

Przetwarzanie Strumieni Danych  
i Data Science  
Projekt

Łukasz Zalewski  
nr indeksu 283655

13 czerwca 2024

# Spis treści

<b>1</b>	<b>Wstęp</b>	<b>3</b>
<b>2</b>	<b>Założenia projektu</b>	<b>3</b>
2.1	Generowanie transakcji . . . . .	3
2.2	Anomalie . . . . .	4
<b>3</b>	<b>Komponenty systemu</b>	<b>4</b>
3.1	Symulator transakcji . . . . .	5
3.2	Detektor anomalii . . . . .	6
3.3	Konsumer wiadomości o anomaliach . . . . .	6
3.4	Panel admina . . . . .	7
<b>4</b>	<b>Architektura systemu</b>	<b>9</b>
4.1	Konfiguracja systemu . . . . .	9
4.2	Diagram architektury . . . . .	9
<b>5</b>	<b>Testy systemu</b>	<b>11</b>
5.1	Przekroczenie wartości limitu karty . . . . .	11
5.2	Przekroczenie dopuszczalnej ilości transakcji dla użytkownika . .	12
5.3	Zbyt duża zmiana położenia w oknie czasowym . . . . .	14
5.4	Wykrycie anomalii względem odchylenia standardowego . . . . .	16

# 1 Wstęp

Niniejszy raport przedstawia projekt rozwiązania na przedmiot “Przetwarzanie Strumieni Danych i Data Science”. W dokumentacji przedstawione są podejście do problemu detekcji anomalii w transakcjach kartami płatniczymi, rozpatrywane typy anomalii, oraz architektura rozwiązania wraz z opisem poszczególnych komponentów.

Ponadto, w dokumentacji zawarte są testy aplikacji. Ukazują one skuteczność działania względem wygenerowanych anomalii, sposób wykrycia oraz metodę, za pomocą której użytkownik dostaje sygnalizację o skutecznej detekcji.

## 2 Założenia projektu

Głównym celem projektu jest stworzenie systemu składającego się z komponentu generującego transakcje płatnicze, detektora anomalii wykrywającego wszelkie możliwe próby oszustwa, oraz komponentu odpowiedzialnego za wizualizację anomalii i powiadomienia użytkownika o znalezionych problemach. System ma działać w czasie rzeczywistym, i zarazem w takim trybie mają być generowane dane.

W celu stworzenia aplikacji poprawnie symulującej przetwarzanie transakcji, konieczne było ustanowienie pewnych założeń w kontekście wykrywanych anomalii oraz generowania danych testowych.

### 2.1 Generowanie transakcji

W celu stworzenia wiarygodnej symulacji oraz ukazania faktu, iż system przetwarza dane generowane w czasie rzeczywistym, transakcje są generowane przez cały czas. W związku z tym postanowiono przyjąć następujące założenia.

- W każdej sekundzie generowana jest losowa liczba transakcji znajdującą się w przedziale  $[0, 5]$
- Dane są generowane losowo dla liczby 2000 użytkowników i przydzielonych im losowo 10000 kartom płatniczym
- Wygenerowane w naturalny sposób transakcje mogą - ale nie muszą - być transakcjami wykazującymi cechy transakcji podejrzanych o bycie anomaliami
- Dane transakcji będą zapisywane w formacie JSON i następnie będą wysyłane przez producenta w formie wiadomości na topic Kafki o nazwie **Transakcje**

## 2.2 Anomalie

Poza samymi w sobie transakcjami, w kontekście detekcji anomalii konieczne jest ustalenie ich pewnego rodzaju które mogłyby być wykrywane w systemie, oraz konieczne jest też ustalenie tego, w jaki sposób anomalie mają być generowane celem przetestowania skuteczności systemu.

W zaimplementowanym systemie zaproponowane są następujące typy anomalii.

- Przekroczenie limitu karty - anomalia związana z tym, że na karcie powiązanej z użytkownikiem wartość transakcji przekroczyła limit karty
- Zbyt duża liczba transakcji w krótkim odstępie czasowym dla jednego użytkownika - anomalia wykryta w sytuacji, gdy w krótkim czasie zostaną wykryte transakcje z przynajmniej trzech kart należących do jednego użytkownika
- Nagła zmiana lokalizacja w krótkim odstępie czasowym - sytuacja, gdy dla blisko położonych w czasie transakcji nagle dochodzi do dużej zmiany lokalizacji
- Wykrycie odchylenia - sytuacja, w której wartość transakcji jest zakwalifikowana jako anomalia gdy weźmie się pod uwagę odchylenie standardowe dotychczas zebranych danych

Ponadto, w zakresie generowania samych anomalii celem ich detekcji postawiono następujące założenia.

- Anomalie mogą być generowane losowo w każdej sekundzie, z prawdopodobieństwem wystąpienia wynoszącym 1%
- Celem możliwości przetestowania losowego występowania transakcji, jeżeli wylosowane zostanie generowanie anomalii, kolejne transakcje będące anomalie zostaną wygenerowane w kolejnych sekundach działania generatora
- Poza losowym wystąpieniem anomalii powinno być również możliwe ich generowanie na żądanie przez użytkownika celem przetestowania systemu pod kątem ich detekcji
- Dane wykryte i przefiltrowane przez detektora anomalii będą wysyłane w formacie JSON na topic Kafki o nazwie **Alerty**

## 3 Komponenty systemu

W architekturze aplikacji wyszczególnione zostały następujące komponenty.

### 3.1 Symulator transakcji

Symulator transakcji to aplikacja, której głównym celem jest tworzenie co sekundę losowych transakcji, a następnie zapisywanie ich na topic Kafki **Transakcje**. Symulator co sekundę generuje od 0 do 5 transakcji, a jednocześnie może też generować anomalie - w sposób losowy, jak i również na żądanie użytkownika.

Od strony technicznej komponent został zaimplementowany w języku **C#**. Biblioteka, za pomocą której zostały zaimplementowana funkcjonalność producenta Kafki, nosi nazwę **Confluent.Kafka**. Do wygenerowania danych użytkowników, kart i transakcji została użyta biblioteka **Bogus**. Ponadto, aplikacja umożliwia wysyłanie żądań zgodnie z protokołem HTTP w celu na przykład wygenerowanie konkretnego typu anomalii, albo pobrania danych.

W kontekście symulatora poczynione zostały następujące założenia.

- Na samym początku działania aplikacja generuje dane dla 2000 użytkowników i 10000 kart płatniczych, przydzielając przy tym karty do tych użytkowników
- Po wstępnym wygenerowaniu danych rozpoczyna się generowanie transakcji co sekundę
- Generowane dane transakcji są w czasie rzeczywistie wysyłane na odpowiedni topic Kafki, po czym możliwe jest ich dalsze przetworzenie
- W każdym momencie działania aplikacji możliwe jest wysłanie żądania zgodnie z protokołem HTTP celem wygenerowania anomalii na żądanie

Dostępne URL do żądań, jakie można wysłać korzystając z protokołu HTTP do symulatora, wraz z ewentualnymi parametrami wyglądają w następujący sposób.

- `api/users/random (GET)` - endpoint pozwalający na pobranie podanej przez użytkownika liczby losowych użytkowników celem sprawdzenia poprawności wygenerowanych danych
- `api/transactions (GET)` - endpoint pozwalający na pobranie transakcji dla zadanego id klienta i karty - przydatne w celu identyfikacji transakcji po wykryciu anomalii
- `api/anomalies/over-the-limit (POST)` - endpoint pozwalający na wygenerowanie anomalii związanej z przekroczeniem limitu karty w trakcie transakcji
- `api/anomalies/multiple-transactions (POST)` - endpoint pozwalający na wygenerowanie anomalii związanej ze zbyt dużą ilością transakcji dla jednego użytkownika w krótkim czasie
- `api/anomalies/location-change (POST)` - endpoint pozwalający na wygenerowanie anomalii związanej ze zbyt dużą zmianą lokalizacji w krótkim oknie czasowym dla danego użytkownika i karty

### 3.2 Detektor anomalii

Detektor aplikacji to aplikacja, której celem jest pobieranie danych z topica **Transakcje**, wykrycie anomalii, a następnie wysłanie ich po przefiltrowaniu do topica Kafki **Alerty**. Wiadomości są przetwarzane na bieżąco, i też w takim trybie są wysyłane na odpowiedni topic po poprawnej detekcji.

Od strony technicznej detektor został zaimplementowany w języku **Java**. Aplikacja jest po kompilacji uruchamiana jako job w środowisku **ApacheFlink**.

Podczas implementacji następujące założenia zostały poczynione.

- Detektor działa na bieżąco, i w czasie rzeczywistym przetwarza wiadomości
- Po odczynie przed detektora wiadomości nie są one usuwane z topica Kafki
- Dla każdego rodzaju anomalii stworzone są osobno kolejne zasady detekcji, które są potem wykorzystane przy przetwarzaniu danych
- Informacje o anomaliiach są wysyłane w formacie JSON na topic o nazwie **Alerty**

### 3.3 Konsumer wiadomości o anomaliiach

Kolejnym komponentem zaimplementowanym w ramach systemu jest konsumer wiadomości, który jest subskrybentem topica Kafki o nazwie **Alerty**. To z niego czyta informacje o anomaliiach, i udostępnia je klientom, którzy przy wykorzystaniu protokołu HTTP są w stanie uzyskać w ramach odpowiedzi na żądanie.

Komponent nie był elementem pierwotnie proponowanym w założeniach zadania jako części systemu, ale został zaimplementowany w ramach lepszego rozdzielenia pod względem architektonicznym aplikacji wizualizującej detekcję od rzeczywistego konsumenta tych informacji. Ponadto, dzięki temu uzyskana jest możliwość lepszego przetestowania tej części aplikacji dzięki implementacji endpointów zgodnych z protokołem HTTP.

Od strony technicznej aplikacja została - tak samo jak w przypadku symulatora - w języku **C#**. Biblioteka, za pomocą którego zostały zaimplementowana funkcjonalność producenta Kafki, to **Confluent.Kafka**.

Podczas implementacji następujące założenia zostały poczynione.

- Aplikacja konsumuje wiadomości z topica Kafki **Alerty**, po czym zapisuje je w swojej pamięci
- Zapisane alerty są dostępne w każdym momencie działania tej aplikacji, i mogą być pobrane przez żądanie zgodnie z protokołem HTTP

Dostępne URL do żądań, jakie można wysłać korzystając z protokołu HTTP do symulatora, wraz z ewentualnymi parametrami wyglądają w następujący sposób.

- `api/get-all` (GET) - endpoint który zwraca wszystkie wykryte anomalie

### 3.4 Panel admina

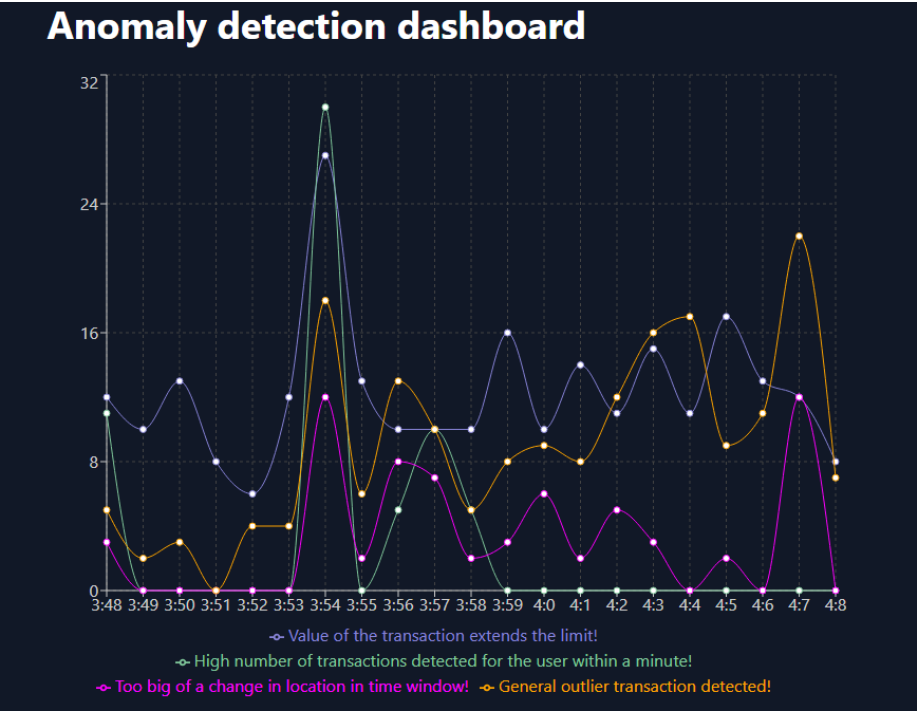
Ostatnim komponentem w systemie jest panel admina. To komponent odpowiedzialny za informowanie użytkownika o anomaliach, i ich wizualizację w określonych odstępach czasowych.

Od strony technicznej aplikacja została napisana przy wykorzystaniu frameworka **React.js**. Wykresy liniowe zaś zostały stworzone przy wykorzystaniu biblioteki **Recharts**.

Podczas implementacji następujące założenia zostały poczynione.

- Panel co minutę wysyła żądanie do konsumera anomalii celem pobrania informacji o wykrytych nieprawidłowościach
- Panel musi zawierać wykres ilości anomalii danego typu od przedziału czasowego, w którym taka anomalia wystąpiła
- Panel musi zawierać tabelę z informacjami na temat konkretnych anomalii
- Dane wyświetlane w tabeli i na wykresie powinny się aktualizować w czasie rzeczywistym

Poniżej ukazany jest wykres liniowy w aplikacji, który ukazuje ilość wystąpień danych anomalii w czasie.



Rysunek 1: Wykres liniowy aplikacji

Z kolei poniżej pokazana jest tabela pokazująca konkretne anomalie wraz z informacjami, po których można je zidentyfikować.

Previous

All Reasons

All Reasons

Value of the transaction extends the limit!

High number of transactions detected for the user within a minute!

Too big of a change in location in time window!

General outlier transaction detected!

Next

Id	CardId		stamp
1	4545		2024-06-13T03:48:01
2	9301	673	2024-06-13T03:48:02
3	2034	224	2024-06-13T03:48:04
4	1802	1419	2024-06-13T03:48:04
5	9693	676	2024-06-13T03:48:07

Rysunek 2: Tabela z informacjami na temat anomalii

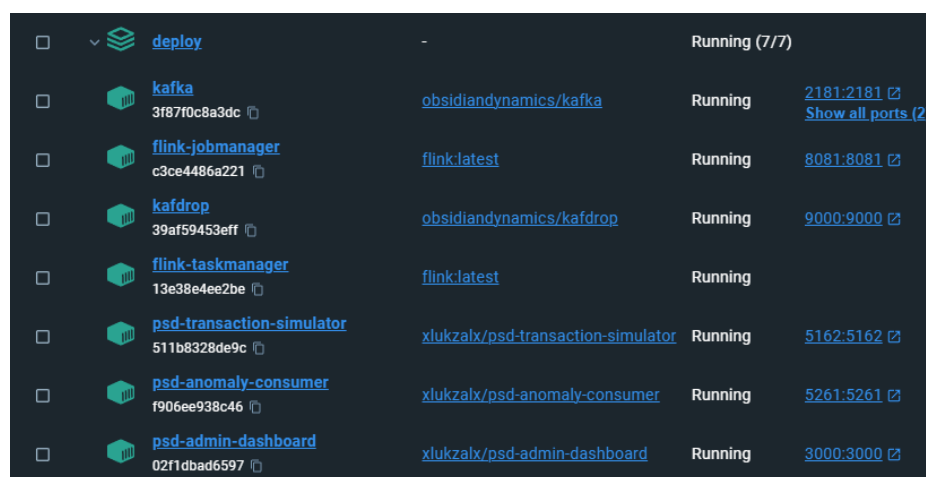























## 4 Architektura systemu

Po zdefiniowaniu komponentów systemu konieczne było zdefiniowanie konfiguracji, i logiczne usytuowanie ich w ramach wspólnej architektury. Ten dział przedstawia te kwestie.

### 4.1 Konfiguracja systemu

W celu uzyskania dobrego działania aplikacji na każdym systemie, postanowiono utworzyć obrazy Dockerowe dla konsumenta i producenta wiadomości oraz dla panelu admina. Obrazy zostały zbudowane i wysłane na publiczne repozytoria autora rozwiązania. Jedynym wyjątkiem od reguły jest detektor anomalii, który wymaga zainstalowanej Javy w wersji 11 na systemie. Wynika to z tego, że konieczne jest lokalna kompilacja aplikacji, a następnie ręczne jej uruchomienie w kontenerze właściwym dla managera zadań Flink, co jest wykonane za pomocą specjalnego skryptu.

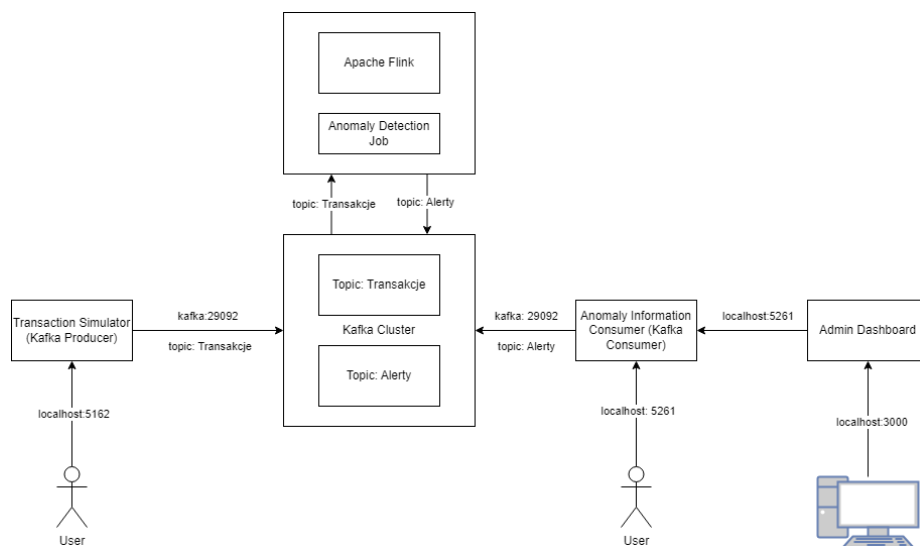


<input type="checkbox"/>	▼  <b>deploy</b>	-	Running (7/7)
<input type="checkbox"/>	 <b>kafka</b> 3f87f0c8a3dc 	<a href="#">obsidiandynamics/kafka</a>	Running <a href="#">2181:2181</a>  <a href="#">Show all ports (2)</a>
<input type="checkbox"/>	 <b>flink-jobmanager</b> c3ce4486a221 	<a href="#">flink:latest</a>	Running <a href="#">8081:8081</a> 
<input type="checkbox"/>	 <b>kafdrop</b> 39af59453eff 	<a href="#">obsidiandynamics/kafdrop</a>	Running <a href="#">9000:9000</a> 
<input type="checkbox"/>	 <b>flink-taskmanager</b> 13e38e4ee2be 	<a href="#">flink:latest</a>	Running
<input type="checkbox"/>	 <b>psd-transaction-simulator</b> 511b8328de9c 	<a href="#">xlukzalx/psd-transaction-simulator</a>	Running <a href="#">5162:5162</a> 
<input type="checkbox"/>	 <b>psd-anomaly-consumer</b> f906ee938c46 	<a href="#">xlukzalx/psd-anomaly-consumer</a>	Running <a href="#">5261:5261</a> 
<input type="checkbox"/>	 <b>psd-admin-dashboard</b> 02f1dbad6597 	<a href="#">xlukzalx/psd-admin-dashboard</a>	Running <a href="#">3000:3000</a> 

Rysunek 3: Konfiguracja systemu w Dockerze

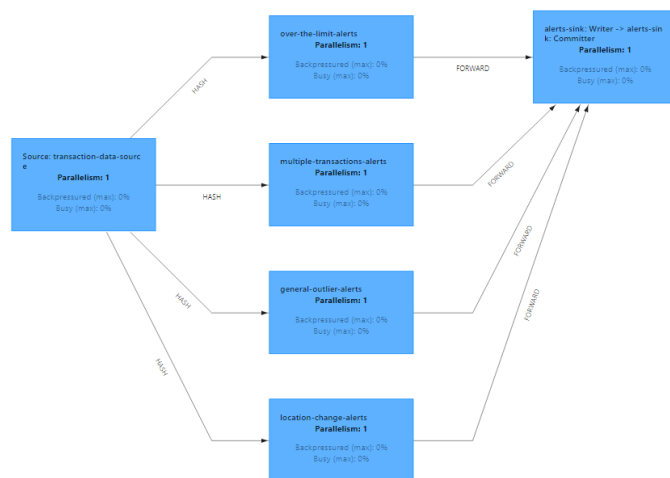
### 4.2 Diagram architektury

Dla systemu stworzony został uproszczony diagram architektury. Ukazuje on zależności między kolejnymi komponentami systemu, przepływ informacji, i jednocześnie konfigurację sieciową od strony serwera i wykorzystanych portów.



Rysunek 4: Uproszczony diagram architektury

W kontekście tego, jak wygląda przetwarzanie danych w Apache Flink, konfiguracja source'ów i sinków względem ustalonych konwencji jest przedstawiona poniżej.



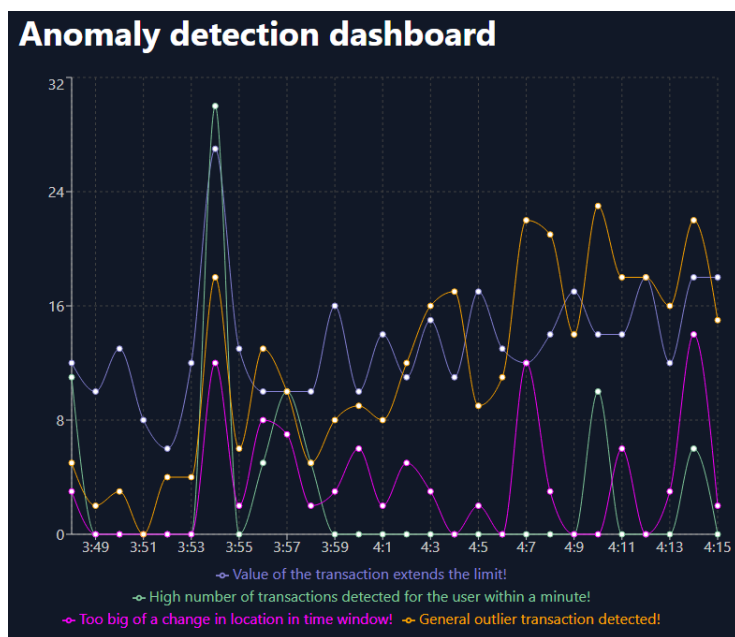
Rysunek 5: Apache Flink

## 5 Testy systemu

Dla stworzonego systemu przeprowadzono testy oraz obserwacje w czasie celem sprawdzenia, czy anomalie są w poprawny sposób wykrywane.

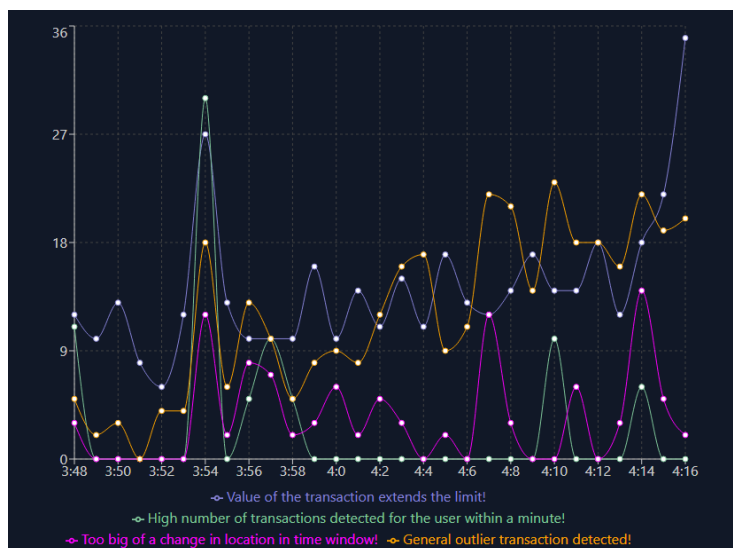
### 5.1 Przekroczenie wartości limitu karty

W przypadku przekroczenia dostępnego limitu na karcie sprawdzenie warunku jest trywialne - jest to zwykle porównanie wartości transakcji oraz wartości limitu karty. Każda transakcja, której wartość przewyższa limit, będzie zakwalifikowana jako anomalia. W tym celu postanowiono użyć endpointa dla symulatora transakcji celem sprawdzenia skuteczności wykrycia anomalii.



Rysunek 6: Wykres liniowy anomalii

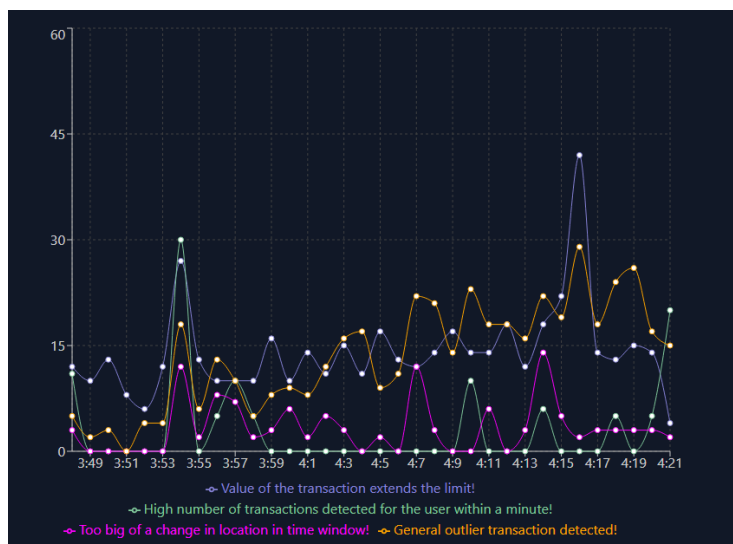
W krótkim czasie wykonano ponad 15 żądań o wygenerowanie anomalii tego typu. Chwilę później można było zobaczyć zmieniony wykres o wykryte anomalie.



Rysunek 7: Wykres liniowy anomalii

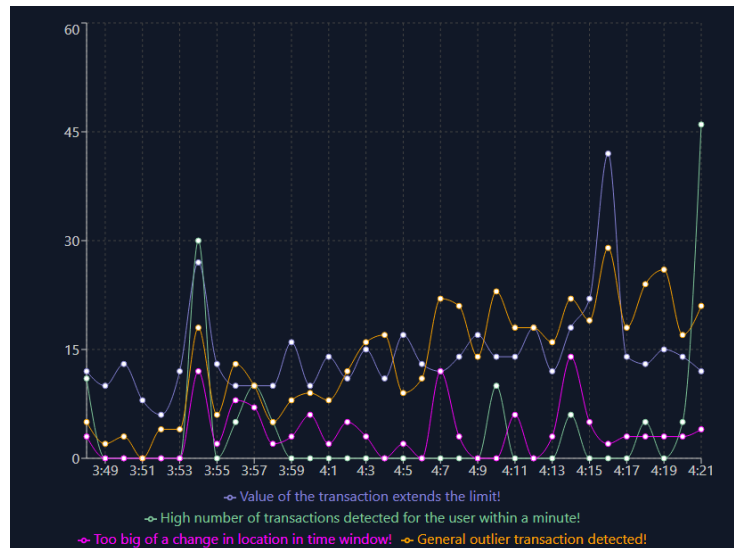
## 5.2 Przekroczenie dopuszczalnej ilości transakcji dla użytkownika

W przypadku przekroczenia dopuszczalnej ilości transakcji mamy tutaj do czynienia z generowaniem dużej ilości transakcji w bardzo małym oknie czasowym. Jako minimalną ilość transakcji uznaną za podstawy do znalezienia anomalii wybrano 5.



Rysunek 8: Wykres liniowy anomalii

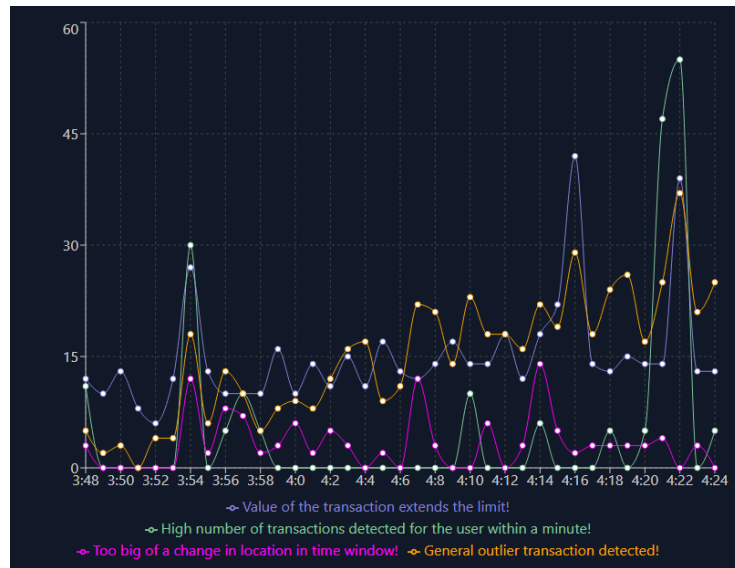
W kontekście własnego generowania anomalii zawsze generowanych jest 10 transakcji. Są one generowane co sekundę i mają reprezentować właśnie dokonanie transakcji w krótkim odstępie czasowym. W tym celu ponownie wykonano wiele żądań, które - jak widać po wykresie - zostały poprawnie zakwalifikowane jako anomalie tego typu. Przy generowaniu tej anomalii jest założenie, ich te transakcje są dokonywane w tej samej lokalizacji - ma to na celu odróżnienie tej anomalii od anomalii związanej ze zbyt dużą zmianą położenia w krótkim oknie czasowym.



Rysunek 9: Wykres liniowy anomalii

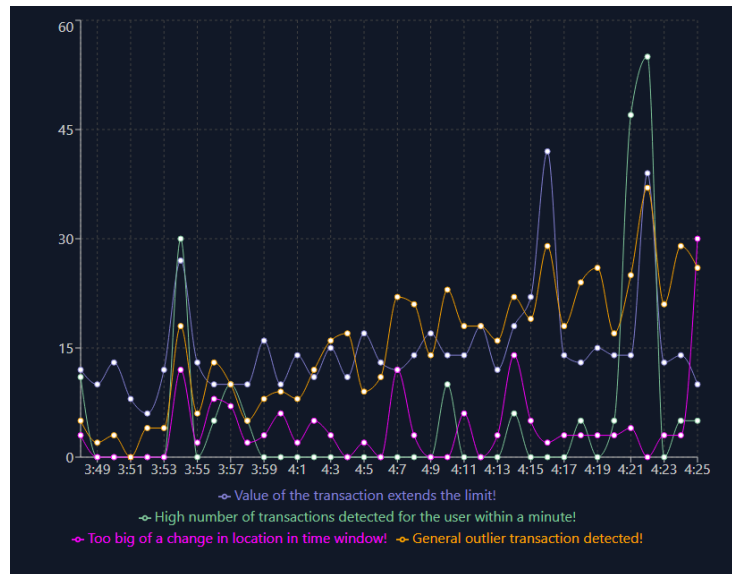
### 5.3 Zbyt duża zmiana położenia w oknie czasowym

W tym przypadku mamy do czynienia z anomaliami powodowanymi tym, iż transakcje dla danego użytkownika mogą czasem być przeprowadzone w odbiegających od siebie miejscach i być dokonane przy tym w krótkim odstępie czasowym. Anomalie te są ukazane na wykresie kolorem fioletowym.



Rysunek 10: Wykres liniowy anomalii

W tym przypadku również postanowiono przetestować działanie detektora przez wykonanie żądań celem wygenerowania anomalii. Po wykonaniu żądań wykres liniowy mocno poszedł do góry, co dowodzi poprawnemu wykrywaniu tej anomalii.

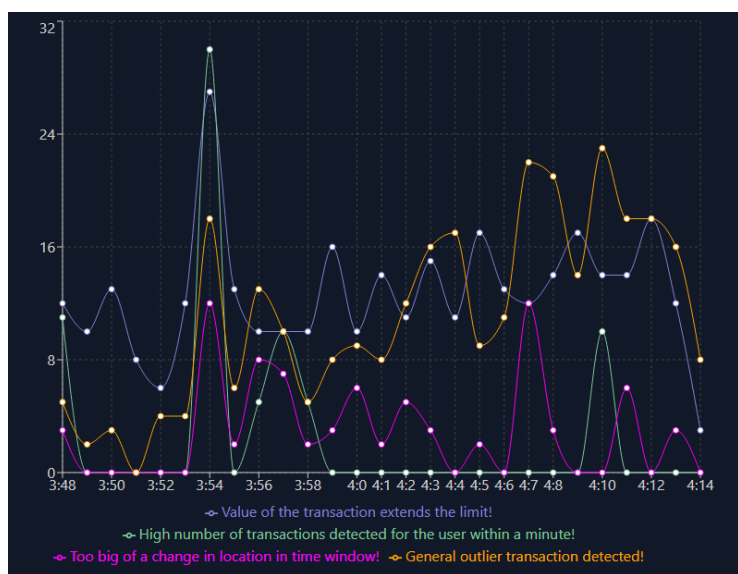


Rysunek 11: Wykres liniowy anomalii

## 5.4 Wykrycie anomalii względem odchylenia standardowego

Odnośnie odkrywania anomalii wykrytych przy użyciu danych statystycznych takich jak odchylenie standardowe, nie ma konkretnej dedykowanej metody do generowania anomalii tylko i wyłącznie tego typu. Jednakże warto zauważyć na wykresie, iż bardzo często zwiększenie się ilości tak wykrytych anomalii było powiązanie z wygenerowaniem anomalii pozostałego typu. Można tym samym dojść do wniosku, iż ta metoda detekcji anomalii również wypełnia swoje zadanie.





Rysunek 12: Wykres liniowy anomalii