MVJ COLLEGE OF ENGINEERING
Since 1982
An Autonomous Institute
(Affiliated to Visvesvaraya Technological University, Belagavi
Approved By AICTE, New Delhi,
Recognized by UGC under 2(f) & 12(B)
Accredited by NBA and NAAC)

A MINI PROJECT REPORT
ON

CREDIT CARD FRAUD DETECTION SYSTEM

Submitted in partial fulfillment of requirements for the award of 6th Sem degree,

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING

Submitted By: SANTHOSH RAAJ G
1MJ19CS145

Under the Guidance of
Ms. THEJASHWINI.M
Assistant Professor, Department of Computer Science & Engineering,

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
MVJ COLLEGE OF ENGINEERING
BANGALORE-67
ACADEMIC YEAR 2021-22

MVJ COLLEGE OF ENGINEERING

Near ITPB, Whitefield, Bangalore-67

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COLLEGE OF ENGINEERING
Since 1982
An Autonomous Institute
(Affiliated to Visvesvaraya Technological University, Belagavi
Approved By AICTE, New Delhi,
Recognized by UGC under 2(f) & 12(B)
Accredited by NBA and NAAC)

CERTIFICATE

This is to certify that the project work, entitled “**CREDIT CARD FRAUD DETECTION SYSTEM”** is a bonafide work carried out by SANTHOSH RAAG G (1MJ19CS145) inpartial fulfillment for the award of degree of Bachelor of Engineering in Computer Science& Engineering during the academic year 2021-22. It is certified that all the corrections/suggestions indicated for Internal Assessment have been incorporated in the Report. The project report has been approved as it satisfies the academic requirements.

Signature of the Guide	**Signature of the HOD**
Ms. Thejashwini.M	**Mrs. Tamilarasi R**

Name of examiners: Signature with date:

1.
2.

MVJ COLLEGE OF ENGINEERING
Whitefield, Near ITPB, Bangalore-67

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MVJ COLLEGE OF ENGINEERING Since 1982
An Autonomous Institute
(Affiliated to Visvesvaraya Technological University, Belagavi
Approved By AICTE, New Delhi,
Recognized by UGC under 2(f) & 12(B)
Accredited by NBA and NAAC)

DECLARATION

I, **SANTHOSH RAAJ G** hereby declare that the entire work titled “**CREDIT CARD FRAUD DETECTION”** embodied in this project report has been carried out by us during the 6th semester of BE degree at MVJCE, Bangalore under the esteemed guidance of **Ms. Thejashwini.M**, Assistant Prof, Dept. of CSE, MVJCE. The work embodied in this dissertation work is original and it has not been submitted in part of full for any other degree in any University.

SANTHOSH
RAAJ G
1MJ19CS145

Place:

Date:

ABSTRACT

Credit card fraud is an inclusive term for fraud committed using a payment card, such as a credit card or debit card.[1] The purpose may be to obtain goods or services or to make payment to another account, which is controlled by a criminal. The Payment Card Industry Data Security Standard (PCI DSS) is the data security standard created to help financial institutions process card payments securely and reduce card fraud

Credit card fraud can be authorised, where the genuine customer themselves processes payment to another account which is controlled by a criminal, or unauthorised, where the account holder does not provide authorisation for the payment to proceed and the transaction is carried out by a third party. In 2018, unauthorised financial fraud losses across payment cards and remote banking totalled £844.8 million in the United Kingdom. Whereas banks and card companies prevented £1.66 billion in unauthorised fraud in 2018. That is the equivalent to £2 in every £3 of attempted fraud being stopped

The challenge is to recognize fraudulent credit card transactions so that the customers of credit card companies are not charged for items that they did not purchase.

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany a successful completion of any task would be incomplete without the mention of people who made it possible. So, with gratitudewe acknowledge all those whose guidance and encouragement served as beacon of light and crowned our effort with success.

We are thankful to the **Management of MVJ College of Engineering Bangalore** fortheir continuous support and encouragement for carrying out the project work

We are thankful to our **Dr. Mahabaleswarappa,** Principal, MVJCE, Bengaluru for being a constant inspiration and providing all the facilities that was needed throughout the project work.

We like to express our gratitude to our **Dr. M. Brindha**, Vice Principal, MVJCE, Bengaluru, for constant encouragement throughout the course.

We also like to express our sincere gratitude to our **Dr. M A Lourdu Antony Raj**, Registrar and Controller of Examinations, MVJCE, Bengaluru, for persistent guidance.

We are thankful to our **Mrs. Tamilarasi R**, HOD of Computer Science Engineering, MVJCE, Bengaluru, for being a constant support and providing all the facilities that was needed throughout the project work.

We consider it as a privilege and honor to express our sincere gratitude to our guide **Ms. Thejashwini.M, Assistant Professor,** Dept. of CSE, MVJCE, for her encouragement that has been a constant source of motivation to us for successful completion of our project.

It’s also a great pleasure to express our deepest gratitude to all the other faculty members of our department for their cooperation and constructive criticism offered, which helped us a lot during our project work.

Finally, we would like to thank all our family members and friends whose encouragement and support was invaluable.Thanking you

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Credit Card Fraud Detection 7 Tamojit Das -tamo.das.97@gmail.com Introduction Credit Card Fraud can be defined as a case where a person uses someone else's credit card for personal reasons while the owner and the card issuing authorities are unaware of the fact that the card is being used. Due to rise and acceleration of E-Commerce, there has been a tremendous use of credit cards for online shopping which led to High amount of frauds related to credit cards. In the era of digitalization, the need to identify credit card frauds is necessary. Fraud detection involves monitoring and analyzing the behavior of various users in order to estimate detect or avoid undesirable behavior. In order to identify credit card fraud detection effectively, we need to understand the various technologies, algorithms and types involved in detecting credit card frauds. Algorithm can differentiate transactions which are fraudulent or not. Find fraud, they need to passed dataset and knowledge of fraudulent transaction. They analyze the dataset and classify all transactions. Fraud detection involves monitoring the activities of populations of users in order to estimate, perceive or avoid objectionable behavior, which consist of fraud, intrusion, and defaulting.

Machine learning algorithms are employed to analyses all the authorized transactions and report the suspicious ones. These reports are investigated by professionals who contact the cardholders to confirm if the transaction was genuine or fraudulent. The investigators provide a feedback to the automated system which is used to train and update the algorithm to eventually improve the fraud-detection performance over time

CHAPTER 2

LITERATURE SURVEY

S P Maniraj [1] In this paper, they describe Random forest algorithm applicable on Find fraud detection. Random forest has two types. They describe in detail and their accuracy 91.96% and 96.77% respectively. This paper summaries second type is better than the first type.   Suman Arora [2] In this paper, many supervised machine learning algorithms apply on 70% training and 30% testing dataset. Random forest, stacking classifier, XGB classifier, SVM, Decision tree and KNN algorithms compare each other i.e. 94.59%, 95.27%, 94.59%, 93.24%, 90.87%, 90.54% and 94.25% respectively. Summaries of this paper, SVM has the highest ranking with 0.5360 FPR, and stacking classifier has the lowest ranking with 0.0335.   Kosemani Temitayo Hafiz [3] In this paper, they describe flow chart of fraud detection process. i.e. data Acquisition, data pre-processing, Exploratory data analysis and methods or algorithms are in detail. Algorithms are K- nearest neighbor (KNN), random tree and Logistic regression accuracy are 96.91%, 94.32%, 57.73% and 98.24% respectively.

CHAPTER 3

PROBLEM ANALYSIS

The challenge is to recognize fraudulent credit card transactions so that the customers of credit card companies are not charged for items that they did not purchase.

Main challenges involved in credit card fraud detection are:

- Enormous Data is processed every day and the model build must be fast enough to respond to the scam in time.
- Imbalanced Data i.e most of the transactions (99.8%) are not fraudulent which makes it really hard for detecting the fraudulent ones
- Data availability as the data is mostly private.
- Misclassified Data can be another major issue, as not every fraudulent transaction is caught and reported.
- Adaptive techniques used against the model by the scammers.

CHAPTER 4

IMPLEMENTATION

Since we will be building a complete web application there is a number of tools that you will need to install before getting started:

- Flask: a very powerful web framework that provides you with tools, libraries and technologies used in web development. A Flask application can be as small as a single web page or as complex as a management interface.
- Docker and Compose: an open platform for developing, shipping, and running applications. It enables you to separate your application from your infrastructure (host machine). If you are installing Docker on Windows, Compose will be already included. For Linux and macOS visit this site.
- Memgraph DB: a native fully distributed in-memory graph database built to handle real-time use-cases at enterprise scale.

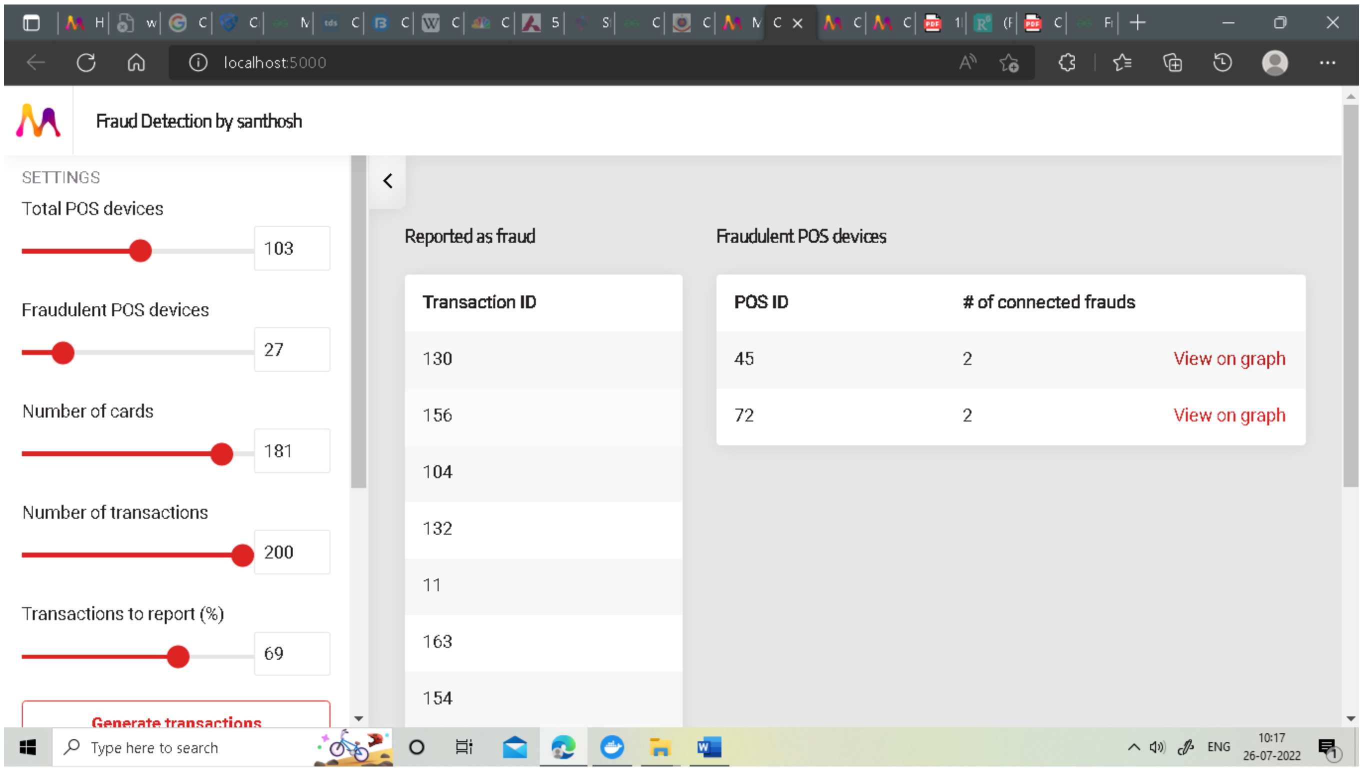
All the roles in this scenario:

- **Card - a credit card used for payment.**
- **POS - a point of sale device that uses a card to execute transactions.**
- **Transaction - a stored instance of buying something.**

Your application will simulate how a POS device gets compromised, then a card in contact with that POS device gets compromised as well and in the end, a fraudulent transaction is reported. Based on these reported transactions, Memgraph is used to search for the root-cause (a.k.a. the compromised POS) of the reported fraudulent transactions and all the cards that have fallen victim to it as shown below.

CHAPTER 5

RESULT



Fraud Detection by santhosh

SETTINGS

Total POS devices: 103

Fraudulent POS devices: 27

Number of cards: 181

Number of transactions: 200

Transactions to report (%): 69

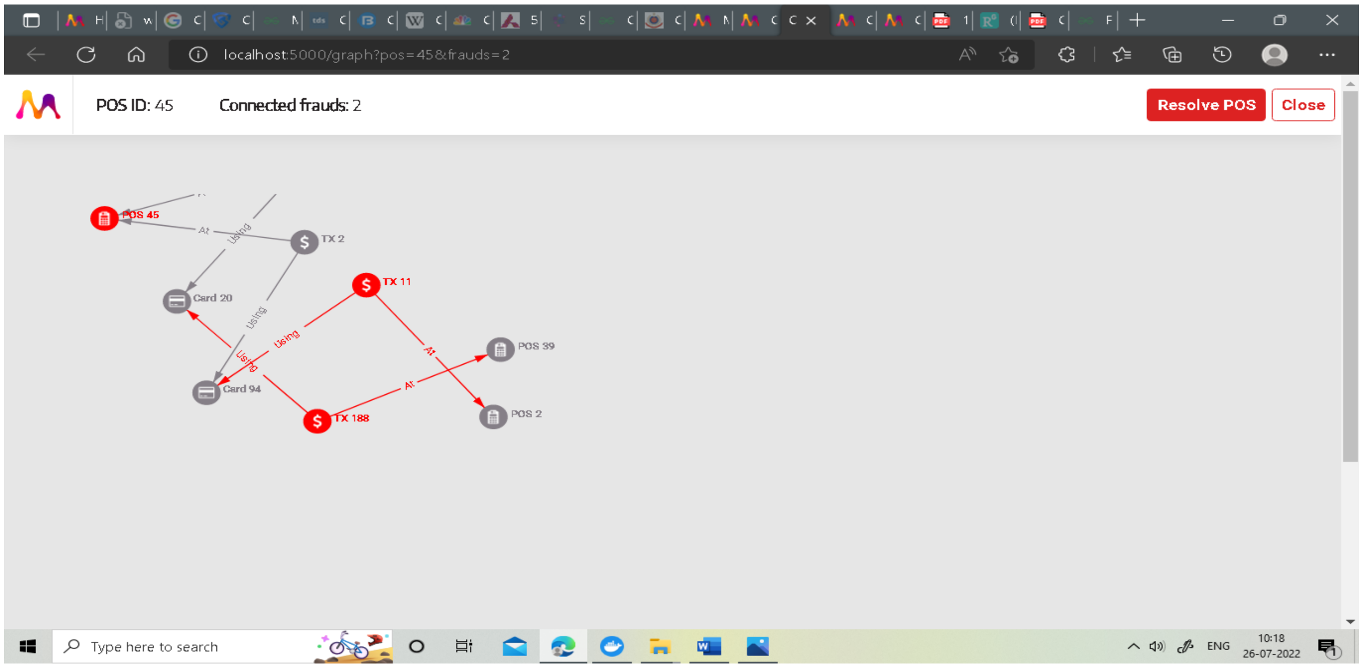
Reported as fraud

Transaction ID
130
156
104
132
11
163
154

Fraudulent POS devices

POS ID	# of connected frauds	
45	2	View on graph
72	2	View on graph

HOME PAGE



RECCOMENDATION PAGE

CHAPTER 6

APPLICATIONS

Credit card fraud is increasing considerably with the development of modern technology and the global superhighways of communication. Credit card fraud costs consumers and the financial company billions of dollars annually, and fraudsters continuously try to find new rules and tactics to commit illegal actions. Thus, fraud detection systems have become essential for banks and financial institution, to minimize their losses. However, there is a lack of published literature on credit card fraud detection techniques, due to the unavailable credit card transactions dataset for researchers. The most commonly techniques used fraud detection methods are Naïve Bayes (NB), Support Vector Machines (SVM), K-Nearest Neighbor algorithms (KNN). These techniques can be used alone or in collaboration using ensemble or meta-learning techniques to build classifiers. But amongst all existing method, ensemble learning methods are identified as popular and common method, not because of its quite straightforward implementation, but also due to its exceptional predictive performance on practical problems. In this paper we trained various data mining techniques used in credit card fraud detection and evaluate each methodology based on certain design criteria. After several trial and comparisons; we introduced the bagging classifier based on decision three, as the best classifier to construct the fraud detection model. The performance evaluation is performed on real life credit card transactions dataset to demonstrate the benefit of the bagging ensemble algorithm.

CHAPTER 7

CONCLUSION

Fraud detection is a complex issue that requires a substantial amount of planning before throwing machine learning algorithms at it. Nonetheless, it is also an application of data science and machine learning for the good, which makes sure that the customer’s money is safe and not easily tampered with.

Future work will include a comprehensive tuning of the Random Forest algorithm I talked about earlier. Having a data set with non-anonymized features would make this particularly interesting as outputting the feature importance would enable one to see what specific factors are most important for detecting fraudulent transactions.

REFERENCES

- Credit Card Fraud Detection Based on Transaction Behavior -by John
- Richard D. Kho, Larry A. Vea published by Proc. of the 2017 IEEE Region
- 10 Conference (TENCON), Malaysia, November 5-8, 2017
- L.J.P. van der Maaten and G.E. Hinton, Visualizing High-Dimensional
- Data Using t-SNE (2014), Journal of Machine Learning Research
- Machine Learning Group — ULB, Credit Card Fraud Detection (2018),
- Kaggle
- Nathalie Japkowicz, Learning from Imbalanced Data Sets: A Comparison of Various Strategies (2000), AAAI Technical Report WS-00–0

APPENDIX

```
import json

import logging

import models

import os

import time

from argparse import ArgumentParser

from flask import Flask, render_template, request,
    Response

from gqlalchemy import Memgraph

from random import randint, sample

from time import sleep



MEMGRAPH_HOST =
    os.getenv("MEMGRAPH_HOST", "memgraph")

MEMGRAPH_PORT =
    int(os.getenv("MEMGRAPH_PORT", "7687"))


# Connect to Memgraph
```

```
def connect_to_memgraph(memgraph_ip,
    memgraph_port):
  memgraph = Memgraph(host=memgraph_ip,
    port=int(memgraph_port))
  while True:
    try:
      if
    memgraph._get_cached_connection().is_active():
        return memgraph
    except:
      sleep(1)


memgraph =
    connect_to_memgraph(MEMGRAPH_HOST,
    MEMGRAPH_PORT)


log = logging.getLogger(__name__)


def init_log():
  logging.basicConfig(level=logging.INFO)
  log.info("Logging enabled")
  # Set the log level for werkzeug to WARNING
    because it will print out too much info otherwise
```

```
        logging.getLogger("werkzeug").setLevel(logging.W
        ARNING)



# Parse the input arguments for the app
def parse_args():
    """
    Parse command line arguments.
    """
    parser = ArgumentParser(description=__doc__)
    parser.add_argument("--host",           default="0.0.0.0",
        help="Allowed host addresses.")
    parser.add_argument("--port",  default=5000,  type=int,
        help="App port.")
    parser.add_argument(
        "--template-folder",
        default="public/template",
        help="The folder with flask templates.",
    )
    parser.add_argument(
        "--static-folder", default="public", help="The folder
        with flask static files."
    )
    parser.add_argument(
        "--debug",
        default=True,
        action="store_true",
        help="Run web server in debug mode",
    )
    print(__doc__)
    return parser.parse_args()


args = parse_args()


# Create the Flask server instance
app = Flask(
    __name__,
    template_folder=args.template_folder,
    static_folder=args.static_folder,
    static_url_path="",
)


def init_data(card_count, pos_count):
    """Populate the database with initial Card and POS
        device entries."""

    log.info("Initializing {} cards and {} POS
        devices".format(card_count, pos_count))
    start_time = time.time()
    # TODO: change this to query builder
    memgraph.execute(
        "UNWIND range(0, {} - 1) AS id "
        "CREATE (:Card {{id: id, compromised:
        false}})".format(card_count)
    )
    memgraph.execute(
        "UNWIND range(0, {} - 1) AS id "
        "CREATE (:Pos {{id: id, compromised:
        false}})".format(pos_count)
    )

    log.info("Initialized data in %.2f sec", time.time() -
        start_time)



def compromise_pos(pos_id):
    """Mark a POS device as compromised."""

    memgraph.execute("MATCH (p:Pos {{id: {}}}) SET
        p.compromised = true".format(pos_id))
    log.info("Point of sale %d is compromised", pos_id)



def compromise_pos_devices(pos_count, fraud_count):
    """Compromise a number of random POS devices."""


    log.info("Compromising {} out of {} POS
        devices".format(fraud_count, pos_count))
    start_time = time.time()


    compromised_devices = sample(range(pos_count),
        fraud_count)
    for pos_id in compromised_devices:
        compromise_pos(pos_id)


    log.info("Compromisation took %.2f sec", time.time()
        - start_time)



def pump_transactions(card_count, pos_count, tx_count,
        report_pct):
    """Create transactions. If the POS device is
        compromised,
    then the card in the transaction gets compromised too.
    If the card is compromised, there is a 0.1% chance the
    transaction is fraudulent and detected (regardless of
    the POS device)."""


    log.info("Creating {} transactions".format(tx_count))
    start_time = time.time()
    query = (
        "MATCH (c:Card {{id: {}}}), (p:Pos {{id: {}}}) "
        "CREATE (t:Transaction "
        "{{id: {}, fraudReported: c.compromised AND
        (rand() < %f)}}) "
        "CREATE (c)<-[:Using]-(t)-[:At]->(p) "
        "SET c.compromised = p.compromised" %
        report_pct
    )

    def rint(max):
        return randint(0, max - 1)

    for i in range(tx_count):
        memgraph.execute(query.format(rint(card_count),
        rint(pos_count), i))

    duration = time.time() - start_time
    log.info("Created %d transactions in %.2f seconds",
        tx_count, duration)


@app.route("/resolve-pos", methods=["POST"])
def resolve_pos():
    """Resolve a POS device and card as not
        compromised."""

    data = request.get_json(silent=True)
    start_time = time.time()

    memgraph.execute(
        "MATCH (p:Pos {{id: {}}}) "
        "SET p.compromised = false "
        "WITH p MATCH (p)--(t:Transaction)--(c:Card) "
        "SET t.fraudReported = false, c.compromised =
        false".format(data["pos"])
    )

    duration = time.time() - start_time
    log.info(
        "Compromised Point of sale %s has been resolved in
        %.2f sec",
        data["pos"],
        duration,
    )

    response = {"duration": duration}
    return Response(json.dumps(response), status=200,
        mimetype="application/json")
```

```
@app.route("/get-compromised-pos", methods=["GET"])
def get_compromised_pos():
    """Get compromised POS devices."""

    log.info("Getting compromised Point Of Service IDs")
    start_time = time.time()

    data = memgraph.execute_and_fetch(
        "MATCH (t:Transaction {fraudReported: true})-
        [:Using]->(:Card)"
        "<-[:Using]-(:Transaction)-[:At]->(p:Pos) "
        "WITH p.id as pos, count(t) as connected_frauds "
        "WHERE connected_frauds > 1 "
        "RETURN pos, connected_frauds ORDER BY
        connected_frauds DESC"
    )
    data = list(data)

    log.info(
        "Found %d POS with more then one fraud in %.2f
        sec",
        len(data),
        time.time() - start_time,
    )

    return json.dumps(data)
```

```
@app.route("/get-fraudulent-transactions",
        methods=["GET"])
def get_fraudulent_transactions():
    """Get fraudulent transactions."""

    log.info("Getting fraudulent transactions")
    start_time = time.time()

    data = memgraph.execute_and_fetch(
        "MATCH (t:Transaction {fraudReported: true})
        RETURN t.id as id"
    )
    data = list(data)

    duration = time.time() - start_time
    log.info("Found %d fraudulent transactions in %.2f",
        len(data), duration)

    response = {"duration": duration, "fraudulent_txs":
        data}
    return Response(json.dumps(response), status=200,
        mimetype="application/json")
```

```
@app.route("/generate-data", methods=["POST"])

def generate_data():

    """Initialize the database."""


    data = request.get_json(silent=True)


    if data["pos"] < data["frauds"]:

        return Response(

            json.dumps({"error": "There can't be more frauds
        than devices"}),

            status=418,

            mimetype="application/json",

        )


    start_time = time.time()


    memgraph.drop_database()

    init_data(data["cards"], data["pos"])

    compromise_pos_devices(data["pos"], data["frauds"])

    pump_transactions(data["cards"],              data["pos"],
        data["transactions"], data["reports"])


    duration = time.time() - start_time


    response = {"duration": duration}

    return    Response(json.dumps(response),    status=201,
        mimetype="application/json")



@app.route("/pos-graph", methods=["POST"])
def host():
    log.info("Client fetching POS connected components")

    request_data = request.get_json(silent=True)
    data = memgraph.execute_and_fetch(
        "MATCH           (p1:Pos)<-[:At]-(t1:Transaction
        {{fraudReported: true}})-[:Using] "
        "->(c:Card)<-[:Using]-(t2:Transaction)-[:At]-
        >(p2:Pos {{id: {}}})"
        "RETURN      p1,      t1,      c,      t2,
        p2".format(request_data["pos"])
    )
    data = list(data)

    output = []
    for item in data:
        p1, p2, t1, t2, c = {}, {}, {}, {}, {}
        p1["id"] = item["p1"].id
        p1["compromised"] = item["p1"].compromised
        p2["id"] = item["p2"].id
        p2["compromised"] = item["p2"].compromised
        t1["id"] = item["t1"].id
        t1["compromised"] = item["t1"].fraudReported
        t2["id"] = item["t2"].id
        t2["compromised"] = item["t2"].fraudReported
        c["id"] = item["c"].id
        c["compromised"] = item["c"].compromised
        output.append({"p1": p1, "t1": t1, "c": c, "t2": t2,
        "p2": p2})

    return Response(json.dumps(output), status=200,
        mimetype="application/json")


# Retrieve the home page for the app
@app.route("/", methods=["GET"])
def index():
    return render_template("index.html")


@app.route("/graph", methods=["GET"])
def graph():
    return render_template(
        "graph.html", pos=request.args.get("pos"),
        frauds=request.args.get("frauds")
    )
```

```
# Entrypoint for the app that will be executed first
def main():
    # Code that should only be run once
    if    os.environ.get("WERKZEUG_RUN_MAIN")    ==
        "true":
        init_log()
    app.run(host=args.host,                port=args.port,
        debug=args.debug)



if __name__ == "__main__":
    main()
```