├── AdminDashboard.jsx
│   │   │   └── ...
│   │   ├── components/           # Reusable UI components
│   │   ├── context/              # React Context (Auth)
│   │   └── lib/                  # Utility functions & Firebase config
│   └── package.json
│
├── README.md
└── SystemDesignDiagrams/
```

---

## 3. System Architecture

The system follows a classic client-server architecture with an embedded Machine Learning service.

```
flowchart TB
    A[👤 User] --> B[🌐 React Frontend]
    B -- "HTTP Request" --> C[⚡ Flask API Server]

    C --> D{🔀 Route Handler}

    D -- "/api/auth/*" --> E[🔐 Auth Middleware]
    E -- "Verify Token" --> F[🔥 Firebase Auth]

    D -- "/api/transactions/*" --> G[💳 Transaction Service]
```

```
    G --> H[🛡️ Fraud Detection Service]
    H --> I[🤖 Random Forest Model]
    I -- "Prediction" --> H

    H -- "Query History" --> J[(📊 CSV Data Service)]
    G -- "Read/Write" --> K[(🗄️ SQLite Database)]

    G -- "If Fraud Detected" --> L[🚨 Create Fraud Alert]
    L --> K

    C -- "Real-time Push" --> M[📡 SocketIO]
    M --> B

    G -- "Result" --> C
    C -- "HTTP Response" --> B
    B --> A
```

### 3.1 Component Descriptions

| Component | Description |
| --- | --- |
| **React Frontend** | Single Page Application (SPA) providing the user interface for dashboards, sending money, and admin review. |
| **Flask Backend** | The main server handling all API requests, business logic, and coordination. |
| **Auth Middleware** | Validates Firebase ID tokens to secure protected API routes. |
| **Transaction Service** | Manages the lifecycle of a transaction: creation, validation, fraud checking, and completion/blocking. |
| **Fraud Detection Service** | A singleton service that wraps the pre-trained ML model for efficient prediction calls. |
| **SocketIO** | Enables real-time, bidirectional communication to push transaction status updates to the client instantly. |
| **SQLite Database** | Stores persistent data for users, transactions, fraud alerts, and user risk profiles. |
| **CSV Data Service** | Loads and queries the large synthetic datasets (`upi_users.csv`, `upi_transactions.csv`) for historical data and demo recipients. |

---

## 4. Hybrid ML Pipeline Flow

The fraud detection mechanism is a **Hybrid System** that combines the power of a pre-trained Machine Learning model with explicit, rule-based checks. This ensures that the system is both intelligent and predictable.

```
flowchart LR
    subgraph "1. Transaction Initiation"
        A[User Sends Money Request] --> B{Validate Input}
```

```
    B -- Invalid --> C[Return Error]
    B -- Valid --> D[Create Transaction Record]
end

subgraph "2. Feature Engineering"
    D --> E["Gather Dynamic Features\n(from DB & CSV)"]
    E --> F["Normalize Features\n(MinMaxScaler logic)"]
end

subgraph "3. Hybrid Fraud Detection"
    F --> G{{"ML Model Prediction\n(Random Forest)"}}
    G -- "Output: Probability (0-1)" --> H{Rule-Based Boosters}

    H -- "Is Blacklisted?" --> I[Force Fraud = True]
    H -- "Trust Score < 15?" --> I
    H -- "Fraud Flags >= 3?" --> I
    H -- "No Rule Triggered" --> J{Probability > 0.3?}

    I --> K[Transaction is FRAUD]
    J -- "Yes" --> K
    J -- "No" --> L[Transaction is SAFE]
end

subgraph "4. Outcome"
    K --> M[Block Transaction]
    M --> N[Create Fraud Alert]
    N --> O[Notify Admin via WebSocket]

    L --> P[Complete Transaction]
    P --> Q[Update Balances]
end

style G fill:#bbf,stroke:#333,stroke-width:2px
style H fill:#fbf,stroke:#333,stroke-width:2px
```

### 4.1 Step-by-Step Breakdown

1. **Transaction Initiation:** The user submits a transaction request from the UI. The backend validates required fields (receiver UPI, amount).
2. **Feature Engineering:**
    - The system gathers **22 features** required by the ML model.
    - Features are sourced dynamically from:
        - **SQL Database:** Receiver's risk profile (trust score, fraud flags, account age).
        - **CSV Data Service:** Historical transaction frequency and time since the last transaction.
    - All numerical features are normalized to a 0-1 scale using pre-defined min/max values from the training data, ensuring the model receives data in the same format it was trained on.
3. **Hybrid Fraud Detection:**

- **ML Model Prediction:** The 22-feature array is passed to the `FraudDetectionService`, which returns a fraud probability (e.g., `0.72`).
- **Rule-Based Boosters:** A set of hard-coded business rules check for critical red flags. If any of these conditions are met, the transaction is immediately flagged as fraud, regardless of the ML score:
  - Recipient is on the **blacklist**.
  - Recipient's **Trust Score** is critically low (< 15).
  - Recipient has multiple **past fraud flags** (>= 3).
  - Recipient has many **fraud complaints** (>= 5).
- **Threshold Check:** If no rules force fraud, the ML probability is checked against a threshold of **0.3** (30%). This threshold is intentionally low to catch more potential threats.

4. **Outcome:**
   - **If Fraud:** The transaction status is set to `BLOCKED`. A `FraudAlert` record is created for admin review. The user is notified.
   - **If Safe:** The transaction status is set to `COMPLETED`. The sender's daily spent limit is updated, and the receiver's balance is credited.

---

## 5. The 22-Feature Model

The Random Forest model is trained on 22 specific features that capture the risk profile of a transaction.

| # | Feature Name | Type | Description |
|---|---|---|---|
| 1 | Transaction Amount | Normalized | The value of the transaction, scaled. |
| 2 | Transaction Frequency | Normalized | Number of transactions involving the recipient in the last 24 hours. |
| 3 | Recipient Blacklist Status | Binary | 1 if recipient is on a blacklist, 0 otherwise. |
| 4 | Device Fingerprinting | Binary | 1 if the device is unknown/mismatched, 0 otherwise. |
| 5 | VPN or Proxy Usage | Binary | 1 if a VPN/proxy is detected, 0 otherwise. |
| 6 | Behavioral Biometrics | Normalized | Anomaly score from user interaction patterns. |
| 7 | Time Since Last Transaction with Recipient | Normalized | Hours since the last transaction with this recipient. |
| 8 | Social Trust Score | Normalized | The recipient's overall trust score (0-100). |
| 9 | Account Age | Normalized | The age of the recipient's account in years. |
| 10 | High-Risk Transaction Times | Binary | 1 if the transaction is at a high-risk hour (11 PM - 5 AM). |
| 11 | Past Fraudulent Behavior Flags | Binary | 1 if the recipient has any past fraud flags. |
| 12 | Location-Inconsistent Transactions | Binary | 1 if the user's location is geographically inconsistent. |

| | | | |
|---|---|---|---|
| 13 | Normalized Transaction Amount | Normalized | A secondary normalization of the amount (amount/5000). |
| 14 | Transaction Context Anomalies | Normalized | Anomaly score based on transaction context. |
| 15 | Fraud Complaints Count | Normalized | Number of fraud complaints filed against the recipient. |
| 16 | Merchant Category Mismatch | Binary | 1 if the transaction category mismatches the merchant's profile. |
| 17 | User Daily Limit Exceeded | Binary | 1 if the transaction exceeds daily limits. |
| 18 | Recent High-Value Transaction Flags | Binary | 1 if a recent high-value transaction was made. |
| 19 | Recipient Verification: Suspicious | One-Hot | 1 if recipient status is 'suspicious'. |
| 20 | Recipient Verification: Verified | One-Hot | 1 if recipient status is 'verified'. |
| 21 | Geo-Location Flags: Normal | One-Hot | 1 if geo-location is 'normal'. |
| 22 | Geo-Location Flags: Unusual | One-Hot | 1 if geo-location is 'unusual'. |

---

## 6. Data Models (Database Schema)

The application uses SQLAlchemy to define the following database tables.

```
erDiagram
    USER ||--o{ TRANSACTION : "sends"
    USER ||--o{ TRANSACTION : "receives"
    USER ||--o| USER_RISK_PROFILE : "has"
    TRANSACTION ||--o{ FRAUD_ALERT : "triggers"
    USER ||--o{ FRAUD_ALERT : "reviews"

    USER {
        int id PK
        string upi_id UK
        string display_name
        string email UK
        string firebase_uid UK
        enum verification_status
        bool is_active
        bool is_admin
        decimal account_balance
        decimal daily_spent
        datetime created_at
    }
```

```
USER_RISK_PROFILE {
    int user_id PK, FK
    float trust_score
    int fraud_flags
    int fraud_complaints_received
    bool blacklist_status
    string geo_location_flag
    int total_transactions
}

TRANSACTION {
    int id PK
    string transaction_ref UK
    int sender_id FK
    int receiver_id FK
    decimal amount
    string description
    enum status
    float fraud_score
    bool is_fraud
    json risk_factors
    datetime created_at
}

FRAUD_ALERT {
    int id PK
    int transaction_id FK
    string alert_type
    enum severity
    string description
    bool reviewed
    int reviewed_by FK
    datetime created_at
}
```

---

## 7. API Endpoints

The backend exposes a RESTful API under the `/api` prefix.

### 7.1 Authentication

| Method | Endpoint | Auth | Description |
|---|---|---|---|
| POST | `/api/auth/register` | No | Register a new user and generate a UPI ID. |
| GET | `/api/auth/me` | Yes | Get the current authenticated user's info. |

### 7.2 User & Recipient

| Method | Endpoint | Auth | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| GET | `/api/users/<upi_id>` | Optional | Lookup user by UPI ID for recipient verification. |
| GET | `/api/users/balance` | Yes | Get current user's balance and limits. |
| GET | `/api/recipients/demo` | No | Get demo recipients for exhibition purposes. |

### 7.3 Transactions

| Method | Endpoint | Auth | Description |
|---|---|---|---|
| POST | `/api/transactions/send` | Yes | Process a new money transfer with fraud check. |
| GET | `/api/transactions/history` | Yes | Get paginated transaction history. |
| GET | `/api/transactions/<ref>` | Yes | Get details of a single transaction. |

### 7.4 Admin

| Method | Endpoint | Auth | Description |
|---|---|---|---|
| GET | `/api/admin/stats` | Admin | Get system-wide statistics. |
| GET | `/api/admin/alerts` | Admin | Get pending fraud alerts for review. |
| PUT | `/api/admin/alerts/<id>/review` | Admin | Mark a fraud alert as reviewed. |
| POST | `/api/admin/users/<id>/suspend` | Admin | Suspend a user account. |

---

## 8. Frontend Structure

The React frontend is a modern SPA built with Vite for fast development.

### 8.1 Key Pages

| Page | Route | Description |
|---|---|---|
| `Dashboard.jsx` | `/dashboard` | Main user dashboard showing balance, recent transactions, and quick actions. |
| `SendTransaction.jsx` | `/send` | The primary page for initiating a money transfer. |
| `TransactionHistory.jsx` | `/transactions` | Full, paginated history of all user transactions. |
| `AdminDashboard.jsx` | `/admin` | Admin panel for viewing stats and reviewing fraud alerts. |
| `LiveDemo.jsx` | `/demo` | A special exhibition mode for showcasing the app. |

### 8.2 Key Libraries

- **Tailwind CSS:** Utility-first CSS framework for rapid styling.
- **Framer Motion:** Provides smooth, animated transitions between UI states.
- **Radix UI:** Accessible, unstyled component primitives (Dialog, Avatar, etc.).
- **Socket.IO Client:** Listens for real-time transaction status updates from the server.

---

## 9. How to Run

### 9.1 Backend Setup

```
# Navigate to backend directory
cd FraudDetectionUsingGAN/AI_model_server_Flask

# Create and activate virtual environment
python -m venv venv
source venv/bin/activate  # On Windows: .\venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt

# Run the server
python app.py
```

The API will be available at `http://127.0.0.1:5001/`.

### 9.2 Frontend Setup

```
# Navigate to frontend directory
cd FraudDetectionUsingGAN/fraudAI_Frontend_React

# Install dependencies
npm install

# Run development server
npm run dev
```

The app will be available at `http://localhost:5173/`.

---

## 10. Appendix: Synthetic Data Generation

The `generate_synthetic_data.py` script is used to populate the system with realistic demo data.

### Process

1. **Load Trained Model:** The script loads the production Random Forest model (`best_rf_model.pkl`).
2. **Generate User Profiles:** It creates 100+ user profiles with varying risk categories (`safe`, `medium`, `high`). High-risk users are given low trust scores, past fraud flags, and suspicious verification statuses.
3. **Generate Transactions:** For each of 10,000 transactions:
    - A sender and receiver are randomly selected.
    - Transaction features are generated based on the receiver's risk profile.
    - The trained ML model is used to predict the label (`0` for safe, `1` for fraud).
    - All features and the label are saved to a CSV file.

4. **Output:** The script produces `upi_users.csv` and `upi_transactions.csv`, which are loaded by the `DataService` at runtime.

This ensures the demo data is consistent with the model's behavior.

---

**Document Version:** 1.0 **Generated On:** 2025-12-27