Capstone Project

Fraud detection in financial transactions

Description

This project is designed to detect fraudulent financial transactions based on a series of heuristic rules. It tracks user transaction behavior, such as transaction amounts, locations, devices, and times, to identify unusual patterns that may indicate fraud.

Key features include:

* **Transaction Tracking**: Each user has a transaction history, and every new transaction is compared to their previous behavior.
* **Fraud Detection Rules**: Suspicious transactions are flagged based on predefined rules (e.g., large amounts, new devices, unusual locations, or odd times).
* **Risk Scoring**: Each transaction is assigned a risk score based on the factors above, and if the transaction exceeds set thresholds, it is flagged as suspicious.
* **Real-Time Alerts**: The system checks transactions in real-time and alerts users if potential fraud is detected.

The project utilizes classes for **Transaction** (which stores transaction data), **User** (which stores user information and their transaction history), and **TransactionOperation** (which contains the fraud detection logic). The system provides an interactive interface for adding new users, performing transactions, and viewing transaction histories. It also calculates the average risk factor for each user's transactions and provides alerts if necessary.

Features

1. User Registration:

Allows the creation of new user accounts with details like name, account number, bank name, trusted device, and location.

2. Transaction Tracking:

Tracks each user's transactions with unique IDs and stores transaction history using a linked list structure.

3. Fraud Detection Rules:

Flags suspicious transactions based on:

- Amount: Flags if the transaction exceeds twice the user's average.

- Device: Flags if the transaction is from an unfamiliar device.

- Location: Flags if the transaction occurs from an unusual location.

- Time: Flags transactions occurring during non-business hours (12:00 AM - 5:00 AM).

4. Risk Scoring:

Calculates a risk score for each transaction based on suspicious factors (amount, device, location, time).

5. Transaction History and Alerts:

Displays transaction history and generates alerts for potentially fraudulent activities.

6. Real-Time Fraud Detection:

Checks and flags transactions in real-time, aborting suspicious transactions and notifying the user for verification.

Implementation

**Data Structures Used:**

* **User Data:** A map<string, User> is used to store user details, where the key is the account number and the value is the User object containing information like account number, device, location, and transaction history.
* **Transaction Data:** Each transaction is represented by the Transaction class, which includes the transaction amount, time, location, device, and risk factor.
* **Transaction History:** The User class uses a doubly linked list (Node) to store the transaction history, allowing for easy access and manipulation of past transactions.

**Transaction Operation:**

**Fraud Detection Rules:** Several fraud detection rules are applied to each incoming transaction:

* + **Amount Rule:** Flags transactions if the amount exceeds twice the user’s average transaction amount.
  + **Device Rule:** Flags transactions made from an unfamiliar device.
  + **Location Rule:** Flags transactions made from locations that differ from the user’s typical location.
  + **Time Rule:** Flags transactions that occur outside of business hours (12:00 AM to 5:00 AM).
* **Risk Scoring:** Each transaction is assigned a risk score based on the number of rules violated. If multiple fraud rules are triggered, the risk score increases, and the transaction is flagged as suspicious.

**User Interaction:**

* The system alerts the user whenever a potentially fraudulent transaction is detected. If a transaction is flagged as suspicious, the user is prompted to confirm or deny it. If the user denies the transaction, it is considered fraudulent, and the system aborts it.

**Real-Time Processing:**

* Each incoming transaction is processed in real-time by applying the fraud detection rules. If any rules are violated, the system flags the transaction and notifies the user immediately.

**Implementation in Code:**

* · **Transaction Class:** Stores the details of each transaction, including its amount, time, location, device, and risk factor. It uses a static variable for generating unique transaction IDs.
* **User Class:** Stores user-specific data, such as the user’s account number, name, location, device, and transaction history (in a doubly linked list). It also calculates the average transaction amount and manages the user’s transaction history.
* **TransactionOperation Class:** Implements the fraud detection logic. It checks whether a transaction violates any of the predefined rules (amount, location, device, time), calculates a risk factor for the transaction, and decides whether to flag the transaction as suspicious or not.

Code

#include <bits/stdc++.h>

using namespace std;

//

class Transaction

{

private:

static int transactionIdCounter;

int transactionId;

double transactionAmount;

int transactiontime;

string place;

string device;

double riskfactor;

public:

Transaction() {}

Transaction(double amount, int transactiontime, string place, string device, double risk)

{

transactionId = ++transactionIdCounter;

this->transactionAmount = amount;

this->transactiontime = transactiontime;

this->place = place;

this->device = device;

this->riskfactor = risk;

}

int getTransactionId()

{

return transactionId;

}

double getTransactionAmount()

{

return transactionAmount;

}

int getTransactionTime()

{

return transactiontime;

}

string getLocation()

{

return place;

}

string getDevice()

{

return device;

}

double getriskFactor()

{

return riskfactor;

}

friend ostream &operator<<(ostream &os, const Transaction &tr);

};

ostream &operator<<(ostream &os, const Transaction &tr)

{

os << "Transaction ID: " << tr.transactionId << ", "

<< "Amount: " << tr.transactionAmount << ", "

<< "Time: " << tr.transactiontime << ", "

<< "Place: " << tr.place << ", "

<< "Device: " << tr.device << ", "

<< "Risk: " << tr.riskfactor;

return os;

}

int Transaction:: transactionIdCounter = 10011;

class Node

{

public:

Transaction transdata;

Node \*next;

Node \*prev;

Node() {}

Node(Transaction transdata)

{

this->transdata = transdata;

next = nullptr;

prev = nullptr;

}

};

static int nextUsedId = 1111;

class User

{

private:

int userId;

string username;

string accountnumber;

string bankname;

string userDevice;

string location;

Node \*head;

double totalTransactionAmount;

int transactionCount;

public:

User()

{

head = nullptr;

totalTransactionAmount = 0.0;

transactionCount = 0;

}

User(string name, string account, string bankname, string userDevice, string location)

{

userId = getUserId();

username = name;

accountnumber = account;

this->bankname = bankname;

this->location = location;

this->userDevice = userDevice;

head = nullptr;

}

string getUserName(){

return username;

}

string getbankName(){

return bankname;

}

string getuserDevice()

{

return userDevice;

}

string getlocation()

{

return location;

}

static int getUserId()

{

return ++nextUsedId;

}

string getAccountNumber()

{

return accountnumber;

}

double getAverageTransactionAmount()

{

if (transactionCount == 0)

{

return 0.0;

}

return totalTransactionAmount / transactionCount;

}

void AddTransaction(Transaction tr)

{

Node \*newTransaction = new Node(tr);

if (head == nullptr)

{

head = newTransaction;

}

else

{

Node \*temp = head;

while (temp->next != nullptr)

{

temp = temp->next;

}

temp->next = newTransaction;

newTransaction->prev = temp;

}

totalTransactionAmount = totalTransactionAmount + tr.getTransactionAmount();

transactionCount++;

cout << "Current Transaction: " << endl;

cout << newTransaction->transdata << endl;

}

double calculateAverageRisk()

{

if (head == nullptr)

{

return 0.0;

}

Node \*temp = head;

double totalRisk = 0.0;

int count = 0;

while (temp != nullptr)

{

totalRisk += temp->transdata.getriskFactor();

count++;

temp = temp->next;

}

return count > 0 ? totalRisk / count : 0.0;

}

void findUserDetails(User &user){

cout<<"Name : "<< user.getUserName()<<endl;

cout<<"Account Number : "<<user.getbankName()<<endl;

cout<<"User ID : "<<user.userId<<endl;

}

void showHistory()

{

if (head == nullptr)

{

cout << "No transaction history available." << endl;

return;

}

Node \*temp = head;

while (temp != nullptr)

{

cout << "TransactionId : " << temp->transdata.getTransactionId() << endl;

cout << "Transaction Amount : " << temp->transdata.getTransactionAmount() << endl;

cout << "Transaction time : " << temp->transdata.getTransactionTime() << endl;

cout << "Transaction location: " << temp->transdata.getLocation() << endl;

cout << "Transaction device: " << temp->transdata.getDevice() << endl;

cout << "Transaction Risk: " << temp->transdata.getriskFactor() << endl;

cout << endl;

cout << endl;

temp = temp->next;

}

cout << endl;

cout << endl;

}

};

class TransactionOperation

{

public:

void checkTransactions(User &uu, string accnum, double Amount, string location, string currdevice, int currenthour, map<string, User> &userDetails);

bool fraudInAmount(map<string, User> &userDetails, double Amount, string accnum);

bool fraudInLocation(map<string, User> &userDetails, string location, double Amount, string accnum);

bool fraudInDevice(map<string, User> &userDetails, string currDevice, string Location, double Amount, string accnum);

bool fraudInTime(int currentHour);

};

void TransactionOperation::checkTransactions(User &uu, string accnum, double Amount, string location, string currdevice, int currenthour, map<string, User> &userDetails)

{

//Risk Score=Base Risk+(Transaction Amount Factor)+(Device Factor)+(Location Factor)+(Time Factor)

bool flagged = false;

double risk = 0.0;

double baseRisk = 5.0;

if (Amount > 20000)

{

if (!fraudInAmount(userDetails, Amount, accnum))

{

flagged = true;

}

else

{

risk = risk + 10 \* 4.0;

}

}

if (uu.getlocation() != location)

{

if (!fraudInLocation(userDetails, location, Amount, accnum))

{

flagged = true;

}

else

{

risk = risk + 2 \* 2.0;

}

}

if (uu.getuserDevice() != currdevice)

{

if (!fraudInDevice(userDetails, currdevice, location, Amount, accnum))

{

flagged = true;

}

else

{

risk = risk + 3 \* 3.0;

}

}

if (!fraudInTime(currenthour))

{

flagged = true;

}

risk = risk + baseRisk;

if (flagged)

{

cout << "Fraud Detected. Transaction Aborted." << endl;

}

else

{

Transaction tr(Amount, currenthour, location, currdevice, risk);

cout << "Transaction Successful" << endl;

uu.AddTransaction(tr);

}

}

bool TransactionOperation::fraudInAmount(map<string, User> &userDetails, double Amount, string accnum)

{

User &u = userDetails[accnum];

double averageAmount = u.getAverageTransactionAmount();

if (Amount > 2 \* averageAmount)

{

cout << "Suspicious transaction: Amount exceeds twice the user's average transaction amount." << endl;

// return false;

}

char ch;

cout << "Hello, Is this you doing transaction for amount Rs. " << Amount << " (Y/N) : ";

cin >> ch;

if (ch == 'y' || ch == 'Y')

{

return true;

}

else

{

cout << "A transaction exceeding the limit of Rs. 20000 has been attempted on your account by someone other than the account owner. Please review your account activity immediately." << endl;

return false;

}

}

bool TransactionOperation::fraudInLocation(map<string, User> &userDetails, string location, double Amount, string accnum)

{

char ch;

cout << "Hello, Is this you doing transaction from Location = " << location << " for Amount = Rs. " << Amount << " (Y/N) : ";

cin >> ch;

if (ch == 'y' || ch == 'Y')

{

return true;

}

else

{

cout << "A transaction exceeding Rs. 20000 was attempted on your account by someone other than the account owner, from a different location." << endl;

return false;

}

}

bool TransactionOperation::fraudInDevice(map<string, User> &userDetails, string currDevice, string Location, double Amount, string accnum)

{

char ch;

cout << "Hello, Is this you doing transaction from Device = " << currDevice << " from Location = " << Location << " for Amount = Rs. " << Amount << " (Y/N) : ";

cin >> ch;

if (ch == 'y' || ch == 'Y')

{

return true;

}

else

{

cout << "A transaction exceeding Rs. 20000 was attempted on your account by someone other than the account owner, from a different location and device." << endl;

return false;

}

}

bool TransactionOperation::fraudInTime(int currentHour)

{

if (currentHour >= 0 && currentHour <= 5)

{

char ch;

cout << "Hello, Is this you doing a transaction at this time (current time is " << currentHour << " hours)? " << endl;

cout << "The transaction is happening outside normal hours (12 AM to 5 AM). " << endl;

cout << "Is this transaction legitimate? (Y/N) : ";

cin >> ch;

if (ch == 'y' || ch == 'Y')

{

return true;

}

else

{

cout << "Transaction marked as suspicious due to time of transaction. " << endl;

return false;

}

}

return true;

}

int main()

{

map<string, User> userDetails;

TransactionOperation tq;

int choice;

char ch;

string accnum, currdevice, location;

double Amount;

int currentHour;

string uname;

string userAccountnumber;

string bankname;

string trustedDevice;

string trustedlocation;

userDetails["SBI123456"] = User("Honour", "SBI123456", "STATE\_BANK\_OF\_INDIA", "PC", "Nagpur");

userDetails["HDFC987654"] = User("Edward", "HDFC987654", "HDFC\_BANK", "Mobile", "Hyderabad");

userDetails["ICICI763242"] = User("Arno", "ICICI763242", "ICICI\_BANK", "PC", "Raipur");

userDetails["BOI343422"] = User("Shay", "BOI343422", "BANK\_OF\_INDIA", "Tablet", "Pune");

userDetails["UN987654"] = User("Jacob", "UN987654", "UNION\_BANK", "Mobile", "London");

userDetails["SBI1235678"] = User("Kassandra", "SBI1235678", "STATE\_BANK\_OF\_INDIA", "Tablet", "Greece");

do

{

cout << "\*\*\*\*\*\*\*MAIN MENU\*\*\*\*\*\*\*\*" << endl;

cout << "1. ADD NEW USER " << endl;

cout << "2. DO TRANSACTIONS " << endl;

cout << "3. SHOW TRANSACTION HISTORY " << endl;

cout << "4. CALCULATE AVERAGE RISK RATE FOR SPECIFIC USER TRANSACTION " << endl;

cout << "5. SHOW SPECIFIC USER DETAILS " << endl;

cout << "6. EXIT " << endl;

cout << "Enter the choice from the Main Menu: ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter the User Detials " << endl;

cout << "Enter the Name : ";

cin >> uname;

cout << "Enter user Account number : ";

cin >> userAccountnumber;

cout << "Enter user bankname : ";

cin >> bankname;

cout << "Enter user trusted Device : ";

cin >> trustedDevice;

cout << "Enter user trusted location: ";

cin >> trustedlocation;

userDetails[userAccountnumber] = User(uname, userAccountnumber, bankname, trustedDevice, trustedlocation);

cout << "User Details Registered " << endl;

break;

case 2:

cout << "Enter account number: ";

cin >> accnum;

if (userDetails.find(accnum) != userDetails.end())

{

User &user = userDetails[accnum];

cout << "Enter amount: ";

cin >> Amount;

cout << "Enter location: ";

cin >> location;

cout << "Enter device: ";

cin >> currdevice;

time\_t now = time(0);

tm \*ltm = localtime(&now);

currentHour = ltm->tm\_hour;

tq.checkTransactions(user, accnum, Amount, location, currdevice, currentHour, userDetails);

// user.showHistory();

}

else

{

cout << "Account not found!" << endl;

}

break;

case 3:

cout << "Enter account number to view transaction history: ";

cin >> accnum;

if (userDetails.find(accnum) != userDetails.end())

{

User &user = userDetails[accnum];

user.showHistory();

}

else

{

cout << "Account not found!" << endl;

}

break;

case 4:

cout << "Enter the accoutn number you wish to find the average risk rate : ";

cin >> accnum;

if (userDetails.find(accnum) != userDetails.end())

{

User &user = userDetails[accnum];

double avgRisk = user.calculateAverageRisk();

if(avgRisk >= 10.0){

cout<<"ALERT !! Average Risk Factor for User With Account Number => "<<user.getAccountNumber()<<endl;

}

//cout << "Average Risk for User with account " << accnum << ": " << avgRisk << endl;

}

else

{

cout << "Account number not found." << endl;

}

break;

case 5:

cout << "Enter the account Number you wish to find : ";

cin >> accnum;

if (userDetails.find(accnum) != userDetails.end())

{

User &user = userDetails[accnum];

user.findUserDetails(user);

}

else

{

cout << "Account number not found " << endl;

}

case 6:

cout << "Exiting program." << endl;

break;

default:

cout << "Invalid choice, please try again!" << endl;

break;

}

cout << "Do you want to continue? (Y/N): ";

cin >> ch;

} while (ch == 'Y' || ch == 'y');

return 0;

}

Output

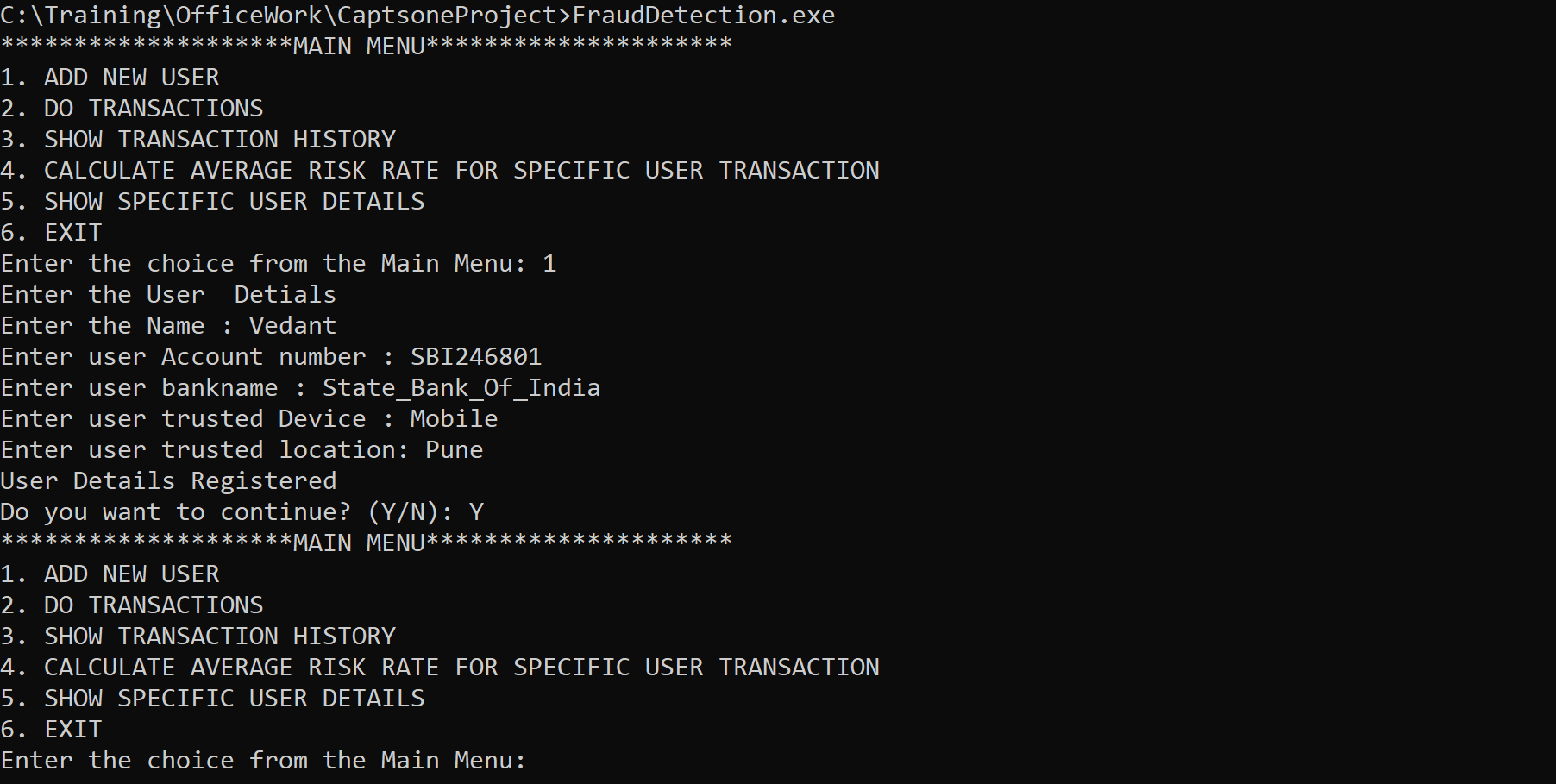


Image 1: Add New User

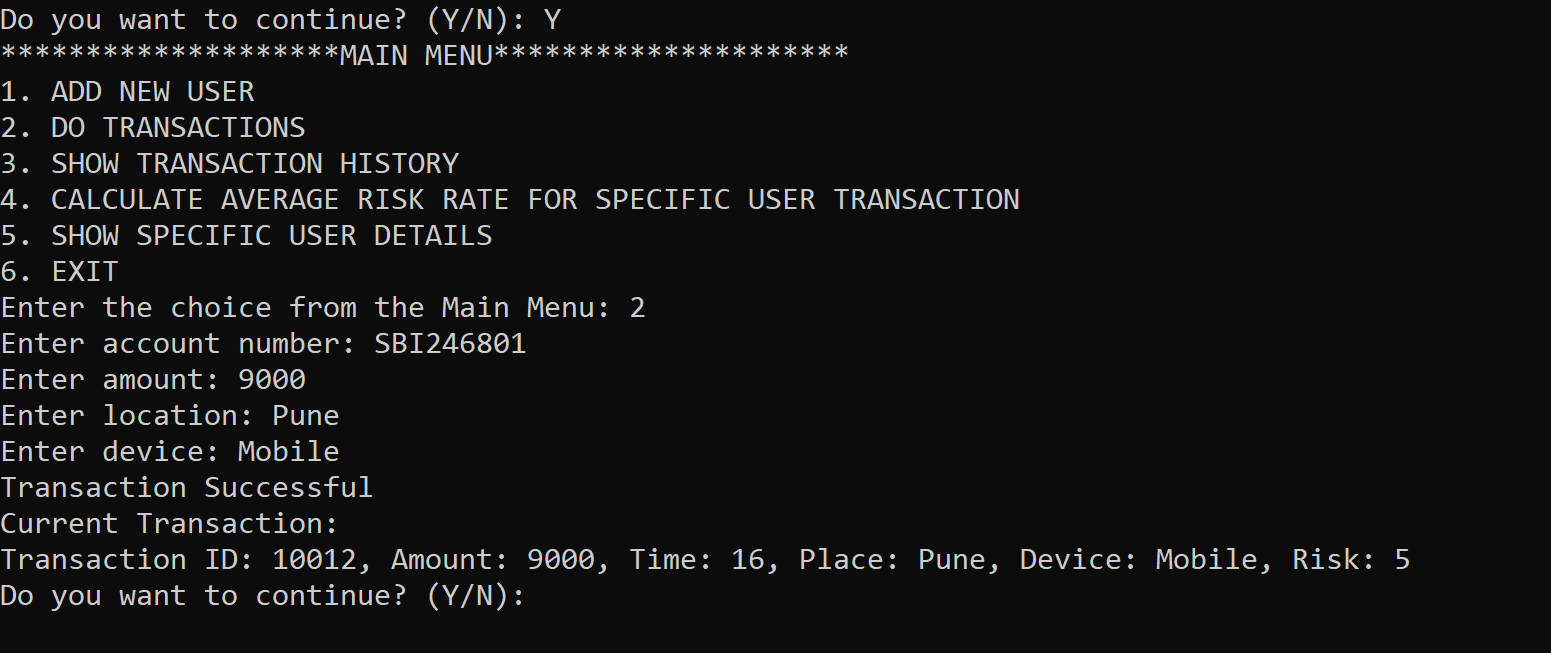


Image 2: Transaction done with valid details

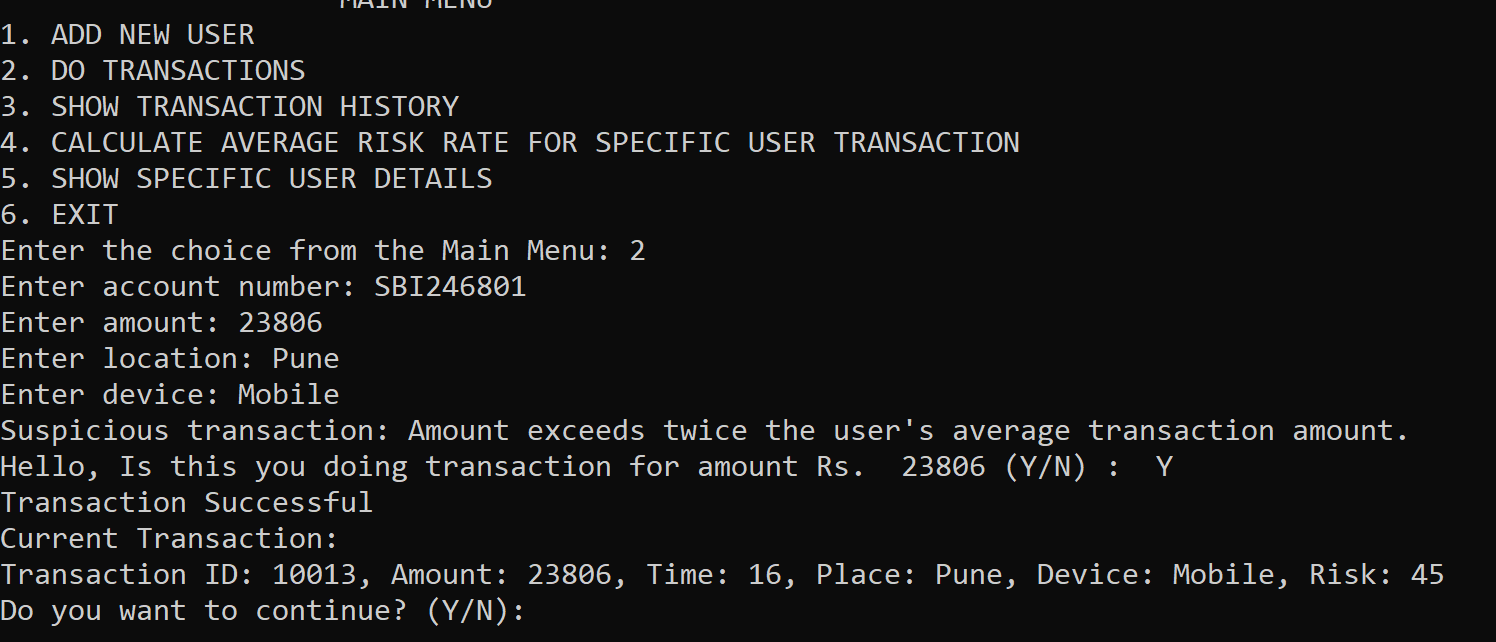


Image 3: Valid Transaction with High Amount

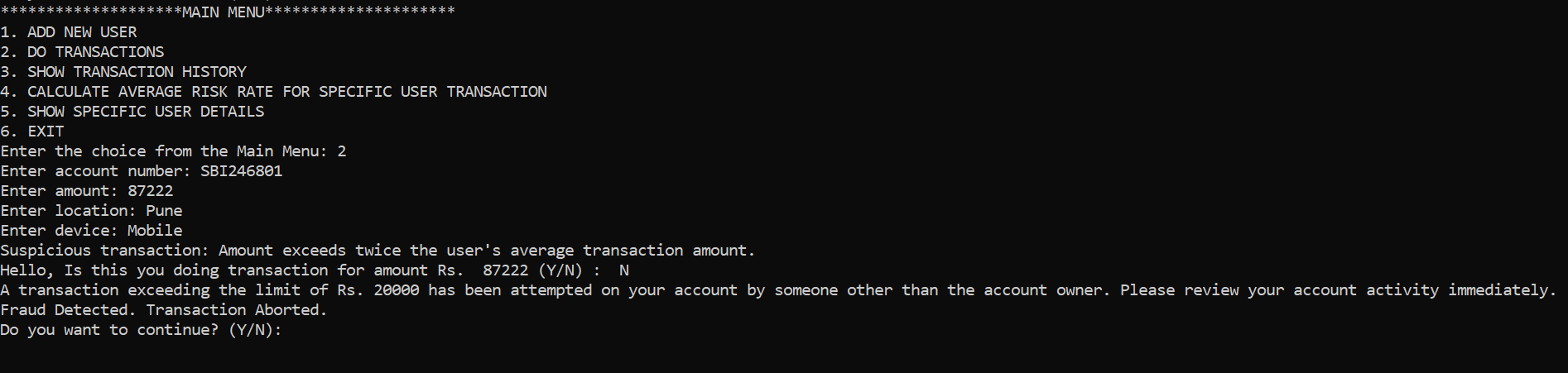


Image 4: Invalid Transaction with High Amount

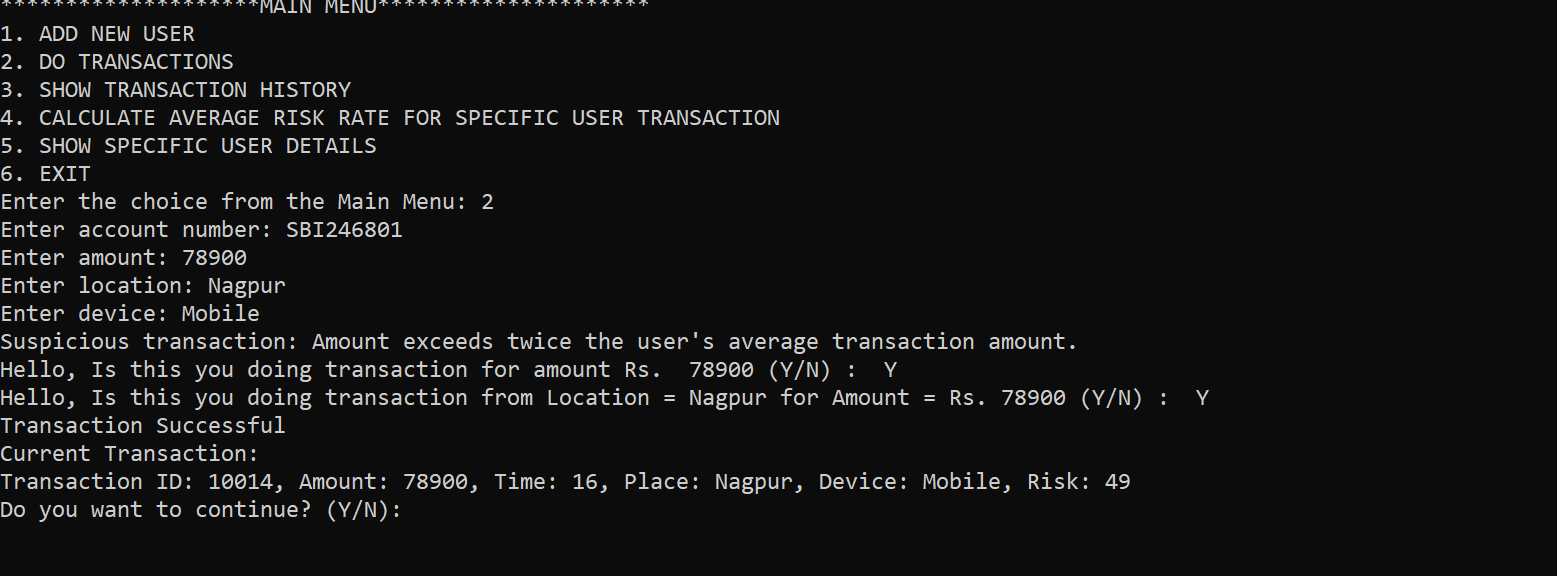


Image 5: Valid Transaction with High Amount & Incorrect Location

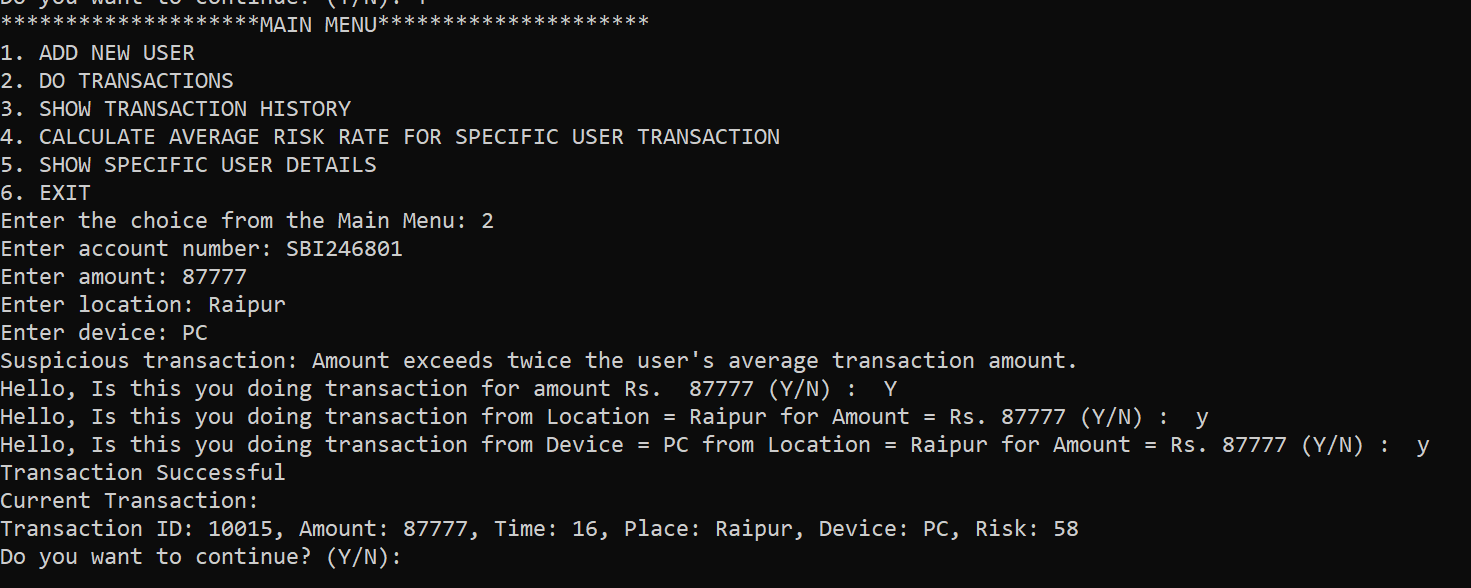


Image 6: Valid Transaction with High Amount, Incorrect Location & wrong time

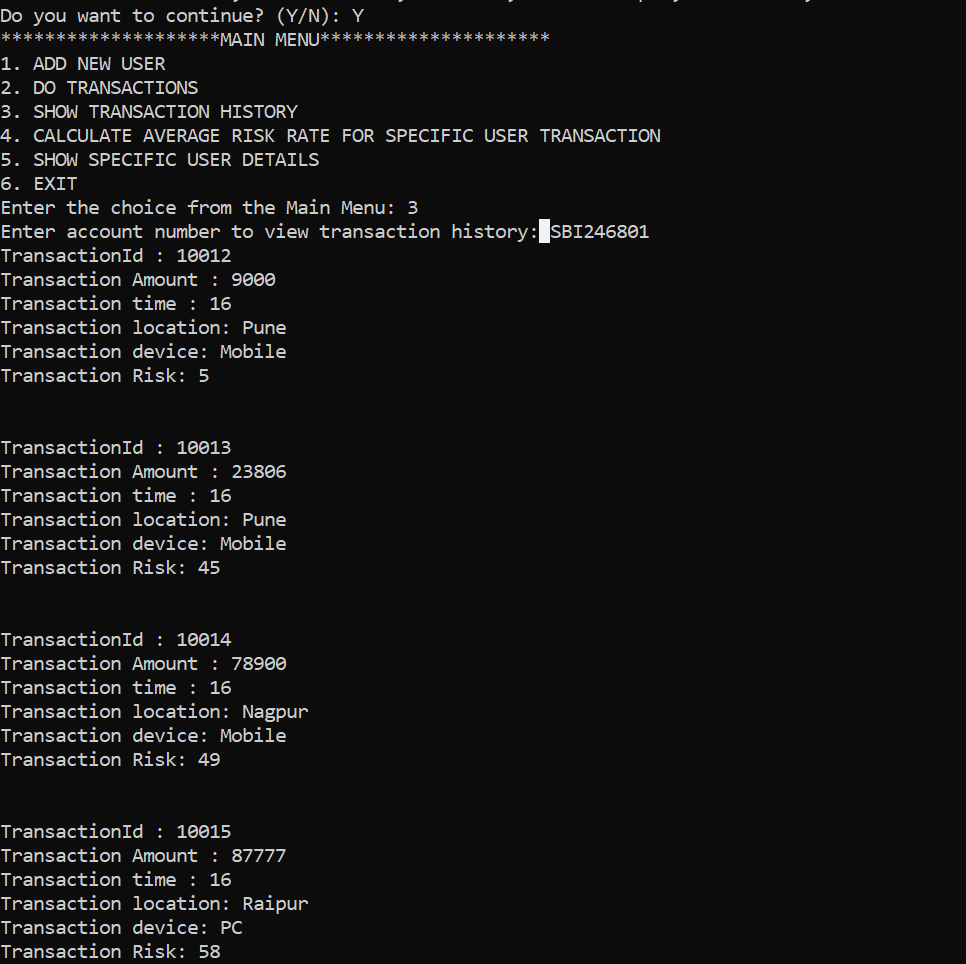


Image 7: Show Transaction history

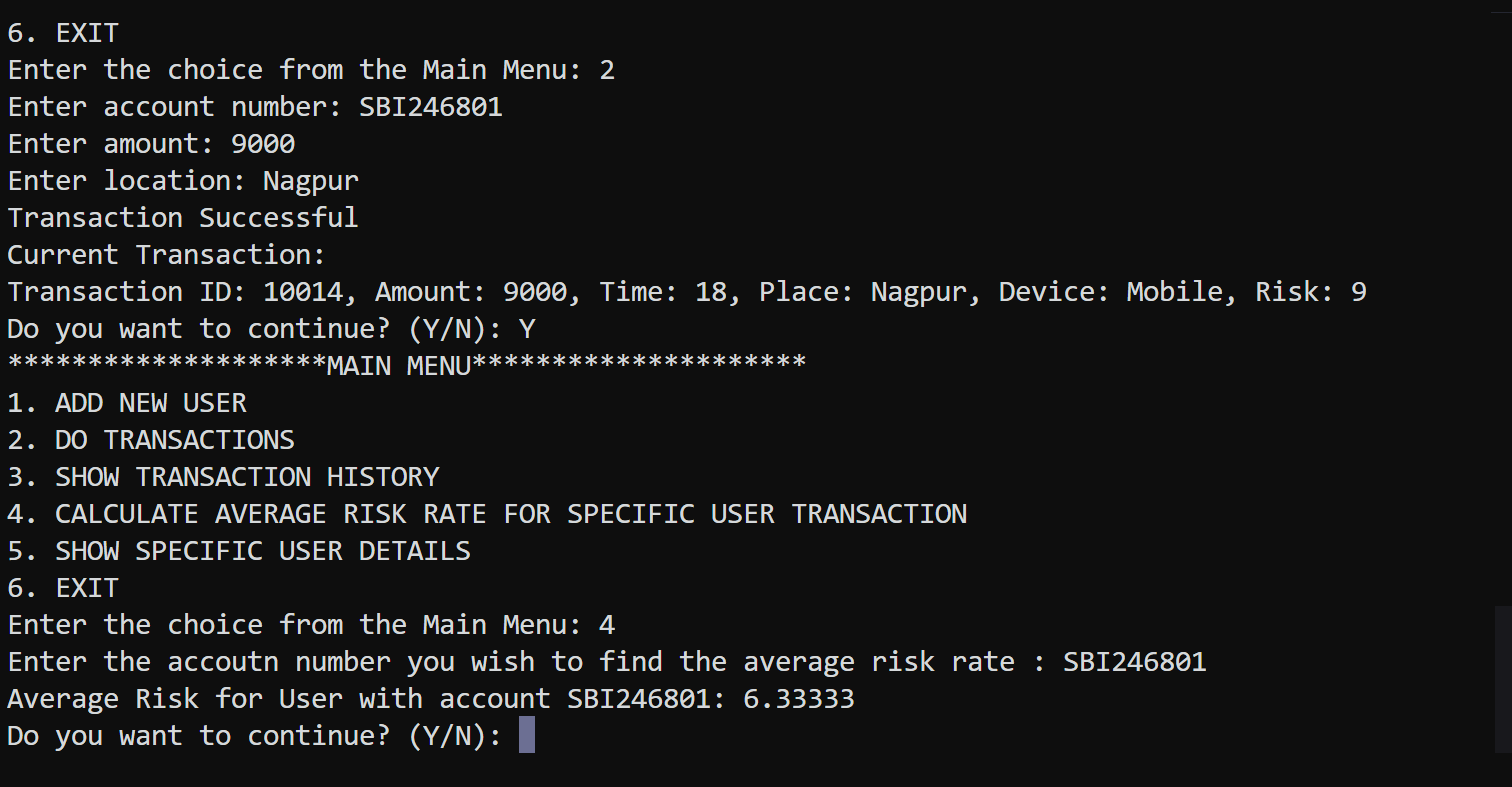


Image 8: Showing Average Risk

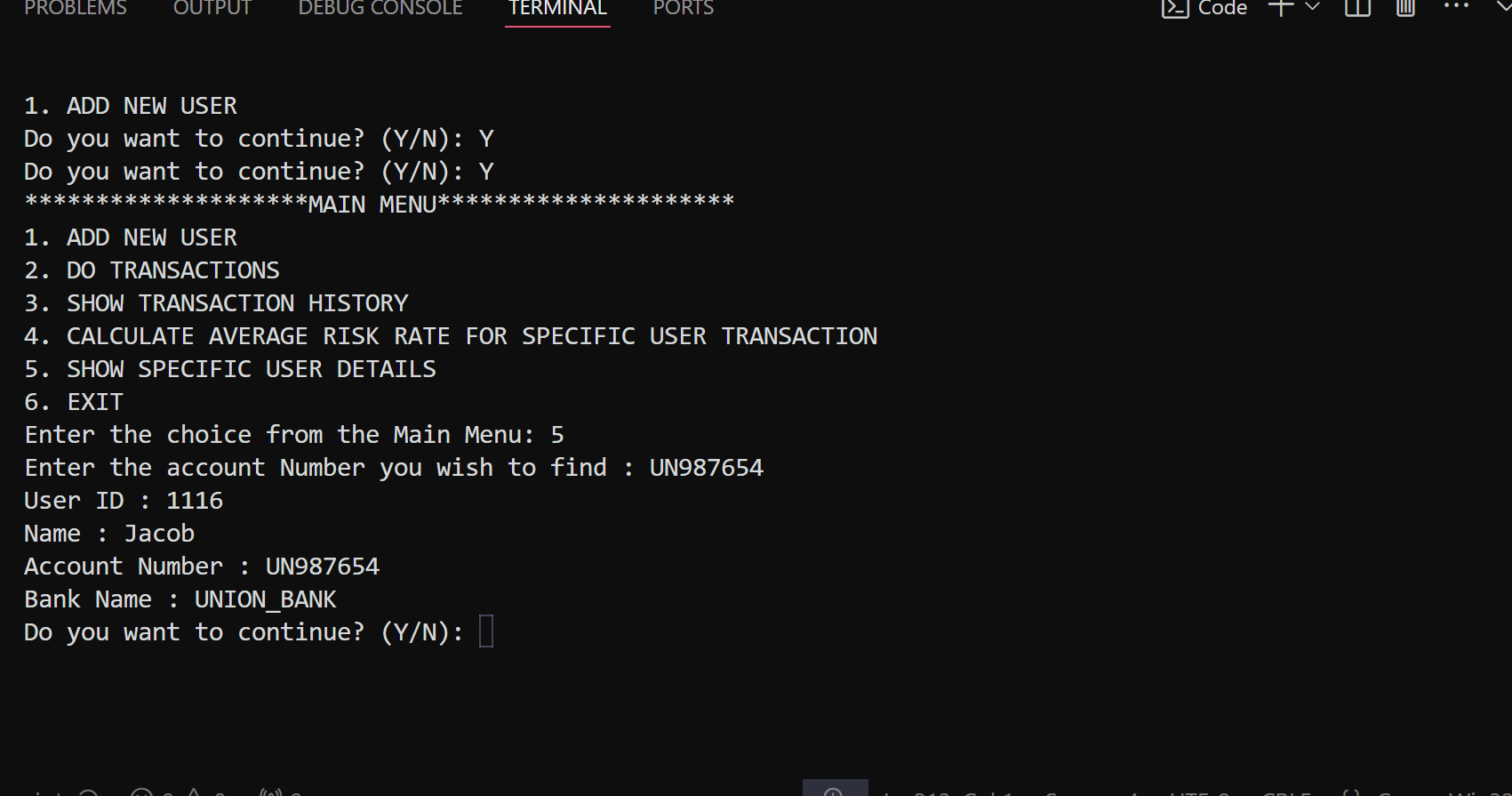


Image 9: Showing User Information

Flow Chart

