# Money Muling Detection - Financial Forensics Engine

## Project Title

**Money Muling Detection: Graph-Based Financial Crime Forensics Engine**

## Live Demo URL

Github(deploy)- https://ankurshuh.github.io/bitCrew/

Github Repo link- https://github.com/Ankurshuh/bitCrew

## Tech Stack

- **Frontend:** Streamlit (Python)
- **Backend:** Python 3.10+
- **Graph Processing:** NetworkX
- **Visualization:** PyVis, Streamlit Tables
- **Deployment:** Streamlit Cloud / Render
- **Data Processing:** Pandas

## System Architecture

```
CSV Upload (Frontend)
       |
       v
   Backend Processing (Python)
       |
       |-- CSV Validation & Preprocessing (validator.py + preprocessing.py)
       |-- Graph Construction (graph_builder.py)
       |-- Pattern Detection (detectors/cycles.py, smurfing.py, shell.py)
       |-- Suspicion Scoring (scoring/suspicion_score.py)
       |-- False Positive Filtering (filters/false_positive.py)
       |-- JSON Formatter (output/json_formatter.py)
       v
Frontend Output:
  - Interactive Graph
  - Fraud Ring Summary Table
  - Downloadable JSON
```

## Algorithm Approach

### 1. Cycle Detection (Circular Fund Routing)

- **Algorithm:** Johnson’s Algorithm (via NetworkX `simple_cycles`)
- **Purpose:** Detect money loops of length 3–5 to find circular fund routing.
- **Complexity:** O((V + E)(C + 1)), where C = number of cycles

### 2. Smurfing Patterns (Fan-in / Fan-out)

- **Algorithm:** Temporal Sliding Window Analysis
- **Purpose:** Detect rapid aggregation or dispersion of funds within 72 hours.
- **Complexity:** O(N) per aggregation, N = number of transactions

### 3. Layered Shell Networks

- **Algorithm:** Bounded DFS / All Simple Paths (NetworkX)
- **Purpose:** Detect 3+ hop chains through low-activity accounts.
- **Complexity:** Exponential in path length, bounded by cutoff = 5

### 4. Suspicion Scoring

- **Algorithm:** Weighted Heuristic Scoring + Normalization
- **Purpose:** Rank accounts 0–100 based on involvement in patterns.
- **Complexity:** O(M), M = number of suspicious accounts

### 5. False Positive Filtering

- **Algorithm:** Frequency-based Outlier Filtering
- **Purpose:** Avoid flagging legitimate high-volume merchants or payroll accounts.
- **Complexity:** O(N), N = number of accounts

## Suspicion Score Methodology

- **Cycles:** 40 points per cycle involvement
- **Smurfing (fan-in/fan-out):** 25 points
- **Shell Networks:** 30 points
- **Normalization:** Weighted sum divided by maximum raw score, scaled to 0–100
- **Optional Upgrade:** Pattern confidence scoring and ring explanation for forensic transparency

## Installation & Setup

1. Clone the repository:

```
git clone <your-repo-url>
cd money_muling_backend
```

2. Create virtual environment:

```
python -m venv venv
source venv/bin/activate    # Linux/Mac
venv\Scripts\activate       # Windows
```

3. Install dependencies:

```
pip install -r requirements.txt
```

4. Run the backend test:

```
python main.py <sample_csv.csv>
```

5. Deploy Streamlit frontend:

```
streamlit run frontend/app.py
```

## Usage Instructions

1. Open the live URL.
2. Upload a CSV with the exact required columns.
3. Click **Analyze**.
4. View interactive graph highlighting suspicious accounts.
5. Check the Fraud Ring Summary Table.
6. Download JSON file with suspicious accounts and fraud rings.

## Known Limitations

- Shell network detection exponential in paths; cutoff limits enforced.
- Thresholds (smurfing/fan-out) are dataset-dependent; adaptive thresholds recommended.
- High transaction datasets >10K may need increased resources for <30s performance.
- Does not detect completely new/unseen fraud patterns outside cycles, smurfing, shell rules.

## Team Members

- **Astha Jain** – B.Tech AIML, 6th Semester, MITS
- **Ronak Chaurasiya** – B.Tech AIML,6th Semester, MITS
- **Ayush Batham** – B.Tech AIML, 6th Semester, GIIT
- **Ankur Shukla** – Btech AIML, 6th Semester, GIIT

---

*Prepared for RIFT 2026 Hackathon, Graph Theory / Financial Crime Track*