МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет _____Комп'ютерних наук_____
(повна назва)

Кафедра_____Програмної інженерії_____
(повна назва)

# КВАЛІФІКАЦІЙНА РОБОТА
## Пояснювальна записка

рівень вищої освіти - другий (магістерський)

**Дослідження методів та засобів аналізу аномальної поведінки мікросервісів**
(тема)

Виконав: Випускник 2 курсу, групи ІПЗм-19-3___

_____Федірко М.А._____
(прізвище, ініціали)

спеціальності 121- Інженерія програмного забезпечення
(код і повна назва спеціальності)

____Освітньо-наукової програми_
(тип програми)

___Інженерія програмного забезпечення___
(повна назва освітньої програми)

Керівник _____проф. каф. ПІ Єрохін А. Л.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____ З.В.Дудар

2021 р.

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет_Комп'ютерних наук_____

Кафедра_Програмної інженерії_____

Рівень вищої освіти - другий (магістерський)

Спеціальність _121-Інженерія програмного забезпечення_____

Тип програми _освітньо-наукова програма_____

Освітня програма_ Інженерія програмного забезпечення_____

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«26»березня 2021 р.

# ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові _Федірку Михайлу Андрійовичу_____

1. Тема роботи Дослідження методів та засобів аномальної поведінки мікросервісів
затверджена наказом університету від "26" березня 2021 р № 385 Ст
2. Термін подання студентом роботи до екзаменаційної комісії
14 травня 2021 р.
3. Вихідні дані до роботи *методи аналізу аномальної поведінки мікросервісів, Python, Process Mining, Visual Studio Code, Databases*
4. Перелік питань, що потрібно опрацювати в роботі *мета завдання, аналіз предметної галузі, постановка задачі, огляд методів та засобів Process Mining, існуючі рішення та бібліотеки*
5. Перелік графічного матеріалу із зазначенням креслеників, схем, слайдів, ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) мета дослідження, обґрунтування доцільності дослідження, підходи до аналізу

процесів, постановка задач дослідження, використані технології, формування набору даних для аналізу, порівняння результатів експериментів, програмна система, висновки _

5 Консультанти розділів роботи

| Найменування розділу | Консультант (посада, прізвище, ім'я, по батькові) | Позначка консультанта про виконання розділу | |
|---|---|---|---|
| | | підпис | дата |
| Спецчастина | Єрохін А.Л. | | |

**КАЛЕНДАРНИЙ ПЛАН**

| № | Назва етапів роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Аналіз предметної галузі | 06.01.2021 | Виконано |
| 2 | Постановка задачі і цілі дослідження | 10.01.2021 | Виконано |
| 3 | Постановка задачі | 11.01.2021 | Виконано |
| 4 | Формування вимог для експериментальної системи | 11.03.2021 | Виконано |
| 5 | Планування експериментальної частини дослідження | 10.04.2021 | Виконано |
| 6 | Проведення експерименту | 17.04.2021 | Виконано |
| 7 | Підготовка пояснювальної записки | 19.04.2021 | Виконано |
| 8 | Нормоконтроль, рецензування | 12.05.2021 | Виконано |
| 9 | Занесення диплома в електронний архів | 14.05.2021 | Виконано |
| 10 | Попередній захист | 15.05.2021 | Виконано |
| 11 | Допуск до захисту у зав. кафедри | 16.05.2021 | Виконано |

Дата видачі завдання __25 січня____2021 р.
Студент _____
(підпис)
Керівник роботи_____ проф., д.т.н. Єрохін А.Л.__
(підпис) (посада, прізвище, ініціали)

# РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота магістра містить: 36 с., 20 рис., 16 дж., 3 табл.

Об’єктом дослідження є методи та засоби аналізу аномальної поведінки мікросервісів.

Метою роботи є пристосування існуючих підходів з аналізу даних для роботи у режимі реального часу.

Методи аналізу базуються на алгоритмах Process Mining.

У результаті роботи здійснено планування експерименту для перевірки гіпотези.

АРХІТЕКТУРА, СЕРВІСИ, ПОДІЇ, ЖУРНАЛ ПОДІЙ, PROCESS MINING, CQRS.

The object of research is the methods and tools for analyzing the abnormal behavior of microservices.

The aim of the work is to adapt existing approaches in data analysis to work in real time manner.

Analysis methods are based on Process Mining algorithms.

As a result, the experiment was planned to test the hypothesis.

ARCHITECTURE, EVENTS, EVENT LOG, PROCESS MINING, CQRS.

Я, Федірко Михайло Андрійович, студент групи ІПЗм-19-3, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів генерації серверного коду у хмарі», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання. Я ознайомлений(а) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

# ПЕРЕЛІК СКОРОЧЕНЬ

API – Application Programming Interface;

ПЗ – Програмне забезпечення;

HTTP – Hypertext Transfer Protocol.

# ЗМІСТ

# ВСТУП

Сьогодні найбільш популярним способом проектувати складні програмні системи є сервісно-орієнтовний підхід [1]. Найчастіше впроваджуємою імплементацією є мікросервісна архітектура [2].

Такі підходи допомагають [3] подолати проблеми розподілення навантаження та масштабування систем.

Також, отримати додаткову користь та приріст у продуктивності системи можна отримати за проектування окремих компонентів розподілених систем за принципом Command Query Responsibility Segregation [4] (CQRS). CQRS підхід пропонує такий спосіб організації роботи сервісу, за яким усі операції можуть бути розділені за типом на операції читання та операції-команди, що змінюють стан даних у базі.

Системи побудовані [5] за парадигмою CQRS полегшують імплементацію іншого концепту – Event Sourcing. Концепт Event Sourcing дозволяє зберігати стан даних у базі за допомогою записів подій. Отже, стан системи у будь якій точці у часі, можна відновити за допомогою послідовного застосування записів подій.

Ще одна проблема розподілених систем – це моніторинг. Найпопулярнішим інструментом для моніторингу програмних систем є логи. Та зазвичай цей інструмент не надто допомагає спеціалістам [6] із якості або аналітикам у пошуку проблем та аналізі роботи системи. Вручну дуже важко «перекопати» сотні або навіть тисячі лог-записів, не кажучи вже про аналіз помилок та вузьких місць у системі. Зазвичай ці записи не приносять жодної користі, а просто лежать похованими на дисках, у разі, якщо одного дня хтось захоче подивитися їх.

Існує цілий розділ Data Science [7] присвячений видобуванню корисної інформації з використанням логів. Це розділ називається Process Mining. Process Mining використовує певні алгоритми та техніки, що можуть екстрагувати певну корисну інформацію про роботу системи, її поведінку та ефективність.

Певні підходи Process Mining дозволяють за допомогою логів побудувати модель процесів у системі, а також знаходити розходження між моделлю та фактичною поведінкою, порівнюючи логи, що продукує система із визначеною моделлю.

Проте є і певні недоліки у існуючих підходах та інструментах. Зазвичай, інструменти для із Process Mining працюють із даними постфактум, тобто спочатку система працює та накопичує логи а потім робота системи зупиняється і ці записи використовуються спеціальним ПЗ для побудування моделі поведінки та аналізу. Такий підхід унеможливлює аналіз у розподілених системах тому що не завжди можливо зібрати дані всіх процесів та сценаріїв поведінки системи. Деякі події можуть бути дуже рідкими, та й не завжди є можливість зупинити систему, якщо вже накопичено достатню кількість логів для використання інструментів Process mining.

Отже постає проблема аналізу поведінки системи та роботи її сценаріїв у режимі онлайн. Такий підхід дозволив би виявляти проблеми та проводити валідацію поведінки системи «на ходу» що може значно допомогти покращувати якість системи.

Зазвичай, у розподілених системах немає особливих вимог до структури логів, тож більшість записів можуть бути просто непридатними для аналізу. Проте із зростом популярності мікросервісної архітектури, також активно впроваджують принципи CQRS та Event Sourcing. Такі системи зберігають дані у вигляді подібному до структурованих логів, що зазвичай містять всю необхідну інформацію.

Метою дослідження є можливість впровадження методів Process Mining для аналізу поведінки розподілених систем за допомогою експерименту.

# 1 **АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ**

## 1.1 Мікросервісна архітектура

Мікросервісний підхід викликав низку змін подходів до архітектури систем у сфері корпоративних додатків. Раніше, вирощені гігантські, монолітні програмні системи досягли межі у можливостях підтримки та масштабованості. Мікросервісний архітектура використовує сервісно-орієнтовний підхід разом із найкращими практиками новітніх розробок у сфері віртуалізації та контейнеризації для того, щоб подолати ці обмеження. Мікросервісний підхід пропонує розбиття монолітної програмної системи на набір менших розподілених сервісів, що значно полегшує роботу із підтримки та масштабування системи. Такі сервіси не зв’язані один з одним і використовують відкриті протоколи для комунікації.

## 1.2 Event Sourcing та CQRS

Event Sourcing – це стиль в архітектурі програмного забезпечення, що часто застосовується у сучасних великих корпоративних системах. Часто цей підхід використовують разом із іншим концептом – CQRS, що дещо спрощує проектування інтерфейсу моделей програмних систем
Далі буде дано короткий опис обох.

### 1.2.1 Event Sourcing

Мартін Фаулер підсумовує [8] підхід Event Sourcing як запис усіх змін стану додатку як послідовність подій. А отже, Event Sourcing не застосовує традиційний концепт для модифікації даних, що змінює значення за допомогою їх заміни, як це зазвичай прийнято робити у підході RESTful [9] для реалізації HTTP PUT запиту. Тож такий підхід зберігає усі події зі зміни даних. На перший погляд, це може виглядати як зайві накладні витрати, проте такий підхід має декілька суттєвих переваг. Завдяки збереженню всіх подій, система має інформацію не тільки відносно нагального стану даних у системі, але й усю історію змін.

Використовуючи записи про події, що змінювали стан даних, можна відтворити актуальний стан даних. Для цього потрібно послідовно «перезапустити» послідовність подій. Це також дає можливість відновити будь-який попередній стан системи в визначеній точці часу у минулому.

Такий підхід може бути покращений за допомогою використання окремої бази даних, що покращить підтримку стану системи за зберігання «знімка» даних. Ця база даних може бути у розташована у пам’яті, що пришвидшить роботу, адже всі дані у такій базі можуть бути відновлені за допомогою послідовного застосування змін.

### 1.2.2 CQRS

У програмному забезпеченні принцип розділення на команди (Commands) за запити (Queries) пропагує підхід відділення операцій запиту даних від операцій модифікацій даних. Command Query Responsibility Segregation (CQRS) є конкретною реалізацією даного принципу. Такий підхід значно спрощує проектування систем, завдяки заохоченню впроваджувати прості методи-обробники команд, до яких належать операції створення, видалення та модифікації даних та методи, що обробляють запити до даних, тобто читання даних. Цей підхід

дозволяє використовувати окремі сховища даних для зберігання команд та запитів. Іншими словами, одне сховище використовується для операцій читання даних, а інше – для операцій запису, оновлення та видалення даних. Зв’язок між цими сховищами даних виконується за допомогою синхронізуючого компонента, що також відповідає за узгодженість даних. На відміну від стандартного багатошарового підходу, CQRS використовує розділення одного шару – бізнес-логіки (див. рис. 1.1).

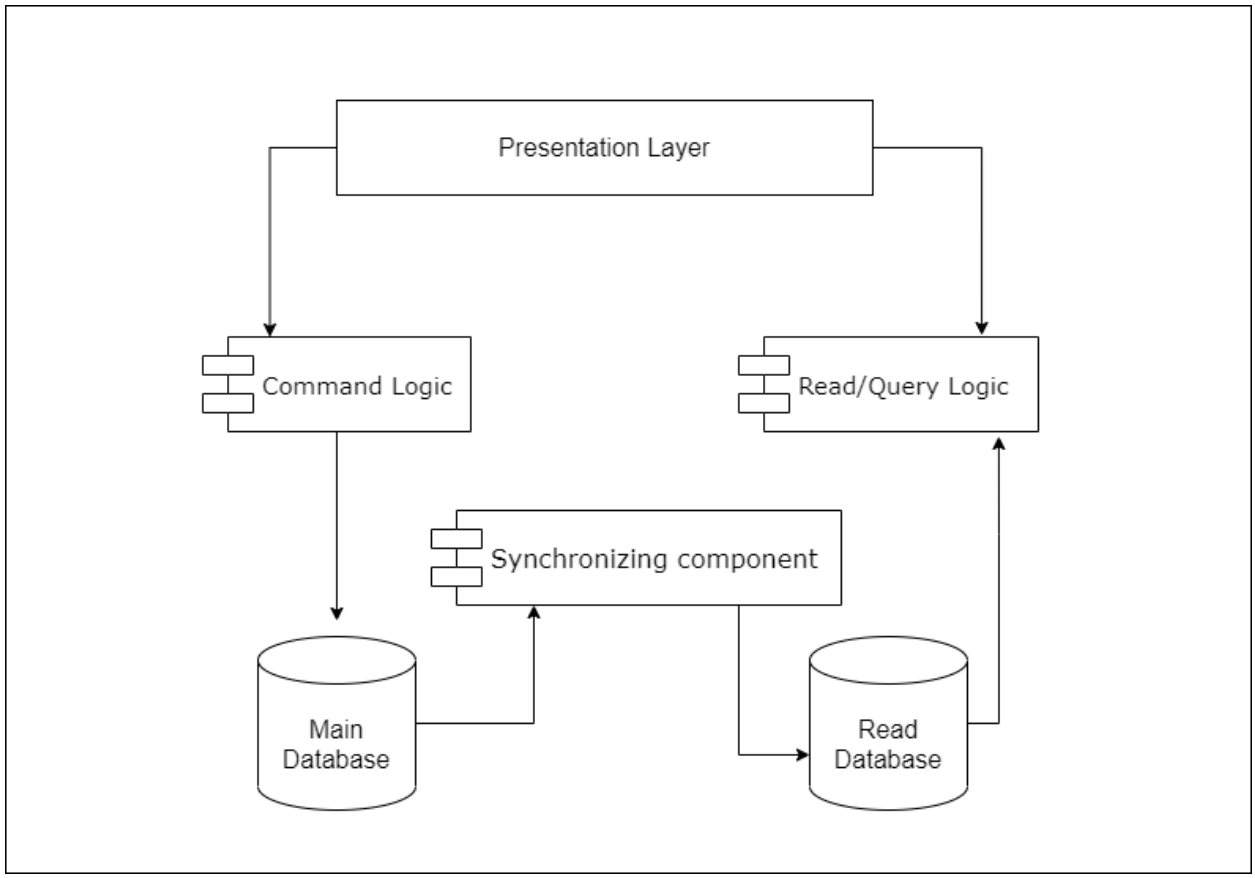


Рисунок 1.1 – Схематичне зображення прикладу CQRS архітектури

Шар презентації залишається незмінним, проте шар бізнес-логіки тепер розділено на два компоненти – Command Logic та Read/Query Logic. Сховище даних використовує дві окремі бази – main та read. Main база поєднана із командами, в той час, коли read база віддає данні до модулю логіки запитів.

Поєднує ці бази – компонент синхронізації, що відповідає за узгодженість даних загалом.

## 1.3 Засоби та підходи до моніторингу у розподілених системах

Із вибуховим ростом популярності розподілених систем, та сервісно-орієнтовного підходу до проектування систем, ми маємо знаходити рішення нових актуальних проблем, яких досі не виникало за використання підходів, до розробки систем-монолітів. Однією такою проблемою є моніторинг програмної системи. На перший погляд це може здивувати, проте більшість сучасних систем покладаються тільки на логи, як на основний засіб моніторингу роботи системи. Зазвичай, логи рідко допомагають при визначенні проблем та пошуку вузьких місць у системи. У більшості випадків, логи великої розподіленої системи складаються із об’ємних змішаних повідомлень із багатьох компонентів системи, шо були написані багатьма незалежними розробниками. Також є деякі підходи до реалізації комплексних систем моніторингу, як наприклад Microsoft Azure Application Insights.

## 1.4 Process Mining

Програмні системи стають дедалі складнішими з кожним днем, та породжують все більше інформації про події, які відбуваються всередині. Тим не менш, організації й досі мають певні труднощі із добуванням корисної інформації із усіх цих даних. Цю проблему допомагають вирішити методи Process Mining.

Process mining можна визначити як міст, який закриває між наукою про дані (Data Science) так моделюванням бізнес-процесів. Тож методи Process Mining використовують логи подій як вхідні дані для автоматичної побудови моделі процесу, перевірки відповідності поведінки системи до визначеної еталонної моделі, пошуку вузьких місць у системі та відхилень, а також – передбачення часу на виконання певних операцій.

Техніки Process Mining застосовують у високотехнологічних системах [10], таких, як, наприклад, апарати комп'ютерної томографії, принтери та сканери, системи оборони, тощо.

Дослідники із IEEE створили [11] маніфест, що має на меті дати вченим розробникам та кінцевим користувачам орієнтир із питань Process Mining.

Організації очікують отримання корисних, точних інсайтів отриманих за допомогою таких технік. Проте, це не завжди можливо, адже для методики Process Mining висувають високі вимоги до якості логів та їх структури. Як вже було зазначено раніше, не всі розподілені системи можуть надати структуровані за єдиним стилем записи подій у системі. У більшості систем якість логів на дуже низькому рівні. Дуже часто вони можуть навіть не відображати дійсність або й зовсім не мати деяких записів про певні події.

### 1.4.1 Попередні етапи Process Mining

Відправною точкою для Process Mining є *подія* та *журнал подій (event log)*. Журнал подій фіксує прояви подій, що були виконані конкретним екземпляром процесу. Також, процес ще називають *case*. Кожна подія має відношення до єдиного процесу і може відноситися до активності (activity) або завдання (task). Події які належать до одного процесу мають бути впорядковані (зазвичай у хронологічному порядку). Запис події також може мати додаткову інформацію

таку, як час, тип транзакції, ресурс, тощо. Така інформація як дата та час, коли відбулася подія, допомагає аналізувати аспекти процесів пов’язані із ефективністю роботи. Ці додаткові властивості прийнято називати атрибутами. Тож підсумовуючи, можна виділити такі вимоги до записів подій:

- журнал подій відображає інформацію про роботу процесу;
- журнал подій містить екземпляр процесу;
- кожен процес складається із впорядкованого списку подій;
- кожен процес може мати атрибут-ідентифікатор;
- кожна подія може відноситися тільки до одного і тільки до одного конкретного процесу;
- подія може мати таки атрибути як активність, час, ресурс, тощо.

### 1.4.2 Виявлення процесів

Виявлення процесів (Process Discovery) – це один із найскладніших етапів у Process Mining. Базуючись на журналі подій, модель процесу будується за спостереженнями за поведінкою процесу. Одним із найбільш придатних інструментів для цього є альфа-алгоритм [12]. Даний алгоритм дозволяє розправитись із паралельністю роботи системи.

### 1.4.3 Моделі процесів

Існує декілька видів нотацій для моделювання процесів із використанням Process Mining підходів.

Найбільш популярними нотаціями є

- Transition Systems;
- Мережі Петрі (Petri Nets);
- Workflow Nets;
- YAWL;
- Business Process Modeling Notation (BPMN);
- Causal Nets.

Базовою нотацією для моделювання є такий вид як Transition Systems (або Системи Переходів). Система переходів складається із станів та переходів (див. рис. 1.2).

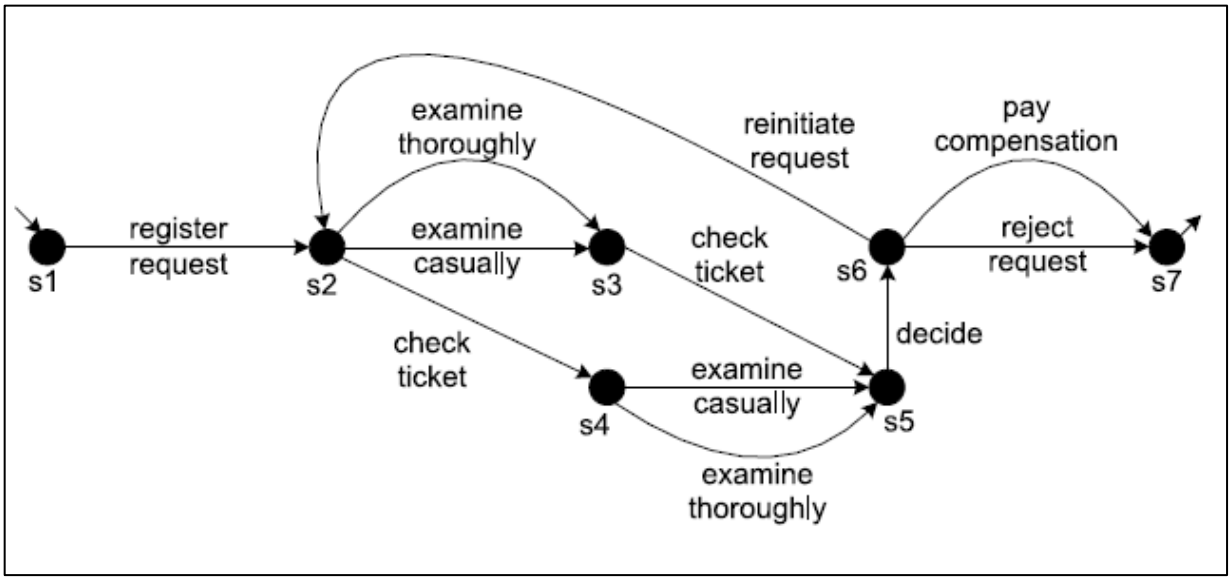


Рисунок 1.2 – Модель процесу у форматі Систем Переходів

Стани системи позначені чорними колами, а переходи поміж станами – стрілками. Кожен стан має унікальне позначення. Це позначення є ідентифікатором стану та не має значення. Значення мають переходи. Переходи поєднують два стани та мають назву активності (activity).

Іншим варіантом для позначення є Мережі Петрі. Це найдавніша та найкраща модель для нотації моделей процесів, що також дає змогу моделювати паралельні процеси. Мережа Петрі являє собою двосторонній граф, що складається із місць (places) та переходів (transitions). Мережа статична, але, керуючись правилом

запуску, токени (tokens) можуть протікати через мережу. Стан мережі Петрі визначається розподілом токенів по місцях і називається позначенням (marking) (див. рис. 1.3). Початкове позначення має тільки один токен у місці «start», тобто у такому стані є можливим лише перехід із місця start і рівно один раз.

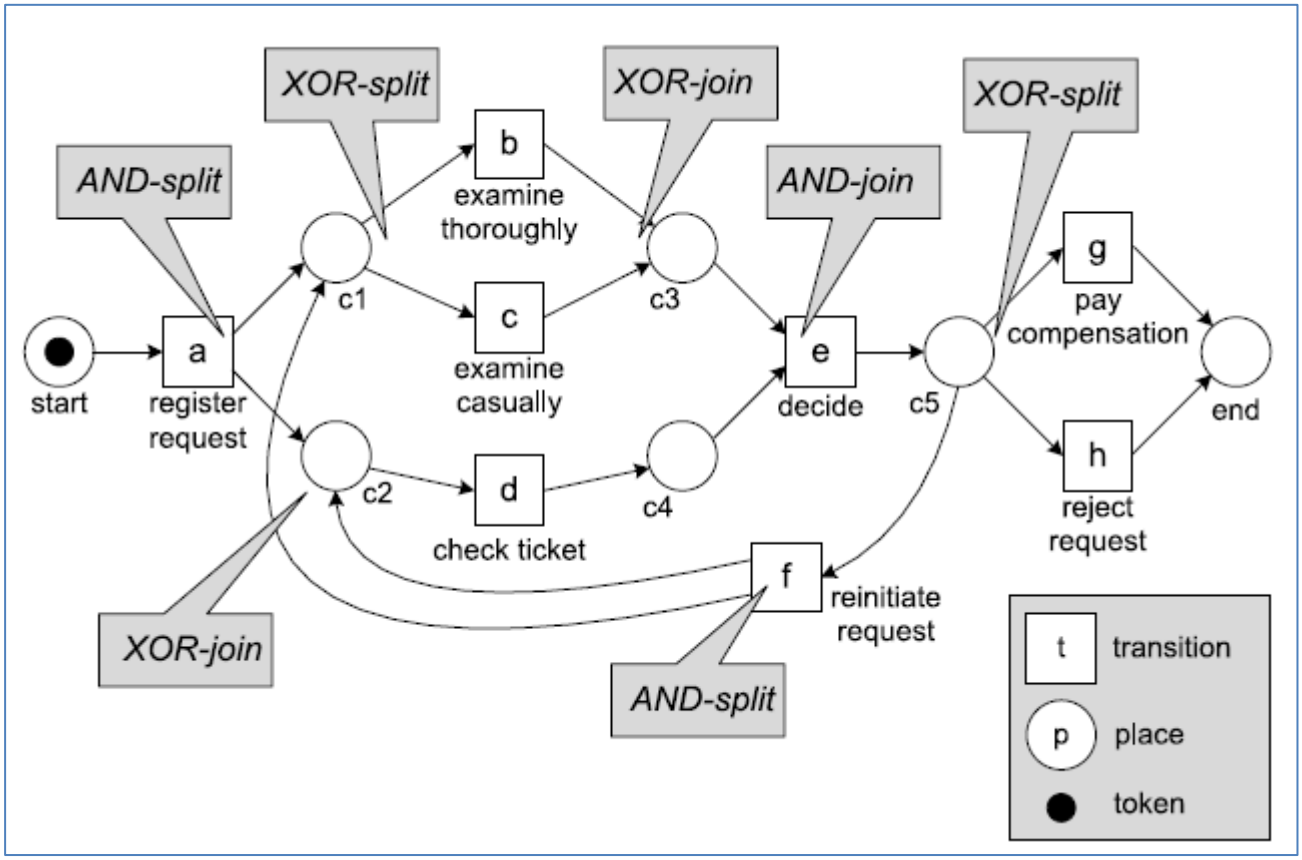


Рисунок 1.3 – Модель процесу у вигляді Мережі Петрі

### 1.4.4 Інструменти Process Mining

Найбільш популярним інструментом для використання технік та алгоритмів Process Mining є ProM Framework [13]. ProM працює із різними типами вхідних даних у форматі XML [14], а також має можливість для розширення функціоналу за допомогою плагінів. Також даний програмний засіб має графічний інтерфейс користувача (див. рис. 1.4).

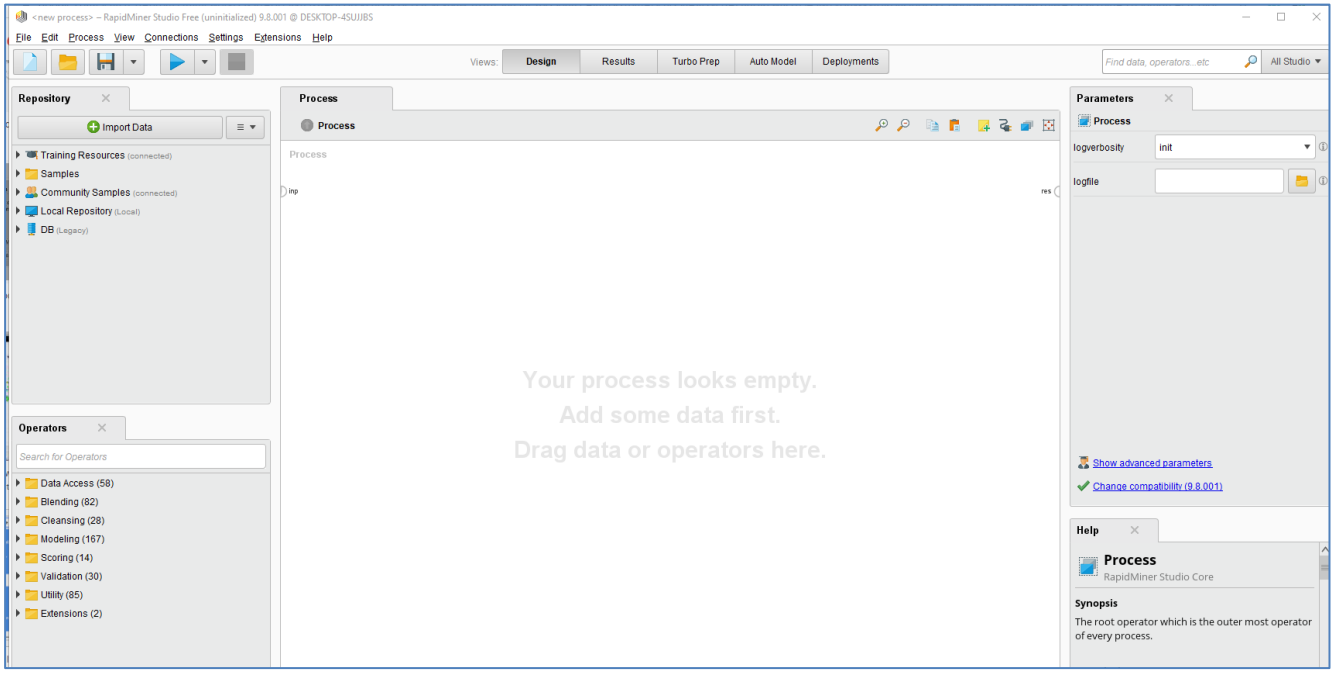


Рисунок 1.4 – Графічний інтерфейс програмного додатку ProM Framework.

## 1.5 Постановка задач дослідження

Отже, для того, щоб мати можливість впроваджувати аналітику з використанням методів Process Mining, необхідно проаналізувати існуючи методи та інструменти для аналізу процесів.

По-перше – необхідно визначити найбільш прийнятну нотацію для опису моделі процесів. Тож до нотацій висуваються такі критерії як простота роботи із нею та можливість моделювання процесів без втрати необхідної точності.

По-друге – необхідно дослідити наявні інструменти для аналізу даних, на можливість їх пристосування для роботи у режимі реального часу. Деякі такі системи мають відкритий вихідний код, що дасть можливість запроваджувати додатковий функціонал. Якщо, ж виявиться, що пристосувати існуючі програмні засоби неможливо, або складність такої задачі недоцільна у рамках даної роботи, то доведеться зробити прототип такого засобу.

Коли буде виявлено або розроблено найбільш придатний програмний засіб для впровадження методів Process Mining, він буде використаний у системі, що симулює певну поведінку.

## 1.6 Експеримент

Для експерименту буде використана існуюча демонстраційна програмна система, шо впроваджує розподілену архітектуру та підходи CQRS та Event Sourcing, адже це дасть нам можливість використовувати записи подій у базі даних як event log. Зазвичай, системи побудовані у стилі CQRS та Event Sourcing мають досі високу якість записів-подій, що дозволить побудувати модель поведінки процесів у середині самої системи. Потім, ця модель буде використана як еталон поведінки експериментальної програмної системи. В експериментальну програмну систему будуть внесені певні зміни, що будуть провокувати помилки у системі та поведінку, шо не відповідає заданій моделі. Потім, за допомогою використання певних засобів автоматизованого тестування, буде симульовано поведінку експериментальної системи. У разі, якщо модифікований програмний засіб, або прототип, зможе у режимі реального часу знаходити помилки або відхилення від очікуваної поведінки, можна буде зробити висновок, що існуючі методи Process Mining можна прилаштувати до використання у режимі реального часу у мікросервісному середовищі.

Такий програмний засіб має відповідати двом критеріям: оброблювати всі події без жодних втрат та виконувати аналіз за прийнятний час, що дозволить використовувати цю систему у режимі реального часу.

# 2 ДОСЛІДЖЕННЯ ТА РОЗРОБКА МЕТОДІВ ВИКОРИСТАННЯ АЛГОРИТМІВ PROCESS MINING У РЕЖИМІ РЕАЛЬНОГО ЧАСУ

## 2.1 Дослідження підходів, до аналізу роботи розподілених систем

Зазвичай, аналіз роботи розподілених систем називають моніторингом. Моніторинг включає декілька наборів технік та підходів, що наведені у таблиці 1.

Таблиця 1 – техніки моніторингу

| № | Назва | приклад |
|---|---|---|
| 1 | Моніторинг продуктивності та якості сервісу | Може відображати так кількісні метрики як кількість HTTP-запитів на секунду, кількість користувачів онлайн, тощо |
| 2 | Системний моніторинг | Відображення завантаженості центрального процесору, вільної пам’яті, тощо |
| 3 | Моніторинг інтеграцій | Перевірка доступності зовнішніх систем, таких як API для інтеграцій, RSS платформ, тощо |
| 4 | Моніторинг безпеки | Моніторинг можливих кібер-атак так контроль доступу до певних елементів систем |

Із наведеної таблиці стає зрозуміло, що повноцінний моніторинг у розподілених системах може включати різні аспекти та механізми впровадження. Деякі постачальники програмного забезпечення мають цілий комплекс рішень для моніторингу розподіленої системи.

До моніторингу у слабо зв’язаних програмних системах можна віднести таки чотири типи активностей, як генерація, обробка, розповсюдження та відображення. Активності з генерації полягають у тому, щоб важливі події були виявлені та були згенеровані відповідні звіти. Такі звіти використовуються для побудування артефактів, що відображають історичні данні роботи системи. Обробні активності мають надавати базовий функціонал для обробки історичних даних. Прикладом такого функціоналу може бути злиття історичних даних, перевірка, фільтрація,

тощо. Такі активності перетворюють сирі та низькорівневі дані у необхідний формат. Процеси розповсюдження у системах моніторингу відповідають за надання таких даних особам що потребують їх. Наприклад, розробникам, аналітикам, інженерам із якості програмного забезпечення, тощо. Процеси пов’язані із активностями відображення, мають на меті відображення зібраних раніше та оброблених даних користувачам у належній формі.

## 2.2 Перевірка гіпотези

Дана робота передбачає проведення експерименту, за результатами якого можна буде зробити висновки, чи дозволяють наявні сучасні інструменти та підходи у сфері Process Mining проводити онлайн-моніторинг роботи сценаріїв використання розподіленої системи.

У рамках даної роботи було висунуто гіпотезу про те, що існуючі інструменти аналізу даних засновані на алгоритмах Process Mining, зокрема – Conformance Checking, можуть надавати корисну інформацію та інсайти роботи системи у режимі реального часу з прийнятною для користувача затримкою.

## 2.3 Архітектура експериментальної системи

У якості основи досліджуємої системи було обрано проект із відкритим вихідним кодом eshop-using-cqrs**.** Варто зазначити, що сама система не є об’єктом дослідження, тому для спрощення роботи, було обрано саме цю програмну реалізацію. Обрана програмна система це прототип інтернет-магазину, що побудований за принципом Service-oriented architecture, та реалізує такі архітектурні підходи до дизайну системи як Command Query Segregation

Responsibility та Event Sourcing. Як зазначалося раніше, засоби Process Mining мають жорсткі вимоги до формату вхідних даних, а системи, що імплементують патерн Event Sourcing, продукують записи придатного формату у базі даних, що містять усі необхідні атрибути для того, щоб бути використаними як вхідні параметри для методів Process Mining.

Вихідний код системи був певним чином дороблений, щоб давати змогу зберігати усі логи в одному сховищі. Такий підхід називається Central Log Storage, або централізоване сховище логів. Даний підхід дозволить використовувати логи, що були раніше записані до сховища, до подання їх у якості вхідних даних для програмного забезпечення, що реалізовує алгоритми Process Mining.

Програмна система складається із декількох мікросервісів, що надають HTTP RESTful API для використання функціоналу системи. Система також має і клієнтську частину, проте ця частина не буде використовуватися в ході експерименту, адже це недоцільно з точки зору використання ресурсів. Можливо, для використання сценаріїв системи, доведеться робити десятки або сотні операцій на хвилину. Для виконання сценаріїв системи буде використано методи автоматизованого API-тестування, що допоможе зекономити час та зусилля, необхідні для того, щоб виконувати ручне тестування.

Граф мікросервісів запропонованої системи можна побачити на рисунку 2.1.

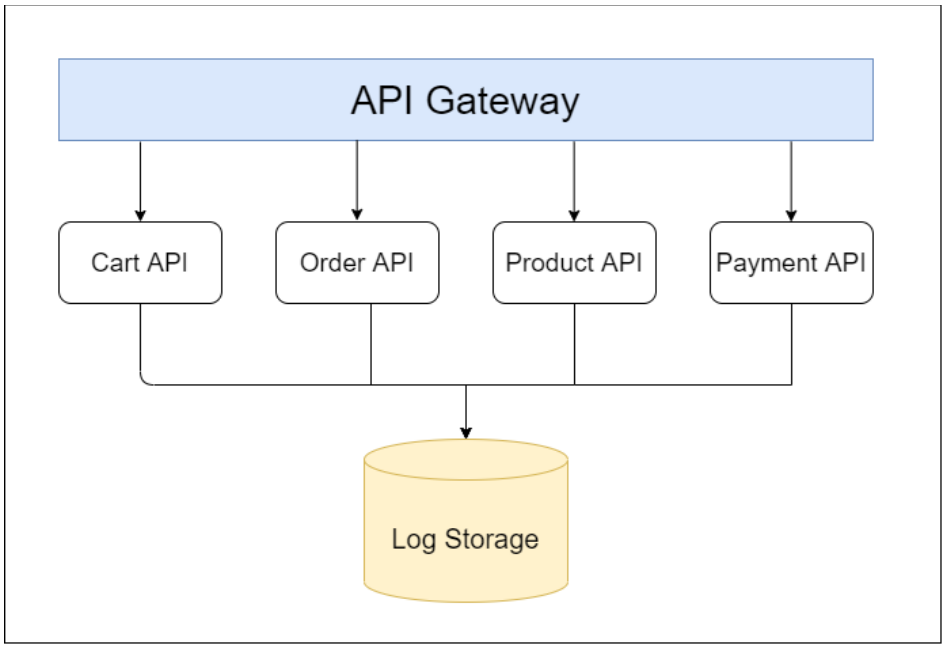


Рисунок 2.1 – Схематичне зображення експериментальної системи

Як видно з рисунку, серверна частина програмного додатку складається із чотирьох *мікросервісов, кожний* з яких має свою зону відповідальності та відправляє записи із подій у сховище логів (Log Storage).

## 2.4 Аналіз існуючих програмних засобів Process Mining

Одним із найбільш популярних засобів, що реалізує методи Data Science (включно із техніками Process Mining) є програмній засіб під назвою ProM Framework Lite. Даний програмний засіб був створений науковцями та дослідниками, що працюють над проблемами науки про данні, та має відкритий вихідний код. Нажаль, ProM Framework Lite надає лише графічний інтерфейс користувача (див. рис. 2.2).

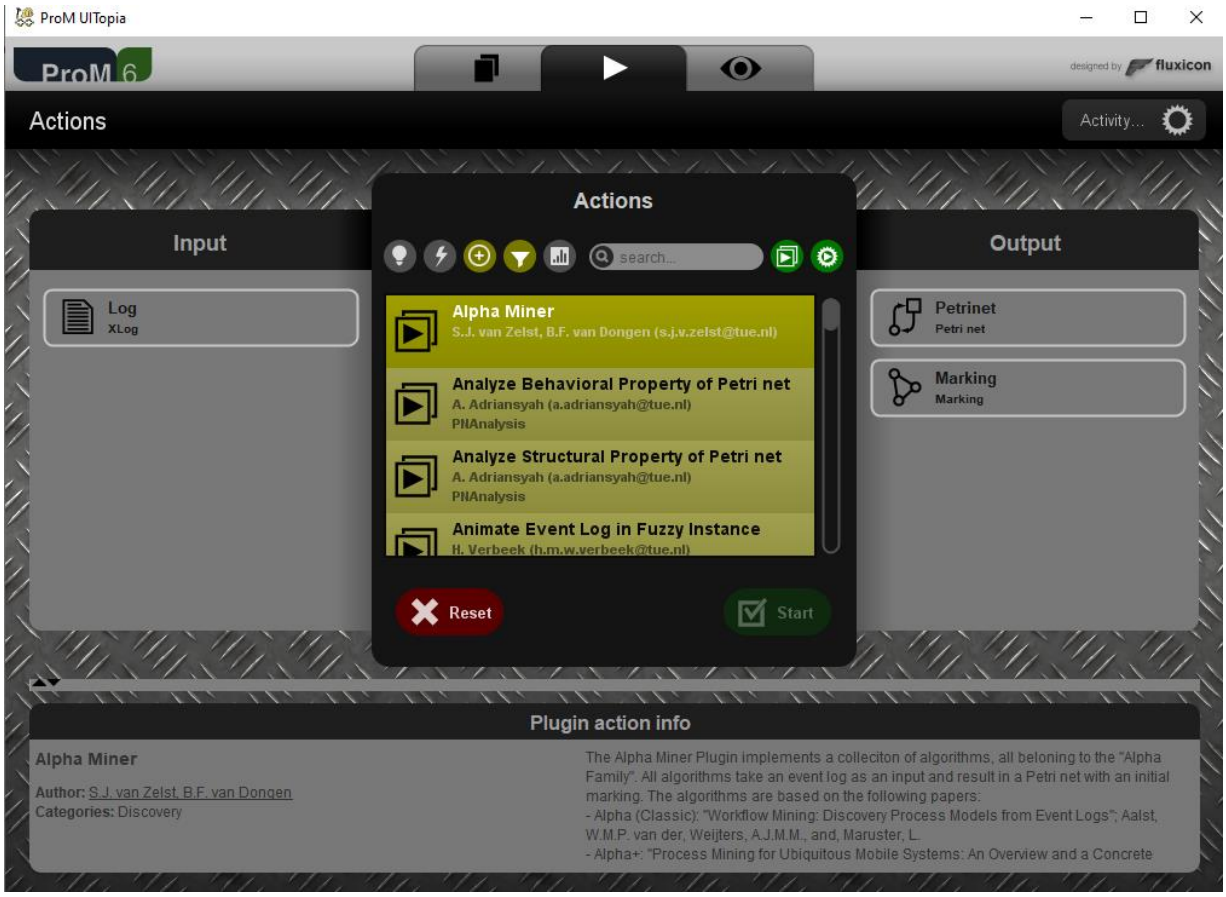


Рисунок 2.2 - Графічний інтерфейс користувача ProM Framework Lite

Нажаль, відсутність програмного інтерфейсу для використання не дає змоги інтегрувати засоби, що були реалізовані у програмному пакеті ProM Framework Lite. Код даного програмного пакету написаний на мові Java та відкритий до редагування. Нажаль, спроби пристосувати вихідний код Prom Framework Like не призвели до успіху через те, що з самого початку програмний код додатку не був спроектований для можливостей подальшого розширення та впровадження нового фунцкіоналу.

Ще одним популярним рішенням для задач із Data Science, зокрема Process Mining є RapidMiner Framework. RapidMiner Framework це комплексний програмний засіб для вирішення задач підготовки та фільтрації даних, машинного навчання, глибокого навчання, text mining та предиктивного аналізу. Даний програмний засіб часто використовується у різних бізнес сферах та для досліджень пов’язаних із аналізом даних.

Інтерфейс програмного засобу RapidMiner Framework наведено нижче (див. рис. 2.3).

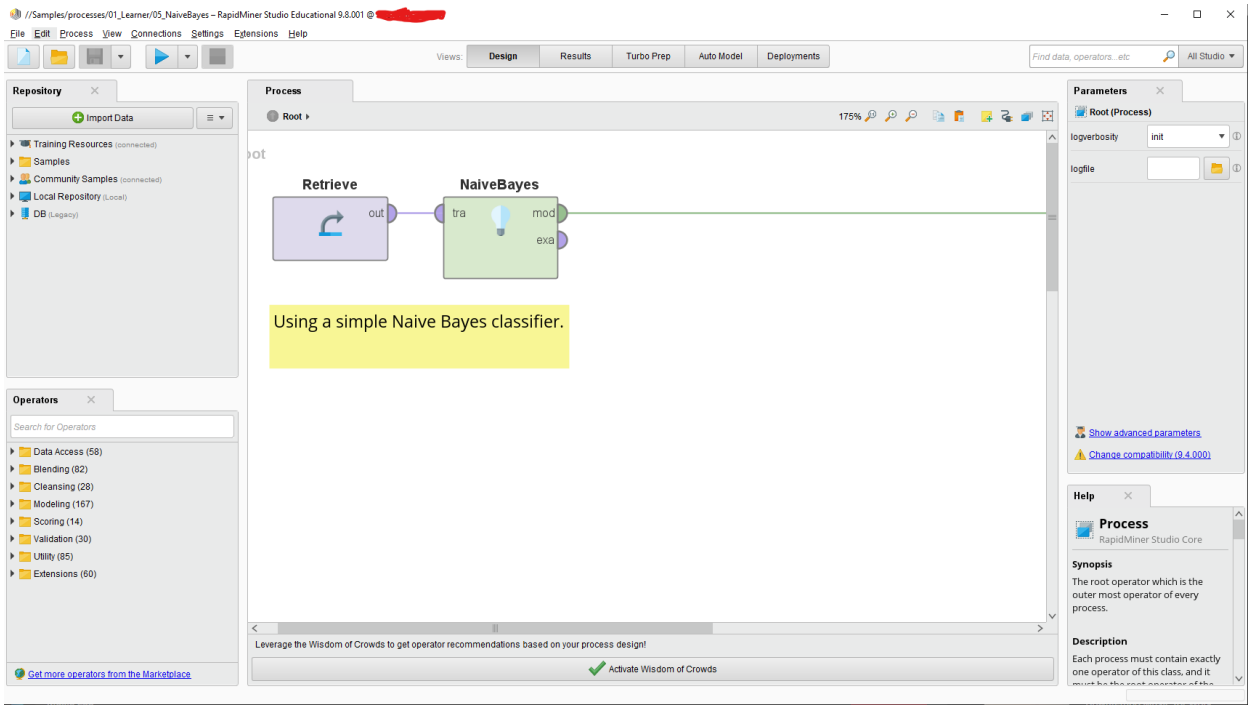


Рисунок 2.3 – Інтерфейс RapidMiner Framework

RapidMiner Framewok так само, як і ProM Framework Lite, написано на мові Java. Даний програмний засіб є безкоштовним для використання в цілях навчання та наукової діяльності, що надає додаткову перевагу у використанні. RapidMiner Framework дозволяє розширювати базові можливості додатку. У якості механізму модифікації пропонується розширення за допомогою плагінів. Документація додатку RapidMiner Framework детально описує способи та прикладні програмні інтерфейси для створення додатків, що надає перевагу RapidMiner Framework перед ProM Framework Lite. Плагіни для розширення можна писати на мові програмування Java та завантажувати їх у репозиторій через спеціальний RapidMiner Marketplace. Marketplace дозволяє встановлювати плагіни інших розробників, та публікувати свої плагіни використання учасниками спільноти RapidMiner. Також, під час роботи було оглянуто декілька плагінів для роботи із алгоритмами Process Mining, проте не було знайдено жодного додатку, який задовольняв би потреби у рамках даної роботи. Також RapidMiner Framework надає програмний інтерфейс розширення за допомогою написання скриптів на мові Python [15].

Недоліком даного засобу є швидкість роботи. RapidMiner Framework розроблюється більше десяти років, тож містить багато залежностей, які потрібно підтримувати, зокрема версію фреймворку .NET 3.5 для використання графічних можливостей додатку на платформі Microsoft Windows. Виконуюча середа (Executing Runtime) використовує багато ресурсів, та дає додаткові затримки, що, зважаючи на вимоги опрацьовувати запити у режимі реального часу, може завадити цілям проекту.

Далі було розглянуто бібліотеку PM4Py [16] (**P**rocess **M**ining for **P**ython), що націлена тільки на використання алгоритмів Process Mining. Бібліотека PM4Py написана на мові Python та надає програмний інтерфейс використання на рівні мови програмування. Бібліотека має відкритий вихідний код та детальну документацію і інструкції з використання. PM4Py надає безкоштовну ліцензію для використання в некомерційних цілях та цілях освіти і виконання досліджень. Також, є окремі ліцензії для комерційного використання. На відміну від RapidMiner та ProM

Framework, PM4Py не потребує встановлення на комп'ютер у якості окремого програмного додатку, а встановлюється як пакет розширення для мови Python за допомогою використання менеджерів пакетів pip.

Мова Python дозволяє швидко проектувати та розробляти програмні додатки, робити прототипи систем, тестувати гіпотези, та змінювати логіку програм. Також, бібліотеку PM4Py легко використовувати для написання клієнт-серверних додатків за допомогою фреймворку Flask**.** На рисунку 2.4 зображено код на мові Python, що наводить приклад використання бібліотеки PM4Py.

```
import os
import pm4py.algo.discovery.inductive import algorithm
import pm4py.objects.log.importer.xes import importer
import pm4py.objects.process_tree.obj import ProcessTree

def execute_script():
    log_path = os.path.join("..", "tests", "input_data",
"running-example.xes")

    log = xes_importer.apply(log_path)
    tree: ProcessTree = inductive.apply_tree(log)
    gviz = pt_vis.apply(tree,
parameters={pt_vis.Variants.WO_DECORATION.value.Parameters.
FORMAT: "svg"})
    pt_vis.view(gviz)

    print("start calculate approximated alignments")
    approx_alignments = align_approx.apply(log, tree)
    pretty_print_alignments(approx_alignments)
```

Рисунок 2.4 – Приклад використання бібліотеки PM4Py

Наведений вище код демонструє функцію, що будує дерево рішень для процесу, логи якого імпортовано за допомогою файлу у форматі XES.

У кожного з оглянутих засобів для Process Mining були свої переваги та недоліки. Для оцінки та вибору рішення, що буде використовуватися у даній роботі, засоби були оцінені на основі декількох критеріїв: наявності якісної документації до програмного додатку, відкритий вихідний код програми,

можливість написання додаткового функціоналу, підтримка мови Python, відсутність застарілих залежностей, швидке виконання алгоритмів, можливість безкоштовного використання. Для наочності подання даних з порівняння можна звернутися до таблиці 2.

Таблиця 2 – Порівняння засобів Process Mining

| | RapidMiner | ProM Framework | PM4Py |
|---|---|---|---|
| Наявність документації | Так | Так | Так |
| Відкритий вихідний код | Так | Так | Так |
| Придатність до модифікації | Так | Ні | Так |
| Підтримка Python | Так | Ні | Так |
| Відсутність застарілих залежностей | Ні | Так | Так |
| Швидке виконання алгоритмів | Ні | Так | Так |
| Можливість безкоштовного використання | Так | Так | Так |

Як можна побачити з таблиці, усі три засоби мають докладну документацію, відкритий вихідний код, та можливість безоплатного використання для дослідницьких цілей. Одразу можна виключити ProM Framework, адже розробники заздалегідь не передбачили механізмів розширення свого додатку. Якщо обирати між RapidMiner та PM4Py, то PM4Py має явну перевагу у швидкості виконання алгоритмів, та не містить застарілих залежностей. Також, PM4Py поставляється у вигляді звичайної Python-бібліотеки, що значно спрощує роботу із засобом. Отже, було вирішено використовувати Python-бібліотеку PM4Py у якості засобу для роботи із алгоритмами Process Mining.

## 2.5 Формати лог-записів

Бібліотека PM4Py дозволяє використовувати вхідні лог-записи для аналізу у двох форматах CSV та XES. XES формат розшифровується як eXtensible Event Format, та є доволі популярним форматом даних, що використовується для аналізу процесів. Приклад вмісту XES файлу зображено на рисунку 2.5.

```
<?xml version="1.0" encoding="utf-8" ?>
<log>
    <extension name="Time" prefix="time" uri="http://www.xes-standard.org/time.xesext" />
    <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.xesext" />
    <string key="origin" value="csv" />
    <trace>
        <string key="concept:name" value="1" />
        <event>
            <string key="concept:name" value="Add product to card" />
            <date key="time:timestamp" value="2021-05-04T20:02:21+00:00" />
            <string key="resource" value="Cart API" />
            <int key="@@index" value="0" />
        </event>
    </trace>
</log>
```

Рисунок 2.5 – Приклад вмісту XES файлу

Наведений зразок коду демонструє запис однієї події у форматі XES. Як можна побачити, даний спосіб запису логів є надлишковим для формату задач у даній роботі. Іншим способом подання логів для бібліотеки PM4Py є формат запису логів, що має назву CSV. На рисунку 3.6 зображено той самий запис у форматі CSV.

```
case_id,activity,timestamp,resource
1,Add product to card,04-May-21 20:02:21,Cart API
```

Рисунок 2.6 – Приклад вмісту CSV файлу

Як можна побачити з рисунку, записи у форматі CSV мають значно менше надлишкового тексту, а отже мають менший розмір, та можуть бути зчитані та записані швидше за файли у форматі XES.

## 2.6 Автоматизація використання системи

Для використання сценаріїв роботи тестової системи були застосовані певні засоби автоматизації. Як було зазначено в попередніх розділах, інтерфейс для взаємодії із тестовою системою являє собою набір RESTful HTTP API. Тобто, взаємодія із тестовою системою відбувається за допомогою HTTP запитів до API. Отже для автоматизації сценаріїв використання тестової системи було створено окрему програму, що за певним сценарієм робить HTTP запити до тестової системи, імітуючи роботу користувача із системою. Програма для автоматизованого тестування написана на мові C# та представляє собою консольний додаток, шо із певною періодичністю робить HTTP виклики з певною послідовністю та затримкою.

## 2.7 Модель процесу

Для того, щоб видобути модель процесу, запустимо автоматизовані сценарії використання системи, для того, щоб накопичити лог-записи.

Для визначення моделі процесів будем використовувати інший набір технік Process Mining – Process Discovery. Алгоритми Process Discovery дозволяють знаходити відповідну модель процесу, що описує послідовність подій та активностей системи, що виконуються під час виконання певного процесу.

Існує декілька алгоритмів із Process Discovery. Найбільш відомими є Alpha, Alpha+, Heuristic, Inductive. Порівняння цих алгоритмів наведено у таблиці 3.

Таблиця 3 – порівняння алгоритмів Process Discovery

| | Alpha | Alpha+ | Heuristic | Inductive |
|---|---|---|---|---|
| Може працювати із «дублікатами» записів | Ні | Ні | Так | Так |
| Може оброблювати цикли довжиною 0 та 1 | Ні | Ні | Так | Так |
| Може працювати із «шумом» | Ні | Ні | Так | Так |
| Гарантує «Sound» модель | Ні | Ні | Ні | Так |

Отже, порівнявши алгоритми Process Discovery, було прийнято рішення використовувати Inductive, алгоритм. Даний алгоритм може знаходити цикли із кількістю ітерацій 0 та 1, які також називають короткими. Inductive алгоритм також дозволяє працювати із «шумом» та гарантує Soundness Model. Модель, яка відповідає твердженню Soundness Model, має задовольняти трьом критеріям: всі частини моделі процесу можуть бути доступними, як тільки процес закінчився, модель процесу не має посилання на сам процес, щоб не сталося із процесом, він завжди може дійти до фінального стану.

Після того, як система попрацювала деякий час, ми можемо із сховища логів експортувати наявні на теперішній час лог-записи до файлу у форматі CSV, так використати їх як вхідні дані для алгоритму Process Discovery, а конкретно – Inductive Process Discovery Algorithm. Для запуску алгоритму використаємо можливості бібліотеки PM4Py. Код, що виконує дану роботу наведено на рисунку 2.7.

```
import pandas as pd
from pm4py.objects.log.util import dataframe_utils
from pm4py.objects.conversion.log import converter as log_converter
import pm4py

def import_from_csv(path, separator=','):
    # Load data from csv
    log_csv = pd.read_csv(path, sep=separator)
    log_csv = dataframe_utils.convert_timestamp_columns_in_df(log_csv)
    # Convert to event log
    event_log = log_converter.apply(log_csv)
    # Discover process tree
    process_tree = pm4py.discover_tree_inductive(event_log)
    bpmn_model = pm4py.convert_to_bpmn(process_tree)
    # View model
    pm4py.view_bpmn(bpmn_model)
```

Рисунок 2.7 – Фрагмент коду програми, що визначає модель процесу

Як можна побачити із фрагменту коду наведеного вище, функція для імпорту лог-файлів у форматі CSV – “import_from_csv”, приймає у якості аргументів шлях до файлу та символ, що розділяє стовпці даних у файлі. Файл CSV завантажується як Pandas Dataframe, та згодом конвертується у лог-формат, з яким працює бібліотека PM4Py. Білбіотека на основі поданого логу визначає дерево процесу, що відповідає визначеній моделі та виводить на екран графічну репрезентацію моделі. Для наочності подання, модель була збережена у форматі PNG. Зображення моделі на ведено на рисунку 2.8.

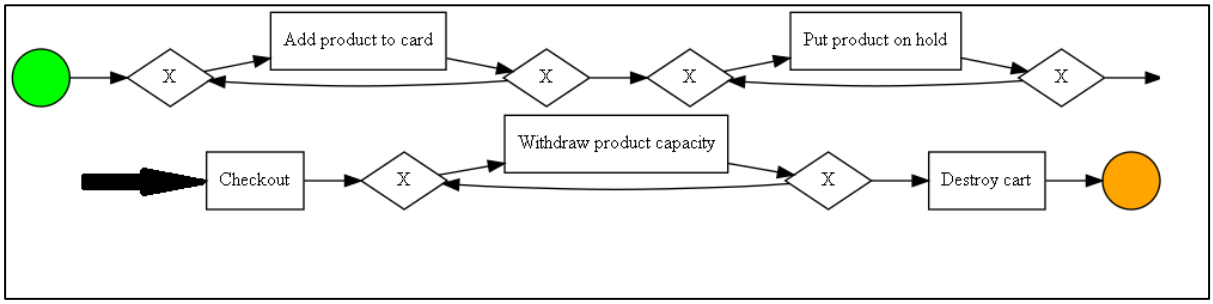


Рисунок 2.8 – Зображення моделі процесу визначеної із логів.

Отримана модель описує процес покупки в інтернет магазині як певну послідовність дій різних мікросервісів системи. Типовий процес складається із таких кроків: додати продукт у кошик (Add product to card), зарезервувати продукт

(Put product on hold), оплата (Checkout), зменшення кількості продуктів (Withdraw product capacity) та видалення кошику (Destroy cart).

Також можна побачити, що кроки Add product to card, Put product on hold та Withdraw product capacity, можуть бути виконані більше одного разу. Це не помилка у рисунку чи чомусь іншому. Так насправді виконується сценарій покупки у тестовій системі. Користувач може додати до кошика один або більше товарів, тому лог-файл може містити декілька записів однакової дії, що в свою чергу призводить до повторного виконання кроків Put product on hold та Withdraw product capacity.

Репрезентацію даної моделі збережемо у форматі XES-файлу та будемо використовувати як еталонну модель сценарію покупки у тестовій системі. Далі ця модель буде використовуватися для аналізу поведінки тестової системи.

## 2.8 Проектування моделі автоматизованого тестування

Як було зазначено у розділі 3.6, для того, щоб проводити виконання певних сценаріїв використання системи, було написано додатковий код на мові C#, який симулює роботу користувача із системою за допомогою HTTP викликів. У розділі 2.7 було наведено етапи роботи системи під час процесу покупки. Тож код автоматизованого тестування робить HTTP виклики, які відповідають цим крокам. Між HTTP викликами код автоматизованого тестування робить паузу у певний проміжок часу таким чином, що весь сценарій покупки проходить від 5 до 130 секунд. Для контролю навантаження на систему було розроблено певний алгоритм контролю автоматизованих сценаріїв використання. Код автоматизації через певні проміжки часу запускає певну кількість паралельних сценаріїв. Для спрощення далі ці проміжки часу будуть названі змінною TimeFrame, Execution Rate per TimeFrame. Графік розподілення часу виконання одного сценарію зображено на рисунку 3.9.

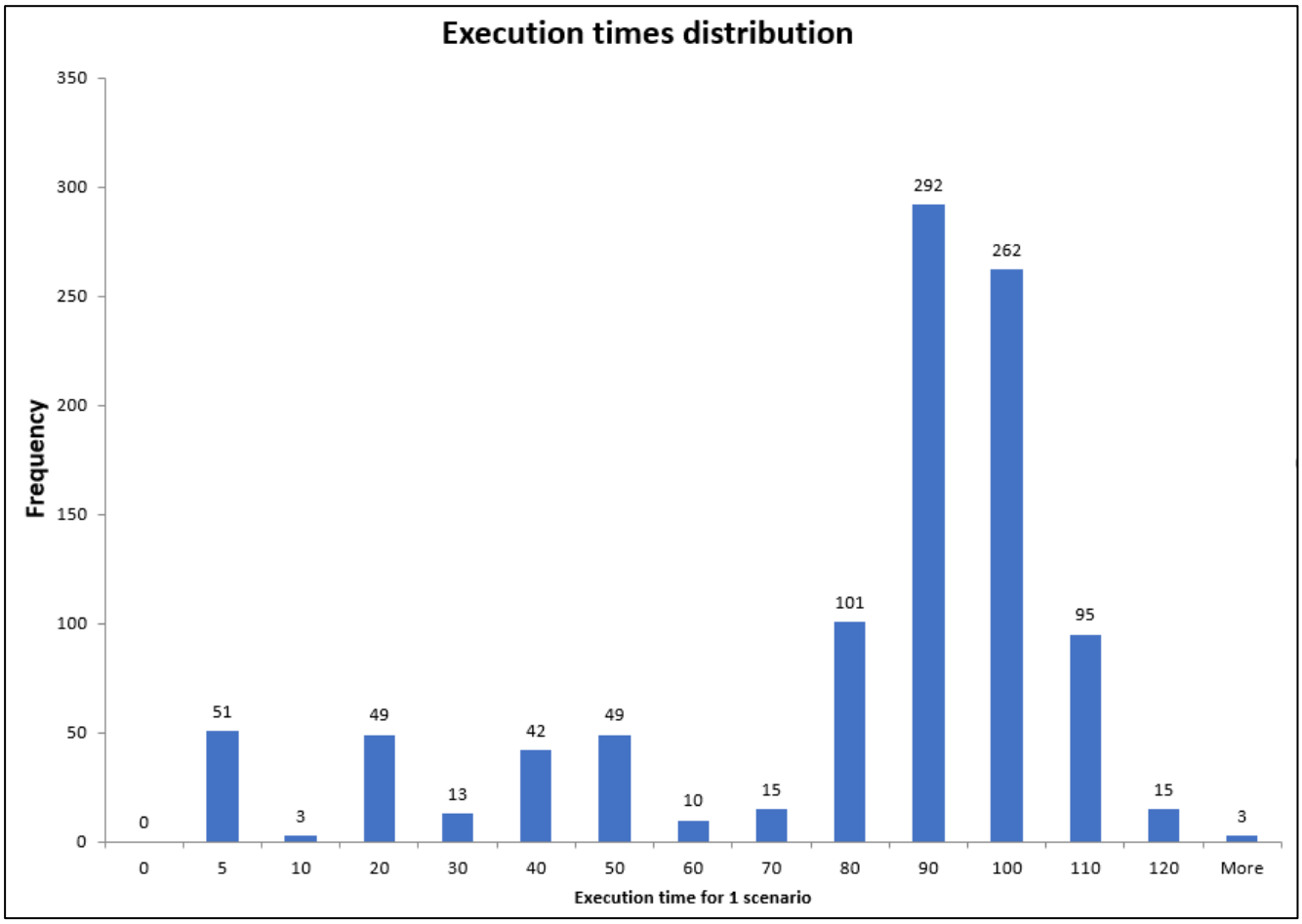


Рисунок 2.9 – Частотна діаграма часу виконання сценарію покупки.

Як зображено на рисунку 2.9, більше половини сценаріїв було виконано за час, що лежить у проміжку від 80 секунд до двох хвилин. Медіанне значення – 86.02 секунди. Для побудування діаграми було використано вибірку в тисячу процесів, що були розпочати під час одного проміжку часу в 30 секунд.

## 2.9 Conformance Checking

Conformance Checking – це ще одна техніка із набору технік Process Mining. Простими словами, алгоритми Conformance Checking порівнюють надані логи подій із заданою моделлю. Ціль роботи таких алгоритмів – перевірити, що логи відповідають моделі, чи навпаки – модель відповідає логам. Найбільш

популярними алгоритмами Conformance Checking є Token-based replay, Decomposition of alignments, Footprints, Log Skeleton.

Для виконання Conformance Checking будемо використовувати TBR (Token-based Replay) алгоритм. TBR алгоритм дозволяє діагностувати такі проблеми: знаходження активностей, що не містяться у моделі, знаходження активностей, що були виконані у спосіб, що не відповідають моделі, неправильний порядок активностей відповідно до моделі.

Бібліотека PM4Py реалізує більшість популярних алгоритмів із Conformance Checking, у тому числі й алгоритм Token-based Replay. Для того, щоб виконати алгоритм Token-based Replay за допомогою бібліотеки PM4Py, необхідно імпортувати модель процесу, імпортувати лог та запустити виконання алгоритму Conformance Checking. Фрагмент коду, що виконує дії описані вище можна побачити на рисунку 2.10.

```
import os
from pm4py.objects.log.importer.xes import importer as xes_importer
from pm4py.algo.discovery.inductive import algorithm as
inductive_miner
from pm4py.algo.filtering.log.auto_filter.auto_filter import
apply_auto_filter
from pm4py.algo.conformance.tokenreplay import algorithm as
token_based_replay


#load log
log = xes_importer.apply("receipt.xes")
filtered_log = apply_auto_filter(log)

#apply process discovery
net, initial_marking, final_marking =
inductive_miner.apply(filtered_log)
parameters_tbr =
{token_based_replay.Variants.TOKEN_REPLAY.value.Parameters.DISABLE_VAR
IANTS: True,
token_based_replay.Variants.TOKEN_REPLAY.value.Parameters.ENABLE_PLTR_
FITNESS: True}

#apply conformance checking
replayed_traces, place_fitness, trans_fitness, unwanted_activities =
token_based_replay.apply(log, net, initial_marking, parameters_tbr)
```

Рисунок 2.10 – Виконання алгоритму Conformance Checking PM4Py

## 2.10 Проектування експерименту

Отже, як зазначено у розділі 3.8, більшість сценаріїв працювали 120 секунд, або менше, тож для змінної TimeFrame у коді автоматизації тестування було вирішено обрати значення у 120 секунд, або дві хвилини. Запуск процесів автоматизованого тестування схематично зображено на рисунку 2.11.

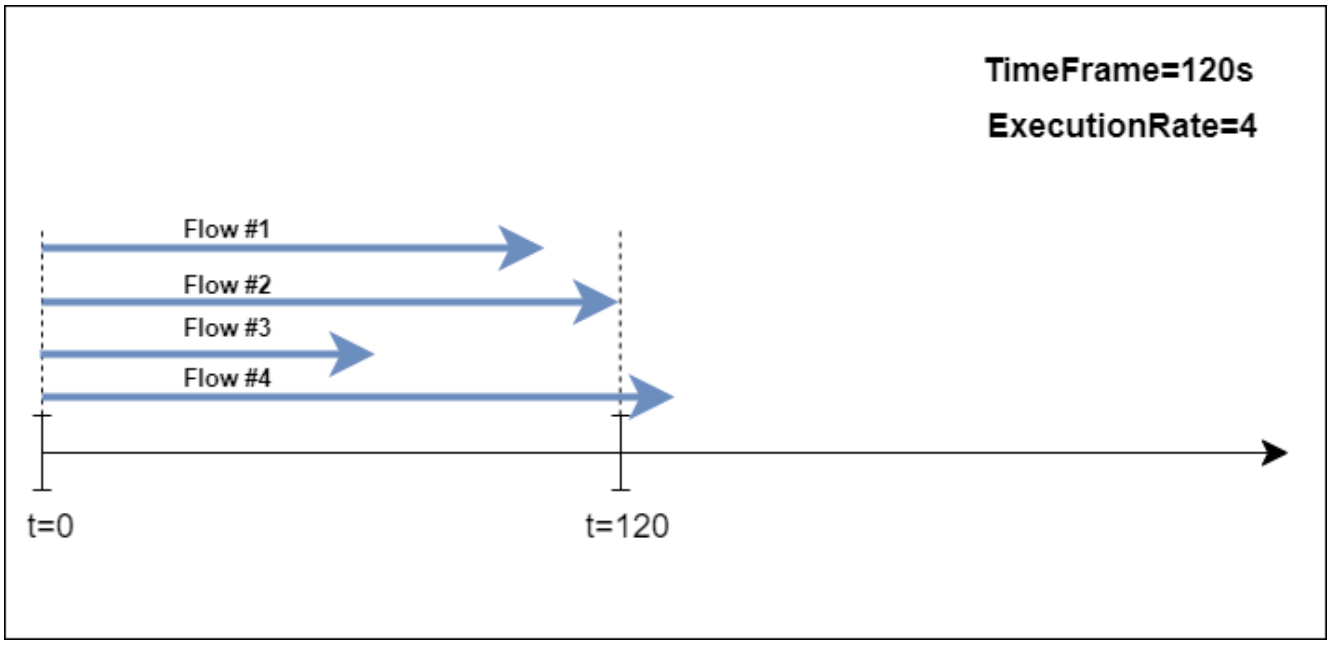


Рисунок 2.11 – Схематичне зображення моделі автоматизованого тестування

На рисунку 2.11 зображено як виконуються сценарії у проміжки часу. У даному випадку, час поділений на проміжки по 120 секунд і за проміжок часу виконується чотири інстанси сценарію. Жоден новий сценарій не буде запущено, доки не закінчиться проміжок часу довжиною визначеною у змінній TimeFrame. Також, можливі випадки, коли сценарій буде виконуватися довший проміжок часу, аніж визначений у змінній TimeFrame. У такому випадку, запуск наступних сценаріїв не буде відкладений до часу завершення сценарію, що виконується більше секунд, аніж визначено у змінній TimeFrame, а буде виконано у свій визначений проміжок часу.

Тестова система буде виконувати певні сценарії поведінки, та продукувати лог-записи під час роботи. Для того, дізнатися, чи можливо виконувати Conformance Checking у режимі реального часу, що є ціллю даної роботи, було створено ще два програмні додатки – Log Exporter та Log Processor. Log Exporter раз на п’ятнадцять секунд буде вивантажувати зі сховища логів записи у форматі CSV та передавати у Log Processor, що буде виконувати алгоритми Conformance Checking та перевіряти подані на вхід лог-файли на відповідність до заданої раніше моделі.

Log Processor представляє собою простий серверний додаток, що оброблює HTTP запити, написаний на мові Python та фреймворку Flask. Log Processor буде використовувати бібліотеку PM4Py для виконання алгоритмів Process Mining, зокрема – Conformance Checking.

Log Exporter це програмний додаток написаний на мові C#, що через певні проміжки часу буде завантажувати дані зі сховища логів, експортувати до файлу у форматі CSV та передавати до Log Processor за допомогою HTTP запиту.

Також, Log Exporter буде вимірювати час необхідний на обробку лог-файлу програмою Log Processor. Послідовно збільшуючи значення ExecutionRate, та виконуючі сценарії автоматизованого тестування, буде досліджена залежність швидкості обробки лог файлу від кількості паралельно запущених сценаріїв.

Звичайно, постійний аналіз лог-файлу не може бути процесом із нульовими затримками у часі, тож визначимо максимально допустимий час обробки в 25 секунд. Отже, якщо додати до цього ще 5 секунд проміжку між запуском Log Exporter, то отримаємо максимальну затримку від реального часу на визначення помилок у 30 секунд, що є допущенням про прийнятний час у рамках даної роботи.

Також, у код тестової системи були внесені певні зміни, що будуть призводити до помилок під час виконання сценарію. У середньому, помилка відбувається у одному з двадцяти сценаріїв та потенційно призводить до відсутності частини лог-записів у відповідному сценарії. Саме ці помилки і мають знайти алгоритми Conformance Checking.

# 3 ПРАКТИЧНА ЧАСТИНА. ПРОВЕДЕННЯ ЕКСПЕРИМЕНТІВ

## 3.1 Перший експеримент

Експеримент був запущений із значенням змінної Execution Rate 1000. Тобто за один проміжок часу виконувалася тисяча сценаріїв покупки. Приблизно, 50 сценаріїв були неуспішними через запрограмовані помилки. Після 717 секунд роботи експерименту, затримка на виконання алгоритму Token-based Replay, вийшла за встановлений максимальний поріг у 30 секунд, та становила 30.05 секунд. Експеримент був запущений напротязі близько 5000 секунд.

Детальніше графік часу витраченого на обробку лог-файлу з плином часу зображено на рисунку 3.1.

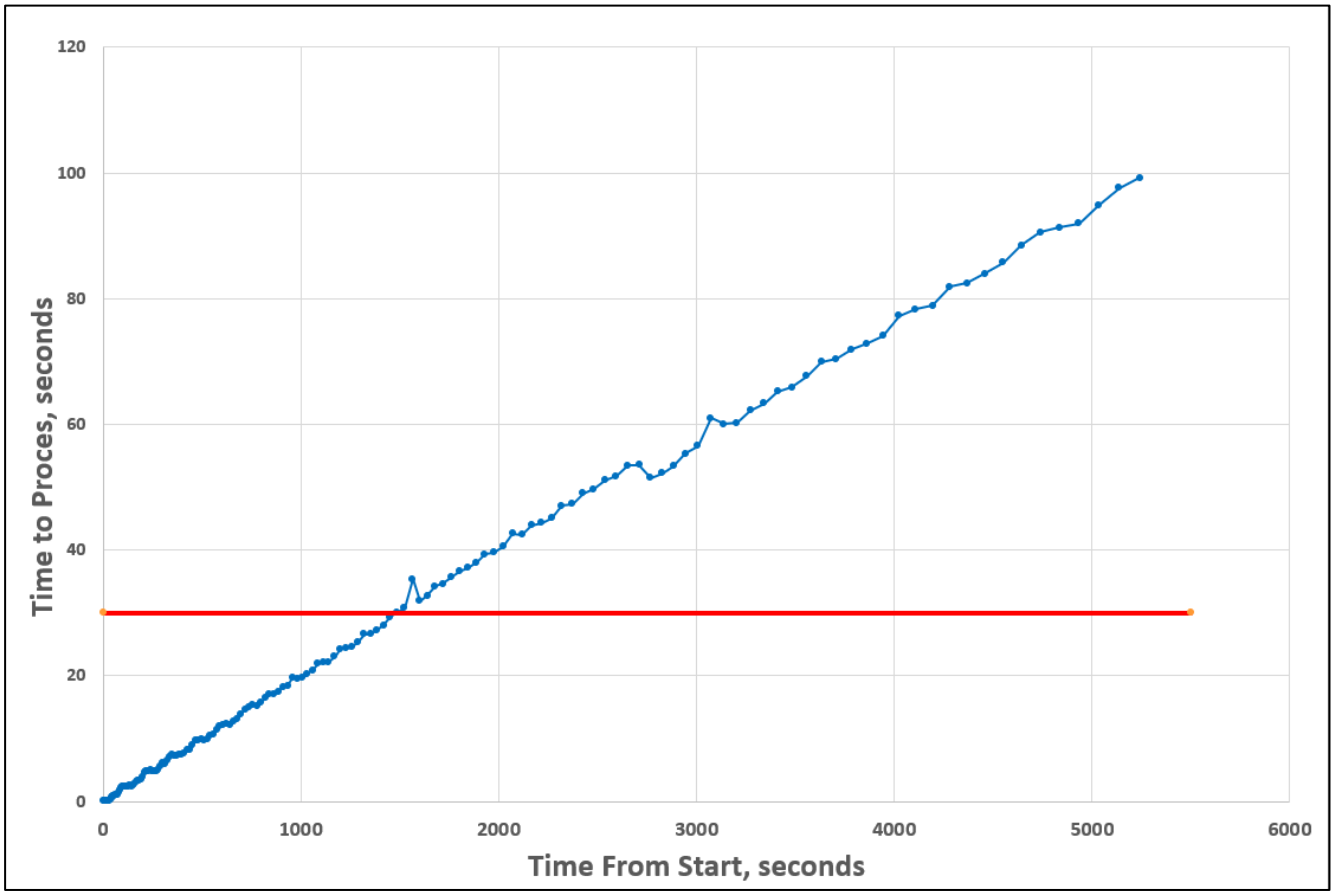


Рисунок 3.1 – Час витрачений на обробку лог-файлу.

Як видно з графіку, приблизно після проміжку в 1500 секунд від початку експерименту, час, необхідний на обробку лог-файлу, що накопичився став перевищувати визначену межу у 30 секунд.

## 3.2 Оптимізація передачі лог-файлу

З метою зменшення часу потрібного на обробку лог-файлу було вирішено оптимізувати передачу лог-файлу до LogProcessor. Щоб зменшити затримки при обробці лог-файлу, змінимо спосіб передачі із HTTP-Upload на вказання шляху до лог-файлу та передачі цього параметру в запиті.

На рисунку 3.2 зображено графік із порівнянням швидкості обробки лог-файлу, використовуючи обидва способи завантаження.

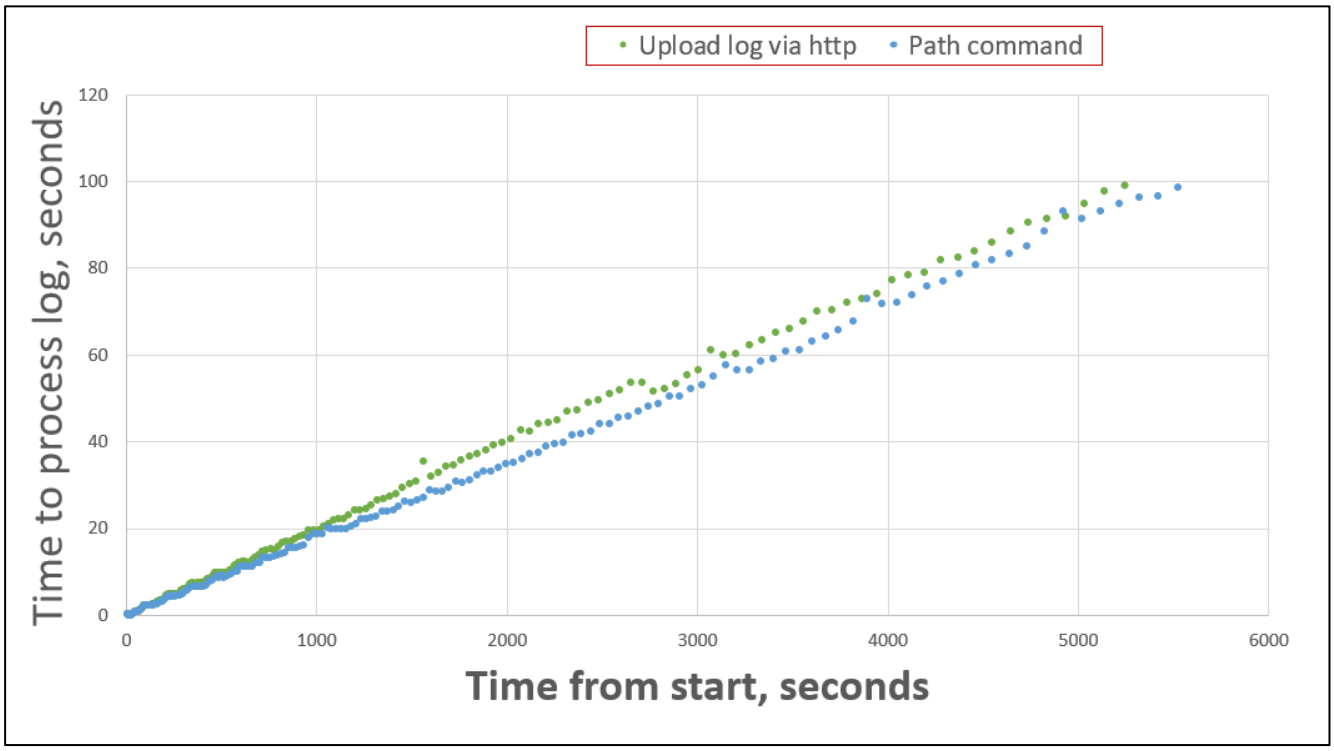


Рисунок 3.2 – Час обробки лог-файлів обома способами.

Зеленими маркерами позначений час обробки лог-файлу, при використанні типу передачі HTTP-Upload, синіми – без безпосереднього завантаження до LogProcessor.

Оптимізація передачі лог-файлу не надала великого покращення, проте така різниця може бути вагомою, коли розмір лог-файлів досягає розміру гігабайтів.

## 3.3 Оптимізація вибірки

Як було зазначено у минулих розділах, показник швидкості обробки, досить швидко перетинає межу в 30 секунд на обробку лог-файлу, а отже, відносно вимог до такої системи, визначених раніше, це заважає роботі у режимі реального часу.

Оптимізація передачі лог-файлу також не дала суттєвого результату, проте все ж показала певні покращення, отже вирішено залишити цей спосіб передачі лог файлу до Log Processor.

Досліджуючи проблему довгої обробки лог-файлів алгоритмами Process Mining, було визначено, що з часом, більшість лог-записів у сховищі логів належать успішно завершеним процесам, отже немає сенсу перевіряти ці сценарії знову і знову, щойно вони успішно завершилися.

Під час кожного запуску Conformance Checking, ми отримуємо інформацію, щодо успішного чи не успішного завершення, отже для того, щоб зменшити об’єм даних для Log Processor, було вирішено зберігати ідентифікатори сценаріїв, що були успішно виконані, та не включати їх до вивантаження у Log Processor.

Щоб перевірити цю гіпотезу, було виконано ще один експеримент. Значення усіх змінних відповідають значенням із минулого експерименту. Спосіб передачі файлів залишився з оптимізацією.

Результати експерименту показали, що така оптимізація дає суттєве покращення. За 5000 секунд експерименту, значення часу використаного на

обробку лог-файлу жодного разу не перевищило визначений поріг у 30 секунд. Графік часу витраченого на обробку файлів зображено на рисунку 3.3.

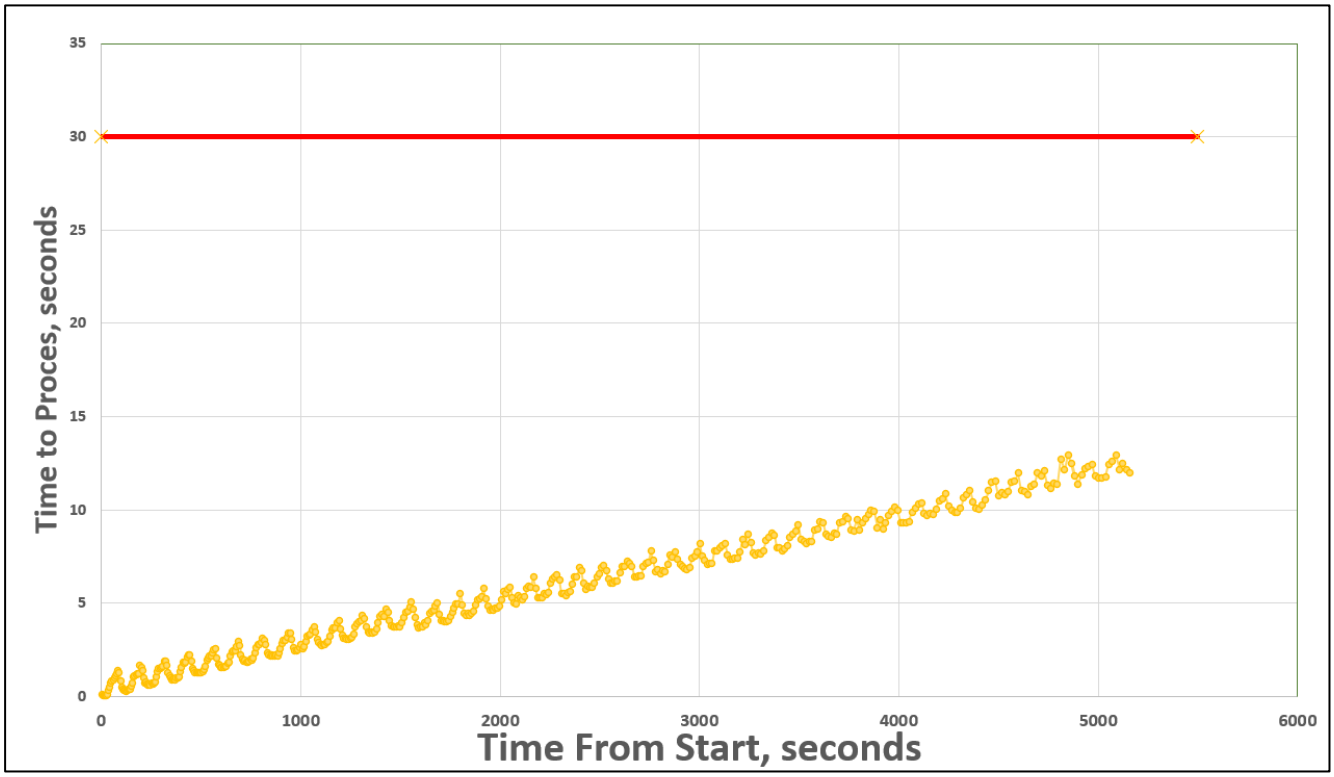


Рисунок 3.3 – Час затрачений на обробку лог-файлу

З графіку видно, що показник часу необхідного для роботи Conformance Checking значно зменшився, і не сягнув порогу в 30 секунд. Проте, все одно можна бачити поступовий зріст цього показника.

## 3.4 Експеримент зі збільшеним навантаженням.

В минулому експерименті, за весь час роботи системи, швидкість не перевищувала визначеного порогу в 30 секунд, проте графік продемонстрував тенденцію для зростання. Можна припустити, що зі збільшенням навантаження,

показник часу необхідного на обробку, перетне межу у 30 секунд за значно менший час від початку експерименту, тож було вирішено збільшити значення ExecutionRate до 10 000. Це фактично означає, шо кожні 120 секунд, будуть запущені 10 000 нових екземплярів сценаріїв виконання.

Вже через 1200 секунд роботи експериментальної системи, час виконання процесу Conformance Checking перетнув відмітку в 30 секунд. Детальніше на графіку на рисунку 3.4.

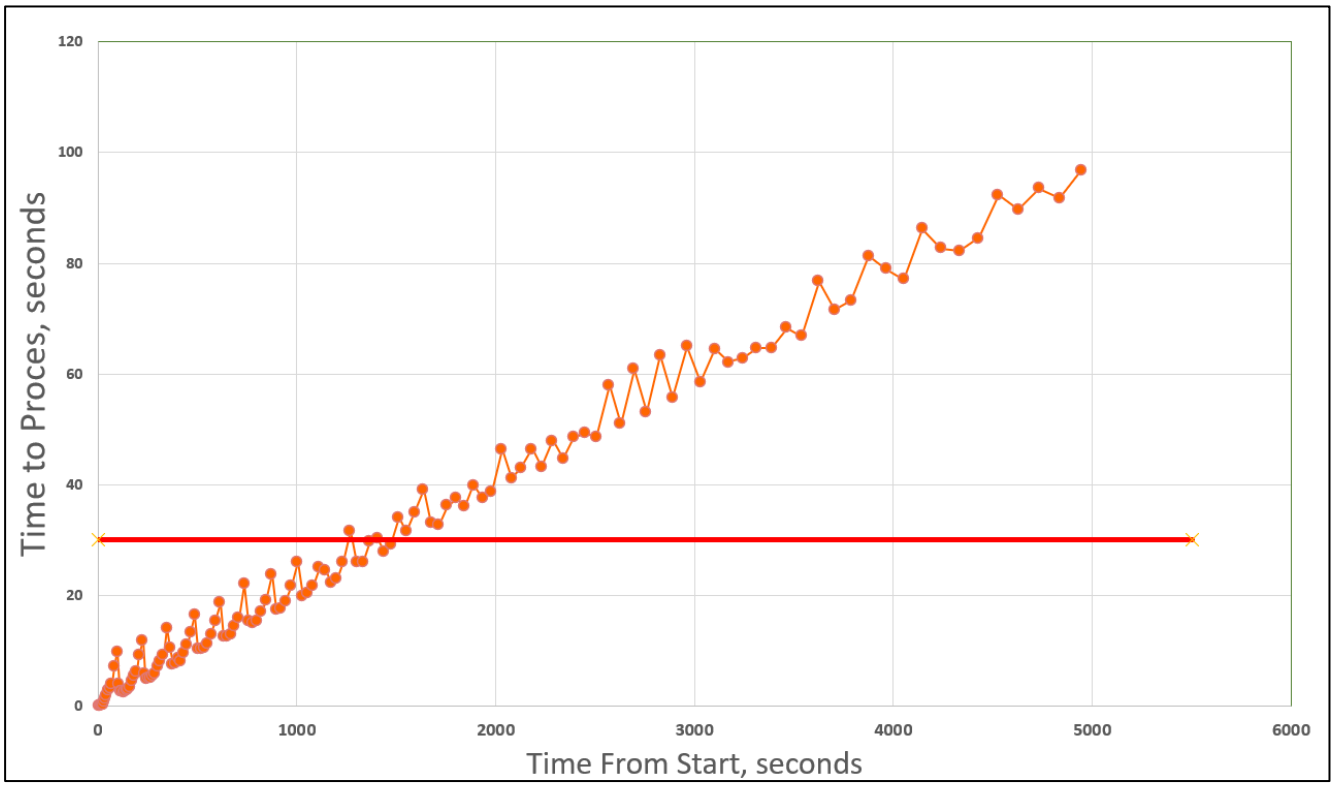


Рисунок 3.4 – Час виконання Conformance Checking при значені Execution Rate в 10 000

Не зважаючи на виконані оптимізації, час на обробку лог-файлу тестової системи зростає. Як було зазначено раніше, тестова система була змінена таким чином, щоб у 5 відсотках усіх сценаріїв, давати помилку. Ці екземпляри залишаються у лог-файлі, та передаються кожного разу для аналізу. Зі збільшенням

значення ExecutionRate, спостерігаємо стрімке зростання кількості таких сценаріїв, що містять помилки.

Такі сценарії не можна вилучати, при вивантаженні, як це було зроблено із завершеними, адже такі сценарії можуть бути як незавершеними, так і такими, що містять помилку.

На рисунку 3.5 зображено графік, на якому відображено час роботи алгоритмів Conformance Checking за різних значень змінної Execution Rate.

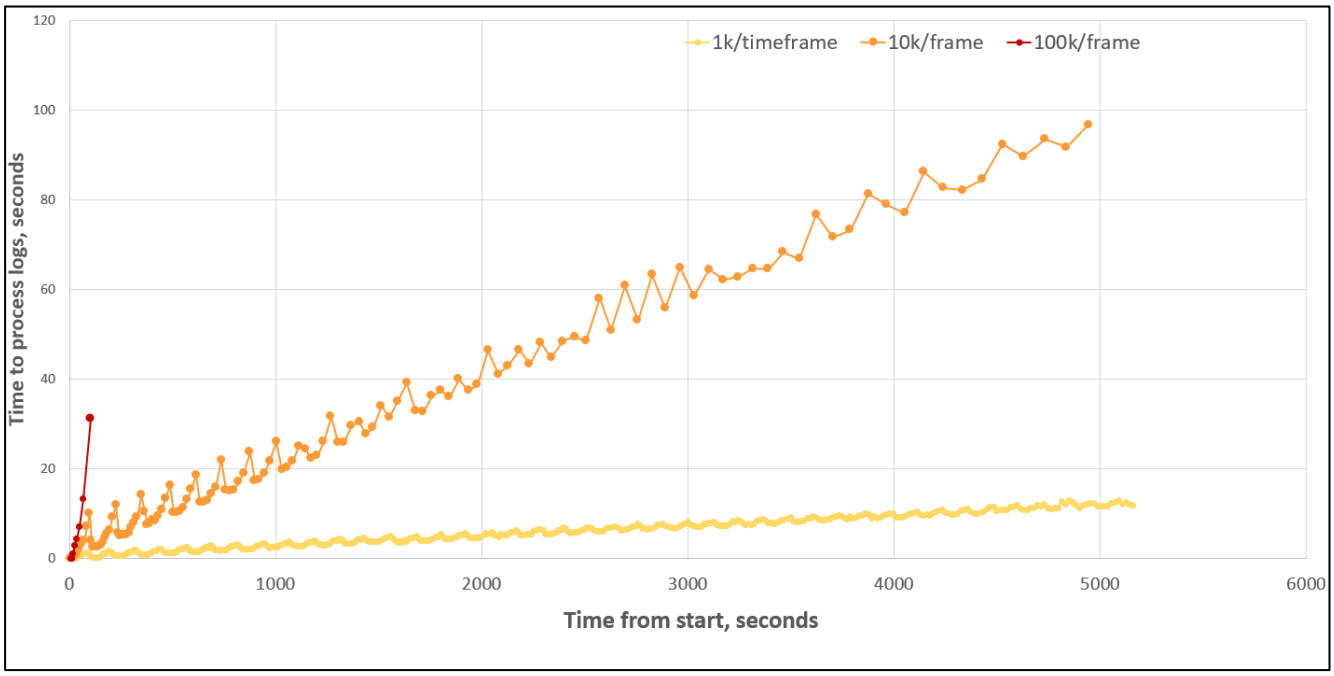


Рисунок 3.5 – Порівняння продуктивності LogProcessor за різних значень Execution Rate

За графік на рисунку 3.5 можна побачити тенденцію до експоненціального зростання показника часу роботи LogProcessor, особливо за значення ExecutionRate у сто тисяч одиниць.

Можна значно зменшити значення змінної ExecutionRate та деякий час спостерігати задовільну швидкість роботи LogProcessor, проте час роботи невпинно росте, та через деякий час неодмінно перевищить заданий показник у 30 секунд.

## 3.5 Тлумачення результатів експерименту

Були проведені заміри швидкості роботи LogProcessor під різним навантаженням. Попри те, що експерименти проводилися із достатньо високою інтенсивністю роботи системи, система LogProcessor була здатна аналізувати досить велику кількість сценаріїв використання за прийнятний час.

Проте, яким малим не було б навантаження, з плином часу, LogProcessor буде усе повільніше оброблювати лог-записи.

Оптимізація передачі даних до LogProcessor надала лише невелике прискорення, проте саме оптимізований спосіб передачі було використано у наступних ітераціях експерименту.

Добрих результатів дала оптимізація вибірки лог-записів, що передаються до LogProcessor. При тому самому значені часу від початку експерименту, час роботи LogProcessor був у 6-9 разів менший, за використання оптимізації.

При порядковому збільшенню значення ExecutionRate, ми спостерігали збільшення необхідного часу для LogProcessor для виконання Conformance Checking.

Проблема полягає у тому, що кількість лог-записів постійно зростає, навіть якщо не включати до Conformance Checking записи, що належать успішно завершеним екземплярам процесу. З плином часу, лог-сховище заповнюється записами, що належать до екземплярів процесів із помилкою, і тому час роботи LogProcessor зростає.

## 3.6 Інші можливі оптимізації

Не зважаючи на те, що у рамках даної роботи не було вирішено проблему поступового збільшення часу роботи LogProcessor, є декілька оптимізацій, що могли б допомогти побороти цю проблему.

По-перше, можна використати можливості паралельних обчислень у LogProcessor, та одночасно виконувати декілька процесів Conformance Checking.

Також, можливо розробити механізми архівації застарілих записів. Якщо процес покупки не було завершено за годину, то є досить низька вірогідність, що він буде завершений взагалі і записи, які відповідають таким застарілим екземпляром процесу можна не включати у вибірку для Conformance Checking.

В роботі не було використано можливості трансляції даних, які, наприклад надає Apache Kafka. Можливо, використання таких засобів може надати додатковий приріст у продуктивності.

Все ж таки, які б прискорення не застосовувалися би, із плином часу, вибірка лог-записів зростає, і потрібно розробити алгоритми, що дозволять не використовувати ту частину даних, яку можна вважати застарілою у рамках системи.

# ВИСНОВКИ

Під час роботи було досліджено підходи до проектування та архітектури розподілених систем, що використовуються у сфері корпоративного програмного забезпечення. Також було досліджено поєднання декількох архітектурних патернів, та зміни в системі, що вони вимагають.

Було досліджено сучасні підходи до моніторингу систем, що побудовані із використанням мікросервісної архітектури. У такий підходах були знайдені певні недоліки, що стають на заваді якісному використанню систем та швидким аналізом стану системи та її поведінки.

Під час аналізу предметної області вдалося знайти підходи та інструменти, що дають можливість використовувати логи систем для побудування моделей поведінки систем та сценаріїв їх використання. Були досліджені алгоритми та вимоги що виникають під час використання певних технік Process Mining. Також було розглянуто спеціальне програмне забезпечення що використовує дані техніки.

Під час роботи були розроблені компоненти, що надавали змогу централізовано зберігати, та агрегувати логи. У якості тестової системи була модифікована уже існуюча система. Тестова система впродовж роботи створювала лог-записи необхідного формату та відправляла їх до сховища логів.

Також, був розроблений механізм автоматизації роботи тестової системи, що спростило роботу із системою, та дозволило імітувати високе навантаження.

Була розроблена система аналізу лог-записів, на основі бібліотеки та засобів Process Mining. Система аналізу працювала у режимі реального часу під час виконання експерименту.

Також, між ітераціями експерименту, було виконано декілька оптимізацій, що дало суттєвий приріст у продуктивності системи.

Результати експерименту в цілому можна назвати успішними, проте, за вимогами поставленими до системи аналізу, цілком досягти мети не вдалося.

При роботі тестової системи під великим навантаженням, з’ясувалося, що із плином часу, відносно вимог до системи, потрібно аналізувати постійно зростаючий об’єм лог-записів, що гальмує роботу компоненту аналізу.

Також, були запропоновані інші можливості оптимізації, що можуть бути виконані поза рамками даної роботи.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ


[1] D. Sprott та L. Wilkes, «Understanding service-oriented architecture,» *The Architecture Journal,* т. 1, p. 10–17, 2004.

[2] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin та L. Safina, «Microservices: yesterday, today, and tomorrow,» *Present and ulterior software engineering,* p. 195–216, 2017.

[3] Andriy Yerokhin, «Usage of Phase Space Diagram to Finding Significant Features of Rhinomanometric Signals,» в *Computer Science & Information Technologies (CSIT'2016),* Lviv, Ukraine, 2016.

[4] D. Betts, J. Dominguez, G. Melnik, F. Simonazzi та M. Subramanian, *Exploring CQRS and Event Sourcing: A journey into high scalability, availability, and maintainability with Windows Azure,* Microsoft patterns & practices, 2013.

[5] Andriy Yerokhin, «F-transform 3D Point Cloud Filtering Algorithm,» в *Proc. of the 2th IEEE International Conference on Data Stream Mining & Processing*, Lviv, Ukraine, 2018.

[6] A. Yerokhin, N. A., B. A та T. A. , «Usage of F-transform to finding informative parameters of rhinomanometric signals,» в *Proc. of the International Conference on Computer Sciences and Information Technologies, IEEE*, Lviv, Ukraine, 2015.

[7] A. Yerokhin, A. Babii, A. Nechyporenko та O. Turuta, *A Lars-Based Method of the Construction of a Fuzzy Regression Model for the Selection of Significant Features,* 2016.

[8] M. Fowler, «Event sourcing,» *Online, Dec,* p. 18, 2005.

[9] L. Richardson та S. Ruby, RESTful web services, " O'Reilly Media, Inc.", 2008.

[10] W. v. d. Aalst, Process Mining: Data Science in Action, Berlin: Springer-Verlag, 2012.

[11] Van Der Aalst, Wil, et al., «Process mining manifesto,» International Conference on Business Process Management, Berlin, 2011.

[12] J. Evermann та G. Assadipour, «Big data meets process mining: Implementing the alpha algorithm with map-reduce,» в *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, 2014.

[13] B. F. Van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters та W. M. P. van Der Aalst, «The ProM framework: A new era in process mining tool support,» в *International conference on application and theory of petri nets*, 2005.

[14] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau та others, *Extensible markup language (XML) 1.0,* W3C recommendation October, 2000.

[15] G. a. F. L. D. J. Van Rossum, Python tutorial, Amsterdam: Centrum voor Wiskunde en Informatica, 1995.

[16] A. S. J. v. Z. Berti та W. v. d. Aalst, «Process mining for python (PM4Py): bridging the gap between process-and data science,» *arXiv preprint arXiv*, 2019.