**TransactionMonitoringApplication.java**

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class TransactionMonitoringApplication {

public static void main(String[] args) {

SpringApplication.run(TransactionMonitoringApplication.class, args);

}

}

**TransactionController.java**

import org.springframework.web.bind.annotation.\*;

import java.time.temporal.ChronoUnit;

import java.util.\*;

import java.util.concurrent.ConcurrentHashMap;

import java.util.stream.Collectors;

@RestController

@RequestMapping("/transactions")

public class TransactionController {

private final Set<String> blacklistedAccounts = Collections.synchronizedSet(new HashSet<>()); // Dynamically updated blacklist

private final List<Transaction> flaggedTransactions = Collections.synchronizedList(new ArrayList<>());

private final Map<String, List<Transaction>> transactionHistory = new ConcurrentHashMap<>();

@PostMapping("/process")

public Map<String, String> processTransaction(@RequestBody Transaction transaction) {

String reason = evaluateTransaction(transaction);

if (reason != null) {

flaggedTransactions.add(transaction);

return Map.of(

"status", "Fraud",

"reason", reason

);

}

recordTransaction(transaction);

return Map.of(

"status", "Success",

"message", "Transaction is valid."

);

}

@GetMapping("/admin/flagged-transactions")

public List<Map<String, String>> getFlaggedTransactions(@RequestHeader("Authorization") String authorization) {

if (!isAdminAuthorized(authorization)) {

throw new UnauthorizedException();

}

return flaggedTransactions.stream()

.map(transaction -> Map.of(

"account", transaction.getAccount(),

"amount", String.valueOf(transaction.getAmount()),

"ip", transaction.getIpAddress(),

"reason", transaction.getReason()

))

.collect(Collectors.toList());

}

@PostMapping("/admin/blacklist")

public Map<String, String> addBlacklistedAccount(@RequestHeader("Authorization") String authorization, @RequestBody String account) {

if (!isAdminAuthorized(authorization)) {

throw new UnauthorizedException();

}

blacklistedAccounts.add(account);

return Map.of("status", "Success", "message", "Account added to blacklist.");

}

@DeleteMapping("/admin/blacklist")

public Map<String, String> removeBlacklistedAccount(@RequestHeader("Authorization") String authorization, @RequestBody String account) {

if (!isAdminAuthorized(authorization)) {

throw new UnauthorizedException();

}

if (blacklistedAccounts.remove(account)) {

return Map.of("status", "Success", "message", "Account removed from blacklist.");

} else {

return Map.of("status", "Failure", "message", "Account not found in blacklist.");

}

}

private String evaluateTransaction(Transaction transaction) {

if (transaction.getAmount() > 100000) {

transaction.setReason("Transaction amount exceeds the limit of ₹100,000.");

return transaction.getReason();

}

if (blacklistedAccounts.contains(transaction.getAccount())) {

transaction.setReason("Account is blacklisted.");

return transaction.getReason();

}

if (!isIpAddressFromIndia(transaction.getIpAddress())) {

transaction.setReason("IP address is outside India.");

return transaction.getReason();

}

List<Transaction> recentTransactions = transactionHistory.getOrDefault(transaction.getAccount(), new ArrayList<>());

long recentCount = recentTransactions.stream()

.filter(t -> ChronoUnit.MINUTES.between(t.getTimestamp(), transaction.getTimestamp()) <= 5)

.count();

if (recentCount >= 3) {

transaction.setReason("More than 3 transactions within 5 minutes from the same account.");

return transaction.getReason();

}

return null;

}

private void recordTransaction(Transaction transaction) {

transactionHistory.computeIfAbsent(transaction.getAccount(), k -> new ArrayList<>()).add(transaction);

}

private boolean isIpAddressFromIndia(String ipAddress) {

// Simplified check: Validate based on predefined IP ranges (use external API or database in real-world scenarios).

return ipAddress.startsWith("49.") || ipAddress.startsWith("103."); // Example ranges for India

}

private boolean isAdminAuthorized(String authorization) {

return "Bearer admin-token".equals(authorization); // Example hardcoded token

}

}

**Transaction.java**

import java.time.LocalDateTime;

public class Transaction {

private String account;

private double amount;

private String ipAddress;

private LocalDateTime timestamp;

private String reason;

// Getters and setters

public String getAccount() {

return account;

}

public void setAccount(String account) {

this.account = account;

}

public double getAmount() {

return amount;

}

public void setAmount(double amount) {

this.amount = amount;

}

public String getIpAddress() {

return ipAddress;

}

public void setIpAddress(String ipAddress) {

this.ipAddress = ipAddress;

}

public LocalDateTime getTimestamp() {

return timestamp;

}

public void setTimestamp(LocalDateTime timestamp) {

this.timestamp = timestamp;

}

public String getReason() {

return reason;

}

public void setReason(String reason) {

this.reason = reason;

}

}

**UnauthorizedException.java**

import org.springframework.http.HttpStatus;

import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(code = org.springframework.http.HttpStatus.UNAUTHORIZED)

class UnauthorizedException extends RuntimeException {

public UnauthorizedException() {

super("Unauthorized access");

}

}