**Money Scam Detector Project Documentation**

**Project Overview**

The **Money Scam Detector** is a rule-based fraud detection system that identifies potential fraudulent activities and scams in online financial transactions and communications. The system is built using the **Flask** framework for the backend, **SQL** for data storage, and provides real-time alerts for suspicious activities, such as transactions or emails that exhibit scam-like patterns.

This project does not rely on machine learning, instead it uses a set of predefined rules to detect scams, making it simpler and more deterministic while still being effective in identifying fraudulent behavior.

**Table of Contents**

1. **Introduction**
2. **System Architecture**
3. **Technologies Used**
4. **Features**
5. **Backend Design**
6. **Frontend Design**
7. **Database Schema**
8. **API Endpoints**
9. **Rule-based Scam Detection**
10. **Admin Dashboard**
11. **Conclusion**

**1. Introduction**

The **Money Scam Detector** is designed to help users identify and prevent online financial scams by analyzing transactions and emails for patterns that match common scam indicators. The system provides alerts and notifications when a suspicious activity is detected. This system is particularly useful for individuals and small businesses to protect themselves from potential online fraud.

**2. System Architecture**

The system is built using a **client-server architecture** where:

* The **Flask** backend serves as the application server, handling requests from users, detecting fraud patterns, and managing real-time notifications.
* The **SQL database** (PostgreSQL/MySQL) stores all transaction and email data, as well as user details.
* The **Frontend** provides a user-friendly dashboard for users and admins to interact with the system and view flagged transactions, emails, and alerts.
* **Real-time notifications** are sent via **Flask-SocketIO** for transactions and emails flagged as suspicious.

**3. Technologies Used**

* **Backend**: Flask (Python web framework)
* **Frontend**: HTML, CSS, JavaScript (React.js or Vue.js)
* **Database**: SQL (PostgreSQL or MySQL)
* **Real-Time Notifications**: Flask-SocketIO
* **Email Notifications**: SendGrid or Flask-Mail
* **SMS Notifications**: Twilio (optional)

**4. Features**

* **User Registration & Authentication**: Secure registration, login, and session management using JWT or Flask-Login.
* **Transaction Monitoring**: Flag suspicious transactions based on predefined rules (e.g., unusually large amounts, frequent transactions).
* **Email Content Analysis**: Detect scam-related keywords and URLs in email content.
* **Rule-Based Scam Detection**: Use a set of predefined rules for detecting fraud patterns in transactions and emails.
* **Real-Time Alerts**: Notify users about suspicious activities via in-app alerts, emails, and SMS.
* **Admin Dashboard**: Admins can view flagged transactions, manage users, and review reports.
* **Audit Logs**: Track all user actions for auditing and transparency.
* **Reporting**: Generate reports summarizing detected fraud, flagged transactions, and resolution statuses.

**5. Backend Design**

The backend is designed to handle:

* **User Authentication**: Secure authentication using Flask-Login or Flask-Security. Passwords are hashed using Flask-Bcrypt.
* **Transaction Analysis**: Transaction data is validated against predefined rules for fraudulent activities.
* **Email Analysis**: Email content is parsed and checked for scam-related keywords or URLs.
* **Alert System**: Real-time alerts are sent via WebSockets (Flask-SocketIO) when suspicious activity is detected.
* **Database Interaction**: SQLAlchemy is used to interact with the database, which stores all user, transaction, email, and alert data.

**6. Frontend Design**

The frontend provides a clean and simple interface:

* **User Dashboard**: Displays the user's transaction history, flagged transactions, and their statuses.
* **Admin Dashboard**: Allows admins to view flagged transactions, review and resolve them, and monitor the performance of the scam detection system.
* **Notifications**: Real-time alerts are shown on the frontend as soon as an activity is flagged.
* **Reports**: Users and admins can generate reports summarizing detected scams and fraud patterns.

**7. Database Schema**

Below is the SQL schema for the project:

1. **Users Table**:
   * user\_id (Primary Key)
   * email (Unique)
   * password\_hash
   * created\_at
   * location (optional)
2. **Transactions Table**:
   * transaction\_id (Primary Key)
   * user\_id (Foreign Key)
   * amount
   * timestamp
   * location
   * status (e.g., "normal", "flagged")
   * scam\_score (A score based on predefined rule checks)
3. **Emails Table**:
   * email\_id (Primary Key)
   * user\_id (Foreign Key)
   * subject
   * content
   * status (e.g., "normal", "flagged")
4. **Alerts Table**:
   * alert\_id (Primary Key)
   * user\_id (Foreign Key)
   * transaction\_id (Foreign Key)
   * alert\_type (e.g., "email", "transaction")
   * status (e.g., "resolved", "unresolved")

**8. API Endpoints**

**User Authentication Endpoints**

* POST /register: Registers a new user.
* POST /login: Authenticates a user and returns a JWT.
* GET /profile: Retrieves user profile information (requires authentication).

**Transaction Endpoints**

* POST /transactions: Submit a new transaction.
* GET /transactions: Retrieve the user's transaction history.
* POST /check\_transaction: Analyzes a transaction for potential fraud.
* GET /alerts: Retrieve a list of alerts for the user.

**Email Endpoints**

* POST /emails: Submit email content for scam analysis.
* GET /emails: Retrieve a list of emails for the user.
* POST /check\_email: Analyze email content for scam-related keywords or links.

**Admin Endpoints**

* GET /admin/transactions: Retrieve all flagged transactions.
* POST /admin/resolve\_transaction: Resolve a flagged transaction (mark it as legitimate or fraudulent).
* GET /admin/reports: Generate a report of fraud detection performance.

**9. Rule-Based Scam Detection**

The fraud detection system uses several rules to flag suspicious activities:

**Transaction Rules:**

1. **Large Transactions**: Flag transactions that are larger than three times the average transaction amount of the user.
2. **Frequent Transactions**: Flag multiple transactions made in a short period (e.g., more than five transactions in one hour).
3. **Unusual Locations**: Flag transactions from IP addresses or locations that differ from the user's typical behavior.
4. **Frequent Transfers to New Accounts**: Flag transactions to new or unusual account numbers.
5. **Round Amounts**: Flag transactions with round numbers (e.g., $1000, $5000), which are sometimes used in scams to obscure fraudulent activities.

**Email Analysis Rules:**

1. **Suspicious Keywords**: Flag emails containing common scam keywords (e.g., “urgent,” “confirm account,” “free money”).
2. **Suspicious Links**: Flag emails that contain URLs known to be linked to phishing or scam websites (using a URL blacklist).
3. **Phishing Phrases**: Flag emails that contain phrases commonly used in phishing attempts, such as “click here” or “verify your account.”

**10. Admin Dashboard**

The admin dashboard allows administrators to:

* View and review flagged transactions and emails.
* Mark transactions as legitimate or fraudulent.
* Generate fraud detection performance reports (e.g., false positives, false negatives).
* Modify rules to improve fraud detection as new scam patterns are identified.

**11. Conclusion**

The **Money Scam Detector** system is a simple yet effective fraud detection solution built with Flask and SQL. By using rule-based heuristics and predefined patterns, it can reliably detect common scams and flag suspicious transactions and emails. This system is flexible and can be easily expanded by adding more rules or integrating with external services for additional verification.

Although this project does not use machine learning, its approach remains effective for many types of fraud and is easy to maintain. As new scam patterns emerge, the rule set can be updated to keep the detection system current.