

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

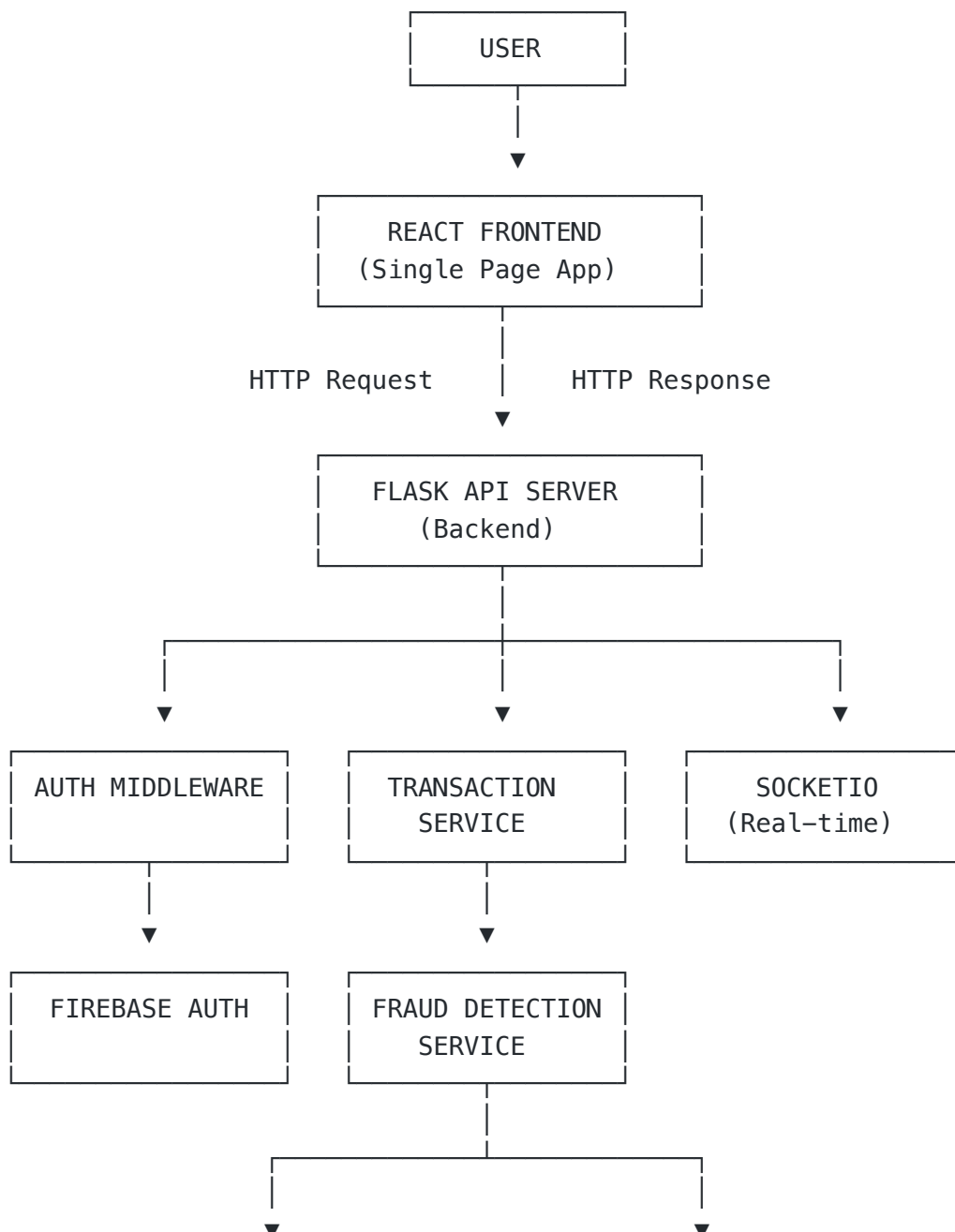
```
FraudDetectionUsingGAN/
├── AI_model_server_Flask/                                # Backend Application
│   ├── app/                                              # Flask application module
│   │   ├── __init__.py                                  # App factory & seeding logic
│   │   ├── api_routes.py                               # 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
```

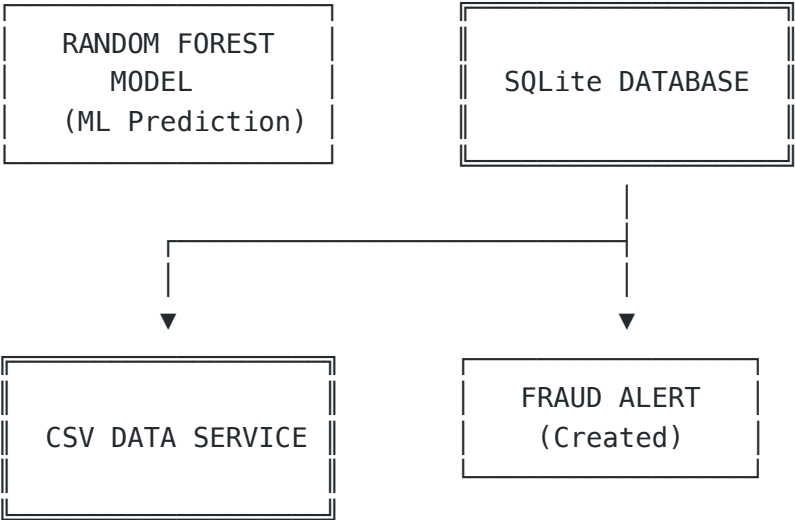
```
|   └─ package.json
|
|   └─ README.md
|   └─ SystemDesignDiagrams/
```

3. System Architecture

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

SAFEPAYAI SYSTEM ARCHITECTURE





LEGEND :



3.1 Component Descriptions

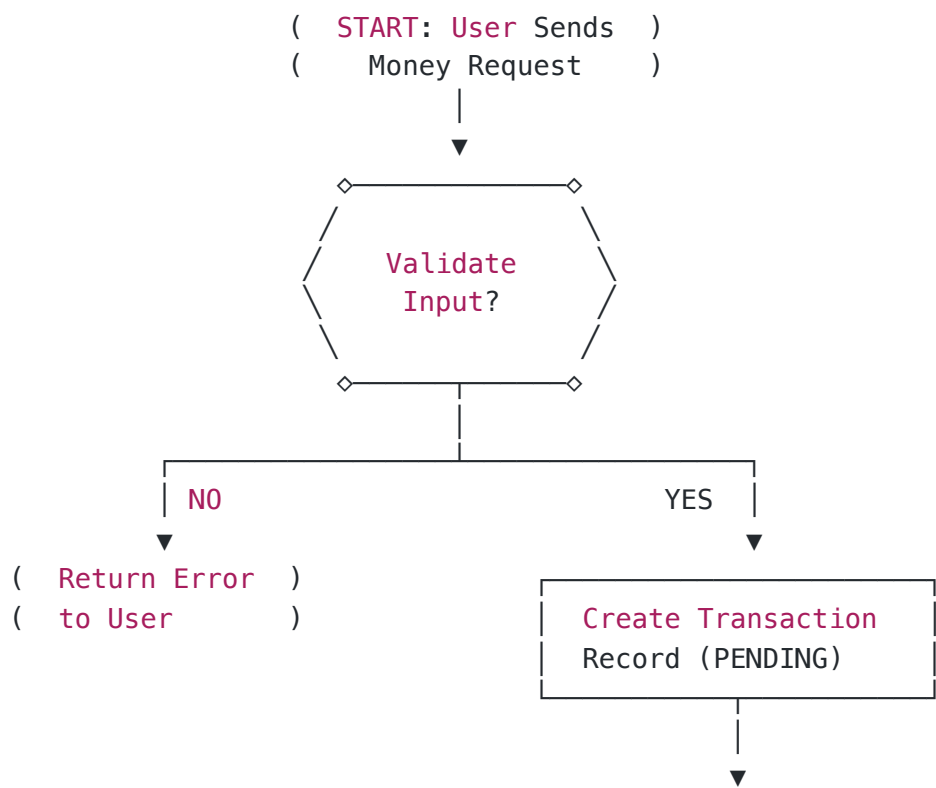
Component	Description
React Frontend	Single Page Application (SPA) providing the user interface for dashboards, sending money, and admin review.
Flask Backend	The main server handling all API requests, business logic, and coordination.
Auth Middleware	Validates Firebase ID tokens to secure protected 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 (<code>upi_users.csv</code> , <code>upi_transactions.csv</code>) for historical data and demo recipients.

4. Hybrid ML Pipeline Flow

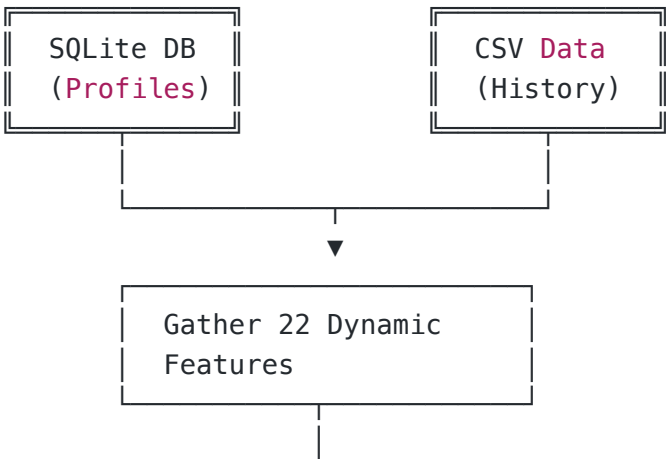
The fraud detection mechanism is a **Hybrid System** that combines the power of a pre-trained Machine Learning model with explicit, rule-based checks. This ensures that the system is both intelligent and predictable.

HYBRID ML PIPELINE FLOWCHART

STAGE 1: TRANSACTION INITIATION



STAGE 2: FEATURE ENGINEERING



Normalize Features
(MinMaxScaler 0-1)

STAGE 3: HYBRID FRAUD DETECTION

ML **MODEL PREDICTION**
(Random Forest)

Input: 22 Features
Output: Probability

RULE-BASED BOOSTERS

Is Blacklisted?
→ FRAUD

Trust Score < 15
→ FRAUD

Fraud Flags >= 3
→ FRAUD

Rule Triggered?

YES

NO

Force: is_fraud
= **TRUE**

Probability > 0.3
(Threshold)

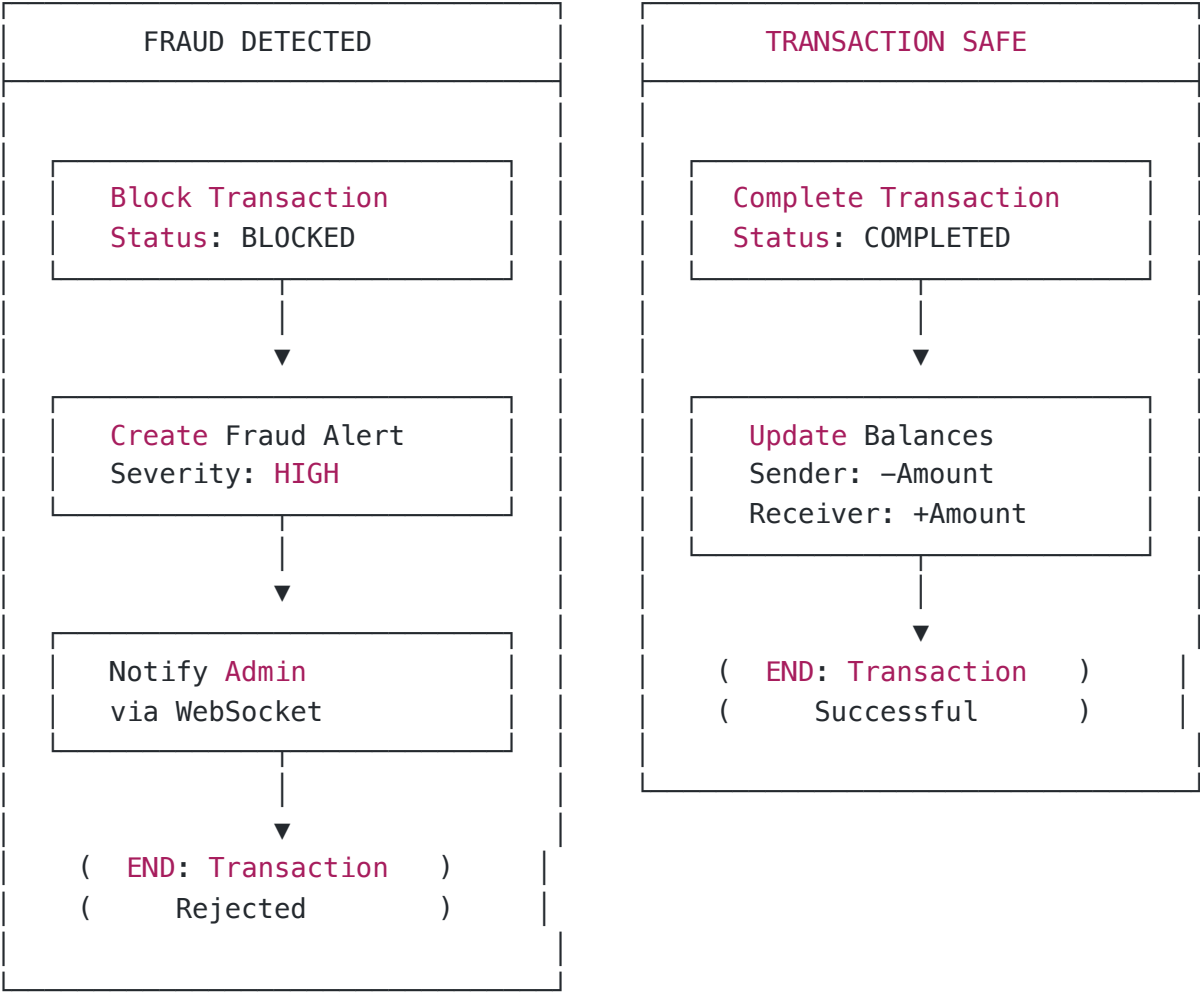
YES

NO

is_fraud = **TRUE**

is_fraud = **FALSE**

STAGE 4: OUTCOME



LEGEND:

Process / Action

Database / Data Store

Decision (Yes/No)

(Terminal)

Start / End Point

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.

#	Feature Name	Type	Description
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'.

#	Feature Name	Type	Description
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.

DATABASE SCHEMA (ER DIAGRAM)

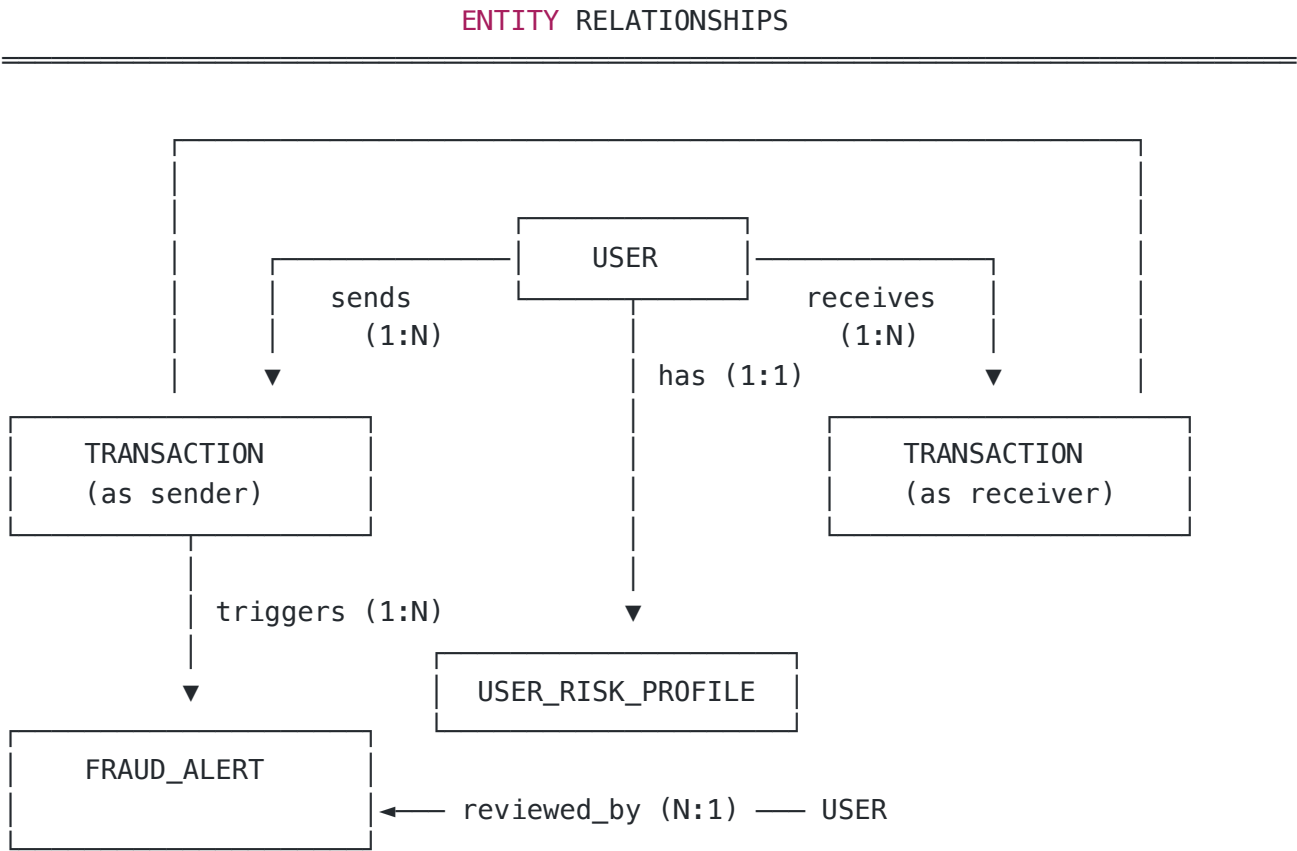


TABLE DEFINITIONS

USER		
Column	Type	Constraints
id	INTEGER	PRIMARY KEY
upi_id	VARCHAR(50)	UNIQUE, NOT NULL
display_name	VARCHAR(100)	NOT NULL
email	VARCHAR(120)	UNIQUE, NOT NULL
firebase_uid	VARCHAR(128)	UNIQUE
verification_status	ENUM	'pending'/'verified'/'suspended'
is_active	BOOLEAN	DEFAULT TRUE
is_admin	BOOLEAN	DEFAULT FALSE
account_balance	DECIMAL(12,2)	DEFAULT 0.00
daily_spent	DECIMAL(12,2)	DEFAULT 0.00
created_at	DATETIME	DEFAULT NOW()

USER_RISK_PROFILE		
Column	Type	Constraints
user_id	INTEGER	PRIMARY KEY, FOREIGN KEY(USER)
trust_score	FLOAT	DEFAULT 75.0 (0-100 range)
fraud_flags	INTEGER	DEFAULT 0
fraud_complaints_received	INTEGER	DEFAULT 0
blacklist_status	BOOLEAN	DEFAULT FALSE
geo_location_flag	VARCHAR(20)	'normal'/'unusual'/'suspicious'
total_transactions	INTEGER	DEFAULT 0

TRANSACTION		
Column	Type	Constraints
id	INTEGER	PRIMARY KEY
transaction_ref	VARCHAR(36)	UNIQUE, NOT NULL (UUID)
sender_id	INTEGER	FOREIGN KEY(USER), NOT NULL
receiver_id	INTEGER	FOREIGN KEY(USER), NOT NULL
amount	DECIMAL(12,2)	NOT NULL
description	VARCHAR(255)	NULLABLE
status	ENUM	'pending'/'completed'/'blocked'
fraud_score	FLOAT	NULLABLE (0.0 - 1.0)
is_fraud	BOOLEAN	DEFAULT FALSE
risk_factors	JSON	NULLABLE
created_at	DATETIME	DEFAULT NOW()

FRAUD_ALERT		
Column	Type	Constraints

id	INTEGER	PRIMARY KEY
transaction_id	INTEGER	FOREIGN KEY(TRANSACTION)
alert_type	VARCHAR(50)	NOT NULL
severity	ENUM	'low'/'medium'/'high'/'critical'
description	TEXT	NOT NULL
reviewed	BOOLEAN	DEFAULT FALSE
reviewed_by	INTEGER	FOREIGN KEY(USER), NULLABLE
created_at	DATETIME	DEFAULT NOW()

7. API Endpoints

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

7.1 Authentication

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

7.2 User & Recipient

Method	Endpoint	Auth	Description
GET	<code>/api/users/<upi_id></code>	Optional	Lookup user by UPI ID for recipient verification.
GET	<code>/api/users/balance</code>	Yes	Get current user's balance and limits.
GET	<code>/api/recipients/demo</code>	No	Get demo recipients for exhibition purposes.

7.3 Transactions

Method	Endpoint	Auth	Description
POST	<code>/api/transactions/send</code>	Yes	Process a new money transfer with fraud check.
GET	<code>/api/transactions/history</code>	Yes	Get paginated transaction history.
GET	<code>/api/transactions/<ref></code>	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