НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки _____
(повна назва інституту/факультету)

Кафедра автоматики та управління в технічних системах _____
(повна назва кафедри)

«На правах рукопису»
УДК 004.75_____

«До захисту допущено»

Завідувач кафедри
_____ О.І. Ролик
(підпис) (ініціали, прізвище)
“___”_____2018р.

Магістерська дисертація

зі спеціальності (спеціалізації) 126 Інформаційні системи та технології
(код і назва спеціальності)

на тему: ”Система моніторингу продуктивності додатків з мікросервісною архітектурою”

Виконав (-ла): студент (-ка) 6 курсу, групи ІАз-72мп
(шифр групи)

Бачкала Богдан Олександрович _____ _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник к.т.н., доцент Букасов М.М. _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет (інститут) інформатики та обчислювальної техніки _____
(повна назва)

Кафедра автоматики та управління в технічних системах _____
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною (освітньо-науковою) програмою

Спеціальність (спеціалізація) 126 Інформаційні системи та технології
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.І. Ролик
(підпис) (ініціали, прізвище)
«___»_____2018 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Бачкала Богдан Олександрович _____
(прізвище, ім'я, по батькові)

1. Тема дисертації "Система моніторингу продуктивності додатків з мікросервісною архітектурою"
науковий керівник дисертації к.т.н., доцент Букасов М.М._____ ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 2018 р. №_____

2. Строк подання студентом дисертації _____

3. Об'єкт дослідження система моніторингу продуктивності додатків з мікросервісною архітектурою

4. Предмет дослідження (вихідні дані для магістерської дисертації за освітньо-професійною програмою) підтримка інтеграції до процесів неперервної доставки; реалізація трьох методів аналізу показників продуктивності; не менше 70 відсотків точності виявлення відхилень продуктивності мікросервісних додатків.

5. Перелік завдань, які потрібно розробити

- провести аналіз існуючих рішень;
- провести аналіз методів виявлення відхилень продуктивності;

- розробити архітектуру системи;
- провести аналіз результатів тестування;
- зробити висновки.

6. Орієнтовний перелік ілюстративного (графічного) матеріалу
   - структурна схема системи моніторингу продуктивності;
   - діаграма класів системи моніторингу продуктивності;
   - діаграма класів модуля типів;
   - діаграма класів модуля сховища;
   - діаграма класів модуля аналізу відхилень продуктивності;
   - діаграма послідовності процесів модуля стресового тестування;
   - діаграма послідовності процесів модуля аналізу продуктивності;
   - діаграма модулів тестового середовища.

7. Орієнтовний перелік публікацій

   - Бачкала Б. О. Виявлення відхилень продуктивності додатків з мікросервісною архітектурою. // VII Міжнародна науково-практична конференція з інформаційних систем та технології. – С. 37–41.

8. Консультанти розділів дисертації*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1.	Аналіз існуючих рішень	02.09.2018	
2.	Аналіз методів виявлення відхилень продуктивності	15.09.2018	
3.	Розробка архітектури системи	12.10.2018	
4.	Проведення тестування та аналіз результатів	09.11.2018	
5.	Оформлення дипломної роботи	20.11.2018	

Студент _____ Б.О. Бачкала
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ М.М. Букасов
(підпис) (ініціали, прізвище)

* Консультантом не може бути зазначено наукового керівника

РЕФЕРАТ

Робота складається з 111 сторінок, робота містить 41 рисунок, 33 таблиці, 44 посилання та 10 додатків.

Мікросервісна архітектура дозволяє розробникам оптимізувати автономність, масштабованість та простежуваність програмних додатків, а також дає можливість виконувати децентралізоване управління системами. Проте, така організація програмної системи має деякі недоліки, одним з таких недоліків є наявність складнощів в виявленні джерел деградації продуктивності додатків.

Метою цієї роботи є розробка системи моніторингу додатків з мікросервісною архітектурою для автоматичного виявлення відхилень продуктивності на етапі тестування.

Об'єктом дослідження є система моніторингу продуктивності додатків з мікросервісною архітектурою.

Предметом дослідження є аналіз змін поведінки продуктивності додатків з мікросервісною архітектурою на етапі тестування програмного забезпечення.

Зміну значень показників продуктивності додатків з мікросервісною архітектурою за різних версій системи було досліджено за допомогою статистичних методів аналізу даних: методу t-критерію Стьюдента, методу контрольних карт та методу асоціативних правил.

В результаті роботи було розроблено систему моніторингу додатків з мікросервісною архітектурою. Основними перевагами системи є наявність засобів інтеграції в процес неперервної доставки програмного забезпечення та підтримка додатків з мікросервісною архітектурою.

Перелік ключових слів: мікросервіс, контейнер, docker, kubernetes, продуктивність, асоціативні правила, контрольні карти.

РЕФЕРАТ

Робота состоит из 111 страниц, робота включает 41 рисунок, 33 таблицы, 44 источника и 10 приложений.

Микросервисная архитектура позволяет разработчикам оптимизировать автономность, масштабируемость и прослеживаемость программных систем, а также дает возможность выполнять децентрализованное управление системами. Однако, такая организация системы имеет некоторые недостатки, одним из таких недостатков является наличие трудностей при поиске первопричин деградации производительности системы.

Целью этой работы является разработка системы мониторинга приложений с микросервисной архитектурой для автоматизации процессов обнаружения отклонений производительности на этапе тестирования.

Объектом исследования является система мониторинга продуктивности приложений с микросервисной архитектурой.

Предметом исследования является анализ изменений поведения производительности приложений с микросервисной архитектурой на этапе тестирования программного обеспечения.

Изменение значений показателей производительности приложений с микросервисной архитектурой при разных версиях системы были исследованы с помощью статистических методов анализа данных: метода t-критерия Стьюдента, метода контрольных карт и метода ассоциативных правил.

Результатом работы является система мониторинга приложений с микросервисной архитектурой. Основными преимуществами системы является наличие средств интеграции в процесс непрерывной доставки программного обеспечения, а также поддержка приложений с микросервисной архитектурой.

Перечень ключевых слов: микросервис, контейнер, docker, kubernetes, производительность, ассоциативные правила, контрольные карты.

ABSTRACT

The work amounts to 111 pages, with 41 figures, 33 tables, 44 references and 10 appendixes.

Microservices enable developers to enhance autonomy, scalability, and traceability of software systems, allow teams to establish decentralized system governance. Such an arrangement of services has a number of disadvantages, one of such shortcomings is the complications in identifying sources of system performance degradations.

The purpose of this work is the development of a performance monitoring system for microservice applications, that would facilitate performance regression detection during the testing phase.

The object of the research is a performance monitoring system for applications with microservice architectures.

The subject of the research is the analysis of changes in the performance of applications with microservice architecture at the stage of software testing.

Changes in the performance counters values of applications with microservice architectures for different versions of the system were investigated using statistical data analysis methods: the Student's t-test method, the control charts method and the association rules method.

The result of the work is a performance monitoring system for applications with microservice architecture. The main advantages of the system include means for simple integration into the process of continuous software delivery and support of applications with microservice architectures.

Key words: microservice, container, docker, kubernetes, performance, association rules, control charts.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ВМ – віртуальна машина

ІН – істино негативний

ІП – істино позитивний

ФС – файлова система

ХН – хибно негативний

ХП – хибно позитивний

CD – continuous delivery

CI – continuous integration

CPU – central processing unit

DI – docker image

DR – docker registry

IP – internet protocol

ВСТУП

Для того щоб підтримувати масштабованість та надійність програм, розробники, як правило, переходять від монолітного стилю архітектури програмного забезпечення до архітектури мікросервісів. Мікросервіси – це невеликі та незалежні програмні служби, які мають обмежений набір функцій, але безпосередньо пов'язані з іншими службами, такий стиль архітектури є популярним при розробці веб-додатків. Підхід проектування додатків як сукупність мікросервісів забезпечує більшу масштабованість і дає можливість розподілити додаток на декількох фізичних та віртуальних системах, що також забезпечує більшу надійність.

Хоча мікросервісні архітектури мають багато переваг перед монолітним стилем, вони також збільшують ускладнюють такі завдання, як управління та моніторинг системи. Розподіляючи додаток на декількох віртуалізованих машинах, аспект моніторингу стає ще складнішим завданням.

В області моніторингу продуктивності додатків часто спостерігається час відгуку викликів окремих методів для виявлення проблем, перш ніж вони створюють реальну загрозу продуктивності системи. Природа розподілених мікросервісних систем породжує середовище, де окремі служби можуть оновлюються кілька разів на день, і де одночасно можуть бути розгорнуті кілька версій програми, це створює кілька нових викликів для інженерів. Оскільки кожне оновлення може змінити час відгуку методів у різних модулях додатка, це може привести до появи багатьох помилкових сигналів систем моніторингу додатків, які використовують історичні дані та, можливо, помилок у системах прогнозування та виявлення аномалій.

Одним з недоліків мікросервісної архітектури є складність тестування системи в цілому. Важливою частиною розробки програмного забезпечення є написання навантажувальних тестів, такі тести дають змогу інженерам впевнитись у відсутності зниження показників продуктивності додатків перед випуском нової версії продукту. Одиницею виміру продуктивності додатка є системний показник продуктивності, що описує споживання окремого ресурсу апаратного забезпечення. В контексті мікросервісної архітектури такі показники збираються для кожної програмної

служби, реальні системи можуть мати сотні незалежних служб. Таким чином, додаток з архітектурою мікросервісів може виробляти численну кількість статистичних даних продуктивності системи. Отже, розробники вимушені аналізувати величезні об’єми інформації по показникам продуктивності додатка кожен раз на етапі випуску нової версії системи, такий процес займає багато часу та схильний породжувати значну кількість помилок.

Моніторинг продуктивності програмного забезпечення під постійними змінами навантаження та системи в цілому є проблемою, яку потрібно вирішити для додатків з мікросервісною архітектурою.

Мікросервісна архітектура пропонує зменшення складності додатків, обіцяє надавати можливість самостійного масштабування окремих сервісів системи, легко видаляти та розгортати незалежні частини системи, підтримувати використання різних мов програмування, збільшити загальну гнучкість системи та, нарешті, підвищити стійкість додатків. Для фахівців з контролю продуктивності систем, такий архітектурний стиль проектування програмного забезпечення представляє численну кількість складних та неочікуваних завдань. Виникає потреба в програмному забезпеченні, здатному вирішувати ці проблеми та водночас задовольняти специфічним потребам мікросервісного середовища.

Система моніторингу продуктивності додатків з мікросервісною архітектурою є прикладом такого програмного забезпечення. Задачею такої системи є виявляння суттєвих змін в продуктивності додатків під час розробки та допомога в уникненні зниження продуктивності системи.

Існує ряд проблем які мають бути вирішені для того щоб забезпечити реєстрацію відхилень продуктивності в мікросервісному середовищі. Такі проблеми стосуються розподіленого характеру мікросервісних систем, складності в розумінні поведінки продуктивності мікросервісних систем, дослідженні наявних показників ефективності мікросервісних додатків та визначенні їх властивостей, в пошуку підходів до тестування масштабованих систем, та складності питання вибору різних показників продуктивності окремих служб для аналізу та виявлення регресії системи. Розподіллений характер мікросервісних систем вимагає використання технологій

віртуалізації та контейнеризації. Ці аспекти самі по собі вносять складність до процесу аналізу продуктивності системи.

Показники продуктивності мають іншу природу та поведінку в контейнеризованих додатках. Так наприклад, показники продуктивності окремої служби можуть змінюватись під впливом іншої служби. Продуктивність є важливим аспектом великих програмних систем, оскільки основна кількість проблем пов’язана з погіршенням продуктивності. Такі проблеми включають в себе зростаючий час відклику програми, збої чи провисання процесів під навантаженням або не відповідність бажаним угодам про рівень обслуговування. Для уникнення таких проблем перед випуском нової версії системи проводять навантажувальне тестування. Такі тести симулюють встановлене навантаження на систему та збирають показники продуктивності. Наступним кроком є вибірковий аналіз системних показників продуктивності з метою виявлення можливих відхилень. Ручне виявлення регресії продуктивності є неефективним та породжує велику кількість помилок через великий обсяг даних для аналізу.

Метою цієї роботи є розробка незалежної системи моніторингу продуктивності додатків у мікросервісному середовищі. Така система має полегшити роботу інженерам, відповідальним за контроль продуктивності додатків. Особливо актуальним є використання розробки в мікросервісному середовищі для виявлення відхилень в продуктивності додатка.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1 Архітектура контейнерів

Віртуалізація апаратних засобів комп'ютера приховує фізичні характеристики обчислювальних платформ від користувачів, представляючи натомість іншу абстрактну обчислювальну платформу. Віртуалізація платформи здійснюється на даній апаратній платформі програмним забезпеченням хост-машини, що створює моделювання комп'ютерного середовища, віртуальну машину (ВМ), для його гостьового програмного забезпечення. Гостьове програмне забезпечення не обмежується користувальницькими додатками; багато хостів дозволяють виконання повноцінних операційних систем. Гостьове програмне забезпечення виконується так, ніби воно було запущено безпосередньо на фізичному обладнанні, з декількома значними застереженнями. Доступ до фізичних ресурсів системи, зазвичай керується на більш обмеженому рівні, ніж хост–процесор та системна пам'ять [1]. Контейнеризація – це сукупність механізмів операційної системи, які дозволяють запускати окремі процеси в ізольованому середовищі [2]. В контейнері можливо запускати окремий процес чи повну операційну систему. Контейнери, у порівнянні з віртуалізацією, потребують набагато менше пам’яті хост машини та мають менший час запуску, ці фактори полегшують передачу образів оточення у мережі [3]. Віртуалізовані системи потребують збереження та установку усіх необхідних бібліотек та програмних залежностей безпосередньо в віртуальну машину, з іншого боку, контейнери можуть містити лише один ізольований процес. Таким чином контейнери пропонують більшість переваг віртуалізованих систем, потребуючи при цьому набагато менше ресурсів хост-машини.

Віртуальна машина може бути легко контрольована і перевірена ззовні, як фізична одиниця, віртуалізація забезпечує високий рівень ізольованості [1]. У випадку контейнерів необхідним є застосунок ряду підходів для забезпечення ізоляції системи [3]. Одним з таких підходів, є використання унікальних ідентифікаторів для окремих контейнерів. Рівень ізоляції контейнерів залежить від реалізації конкретного

способу проектування архітектури контейнерів, це дозволяє використовувати різні типи контейнерів. Так, наприклад, можливо спроектувати контейнер таким чином, щоб в ньому виконувався один ізольований процес, а усі вільні ресурси були розділені між іншими контейнерами. Контейнери забезпечують ізоляцію на рівні файлової системи: кожен процес виконується у повністю окремій кореневій файловій системі (ФС). Також контейнери можуть розділяти одну IP-адресу, адже самі по собі контейнери не мають ідентифікатор мережевого рівня, кожен ізольований процес має доступ тільки до пов'язаного з контейнером мережевого простору імен.

1.2 Мікросервіси

Архітектура мікросервісів являє собою підхід до розробки програмного забезпечення, в основі якого є створення незалежних сервісів, кожний з яких виконує єдине завдання і має змогу взаємодіяти з іншими сервісами системи [4].

Такий підхід дає змогу інженерам працювати незалежно один від одного та розробляти окремі сервіси використовуючи різні підходи та мови програмування. Підхід проектування додатків як сукупності мікросервісів забезпечує масштабованість системи та дає можливість розподілити додаток на декількох фізичних та віртуальних машинах, що також забезпечує більшу надійність.

Зазвичай, мікросервіси запускають в окремих ізольованих контейнерах та застосовують контролер масштабованості, наприклад Kubernetes, для підтримки та масштабування окремих сервісів у відповідності до робочих навантажень системи. В якості платформи контейнеризації популярним вибором є Docker – інструментарій для управління ізольованими контейнерами [5].

Архітектурний стиль проектування програмного забезпечення в якості набору незалежних служб та контейнеризації таких служб для подальшого управління різко набирає популярність серед розробників. Такі технології як Docker та Kubernetes дозволять інженерам розробляти надійні, гнучкі та масштабовані системи.

На відміну від монолітних систем які розгортаються та запускаються на віртуальних машинах, мікросервіси можливо розгортати незалежно один від одного

в окремих ізольованих контейнерах, та перезапускати у разі відмови порізно без необхідності в перезапуску усієї системи. Такі контейнери можливо масштабувати на тисячі екземплярів у відповідності до навантаження на систему. Порівняння мікросервісної та монолітної архітектури показано на рисунку 1.1.

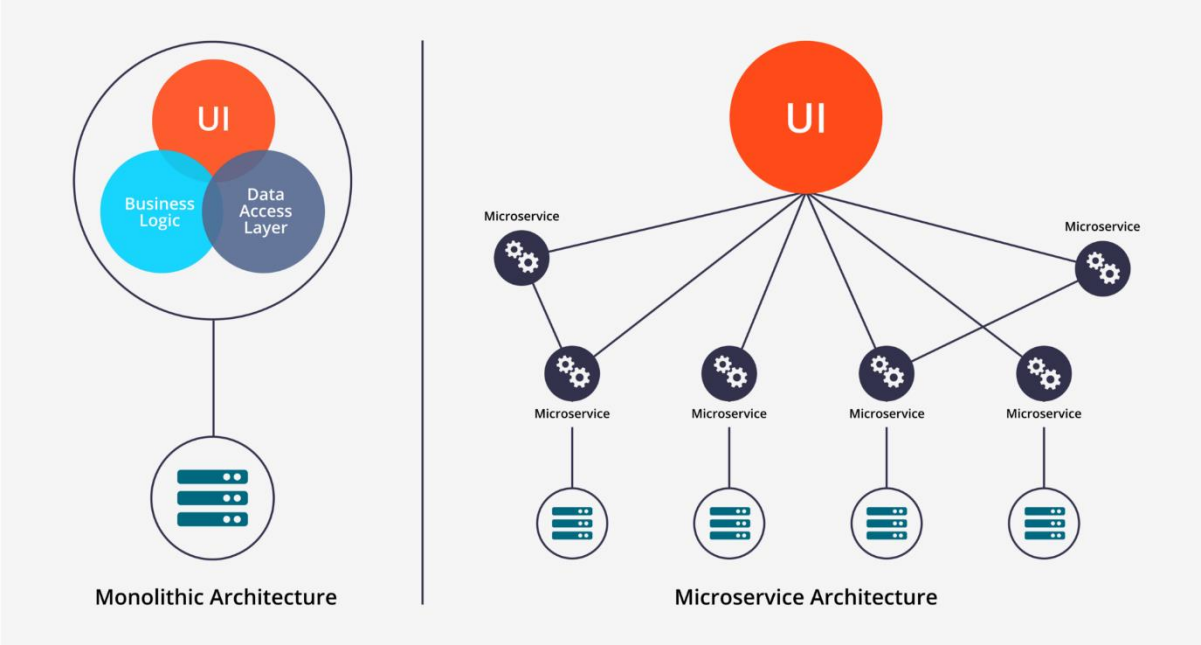


Рисунок 1.1 - Порівняння мікросервісної та монолітної архітектури [6]

Як видно з рисунку 1.1 монолітна архітектура реалізується єдиною програмною системою, яка працює з єдиною базою даних і кодовою базою. Таким чином, кожен сервіс має доступ до всіх ресурсів додатка. Всі сервіси працюють з одним сервером баз даних, що дозволяє кожному сервісу звертатися до бази даних безпосередньо. При реалізації мікросервісної архітектури, кожен сервіс має свою кодову базу та базу даних і тільки він має до них доступ. Оскільки сервіси не мають доступу до бази даних іншого сервісу – доступ до таких даних здійснюється викликом інших сервісів ресурсу. Таким чином, мікросервісній архітектурі притаманний більш складний доступ до ресурсів, які відносяться до інших сервісів [7].

1.2.1 Засоби віртуалізації

Досить поширеним засобом віртуалізації замкнутих систем є інструментарій для управління ізольованими Linux-контейнерами – Docker [5]. Docker використовує контейнерну віртуалізацію (рисунок 1.2), додатки в окремих контейнерах повністю ізольовані один від одного на рівні файлової системи та внутрішніх процесів. Хост-машина дозволяє паралельне виконання декількох контейнерів, забезпечуючи віртуалізацію для додатків без необхідності у віртуалізації операційних систем для кожного додатка. Таким чином один сервер здатен запускати сотні контейнерів одночасно, забезпечуючи масштабованість та розподіл навантаження між окремими сервісами.

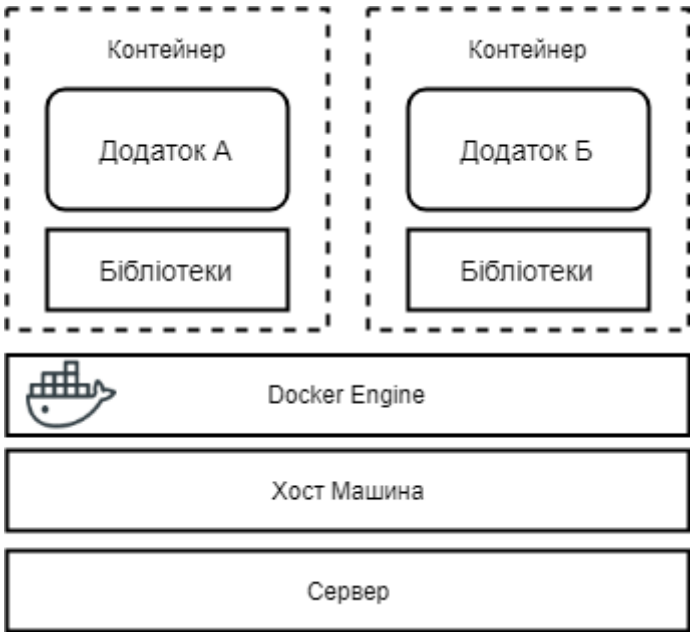


Рисунок 1.2 – Будова Docker-контейнера

Docker дозволяє не переймаючись вмістом контейнера запускати довільні процеси в режимі ізоляції і потім переносити і клонувати сформовані для даних процесів контейнери на інші сервери, беручи на себе всю роботу зі створення, обслуговування і підтримки контейнерів [5]. На рисунку 1.3 зображено архітектуру та основні компоненти Docker.

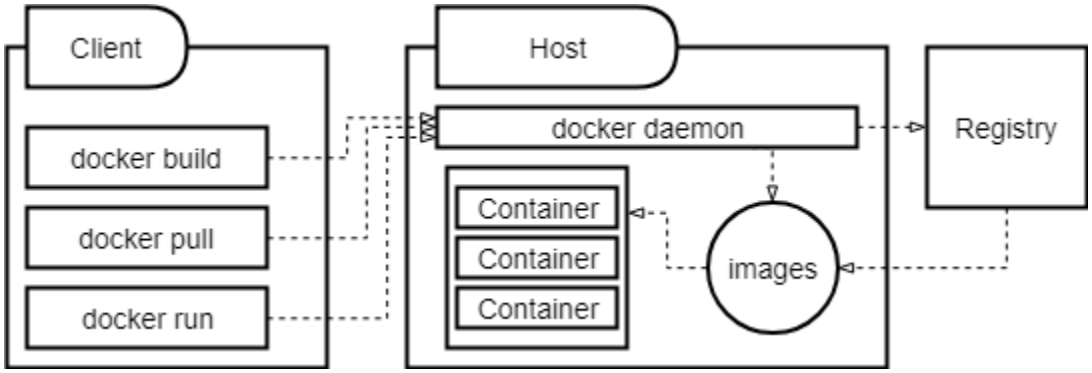


Рисунок 1.3 – Архітектура та компоненти Docker

Docker Client представляє собою інтерфейс користувача Docker, включає в себе набір команд для керування образами та контейнерами [8]. Docker Client дозволяє взаємодіяти з Docker Daemon, демон процесом [9], який відповідає за роботу системи. Docker Host містить контейнери та образи Docker Image (DI), образ містить операційну систему, застосунок і всі його залежності. Образи в Docker складаються з шарів. Якщо нам треба образ з веб-сервером, то ми беремо за основу образ з дистрибутивом операційної системи, додаємо залежність – веб-сервер, і записуємо це як новий образ, який матиме два шари – один з ОС, наступний з веб-сервером [5]. Зібрані Docker образи зберігаються у Docker Registry (DR) для подальшого використання.

1.2.2 Системи керування контейнерами

Kubernetes це відкрита система автоматичного розгортання, масштабування та управління застосунками у контейнерах [10]. Kubernetes дозволяє розгортати Docker контейнери у кластері та має інструментарій для реплікації та масштабування контейнерів. Kubernetes може автоматично розгортати нові контейнери у відповідності до навантаження на систему, підтримує автоматичний перезапуск контейнерів у разі виходу з ладу окремих сервісів. У Kubernetes мікросервіси можливо розгортати незалежно один від одного в окремих ізольованих Docker

контейнерах, та перезапускати у разі відмови порізно без необхідності в перезапуску усієї системи. Діаграму кластеру зображено на рисунку 1.4.

Kubernetes кластер складається с двох компонент:

а) Kubernetes Master – основний модуль контролю кластера;

б) Kubernetes Node – вузол кластера, на якому розгортаються контейнери.

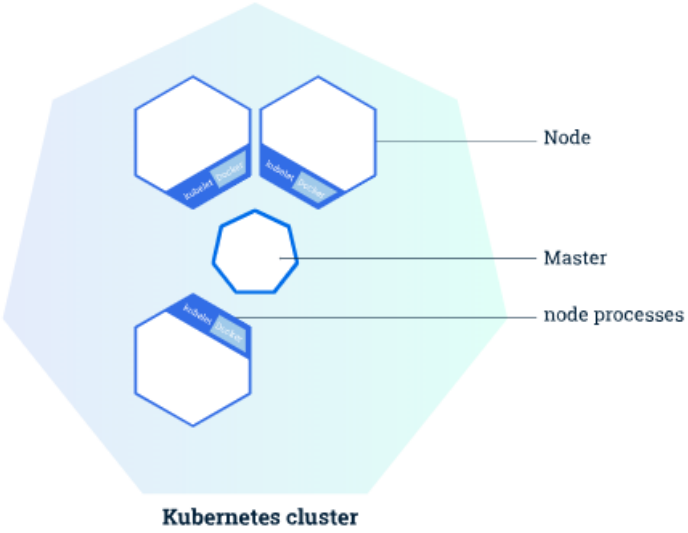


Рисунок 1.4 – Діаграма Kubernetes кластеру [10]

Kubernetes Master керує навантаженням та комунікаціями у системі. Кожен вузол кластера має Kubelet, що відповідає за робочий стан вузла та засоби комунікації з Kubernetes Master.

1.3 Тестування продуктивності

Тестування продуктивності – це один із видів тестування програмного забезпечення, який націлений на визначення пропускної спроможності, надійності та масштабованості програмної системи під заданим навантаженням [11]. Навантаження, частіше за все, являє собою штучно згенерований набір операцій, які виконуються над системою для спостереження роботи додатка. Тестування продуктивності допомагає виявити вузькі місця в системі, полегшує налаштування продуктивності та дає змогу оцінити відповідність системи заданим умовам

продуктивності. Прикладами таких вимог можуть бути гарантований час доступності системи, певні обмеження часу відповіді для певної кількості користувачів або гарантована пропускна спроможність додатка.

В тестуванні продуктивності розрізняють наступні напрямки: навантажувальне, стресове, тестування стабільності, тестування на відмову та відновлення, об'ємне тестування, шип тестування [12]. Ці типи тестування в основному відрізняються метою, яку вони прагнуть досягти. Навантажувальні тести призначені для визначення швидкості, масштабованості та стабільності системи. Масштабованість описує здатність системи адаптуватися до збільшення робочих навантажень за рахунок використання додаткових ресурсів. Навантажувальні тести проводять оцінку поведінки системи за звичайних та пікових умовах навантаження. Стресове тестування оцінює поведінку системи при навантаженнях, які значно перевищують очікуваний максимум. Тестування стабільності перевіряє чи здатна система витримувати очікуване навантаження протягом тривалого часу. Об’ємне тестування проводиться зі збільшенням кількості використовуваних даних, які зберігаються і використовуються в додатка. Шип тестування проводиться раптово, збільшуючи навантаження на невеликий час, створюване за допомогою дуже великого числа користувачів, і спостерігаючи за поведінкою системи [12].

Незалежно від типу навантажувальних тестів, існують різні рівні, за яких тестування продуктивності може бути виконане. Модульні тести зосереджені на тестуванні окремих класів та методів, тоді як модульні тести відповідають за перевірку окремих модулів системи в цілому. Інтеграційні тести зосереджені на тестуванні взаємодії між модулями додатка, тоді як системні тести мають протестувати загальну поведінку системи.

Результатом проведення навантажувального тестування є набір показників продуктивності додатка, які використовуються для подальшого аналізу. Основними показниками продуктивності системи є:

а) використання ресурсів центрального процесора це показник, що відображає скільки часу з заданого певного інтервалу було витрачено процесором на обчислення для обраного процесу;

б) використання оперативної пам’яті це показник, що відображає кількість пам'яті, яка була використана додатком;

в) використання мережевих ресурсів це показник, значення якого відображають межі продуктивності системи в цілому;

г) використання ресурсів файлової системи це показник, що визначає продуктивність роботи додатка з диском. Велика кількість зчитувань або записів може призводити до простоювання процесора в очікуванні обробки даних з диска.

Зазвичай тестування продуктивності займає багато часу. Один прогін налаштованого навантажувального тесту може тривати декілька годин та навіть днів. Навантажувальні тести повинні виконуватись довго для того щоб реєструвати такі відхилення як витіки пам'яті.

Через таку природу навантажувальних тестів багато компаній автоматизують процес тестування продуктивності їх систем [13]. Першим кроком зазвичай є автоматизація за допомогою генераторів навантаження які симулюють реальні сценарії використання системи з налаштованою частотою. HP має проект httperf [14] який може автоматично генерувати навантаження на декілька мережевих протоколів. IBM має розробку Rational Performance Tester [15] за допомогою якої можливо автоматизувати генерацію обраної кількості користувачів системи та простежувати роботу додатка під навантаженням. Наступним кроком є автоматичне налаштування компонентів продуктивності операційних систем таких як Windows System Performance Monitor [16] та Linux proc [17]. Також компанії розробляють власні показники продуктивності для покращення процесу аналізу показників продуктивності [13].

Аналіз отриманих статистичних даних продуктивності окремих компонент системи та системи в цілому зазвичай проводиться вручну інженерами, які відповідають за продуктивність системи. Такий процес займає багато часу, та схильний в результаті мати значну кількість помилок, через великі об’єми зібраних даних.

1.4 Види відхилень продуктивності

Тестування продуктивності системи є методом забезпечення відповідності нової версії додатка до вимог продуктивності системи [18]. Такі тести називають функціональними тестами продуктивності системи. Регресія продуктивності означає значне відхилення показників продуктивності додатка від норми.

Зразками найбільш розповсюджених регресії продуктивності є:

а) зріст використання пам'яті;

б) зріст процесорного часу;

в) зріст часу доступу ввід/вивід;

г) зріст навантаження мережі;

д) надлишкове логування;

е) некоректне налаштування;

Додавання виділення значної кількості оперативної пам’яті у часто використовуваний об’єкт може призвести до значного збільшення використання пам’яті системи. Додаткові розрахунки або алгоритми з поганим часом виконання збільшують навантаження на центральний процесор. Збільшення кількості блокуючих операцій вводу/виводу може значно знизити продуктивність системи. Введення зростаючої кількості мережевих запитів знизить продуктивність системи. Надлишок логування може значно знизити продуктивність системи, через надмірну кількість операцій вводу/виводу. Неправильне налаштування системи може призвести до зниження продуктивності.

1.5 Виявлення відхилень продуктивності

Підходи до виявлення відхилень продуктивності порівнюють показники нової версії програмного забезпечення з показниками минулих версій (рисунок 1.5). Розробка та випуск нової версії системи може означати зміни в коді системи, функціоналі та здатності витримувати різні навантаження.

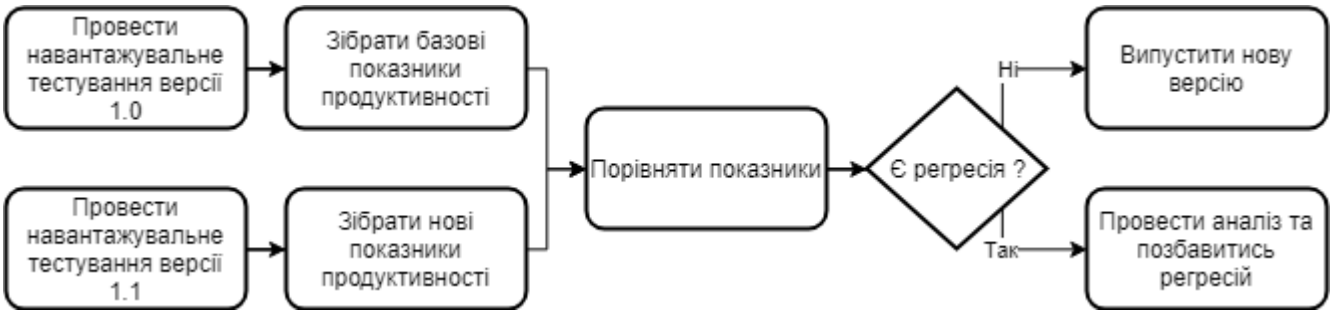


Рисунок 1.5 – Механізм виявлення відхилень продуктивності

Інженери проводять навантажувальне тестування кожного разу перед випуском нової версії системи, для того, щоб впевнитись у відсутності будь-яких погіршень швидкодії сервісів. За своєю природою тести продуктивності сильно відрізняються від функціональних тестів та модульних тестів системи, адже потребують складне налаштування та зазвичай мають довгий час виконання. Через це, навантажувальні тести запускають наприкінці циклу розробки нової версії системи, що в свою чергу означає наявність багатьох змін у різних частинах додатка.

Тому задача визначення першопричини погіршення продуктивності зазвичай є дуже трудомісткою, може займати декілька часів та днів в залежності від досвіду інженерів, складності системи та природи самого відхилення продуктивності. Також важливим фактором аналізу показників продуктивності додатків є великі об’єми статистичних даних навантажувальних тестів які розробники повинні аналізувати наприкінці циклу тестування.

Існує численна кількість підходів до аналізу показників продуктивності додатків та виявлення першопричин деградацій швидкодії програмного забезпечення. Деякі підходи включають в своєму алгоритмі крок зменшення об’єму вхідних показників продуктивності для оптимізації часу виявлення відхилень продуктивності. Інші підходи зосереджуються на аналізі даних логування програмних систем для побудови моделей поведінки додатків та подальшого їх використання на етапі виявлення відхилень продуктивності.

1.6 Оцінка методів виявлення відхилень продуктивності

Існує чотири можливі результати виконання методу виявлення відхилень продуктивності системи.

Метод може вказати на наявність регресії продуктивності, коли система насправді має відхилення. Такий результат є істинно позитивним (ІП). Метод також може не зареєструвати відхилень продуктивності, коли система не має регресії. В такому випадку результат позначають як істинно негативний (ІН). Ці два можливих результати можуть бути використані для оцінки роботи ідеального методу, який не допускає помилок, але в реальному житті необхідно враховувати ще два можливих результати. Метод може вказати на регресію продуктивності, а система при цьому може не мати жодних відхилень. Такий результат є хибно позитивним (ХП). Нарешті, якщо метод не виявляє жодних відхилень продуктивності, але система має регресії, результат позначають як хибно негативний (ХН). На рисунку 1.6 зображено матрицю відношення реальних та розрахованих результатів.

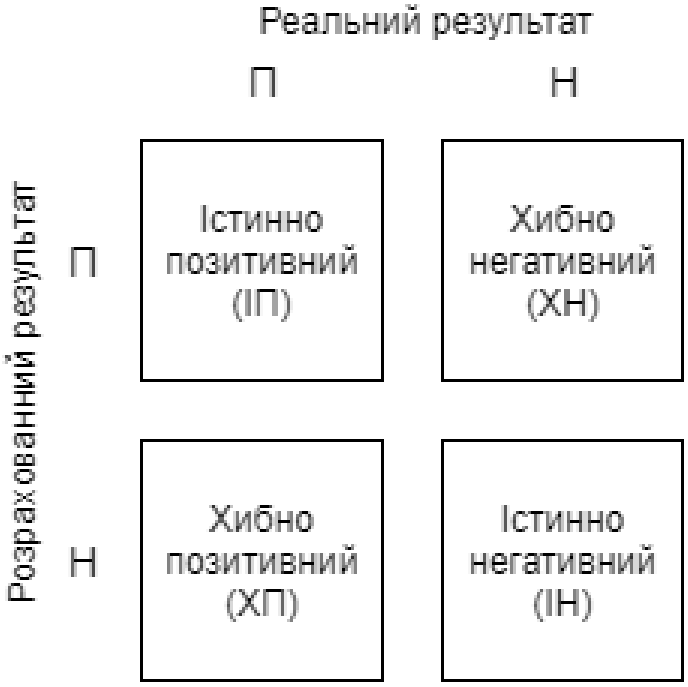


Рисунок 1.6 – Матриця відношення реальних та розрахованих результатів

За допомогою цих чотирьох видів можливих результатів роботи методів виявлення відхилень продуктивності можливо розрахувати параметри точності та повноти.

Точність методу описує скільки виявлених регресії насправді є відхиленнями продуктивності. Точність методу визначається за наступною формулою:

$$\text{Точність} = \frac{\text{ІП}}{\text{ІП} + \text{ХП}}. \tag{1.1}$$

Повнота методу описує скільки реальних відхилень продуктивності було виявлено. Повнота методу визначається за наступною формулою:

$$\text{Повнота} = \frac{\text{ІП}}{\text{ІП} + \text{ХН}}. \tag{1.2}$$

Для обох показників, можливі значення лежать в межах від нуля до одиниці, одиниця вважається оптимальним результатом.

Точність разом з повнотою використовують для визначення F-міри, щоб отримати тільки одну оцінку якості роботи системи. F-міра методу визначається за наступною формулою:

$$F = 2 \cdot \frac{\text{Точність} \cdot \text{Повнота}}{\text{Точність} + \text{Повнота}}. \tag{1.3}$$

Значення F-міри використовується для оцінки ефективності окремого методу виявлення відхилень продуктивності.

2 МЕТОДИ ВИЯВЛЕННЯ ВІДХИЛЕНЬ ПРОДУКТИВНОСТІ

Для розробки системи моніторингу продуктивності додатків з мікросервісною архітектурою необхідно визначити які з існуючих методів реєстрації регресії продуктивності програмного забезпечення можуть бути ефективно використані у контексті мікросервісного середовища.

Для подальшого порівняльного аналізу визначимо критерії відбору методів:

а) швидкість виявлення;

б) адаптивність;

в) інтеграція показників продуктивності;

г) складність;

д) наявність історичних даних;

Критерій швидкодії виявлення відхилень продуктивності описує необхідну кількість прогонів навантажувальних тестів для реєстрації регресії. Перевагою мікросервісного стилю архітектури є можливість швидкої інтеграції змін в окремих сервісах, тому важливо, щоб процес виявлення регресії займав мінімум часу та не гальмував випуск нової версії продукту.

Критерій адаптивності визначає здатність окремого методу пристосовуватись до змін у навантаженні на систему. Додатки з мікросервісною архітектурою представляють собою розподілені системи у кластерах, окремі сервіси яких можуть бути масштабовані у відповідності до навантаження з боку користувачів.

Критерій інтеграції показників продуктивності встановлює можливість підтримки методом нових системних показників. Метод має аналізувати продуктивність системи, враховуючи масштабованість сервісів, коефіцієнти реплікації та особливості внутрішньої комунікації сервісів.

Критерій складності визначає кількість логічних етапів, які необхідно завершити для того щоб зареєструвати регресію системи. Мікросервісна архітектура лише ускладнює процес аналізу відхилень продуктивності, отже методи які мають готові реалізації або є достатньо простими для введення в контексті мікросервісного середовища є більш пріоритетними за інших.

Критерій наявності історичних даних вказує на потребу окремого методу в наявності системних показників минулих навантажувальних тестів для свого аналізу. Така умова вважається недоліком методу, адже накладає додаткові вимоги на розробників системи.

2.1 Статистичні тести

Статистичні тести, такі як t-критерій Стьюдента, часто використовують для порівняння статистичних даних показників системи під час навантажувального тестування [19]. Для застосування t-критерію Стьюдента необхідно аби вхідні дані мали нормальний розподіл. В контексті аналізу продуктивності додатків за статистичними показниками продуктивності додатка доцільно використовувати двовибірковий t-критерій для незалежних вибірок [20].

Важливо зазначити, що на практиці t-критерій показав себе не з кращої сторони та має багато недоліків [19], але через простоту реалізації може бути корисним у використанні разом з іншими методами. Підсумок характеристик методу t-критерій Стьюдента приведено у таблиці 2.1.

Таблиця 2.1 – Підсумок характеристик методу t-критерій Стьюдента

Швидкість виявлення	Адаптивність	Інтеграція показників продуктивності	Складність	Наявність історичних даних
1 прогін	Ні	Так	1 етап	Ні

2.2 Метод контрольних карт

Метод контрольних карт дозволяє автоматично визначати чи відхилення продуктивності додатка зв’язані зі змінами у навантаженні на систему або викликанні дефектами самої системи [13]. Контрольна карта складається з центральної лінії, двох

контрольних меж (над та під центральною лінією) і значень показників продуктивності. Для аналізу відхилень продуктивності необхідно побудувати контрольні карти за базовими показниками системи та за показниками нового тесту, для якого визначаються імовірні регресії. Після побудови відповідних графіків необхідно визначити коефіцієнти відхилення значень показників нового тесту та базових показників системи. Коефіцієнт відхилення описує відсоток показників, які знаходяться за контрольними межами нормальної поведінки системи. Велика різниця у значеннях коефіцієнтів відхилення між новим тестом та базовими показниками може вказувати на наявність можливих регресії продуктивності у новій версії системи. Загальну схему роботи методу контрольних карт зображено на рисунку 2.1.

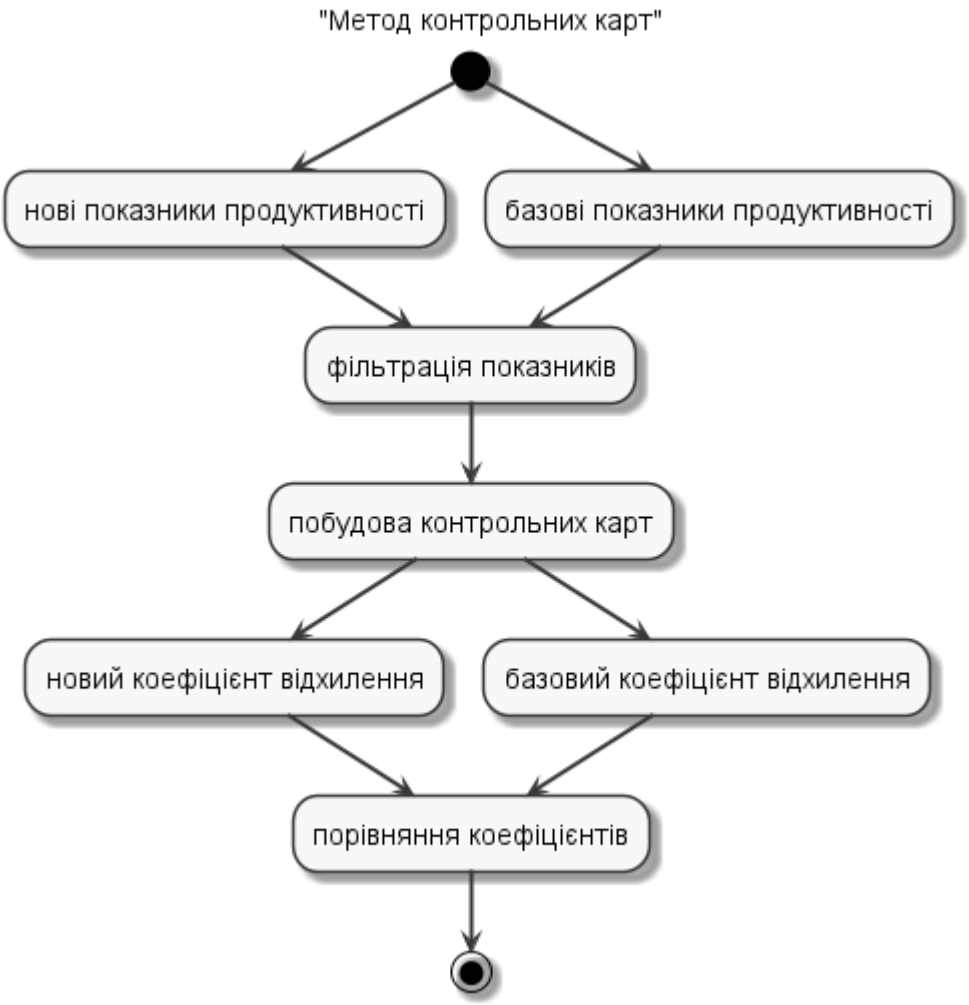


Рисунок 2.1 – Схема роботи методу контрольних карт

Для використання методу контрольних карт необхідно забезпечити стабільність вхідних процесів (системного навантаження) та нормальний розподіл вихідних даних (системних показників). В контексті аналізу навантажувальних тестів такі умови не виконуються. На етапі тестування генератор навантаження може мати випадковий характер та здійснювати різну кількість запитів до системи у кожному окремому тесті. Більш того, реальне навантаження на систему з боку користувачів також може змінюватись с часом. Одним з підходів порівняння системних показників за різних навантажень є застосунок машинного навчання для побудови лінійної моделі залежності навантаження та отриманих показників продуктивності [21]. Результатом виконання навантажувальних тестів є набір системних показників, які частіше за все не мають нормальний характер розподілу. Таким чином для застосування методу контрольних карт в аналізі таких даних необхідно провести деякі коригування. Для рішення цієї проблеми необхідно відфільтрувати значення у відповідності до локального мінімуму двомодальної моделі та отримати нормальний розподіл системних показників продуктивності [21].

Для застосування методу контрольних карт в області аналізу відхилень продуктивності мікросервісних додатків необхідно задовільнити дві вимоги до показників продуктивності. По-перше, системні показники повинні мати нормальний розподіл або має бути можливість скоригувати показники до нормального розподілу. По-друге, необхідно, щоб залежність значень системних показників від навантаження мала лінійний характер. Підсумок характеристик методу контрольних карт приведено у таблиці 2.2.

Таблиця 2.2 – Підсумок характеристик методу контрольних карт

Швидкість виявлення	Адаптивність	Інтеграція показників продуктивності	Складність	Наявність історичних даних
1 прогін	Так	Так	3 етапи	Так

2.3 Метод асоціативних правил

Першим етапом алгоритму методу асоціативних правил є нормалізація вхідних даних, для цього необхідно витягнути значення системних показників продуктивності які відповідають очікуваній тривалості тесту. Потім, розділити час на рівні інтервали, таким чином, щоб кожний інтервал відповідав середньому значенні вектору показників у цьому інтервалі. На наступному етапі необхідно провести дискретизацію даних для побудови асоціативних правил. Третій етап представляє собою виведення асоціативних правил шляхом виявлення частих кореляцій між показниками продуктивності системи [22].

Асоціативне правило складається з попередника та наступника. Правило визначає що у випадку збігу попередників збіг наступників має велику вірогідність [23] Кожний набір асоціативних правил описує поведінку навантажувального тесту. Таким чином для виявлення відхилень продуктивності необхідно зіставити асоціативні правила для набору базових даних з асоціативними правилами нового навантажувального тесту та перевірити зв’язки попередників та наступників. Впевненість асоціативного правила визначає відсоток відповідності попередників наступникам. Затребуваність асоціативного правила вказує наскільки часто набір певних показників продуктивності зустрічається у наборі даних [23]. У випадку коли впевненість асоціативного правила відхиляється від базових показників більше певного порогу відбувається реєстрація регресії продуктивності.

Для аналізу правил співвідношення системних показників використовується алгоритм Apriori [24]. Apriori алгоритм вираховує значення затребуваності та впевненості для асоціативних правил. Для покращення алгоритму виявлення регресії асоціативні правила що мають значення затребуваності нижче 0,3 чи значення впевненості нижче 0,9 не враховуються. Зменшення цих порогових значень може призвести до значного зросту кількості асоціативних правил, що в свою чергу зменшить вірогідність відповідності показників нового тесту показникам історичних даних. Для виявлення можливих регресії продуктивності необхідно порівняти значення впевненості асоціативних правил для набору історичних даних зі

значеннями впевненості асоціативних правил нового тесту. Загальну схему роботи методу асоціативних правил зображено на рисунку 2.2.

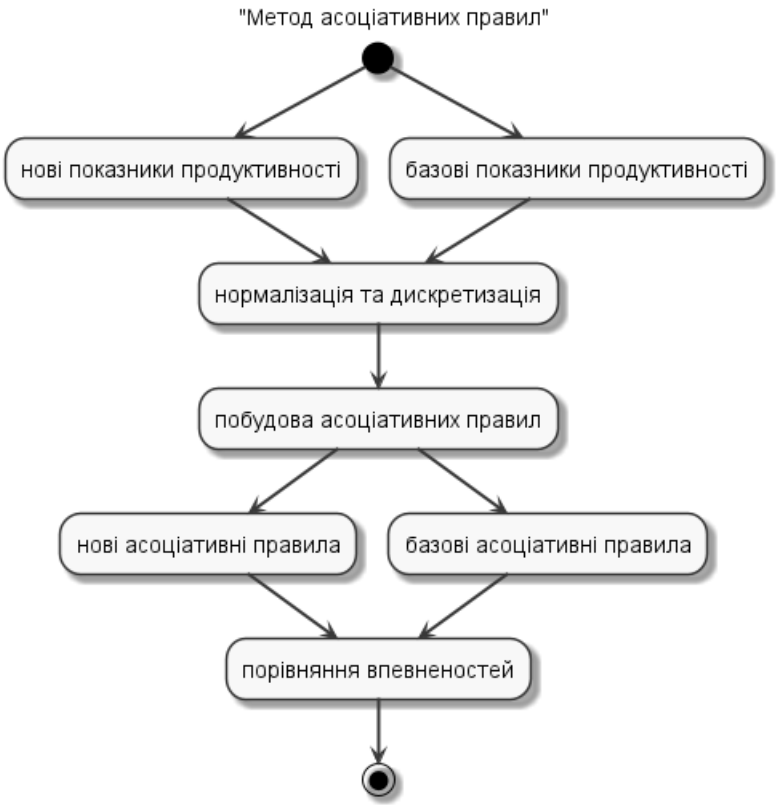


Рисунок 2.2 – Схема роботи методу асоціативних правил

Різниця у значеннях впевненості для асоціативного правила буде мати значення між 0 та 1. Значення нуля вказує на те, що впевненість окремого правила не змінилась у новому тесті, з іншого боку, значення одиниці сигналізує про значні відхилення впевненості правила у новому тесті. Підсумок характеристик методу асоціативних правил приведено у таблиці 2.3.

Таблиця 2.3 – Підсумок характеристик методу асоціативних правил

Швидкість виявлення	Адаптивність	Інтеграція показників продуктивності	Складність	Наявність історичних даних
1 прогін	Ні	Так	3 етапи	Так

3 РОЗРОБКА СИСТЕМИ МОНІТОРИНГУ ПРОДУКТИВНОСТІ ДОДАТКІВ

У цьому розділі описується загальний підхід до розробки системи моніторингу продуктивності додатків з мікросервісною архітектурою. Сформульовано основні вимоги розроблювальної системи, представлено опис робочих процесів системи, структурну схему системи, а також загальну архітектуру та опис окремих модулів системи.

Робочий процес системи моніторингу продуктивності додатків з мікросервісною архітектурою зображено на рисунку 3.1. Система спроектована для додатків з мікросервісною архітектурою, не залежить від кількості сервісів, та особливостей їх реалізації.

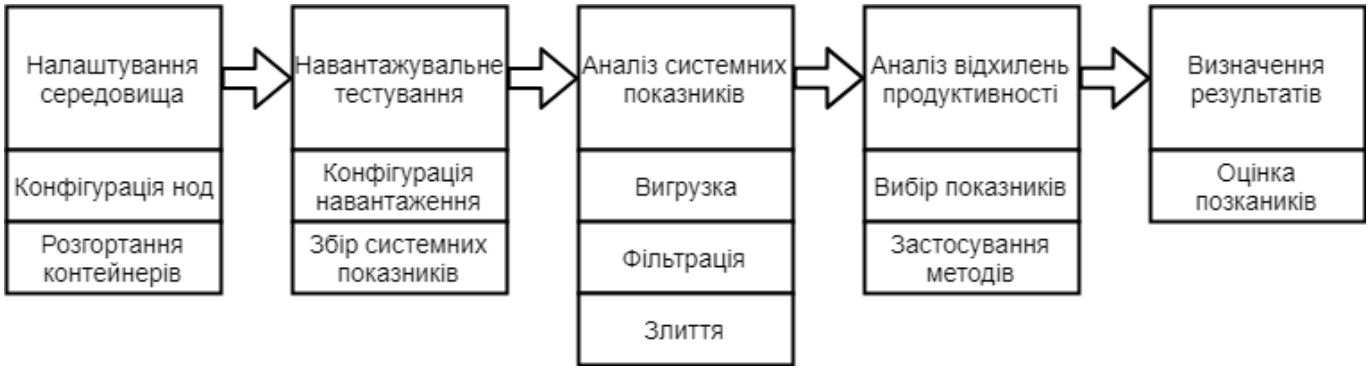


Рисунок 3.1 – Робочий процес системи моніторингу продуктивності

Першим кроком робочого процесу є налаштування середовища для розгортання окремих мікросервісів, налаштування нод кластеру та масштабування окремих контейнерів. Приклад файлу налаштування мікросервісу наведено у додатку А. Наступним кроком є проведення навантажувального тестування мікросервісного додатка. Окрім мікросервісів самого додатка в кластері також присутні спеціальні сервіси збору та збереження показників продуктивності системи. Після проведення навантажувального тестування проводиться завантаження системних показників. Показники продуктивності окремих сервісів системи передаються на аналіз методам виявлення відхилень продуктивності.

Структурну схему системи моніторингу продуктивності додатків з мікросервісною архітектурою наведено у додатку Б. Система спроектована для запуску на локальній машині розробника або на сервері неперервної інтеграції. Система має інтерфейс командного рядка та використовує файлову систему хост-машини в якості основного сховища даних.

На структурній схемі виділено області повноважень різних частин системи. Простір *Monitoring System* включає компоненти, що складають фундамент системи моніторингу та були основним фокусом розробки в даній роботі. Система моніторингу продуктивності має дві основні точки взаємодії з користувачем, це модуль *StressTool*, який використовується для генерації системного навантаження на мікросервісний додаток та модуль *Engine*, основною задачею якого є аналіз системних показників продуктивності мікросервісів.

В основі реалізації модуля генерації системного навантаження лежить інтеграція зовнішньої системи - Locust.io [25], що представляє собою масштабований інструмент навантажувального тестування. Locust.io реалізує архітектуру master-slave [26] для забезпечення запуску розподіленого тестування на декількох хост-машинах. Для цього необхідно налаштувати та запустити master вузол, для того, щоб мати можливість відстежувати статистику породженого навантаження, та декілька worker вузлів для безпосередньої симуляції встановленої кількості користувачів та навантаження на додаток.

Простір *Kubernetes* містить Docker контейнери, якими безпосередньо керує Kubernetes. Такі контейнери включають в себе мікросервіси додатка, для якого необхідно проводити моніторинг продуктивності та додаткові сервіси, які є необхідними для роботи системи моніторингу продуктивності.

Такі додаткові сервіси виділено у підпросторі *Add-ons*, сервіси забезпечують зчитування та збереження статистичних даних продуктивності мікросервісів додатка. Для агрегації актуальних показників продуктивності мікросервісів використовується Kubernetes надбудова – Heapster [27], основною задачею якого є збір та інтерпретація різних сигналів мікросервісів, таких як обчислення використання ресурсів, події життєвого циклу та інше. Для збереження агрегованих даних використовується база

даних InfluxDB [28], яка використовує мову запитів з вбудованими функціями для роботи з часом і структурою даних що складається з вимірів, серій та точок даних [29]. Приклад сценарію встановлення зв’язку між Hepaster та InfluxDB показано на рисунку 3.2.

```
spec:
  serviceAccountName: heapster
  containers:
  - name: heapster
    image: gcr.io/google_containers/heapster-amd64:v1.4.0
    imagePullPolicy: IfNotPresent
    command:
    - /heapster
    - --source=kubernetes:https://kubernetes.default
    - --sink=influxdb:http://monitoring-influxdb:8086
    resources:
      limits:
        cpu: 100m
        memory: 128Mi
      requests:
        cpu: 100m
        memory: 128Mi
```

Рисунок 3.2 – Сценарій встановлення зв’язку між Hepaster та InfluxDB

Система моніторингу продуктивності взаємодіє з Kubernetes кластером через два модулі - *Stress Tool* та *Storage.* Для перевірки пропускної здатності мікросервісного додатка користувач системи через інтерфейс командного рядка формує запит стресового навантаження, тим самим налаштовуючи модуль *Stress Tool* на генерацію встановленої кількості запитів до шлюзу додатка.

Після закінчення часу тестування, користувач системи може запустити модуль аналізу показників продуктивності мікросервісів. Модуль *Engine* має залежність на модуль сховища *Storage*, який виконує запити до бази даних InfluxDB та загружає актуальні показники продуктивності мікросервісів, після чого зберігає їх до локального сховища в файловій системі хост-машини, на якій було запущено систему моніторингу продуктивності, для подальшого моніторингу продуктивності додатка.

3.1 Формування вимог до системи

Основна задача системи моніторингу продуктивності додатків з мікросервісною архітектурою – це забезпечення встановленого рівня продуктивності на етапі випуску нових версій продукту.

При розробці програмного забезпечення часто використовують метод неперервної інтеграції системи, який полягає у виконанні частих автоматизованих складань проекту для якнайшвидшого виявлення та вирішення інтеграційних проблем [30]. Приклад циклу розробки за методом неперервної інтеграції зображено на рисунку 3.3.

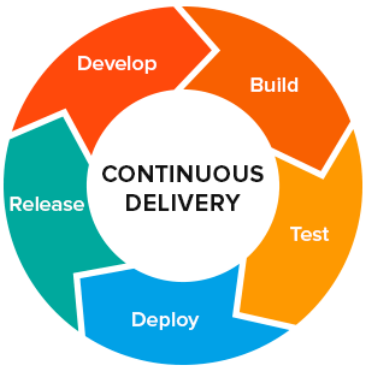


Рисунок 3.3 – Цикл розробки за методом неперервної інтеграції [31]

Практика застосування методу неперервної інтеграції (CI) дозволяє командам розробників швидко тестувати та впроваджувати зміни до системи. Довгі цикли впровадження мають схильність нести велику кількість недоліків, бути більш ненадійними, та не задовольняти кінцевим вимогам.

В процесі неперервної інтеграції розробники часто вносять зміни до коду системи в окремих гілках системи керування версіями, та зливають напрацювання до основної гілки наприкінці циклу розробки. Після введення усіх змін відбувається тестування з метою виявлення можливих регресій продуктивності нової версії додатка. Така практика дозволяє значно зменшити об’єм роботи на наступних фазах розробки та впевнитись в тому, що кодова база залишається в стабільному стані.

Метод неперервної інтеграції також має декілька недоліків. Система контролю версій повинна підтримувати можливість злиття декількох проміжних змін до одного цілого, для того щоб уникнути великої кількості збірок версій системи за незначних змін. Тестування повинно бути повністю автоматизовано та достатньо надійно, організовано таким чином, щоб додаток мав можливість продовжувати функціювати навіть за значних змін коду системи. Тестування також має бути ефективним, після внесення змін до системи, розробники повинні мати можливість обрати конкретні сценарії тестування, які зможуть надати корисну інформацію про наслідки внесених змін до поведінки системи.

Структуруючи систему як набір незалежних модулів можливо досягти значних покращень в швидкості зборки системи, паралельно запускаючи зборку декількох модулів, а також оптимізувати час виконання тестових сценаріїв, ізолюючи тестування лише до модулів які зазнали змін.

Таким чином, система моніторингу продуктивності повинна мати засоби налаштування ізольованого тестування окремих модулів додатка, бути стійкою до значних змін в коді додатка та ефективно проводити тестування додатка на предмет можливих відхилень продуктивності. Необхідно мати можливість інтегрувати систему моніторингу в середовище безперервної доставки (CD).

Мета системи моніторингу продуктивності – виявити проблеми швидкодії додатка безпосередньо на етапі навантажувального тестування. Зокрема, визначити підсистеми які є першопричинами погіршення продуктивності додатка. Таким чином система моніторингу продуктивності повинна дозволити розробникам автоматизувати процес аналізу змін продуктивності додатка під час навантажувального тестування.

3.2 Сценарії використання системи

Зазвичай процес навантажувального тестування за методом неперервної інтеграції виконується за наступними кроками. По перше, розробник, який є відповідальним за випуск нових версій додатка, проводить налаштування та

розгортає попередню та нову версію системи у тестовому середовищі. На наступному етапі інженер з тестування налаштовує штучне навантаження на обидві версії системи, навантаження повинно бути близьким до реального навантаження на систему з боку користувачів та включати в себе перевірку більшості функціонала додатка. Тестування нової та старої версії додатка проводиться за абсолютно ідентичних умов. Після виконання тестів, інженер збирає отримані показники додатка за різних версій та передає результати на аналіз інженеру, який відповідає за продуктивність додатка. Інженер продуктивності на основі отриманих результатів проводить порівняльний аналіз показників продуктивності різних версій додатка. Діаграму сценаріїв навантажувального тестування приведено на рисунку 3.4.

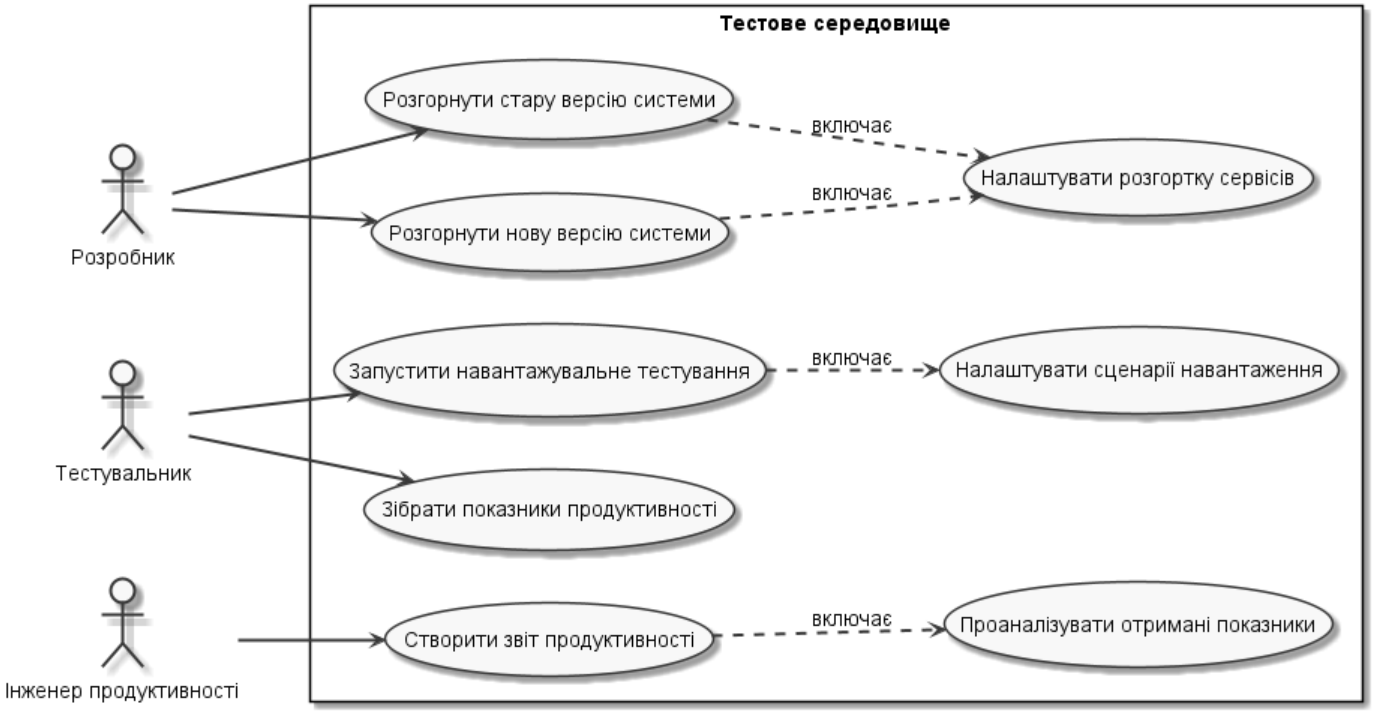


Рисунок 3.4 – Діаграма сценаріїв навантажувального тестування

Для визначення можливих відхилень продуктивності в новій версії додатка, інженер продуктивності проводить аналіз змін окремих системних показників. Перш за все інженер обирає базовий показник продуктивності в якості залежної змінної, наприклад, показник степені навантаження центрального процесора. Вибір показника може бути обґрунтовано досвідом інженера, інженер може обрати показники які

стабільно визначають реальні регресії продуктивності. На основі показників центрального процесора інженер будує базову модель продуктивності додатка, яку використовує надалі для перевірки відповідності показників нової версії додатка бажаним значенням. Такий підхід має значний недолік, адже відхилення продуктивності можуть бути викликані деградацією зовсім інших показників, які мають низьку кореляцію з показниками центрального процесора. Так, наприклад, якщо до нової версії додатка було підключене надмірне логування, показники навантаження файлової системи будуть сильно відрізнятись від базових показників, та стануть причиною відхилення продуктивності усього додатка. Зі зростом кількість агрегованих даних застосування порівняльного аналізу на основі побудови моделей окремих показників продуктивності стає ще менш надійним засобом ефективного виявлення регресій.

Система моніторингу продуктивності інтегрується на етапі запуску навантажувального тестування. Для додатків з мікросервісною архітектурою система повинна мати засоби гнучкого налаштування тестування окремих сервісів, які зазнали значних змін у новій версії додатка. Отже після налаштування тестового сценарію та запуску навантажувального тестування необхідно зібрати показники продуктивності сервісів додатка у мікросервісному середовищі. Для цього необхідно реалізувати підтримку популярних засобів моніторингу мікросервісних додатків та інтегрувати зчитування показників продуктивності у реальному часі. На етапі дослідження отриманих показників продуктивності необхідно враховувати зміни та відхилення усіх системних показників додатка для уникнення проблем модельного підходу до аналізу відхилень продуктивності. Діаграму сценаріїв використання системи моніторингу продуктивності приведено на рисунку 3.5.

Система моніторингу продуктивності має абстрагувати задачі виконання навантажувального тестування та оцінки зміни стану додатка між різними версіями. Розробнику достатньо написати сценарії генерації навантаження на сервіси, які зазнали значних змін у новій версії додатка. Така гнучка система налаштування дозволяє уникнути перенавантаження додатка, та зосередитись на підсистемах тестування яких представляє найбільший інтерес. Більш того, система моніторингу

автоматизує процес збору актуальних показників продуктивності окремих мікросервісів та має засоби збереження та подальшого аналізу отриманої інформації. Для дослідження потенційних змін в показниках продуктивності мікросервісів система моніторингу задіює ефективні методи виявлення відхилень продуктивності та в разі виявлення регресії презентує набір актуальних системних показників, що мають ознаки деградації для подальшого пошуку першопричини появи регресії продуктивності.

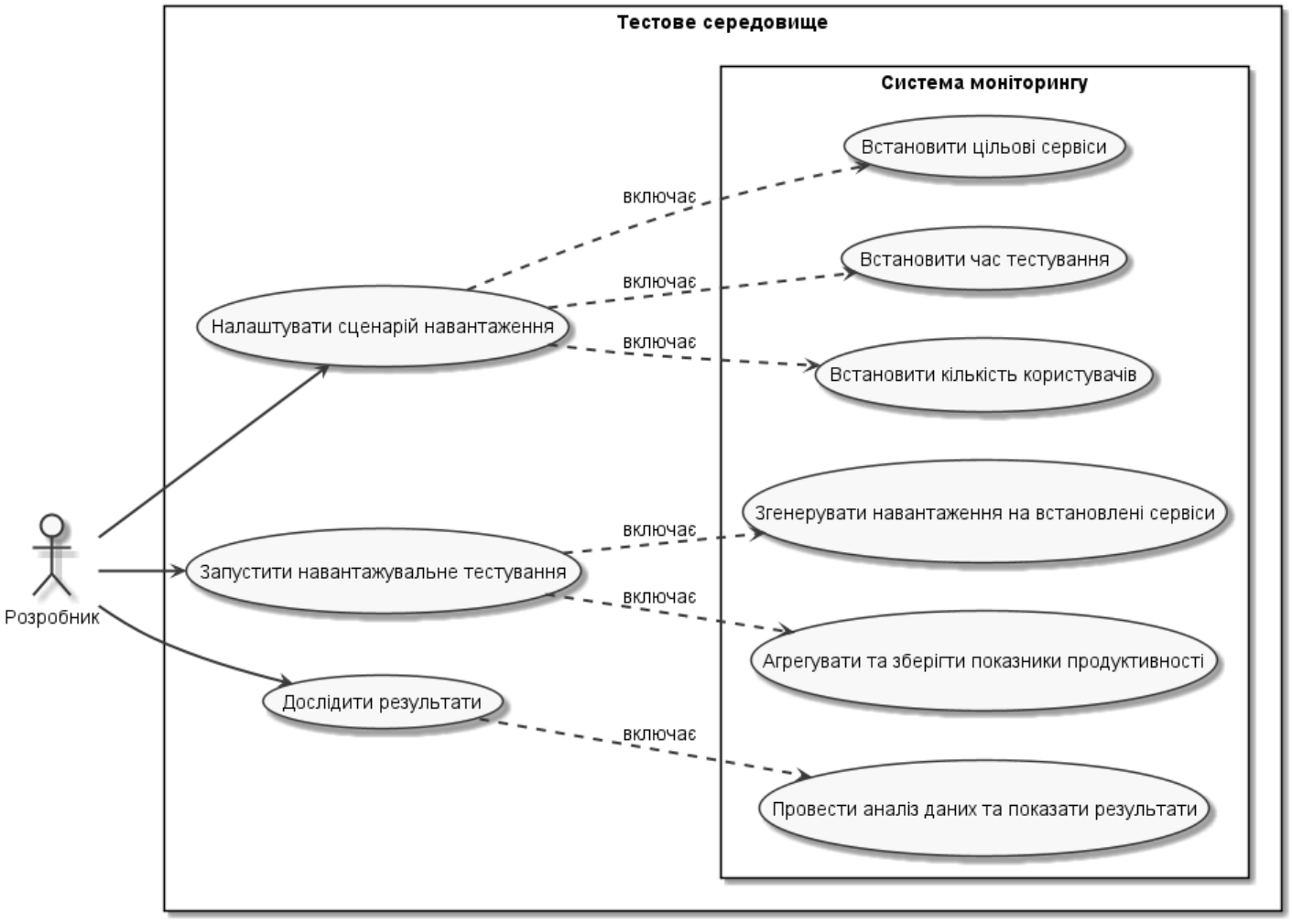


Рисунок 3.5 – Діаграма сценаріїв використання системи моніторингу продуктивності

Таким чином система моніторингу продуктивності мікросервісних додатків автоматизує процеси налаштування тестових сценаріїв при проведенні навантажувального тестування програмних систем. Система моніторингу дозволяє розробникам зосередитись на аналізі актуальних даних про зміни продуктивності додатка на будь-якому етапі тестування додатка, та не перейматись питаннями налаштування тестового середовища та збором показників продуктивності окремих мікросервісів додатка.

Для того щоб не сповільнювати процес безперервної доставки ефективність системи моніторингу продуктивності не повинна залежати від часу обробки результатів. Деякі методи виявлення відхилень продуктивності у своїй основі мають не оптимальні алгоритми аналізу статистичних даних продуктивності додатків. Такі методи не допустимо застосовувати при проектуванні системи моніторингу продуктивності для додатків з мікросервісною архітектурою. Кількість мікросервісів в такому середовищі може бути легко масштабована до декількох сотень реплікацій відповідно до навантаження на систему і таким чином об’єми статистичних даних продуктивності будуть значними і використання не оптимальних алгоритмів аналізу показників продуктивності у такому сценарії може призвести до не бажаних наслідків. В процесі неперервної інтеграції час виконання повного циклу доставки змін в програмному забезпеченні грає вирішальну роль, тому важливо обережно підходити до вибору методів виявлення відхилень продуктивності програмних систем, беручи до уваги час виконання основних кроків алгоритмів окремих методів.

Таким чином необхідно реалізувати рентабельні методи виявлення відхилень продуктивності. Час виконання циклу тестування продуктивності мікросервісної системи не має значно перевищувати час виконання навантажувальних тестів. Також система повинна мати досить високу точність виявлення потенційних регресій продуктивності мікросервісних додатків.

3.3 Архітектура системи

Система моніторингу продуктивності мікросервісних додатків складається з чотирьох модулів: модуль стресового тестування (stress-tool), модуль сховища (storage), модуль типів (model) та модуль аналізу відхилень продуктивності (engine).

На рисунку 3.6 зображено модулі системи та залежності між ними. Основу бізнес логіки системи моніторингу складають модулі стресового тестування та аналізу відхилень продуктивності. Основними класами за допомогою яких розробник може налаштовувати систему моніторингу є класи *StressExecutor* та *EngineExecutor*, які мають інтерфейс командного рядка та відповідають за налаштування навантажувального тестування та аналіз показників продуктивності додатка відповідно. Модулі типів та сховища є допоміжними та використовуються для визначення сутностей та зв'язку з базами даних відповідно. Діаграму класів системи моніторингу продуктивності наведено у додатку В.

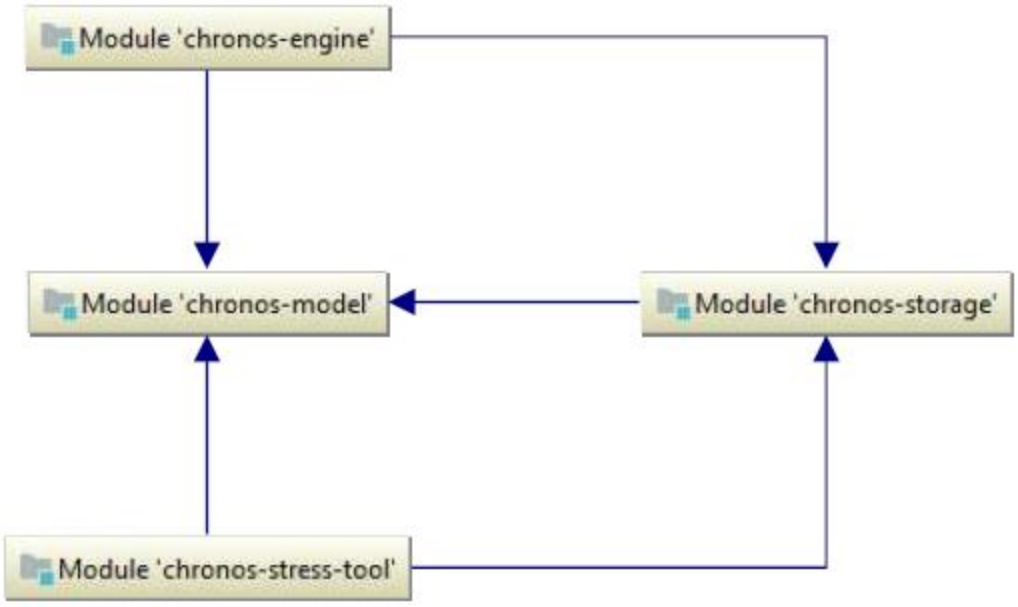


Рисунок 3.6 – Залежності між модулями системи моніторингу

3.3.1 Модуль стресового тестування

Модуль стресового тестування виконує генерацію штучного навантаження на мікросервісний додаток. Модуль було спроектовано таким чином, щоб його можна було налаштовувати для навантажувального тестування широкого спектру мікросервісних додатків. Для того, щоб ефективно використовувати функції модуля стресового тестування необхідно написати сценарій запита на генерацію системного навантаження. Приклад запита генерації системного навантаження показано на рисунку 3.7.

```
class WebTasks(TaskSet):

    @task
    def load(self):
        base64string = base64.encodestring(('%s:%s' % ('user','password')).encode()).decode().replace('\n', '')

        catalogue = self.client.get("/catalogue").json()
        category_item = choice(catalogue)
        item_id = category_item["id"]

        self.client.get("/")
        self.client.get("/login", headers={"Authorization":"Basic %s" % base64string})
        self.client.get("/category.html")
        self.client.get("/detail.html?id={}".format(item_id))
        self.client.delete("/cart")
        self.client.post("/cart", json={"id": item_id, "quantity": 1})
        self.client.get("/basket.html")
        self.client.post("/orders")


class Web(HttpLocust):
    task_set = WebTasks
    min_wait = 0
    max_wait = 0
```

Рисунок 3.7 - Приклад запиту генерації системного навантаження

Модуль стресового тестування є незалежним та має інтуїтивний програмний інтерфейс. Для щоб розробникам було легко інтегрувати навантажувальне тестування в їх процеси перевірки нової версії продукту, модуль має простий інтерфейс командного рядка. Діаграму класів модуля стресового тестування зображено на рисунку 3.8.

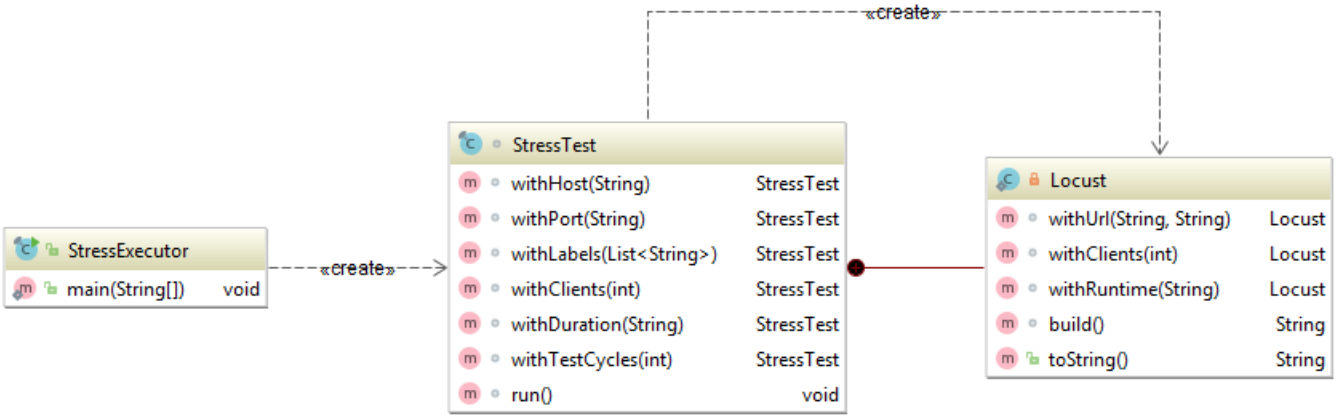


Рисунок 3.8 – Діаграма класів модуля стресового тестування

Модуль стресового тестування представляє собою програму з інтерфейсом командного рядка (CLI). За допомогою інтерфейсу програми можливо налаштувати навантажувальне тестування мікросервісного додатка. Основний клас *StressExecutor* приймає аргументи командного рядка та створює клас *StressTest* з наступними налаштуваннями:

а) *host* – веб адреса за якою було розгорнуто мікросервісний додаток у кластері;

б) *port* – порт сервісу, до якого будуть здійснюватись запити за http протоколом;

в) *labels* – назви окремих сервісів, для яких необхідно зберегти показники продуктивності для подальшого аналізу;

г) *duration* – тривалість навантажувального тестування, значення за замовчуванням 2 години;

д) *clients* – необхідна кількість користувачів системи, значення за замовчуванням 1000;

е) *testCycles* – кількість циклів прогону навантажувальних тестів, значення за замовчуванням 1;

В свою чергу клас *StressTest* інкапсулює допоміжний клас *Locust* який налаштовує запит на генерацію системного навантаження. Також *StressTest* має залежність на модуль сховища через інтерфейси *InfluxService* та *LocalStorage*. За допомогою інтерфейсу *InfluxService* модуль стресового тестування виконує запити до сховища *InfluxDB* для отримання актуальних статистичних даних показників продуктивності певного мікросервісу. Приклад методу класу *InfluxServiceImpl* показано на рисунку 3.9.

```
private InfluxRecord _loadRecord(LoadRun loadRun, String label)
{
   final Map<String, List<DataRecord>> data = new TreeMap<>();
   for (Metrics metric : Metrics.fetchAll())
   {
      final Query query = newQuery("SELECT value FROM $metrics WHERE labels=$label AND time >=
$start AND time <= $end")
            .bind("metrics", metric.signature())
            .bind("label", label)
            .bind("start", loadRun.start())
            .bind("end", loadRun.end())
            .forDatabase("k8s")
            .create();
      final QueryResult queryResult = influxDB.query(query);

      _toSeriesStream(queryResult)
            .forEachOrdered(series ->
                  data.put(metric.signature(),
                        series.getValues().stream()
                              .map(DataRecord::new)
                              .collect(Collectors.toList())));
   }
   return new InfluxRecord(loadRun, data);
}
```

Рисунок 3.9 – Метод класу InfluxServiceImpl

Метод *_loadRecord* виконує запит до бази для отримання показників продуктивності обраного мікросервісу (параметр *label)* за певний період часу навантажувального тестування. Для доступу до бази даних *InfluxDB* використовувалась Java бібліотека з відкритим кодом Influxdb-java [32]. Після збору актуальних даних продуктивності мікросервісів *StressTest* через інтерфейс *LocalStorage* запускає процес збереження показників до локального сховища для подальшого аналізу.

3.3.2 Модуль типів

Усі модулі системи моніторингу продуктивності мають залежність на модуль типів, що описує моделі даних у системі. Основними типами є *Metrics* та *DataRecord*. Діаграму класів перелічуваних типів даних *Metrics*, що описують доступні показники продуктивності мікросервісного наведено у додатку Г. Інтерфейс *Metrics* узагальнює доступ до доступних показників продуктивності мікросервісів. Приклад доступних методів інтерфейсу *Metrics* показано на рисунку 3.10.

```
static List<Metrics> fetchMetrics(Class<? extends Metrics> clazz)
{
   return Arrays.asList(clazz.getEnumConstants());
}

static List<Metrics> fetchAll()
{
   final Stream<CPU> cpuStream = stream(CPU.values());
   final Stream<MEMORY> memoryStream = stream(MEMORY.values());
   final Stream<NETWORK> networkStream = stream(NETWORK.values());
   final Stream<FILESYSTEM> fileSystemStream = stream(FILESYSTEM.values());
   return Stream.concat(
         Stream.concat(cpuStream, memoryStream),
         Stream.concat(networkStream, fileSystemStream))
         .collect(Collectors.toList());
}
```

Рисунок 3.10 – Методи інтерфейсу Metrics

Окрім класів системних показників продуктивності модуль типів містить допоміжні класи, що використовуються для роботи з показниками продуктивності мікросервісів додатка, діаграму класів наведено у додатку Д. Основним класом модуля є клас *DataRecord* який інкапсулює значення показника продуктивності в певний момент часу. Також модуль включає в себе клас *LoadRun*, який відповідає певній ітерації навантажувального тестування та містить показники початку та закінчення циклу тестування. Клас *InfluxRecord* групує зібрані показники продуктивності мікросервісів у відповідності до ітерації навантажувального тестування. Клас *RunResult* містить записи показників продуктивності мікросервісів

з локального сховища та використовується модулем аналізу продуктивності для виявлення можливих регресій.

Також модуль сховища має допоміжний клас *DataRecordMapper* який використовується для перетворення формату даних які були завантажені з *InfluxDB* сховища у строковий формат для подальшого збереження. Приклад методів класу *DataRecordMapper* показано на рисунку 3.11.

```
public static Map<String, String> toString(Map<String, List<DataRecord>> records,
                                           Function<DataRecord, String> mapping)
{
    final String delimiter = ",";
    return new TreeMap<>(records.entrySet().stream()
            .collect(Collectors.toMap(
                    Map.Entry::getKey,
                    e -> e.getValue()
                            .stream()
                            .map(mapping)
                            .collect(Collectors.joining(delimiter)))));
}

public static Map<String, List<DataRecord>> fromString(Map<String, String> records)
{
    final String delimiter = ",";
    return new TreeMap<>(records.entrySet().stream()
            .collect(Collectors.toMap(
                    Map.Entry::getKey,
                    e -> Arrays.stream(e.getValue().split(delimiter))
                            .map(DataRecord::new)
                            .collect(Collectors.toList()))));
}
```

Рисунок 3.11 – Методи класу *DataRecordMapper*

Метод *toString* класу *DataRecordMapper* використовуються для приведення статистичних даних продуктивності мікросервісів до формату який зможе бути оброблено мовою R в реалізації методів t-критерію Стьюдента та методу контрольних карт для статистичного аналізу відхилень продуктивності додатка.

3.3.3 Модуль сховища

Модуль сховища представляє собою набір інструментів для обробки статистичних даних продуктивності мікросервісів з бази даних *InfluxDB* та підтримує операції запису та зчитування таких даних з диску. Діаграму класів модуля сховища наведено у додатку Е.

Модуль сховища має два інтерфейси: *InfluxService*, який використовується модулем стресового тестування для збереження показників продуктивності сервісів системи та інтерфейс *LocalStorage*, який використовується модулем аналізу відхилень продуктивності для зчитування статистичних даних продуктивності сервісів додатка з диска.

3.3.4 Модуль аналізу відхилень продуктивності

Модуль аналізу відхилень продуктивності включає в себе класи, що реалізують наступні методи реєстрації регресій (розділ 2):

а) метод t-критерію Стьюдента реалізує клас *TTestDeviationLocator;*

б) метод контрольних карт реалізує клас *ControlChartsDeviationLocator;*

в) метод асоціативних правил реалізує клас *RulesDeviationLocator*.

Кожен з класів реалізації методів виявлення відхилень продуктивності успадковується від класу *DeviationLocator*, що має засоби додавання статистичних даних продуктивності базових та контрольних тестів. У своїй реалізації класи використовують бібліотеку Rserve [33] для аналізу даних продуктивності. Діаграму класів модуля аналізу відхилень продуктивності наведено у додатку Ж.

Клас *DeviationLocator* має засоби обробки статистичних даних продуктивності мікросервісів, так наприклад метод *_getSteadyRecords* відфільтровує перші 20 значень показників продуктивності, такі дані зазвичай не описують реальну поведінку певного показника продуктивності та часто є нестабільними, метод також фільтрує показники продуктивності значення яких дорівнює нулю для отримання більш чіткої

картини характеру зміни показників продуктивності. Приклад методу класу *DeviationLocator* показано на рисунку 3.12.

```
private List<DataRecord> _getSteadyRecords(List<DataRecord> records)
{
   records.removeIf(dataRecord ->
         Double.parseDouble(dataRecord.getValue()) == 0);

   final List<DataRecord> result = new ArrayList<>();
   for (int k = 20; k < records.size(); k++)
   {
      result.add(records.get(k));
   }
   return result;
}
```

Рисунок 3.12 – Метод класу *DeviationLocator*

Також клас *DeviationLocator* має методи фільтрації показників значення яких мають нульову варіативність. Прикладом таких показників є *cpu/limit, memory/limit.*

3.4 Реалізація бізнес-логіки системи

Система моніторингу продуктивності додатків з мікросервісною архітектурою реалізована за допомогою мови програмування Java. Система має інтерфейс командного рядка для полегшення інтеграції в процес неперервної доставки, систему можливо використовувати разом з Jenkins [34]. Основу бізнес логіки системи складають модуль стресового тестування, який використовуються в якості генератора системного навантаження сервісів додатка та модуль аналізу продуктивності мікросервісів, який використовується для моніторингу продуктивності додатка. На етапі навантажувального тестування розробник має для початку налаштувати та запустити логіку генератора системного навантаження, а вже після цього на етапі аналізу результатів тестування необхідно запустити логіку аналізу зібраних показників продуктивності мікросервісів системи для подальшого дослідження змін поведінки додатка.

3.4.1 Реалізація генератора навантаження

Модуль стресового тестування є відправною точкою для початку роботи з системою моніторингу продуктивності. Для того щоб мати можливість аналізувати показники продуктивності мікросервісів, необхідно для початку згенерувати та застосувати навантаження на додаток.

Перед початком роботи з модулем стресового тестування розробник повинен написати сценарії тестування мікросервісу, поведінка якого представляє найбільший інтерес. Сценарій тестування представляє собою простий сценарій на мові Python, в якому прописуються налаштування генератора навантаження Locust [25]. Розробник має вказати основні сервіси системи для яких плануються проводити аналіз продуктивності, прописати запити до інтерфейсів таких сервісів та передати необхідні параметри авторизації. Також сценарій навантажувального тестування може містити інтеграції з різними допоміжними бібліотеками. У разі необхідності розробник може підключити локальну базу InfluxDB для збору показників виконаного навантаження та їх подальшого аналізу.

Після налаштування сценарію навантаження розробник має запустити сценарій командного рядка *stress_test.sh* та передати параметри навантаження. Після цього буде автоматично приведено в дію процес навантажувального тестування та збору системних показників мікросервісів.

Клас *StressTest* інкапсулює логіку застосування навантажувального тестування та агрегації доступних системних показників в циклі. Кількість ітерацій циклу визначається розробником при передачі параметрів командного рядка. Збільшення кількості ітерації забезпечить більш широкий набір стабільних значень системних показників мікросервісів, але може зайняти багато часу. Приклад методу класу *StressTest* показано на рисунку 3.13.

```
private void _execute(String command)
{
   try
   {
      final String start = ZonedDateTime.now(UTC).format(formatter);
      out.println(format("Starting run on: %s", start));

      final Process p = Runtime.getRuntime().exec(format("cmd /c start /wait %s", command));
      boolean isFinished = p.waitFor(5, TimeUnit.HOURS);

      if (isFinished)
      {
         final String end = ZonedDateTime.now(UTC).format(formatter);
         out.println(format("Finished run on: %s", end));

         _flushToDisk(new HashMap<String, String>()
         {{
            put(start, end);
         }});
      } else
      {
         err.println(format("Command: %s took too long or failed to execute", command));
      }
   } catch (IOException | InterruptedException e)
   {
      err.println(format("Exception on command: %s message: %s", command, e.getMessage()));
   }
}
```

Рисунок 3.13 – Метод класу *StressTest*

Кількість ітерації виклику методу *_execute* класу *StressTest* визначається параметром *testCycles*, що розробник передає під час налаштування навантаження. Метод *_execute* викликає системний процес для запуску сценарію навантаження за тайм-аутом в п’ять годин, якщо навантаження триває довше п’яти годин ітерація переривається та записуються дані за п’ять годин тестування. Така логіка була введена для уникнення висячих процесів. Після закінчення часу тестування відбувається запис показників продуктивності мікросервісу за визначений період до локальної бази системи для подальшого аналізу на наявність потенційних відхилень продуктивності.

Процес агрегації показників продуктивності мікросервісів та подальшого запису до бази виконується на кожній ітерації навантаження. Систему було спроектовано таким чином для уникнення втрат даних у разі відмови нод кластеру та краху додатка. Одна ітерація штучного навантаження може займати декілька годин,

так наприклад, за умови запуску навантажувального тестування на 10 ітерацій по 4 години, у разі успішного виконання дев’яти ітерацій та виникнення системної помилки на десятій можна втратити дані тестування за 36 годин. За існуючого підходу не буде втрачено нічого, усі показники продуктивності мікросервісу за 36 годин буде успішно збережено до локальної бази даних. Діаграму послідовності процесів модуля стресового тестування наведено у додатку И.

3.4.2 Реалізація моніторингу продуктивності

Модуль моніторингу продуктивності було реалізовано за допомогою мови програмування Java, що використовувалась для створення програмного інтерфейсу модуля а також інтеграції з іншими частинами системи та зовнішніми бібліотеками, а також мови програмування R, що використовувалась для статистичного аналізу показників продуктивності мікросервісів системи за допомогою різних методів виявлення відхилень продуктивності а так для побудови інформативних графіків характеру розподілу значень показників продуктивності окремих мікросервісів.

Після закінчення стресового тестування нової версії додатка розробник має провести аналіз змін поведінки додатка під навантаженням з новою версією за допомогою системи моніторингу продуктивності. Для цього розробник має запустити сценарій командного рядка *analyse_performance.sh* та передати назви мікросервісів для яких планується перевірка відхилень продуктивності. Інтерфейс програмного рядка було розроблено для полегшення задач інтегрування системи моніторингу до процесів неперервної доставки, таким чином розробник може легко автоматизувати процес запуску аналізу показників продуктивності системи методом налаштування автоматичної задачі запуску навантажувальних тестів.

Клас *LocalExecutor* інкапсулює логіку загрузки показників продуктивності з локальних сховищ та подальшого аналізу потенційних регресій за допомогою ефективних методів виявлення відхилень продуктивності. Приклад методу класу *LocalExecutor* показано на рисунку 3.14.

```
void run()
{
   final LocalStorage repository = LocalStorage.create(format("%s/%s", REPOSITORY_PATH, config));

   final RunResult latestRecord = _getLatestRecord();
   final List<RunResult> baseRecords = _getBaseRecords(repository);

   if (!baseRecords.isEmpty())
   {
      final List<String> deviatedMetrics = _checkDeviations(latestRecord, baseRecords);

      if (deviatedMetrics.isEmpty())
      {
         log.info("No regression detected.");
         repository.persist(latestRecord);
      } else
      {
         log.debug("Regression detected in: {}", deviatedMetrics);
      }
   } else
   {
      repository.persist(latestRecord);
   }
}
```

Рисунок 3.14 – Метод класу *LocalExecutor*

В якості вхідних параметрів клас *LocalExecutor* приймає значення налаштування, що визначає мікросервіси для яких необхідно провести аналіз продуктивності. Наступним кроком є загрузка останніх актуальних показників продуктивності, які були згенеровано під час навантажувального тестування, з локального сховища. Також метод перевіряє наявність базових показників продуктивності у системі, які необхідні для порівняльного тестування, у разі відсутності базових показників система записує актуальні показники в якості базових та аналізу відхилень не відбувається. Така логіка була введена для випадку першого використання системи, розробники повинні спроектувати першу версію системи, яка буде відповідати бізнес вимогам та прогнати холостий аналіз продуктивності для того, щоб заповнити сховище базових показників. Якщо базові показники присутні запускається процес порівняльного аналізу базових та актуальних показників мікросервісу за допомогою методів виявлення відхилень продуктивності. На рисунку 3.15 зображено приклад застосування метода контрольних карт для порівняльного аналізу показників продуктивності.

```
final DeviationLocator charts = DeviationLocatorFactory.build(LocatorType.CONTROL_CHARTS);
repository.stream()
        .peek(r -> log.debug("loaded run: {}", r))
        .map(RunResult::getData)
        .forEachOrdered(charts::addBaseRecords);
charts.addTargetRecords(targetRecord.getData());

log.debug("target run: {}", targetRecord);

final Set<Deviation> chartDeviations = charts.locateDeviation();
log.debug("Chart evaluation result: {}\n", chartDeviations);
```

Рисунок 3.15 – Застосування метода контрольних карт

Реалізація методу контрольних карт передбачає наявність репозиторію базових тестів, тому перед викликом методу *locateDeviation()* значення усіх базових тестів в системі загружаються та групуються для передачі в метод контрольних карт. Результатом роботи методу є список зареєстрованих відхилень продуктивності, клас *Deviation* містить дані про показники продуктивності, що мають ознаки деградації з новою версією системи, та відомості про метод, що зареєстрував відхилення.

В реалізації методів виявлення відхилень продуктивності для аналізу статистичних даних показників продуктивності мікросервісів було використано мову програмування R [35].

R має значні можливості для здійснення статистичних аналізів, включаючи лінійну і нелінійну регресію, класичні статистичні тести, аналіз часових рядів (серій), кластерний аналіз і багато іншого. R легко розбудовується завдяки використанню додаткових функцій і пакетів. Графічні можливості R дозволяють створювати високоякісні графіки з різними атрибутами, зокрема математичні формули і символи [36]. Код системи моніторингу написано на Java, тому для застосування інструментів статистичного аналізу R було використано бібліотеку з відкритим кодом Rserve [32]. Приклад реалізації методу статистичного аналізу t-критерій Стьюдента зображено на рисунку 3.16.

```
private boolean _ttest(String metric, String base, String target)
{
   RConnection c = null;
   try
   {
      c = new RConnection();

      c.eval(format("base = c(%s)", base));
      c.eval(format("target = c(%s)", target));

      c.eval("bm = mean(base)");
      c.eval("test = t.test(target,mu=bm)");

      double pValue = c.eval("test$p.value").asDouble();

      if (pValue < .05)
      {
         log.debug("metric: {} pValue: {}", metric, pValue);
         return true;
      }
      return false;
   } catch (RserveException | REXPMismatchException e)
   {
      throw new RuntimeException("failed during R processing", e);
   } finally
   {
      if (c != null)
      {
         c.close();
      }
   }
}
```

Рисунок 3.16 – Реалізація методу статистичного аналізу t-критерій Стьюдента

В методі *ttest* відбувається порівняння базових та актуальних значень продуктивності окремих системних показників, наприклад *cpu/usage_rate* чи *memory/working_set*. Статистичний аналіз показників проводиться за допомогою пакету Student's t-Test [37]. Діаграму послідовності процесів модуля аналізу продуктивності наведено у додатку К.

4 ТЕСТУВАННЯ СИСТЕМИ

Цей розділ присвячено опису процесу тестування системи моніторингу продуктивності додатків з мікросервісною архітектурою. Перш за все необхідно обрати існуючий додаток з мікросервісною архітектурою для проведення тестування системи моніторингу продуктивності. Такий додаток повинен мати досить просту реалізацію та відкритий код.

Після налаштування тестового середовища та успішного розгортання додатка у системі управління контейнерами необхідно провести навантажувальне тестування та зібрати показники продуктивності усіх сервісів додатка. Для аналізу роботи системи моніторингу тестування проводиться за різних налаштувань сервісів додатка, що включає в себе впровадження штучних регресій.

Останнім кроком є аналіз зібраних показників продуктивності для перевірки ефективності розробленої системи моніторингу. На останньому етапі тестування також проводиться визначення ефективності роботи окремих методів виявлення регресії продуктивності.

Цей розділ представляє детальний опис налаштування тестового середовища, обґрунтування вибору мікросервісного додатка для тестування та аналіз результатів тестування системи моніторингу продуктивності додатків з мікросервісною архітектурою.

4.1 Об’єкт тестування

У цьому розділі розглядається додаток, на якому проводилось тестування системи моніторингу продуктивності додатків з мікросервісною архітектурою. Додаток був обраний за декількома ключовими факторами.

По-перше, важливо мати можливість розгортати сервіси додатка як окремі контейнери та масштабувати їх за мірою необхідності. Додаток має бути у відкритому доступі та не мати складної структури, необхідно мати можливість змінювати окремі частини системи для тестування.

Усім вимогам відповідає SockShop [38] – повністю контейнеризований додаток з мікросервісною архітектурою та відритим кодом, який розробляється компанією Weaveworks [39].

Додаток складається з мікросервісів написаних на Java, Go та Node.js. Документація також пропонує скрипти для розгортання мікросервісів у Docker, Kubernetes та AWS. Архітектуру додатка Sock-Shop зображено на рисунку 4.1.

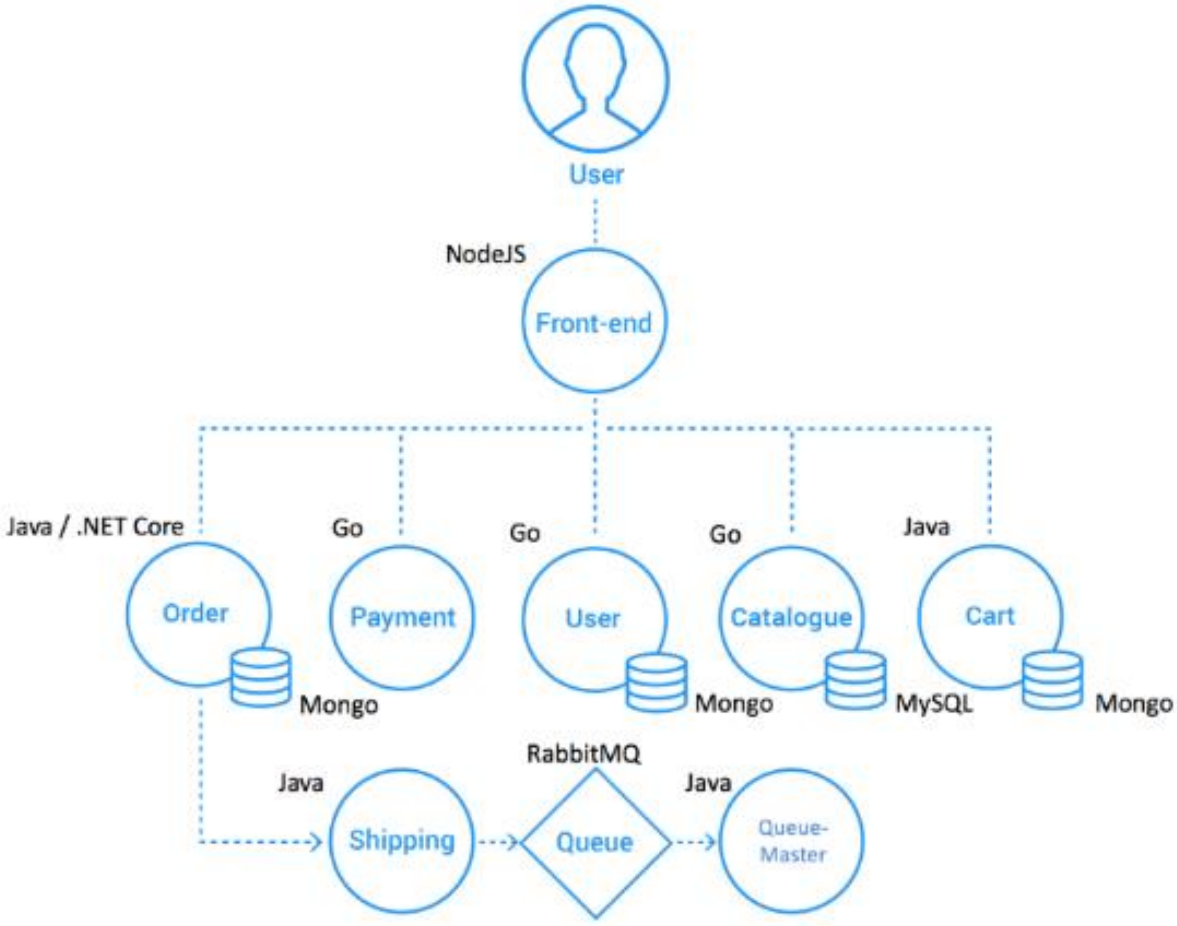


Рисунок 4.1 – Архітектура додатка Sock-Shop [40]

SockShop також має засоби для навантажувального тестування за допомогою Locust [25], що симулює реєстрацію користувача, навігацію по каталогу, здійснення замовлень та оплату товарів.

4.2 Налаштування тестового середовища

Для проведення тестування мікросервіси Sock-Shop було розгорнуто за допомогою Kubernetes та minikube [41] у кластері з однією нодою. Параметри кластеру: 8 гігабайт оперативної пам’яті, 4 ядра центрального процесора.

Мікросервіси які розгортаються у Kubernetes кластері:

а) мікросервіси SockShop;

б) модуль збору показників продуктивності, контейнер Heapster [27], який зчитує актуальні відомості стану сервісів додатка та записує отримані дані до репозиторію;

в) модуль схову, контейнер InfluxDB [28], який використовується для збереження інформації про стан мікросервісів;

Основу кластеру складають контейнери мікросервісів SockShop, які можуть бути автоматично масштабовані у відповідності до навантаження.

Окремо також зображено генератор навантаження Locust, який породжує штучне системне навантаження, симулюючи задану кількість користувачів та посилаючи GET та POST запити до front-end контейнеру.

Heapster контейнер збирає показники продуктивності окремих мікросервісів та записує дані до InfluxDB. Інформація записана до InfluxDB надалі буде використана для аналізу продуктивності мікросервісів. Діаграму модулів тестового середовища наведено у додатку Л.

На діаграмі модулів тестового середовища позначено три основні оточення, це система моніторингу, об’єкт тестування – мікросервісний додаток Sock-Shop, та спільні ресурси системи моніторингу та об’єкта тестування – сховище InfluxDB та модуль збору показників продуктивності Heapster. Оточення системи моніторингу продуктивності містить модулі генерації навантаження, аналізу показників продуктивності та модуль сховища (детальний розгляд архітектури системи наведено в модулі 3.4). Оточення об’єкта тестування містить набір незалежних мікросервісів

налаштованих для роботи у кластері. Статистичні дані продуктивності окремих мікросервісів додатка Sock-Shop збираються модулем Heapster та записуються до спільної бази даних InfluxDB. Система моніторингу продуктивності ініціює процес навантажувального тестування за допомогою налаштування генератора навантаження Locust. Після закінчення етапу навантажувального тестування мікросервісів Sock-Shop запускається модуль аналізу показників продуктивності додатка та за допомогою модуля сховища виконується завантаження актуальних показників продуктивності зі спільної бази даних InfluxDB.

Таке налаштування тестового середовища дозволяє аналізувати продуктивність системи за різних налаштувань окремих мікросервісів. Приклад сценарію розгортки Docker контейнеру для мікросервісів Sock-Shop показано на рисунку 4.2.

```
FROM weaveworksdemos/msd-java:latest

WORKDIR /usr/src/app
COPY *.jar ./app.jar

RUN    chown -R ${SERVICE_USER}:${SERVICE_GROUP} ./app.jar

USER ${SERVICE_USER}

ARG BUILD_DATE
ARG BUILD_VERSION
ARG COMMIT

LABEL org.label-schema.vendor="Weaveworks" \
  org.label-schema.build-date="${BUILD_DATE}" \
  org.label-schema.version="${BUILD_VERSION}" \
  org.label-schema.name="Socks Shop: Cart" \
  org.label-schema.description="REST API for Cart service" \
  org.label-schema.url="https://github.com/microservices-demo/carts" \
  org.label-schema.vcs-url="github.com:microservices-demo/carts.git" \
  org.label-schema.vcs-ref="${COMMIT}" \
  org.label-schema.schema-version="1.0"

ENV JAVA_OPTS "-Djava.security.egd=file:/dev/urandom"
ENTRYPOINT ["/usr/local/bin/java.sh","-jar","./app.jar", "--port=80"]
```

Рисунок 4.2 –Сценарій розгортки Docker контейнеру

Як можна бачити на рисунку 4.2 процес побудови та пакування окремого мікросервісу в Docker контейнер є повністю автоматизованим. В сценарії розгортки прописуються детальні кроки побудови та подальшого використання мікросервісу в контейнерному середовищі. Так на першому рядку сценарію можна бачити застосування створення кореневої файлової системи для контейнеру з з використанням механізму copy-on-write, що дозволяє прискорити розгортання, знижує витрату пам'яті і економить дисковий простір. Змінена файлова система одного контейнера може використовуватися як основа для формування нових базових образів і створення інших контейнерів, без необхідності оформлення шаблонів або ручного налаштування складу образів [5]. Також сценарій має налаштування для автоматичного оновлення мікросервісу у Docker-Hub за умови внесення змін та розгортки нової версії служби у кластері. Завдяки такій гнучкому механізму налаштування розгортки мікросервісів у кластері стає можливим сценарій заміни окремого мікросервісу кластеру без необхідності перезапуску системи в цілому.

Мікросервіси Sock-Shop запущено у Kubernetes кластері, тому є можливість налаштування коефіцієнтів реплікації окремих мікросервісів а також масштабування додатка для проведення тестування системи моніторингу продуктивності при роботі з великими об’ємами показників продуктивності. Також Kubernetes дозволяє розгортати додаткові мікросервіси, які можуть бути використані для збору статистичних даних продуктивності та їх подальшого зберігання, у тому ж середовищі, що й сам додаток.

4.3 Впровадження регресії продуктивності

Для тестування ефективності модуля виявлення відхилень продуктивності необхідно впровадити штучні регресії в кодову базу системи. Тестування проводиться для carts мікросервісу, мікросервіс реалізован на Java та має засоби комунікації з базою даних MongoDB.

Для впровадження були обрані наступні види штучних регресії [42, 43]:

а) надмірне логування;

б) надмірні розрахунки;

в) виділення додаткової пам’яті;

г) некоректне налаштування бази даних;

д) one lane bridge;

е) ramp.

Відхилення продуктивності шляхом надмірного логування досягається за допомогою додавання трасування в більшість методів сервісу, включаючи запис стану об’єктів на всіх етапах обробки.

Додаткові розрахунки у коді який часто викликається можуть призвести до значного зросту використання центрального процесора. Додавання таких кусків коду до декількох методів може викликати значний зріст навантаження процесора.

Імітувати відхилення продуктивності також можна введенням штучних витіків пам’яті, результатом яких буде зменшення доступної програмі оперативної пам'яті.

Неправильне налаштування доступу до бази даних може призвести до зменшення ліміта доступних підключень до бази з 100 до 1.

Для реалізації one lane bridge [43] у коді сервісу штучно проектується вузьке місце для паралельного виконання. Приклад реалізації шаблону one lane bridge для мікросервісу *carts* зображено на рисунку 4.3.

Шаблон ramp [43] імітує зріст часу виконання окремого процесу або зріст часу обробки ресурсу процесом. Приклад реалізації шаблону ramp для мікросервісу *carts* зображено на рисунку 4.4.

```
@RequestMapping(value = "/{customerId}")
public Cart get(@PathVariable String customerId)
{
   synchronized (semaphore)
   {
      return new CartResource(cartDAO, customerId).value().get();
   }
}
```

Рисунок 4.3 – Реалізація шаблону one lane bridge

```
private static final AtomicInteger atomicInt = new AtomicInteger();

@RequestMapping(value = "/{customerId}", produces = MediaType.APPLICATION_JSON_VALUE)
public Cart get(@PathVariable String customerId)
{
   int curVal = atomicInt.incrementAndGet();
   long start = System.currentTimeMillis();
   final Cart res = new CartResource(cartDAO, customerId).value().get();
   long diff = System.currentTimeMillis() - start + 1;
   try
   {
      Thread.sleep((int) Math.max(0, ((diff * curVal * 100 / 90_000.0) - diff) / 2));
   } catch (InterruptedException e)
   {
      e.printStackTrace();
   }
   return res;
}
```

Рисунок 4.4 – Реалізація шаблону ramp

4.4 Системні показники

Необхідно описати за якими системними показниками відбувається аналіз відхилень продуктивності. Основними показниками продуктивності є показники центрального процесора (CPU), показники оперативної пам'яті (Memory), показники мережі (Network) та показники використання файлової системи (Filesystem). Усі дані продуктивності збираються за допомогою Heapster та записуються до InfluxDB сховища.

4.4.1 Показники центрального процесора

Використання ресурсів центрального процесора у Kubernetes описується за допомогою millicores [44], якщо нода має 4 ядра, то місткість такої ноди складає 4000 millicores. Доступні показники центрального процесора приведено у таблиці 4.1. На етапі тестування модуля аналізу показнику продуктивності мікросервісів було виявлено, що значення показників *cpu/limit* та *cpu/request* мають нульову варіативність, тому для аналізу показників центрального процесора використовуються значення показників *cpu/usage_rate.*

Таблиця 4.1 – Показники центрального процесора

Назва	Опис
cpu/limit	Ліміт доступних millicores.
cpu/request	Гарантована кількість ресурсів у millicores.
cpu/request	Використання центрального процесора на всіх ядрах за весь час вимірюється у millicores.
cpu/usage_rate	Використання центрального процесора на всіх ядрах в певний момент часу вимірюється у millicores.
cpu/node_capacity	Об'єм доступних millicores у ноді.
cpu/node_reservation	Зарезервований об'єм millicores у ноді.
cpu/node_utilization	Об'єм задіяних millicores у ноді.
cpu/node_allocatable	Об’єм millicores доступних для обробки

4.4.2 Показники оперативної пам’яті

Використання оперативної пам’яті у Kubernetes визначається в байтах. Доступні показники оперативної пам’яті приведено у таблиці 4.2. На етапі тестування модуля аналізу показнику продуктивності мікросервісів було виявлено, що значення показників *memory/limit* та *memory/request* мають нульову варіативність, тому для аналізу показників оперативної пам’яті використовуються значення показників *memory/major_page_faults_rate, memory/page_faults_rate, memory/usage* та *memory/working_set.*

Таблиця 4.2 – Показники оперативної пам’яті

Назва	Опис
memory/limit	Ліміт пам’яті в байтах.
memory/major_page_faults	Кількість значних похибок сторінок.
memory/page_faults	Кількість похибок сторінок.
memory/major_page_faults_rate	Частота значних похибок сторінок.
memory/page_faults_rate	Частота похибок сторінок.
memory/request	Гарантована кількість доступної пам’яті в байтах.
memory/usage	Загальне використання пам’яті системою в байтах.
memory/working_set	Використання пам’яті робочим процесом в байтах.

Похибки сторінок виникають коли пам’ять є доступною, але має бути оброблена оперативною системою, а значні похибки сторінок виникають у випадку коли системі необхідно завантажувати пам’ять з диску.

4.4.3 Показники мережі

Використання ресурсів мережі у Kubernetes визначається в байтах. Доступні показники мережі приведено у таблиці 4.3. На етапі тестування модуля аналізу показнику продуктивності мікросервісів було виявлено, що значення показників *network/rx, network/rx_errors* та *network/tx, network/tx_errors* через свою природу не можуть відражати зміну поведінки показників мережі, та мають сильно відрізняються за різних версій системи навіть без введення штучних регресій продуктивності, тому для аналізу показників мережі використовуються значення показників *network/rx_rate*, *network/rx_errors_rate* та *network/tx_rate*, *network/tx_errors_rate.*

Таблиця 4.3 – Показники мережі

Назва	Опис
network/rx	Кількість вхідних байтів за весь час.
network/rx_rate	Кількість вхідних байтів за секунду.
network/rx_errors	Кількість помилок вхідного трафіку за весь час.
network/rx_errors_rate	Кількість помилок вхідного трафіку за секунду.
network/tx	Кількість вихідних байтів за весь час.
network/tx_rate	Кількість вихідних байтів за секунду.
network/tx_errors	Кількість помилок вихідного трафіку за весь час.
network/tx_errors_rate	Кількість помилок вихідного трафіку за секунду.

4.4.4 Показники файлової системи

Використання ресурсів файлової системи у Kubernetes визначається в байтах. Доступні показники файлової системи приведено у таблиці 4.4. На етапі тестування модуля аналізу показнику продуктивності мікросервісів було виявлено, що значення показників *filesystem/limit* мають нульову варіативність, тому для аналізу показників файлової системи використовуються значення показників *filesystem/usage.*

Таблиця 4.4 – Показники файлової системи

Назва	Опис
filesystem/limit	Загальний розмір файлової системи в байтах.
filesystem/usage	Загальна кількість байтів, що споживаються файловою системою.

4.5 Аналіз методів виявлення відхилень продуктивності

Для оцінки методів реєстрації регресії продуктивності було проведено навантажувальне тестування мікросервісного додатка (розділ 3.3). Загалом було виконано 50 тестів по 2 години кожний, 20 тестів було проведено за стандартного налаштування сервісів додатка.

З 20 навантажувальних тестів на системі без змін 10 було обрано для створення сховища історичних даних, яке є обов’язковим для деяких методів, 10 тестів було використано для аналізу кількості хибно позитивних відкликів окремих методів виявлення відхилень продуктивності.

Інші навантажувальні тести було проведено за різних налаштувань мікросервісного додатка, з інжекцією штучних регресій (розділ 3.5) до окремих сервісів, по 4 тести на кожен з 6 видів регресій. Налаштування тестового навантаження залишалось незмінним на кожному етапі.

На рисунку 4.5 зображено середні значення відхилень показників використання оперативної пам’яті мікросервісом за різних версій системи.

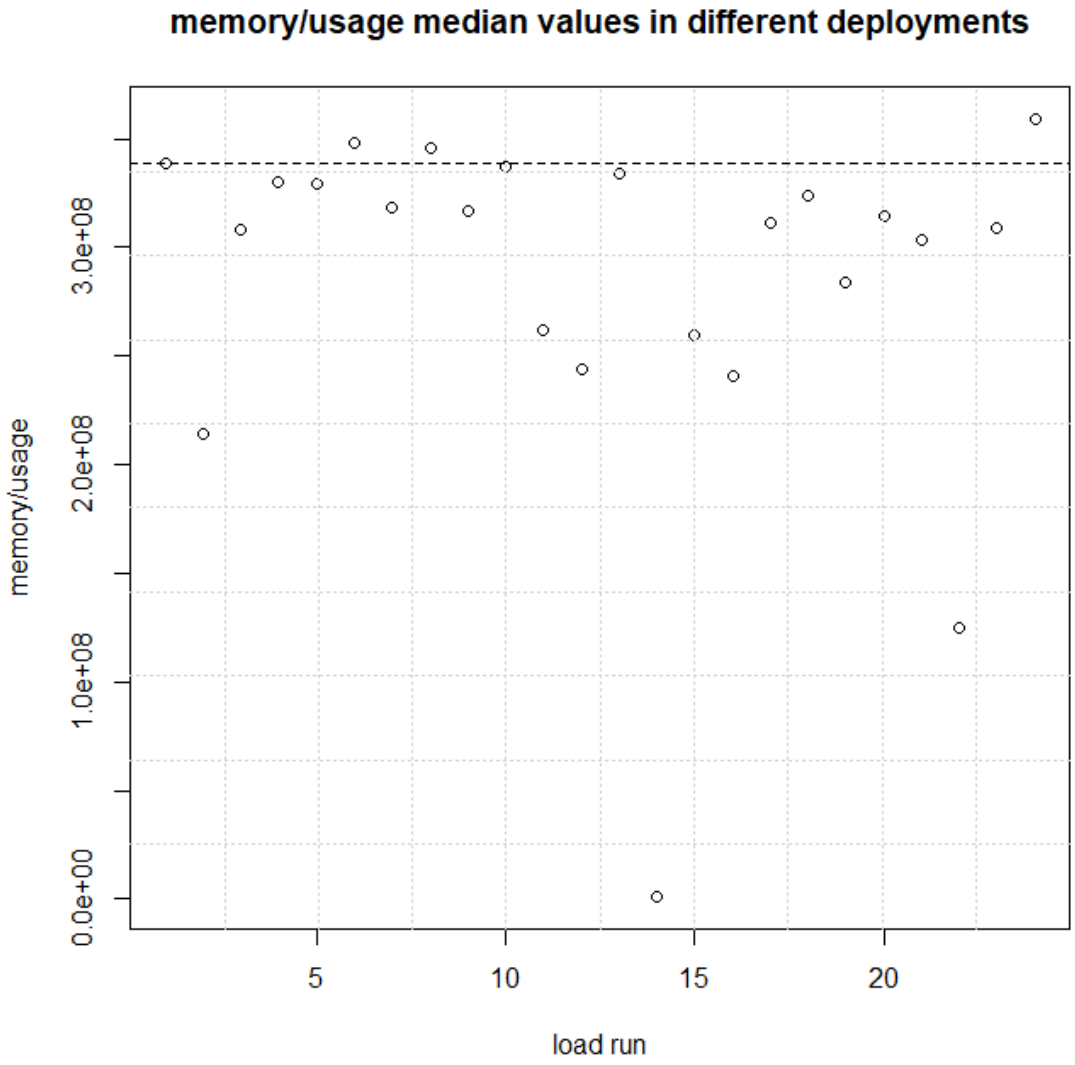


Рисунок 4.5 – Середні значення відхилень показників оперативної пам’яті

Як видно з рисунку 4.5 в основному значення показників утилізації ресурсів оперативної пам’яті мікросервісом carts лежать в межах середнього значення для всіх версій розгортки системи, але також є деякі відхилення, що можуть бути викликанні перезапуском контейнера на певному етапі тестування.

Для перевірки характеру поведінки системних показників мікросервісів було проведено додаткове навантажувальне тестування стабільної версії додатка. Тестування було налаштовано на симуляцію роботи з додатком 1000 користувачів на

протязі 40 годин, було зібрано близько 10000 показників продуктивності центрального процесора, оперативної пам’яті, мережі та файлової системи. На рисунку 4.6 та рисунку 4.7 представлено графіки розподілу значень показників оперативної пам’яті системи для *carts* та *orders* мікросервісів додатка відповідно.

Рисунок 4.6 - Графік розподілу значень показників оперативної пам’яті для *carts* мікросервісу

Рисунок 4.7 - Графік розподілу значень показників оперативної пам’яті для *orders* мікросервісу

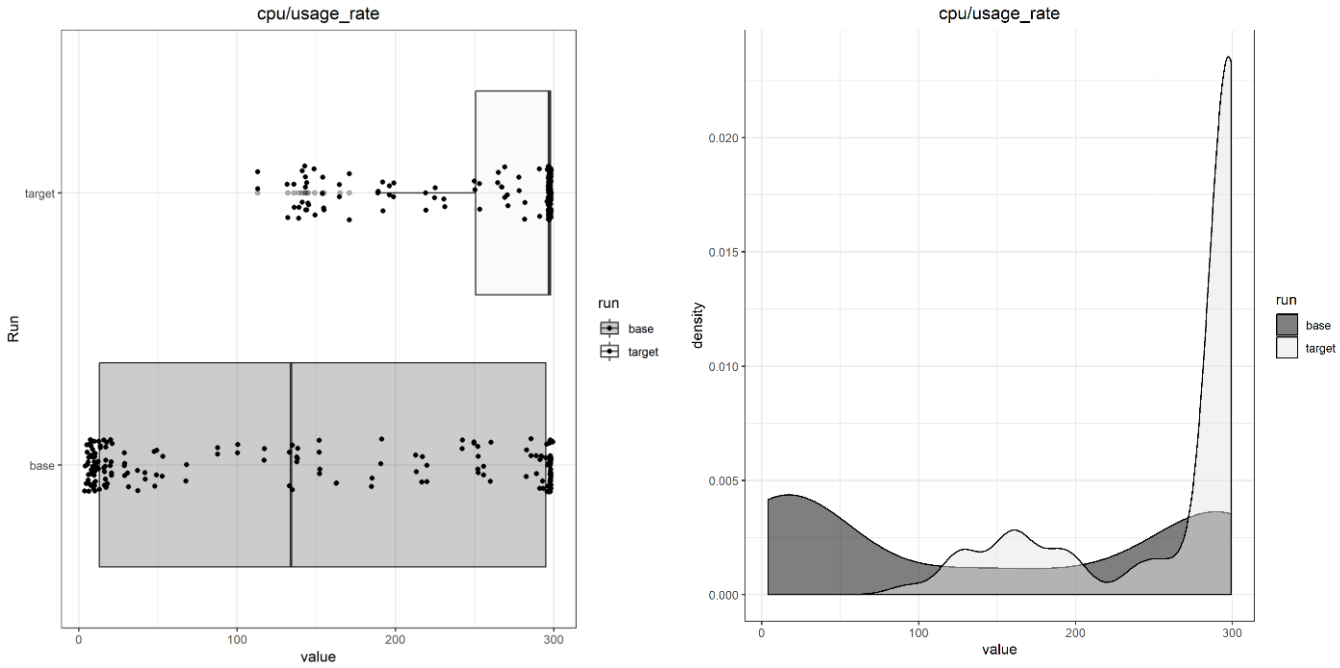
4.5.1 Результати методу статистичних тестів

В якості статистичного тесту було обрано t-критерій Стьюдента. Загалом було проведено порівняння системних показників кожної пари 20 базових тестів без регресій, та порівняння показників одного базового тесту з показниками кожного з 24 тестів з штучними регресіями. Результати тестування наведено у таблиці 4.5.

Таблиця 4.5 – Результати тестування методу t-критерій Стьюдента

ХП	ІН	ІП	ХН	Точність	Повнота	F-міра
15	5	24	0	62%	100%	76%

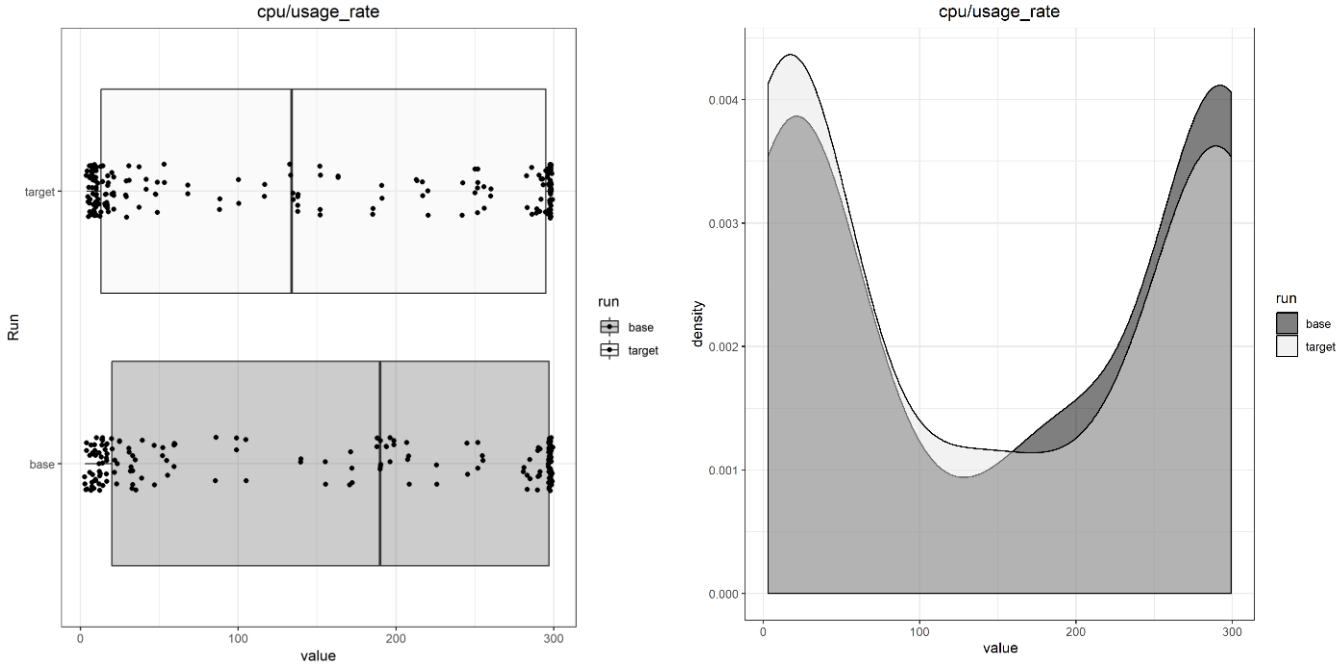
Результати тестування показали високий рівень хибно позитивних спрацювань, метод t-критерію Стьюдента за встановленого рівня значимості 0.05 зареєстрував відхилення продуктивності в 15 з 20 тестів без регресії. Метод також був здатний зареєструвати усі види штучних регресій. Загалом метод t-критерію Стьюдента має 62% точності та 100% повноти, що зумовлено відсутністю хибно негативних спрацювань. Такий результат свідчить про низьку надійність використання методу t-критерію Стьюдента для виявлення відхилень продуктивності у мікросервісному середовищі. Порівняльні графіки розподілу значень навантаження центрального процесора за наявності регресії та без регресії зображено на рисунку 4.8 та рисунку 4.9 відповідно.



а б

Рисунок 4.8 – Розподіл показників центрального процесора за наявності регресії

а) коробковий графік б) графік густини



а б

Рисунок 4.9 – Розподіл показників центрального процесора без регресії

а) коробковий графік б) графік густини

Високий рівень хибно позитивних спрацювань методу t-критерію Стьюдента обумовлено обмеженою кількістю базових показників та характером розподілу вхідних даних. В наявній реалізації метод t-критерію Стьюдента використовує лише показники минулого та наявного тестів для визначення регресій ігноруючи решту репозиторію. Однією з умов застосування даного критерію є нормальний характер розподілу початкових даних. Показники продуктивності окремих сервісів у мікросервісному середовищі не мають нормальний характер розподілу, тому метод часто реєструє регресії навіть якщо їх немає. Метод виявлення відхилень продуктивності з настільки високим значенням хибно позитивних спрацювань може бути використаний лише у сукупності з іншими методами та не може характеризувати поведінку продуктивності мікросервісного додатка в цілому.

4.5.2 Результати методу контрольних карт

Для проведення оцінки роботи методу контрольних карт було сформовано сховище історичних даних з показників продуктивності 10 тестів без регресій. Сховище історичних даних було використано для проведення тестування 20 тестів без регресій та 24 тестів з штучними регресіями, по 4 тести для кожного з 6 видів регресій. Результати тестування наведено у таблиці 4.6.

Таблиця 4.6 – Результати тестування методу контрольних карт

ХП	ІН	ІП	ХН	Точність	Повнота	F-міра
6	14	16	8	72%	67%	69%

Загалом за допомогою методу контрольних карт було зареєстровано 16 з 24 штучних регресій, також метод має 6 хибно позитивних спрацювань. Метод зареєстрував кожен з видів штучних регресій хоча б один раз. В порівнянні з статистичним методом t-критерій Стьюдента метод контрольних карт має кращі результати, набагато нижчий рівень хибно позитивних спрацювань. Метод має досить

високий показник точності 72% та низьку повноту 67%. Тестування показало, що більша частина системних показників не мають нормальний розподіл, що є одним з критеріїв до вхідних даних для методу контрольних карт. Для фільтрування даних з метою приведення їх до нормального розподілу була застосована техніка відсікання показників зліва від локального мінімуму [15]. На рисунку 4.10 зображено порівняльний графік розподілу густини значень показників використання системної пам’яті до та після фільтрування.

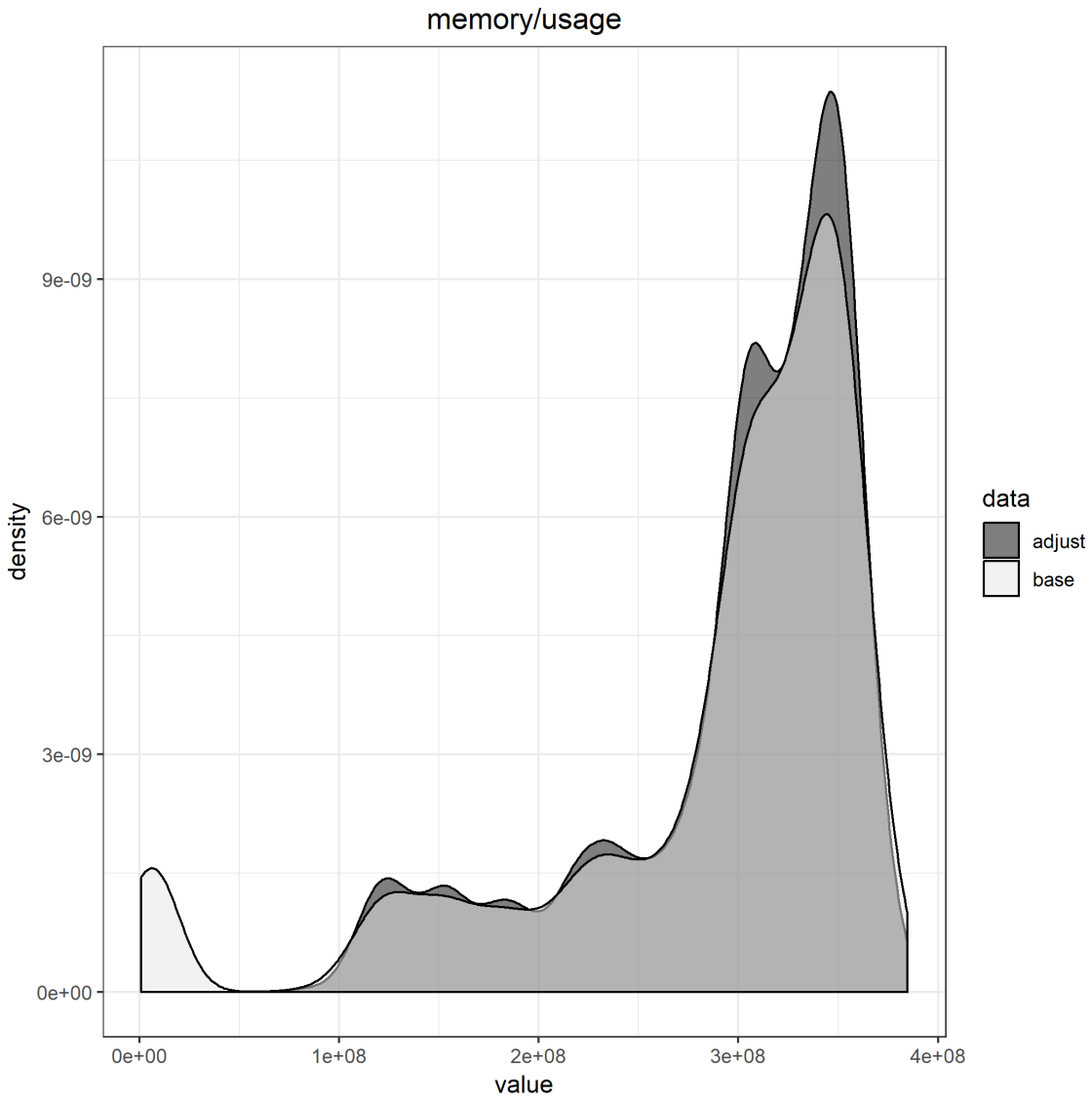
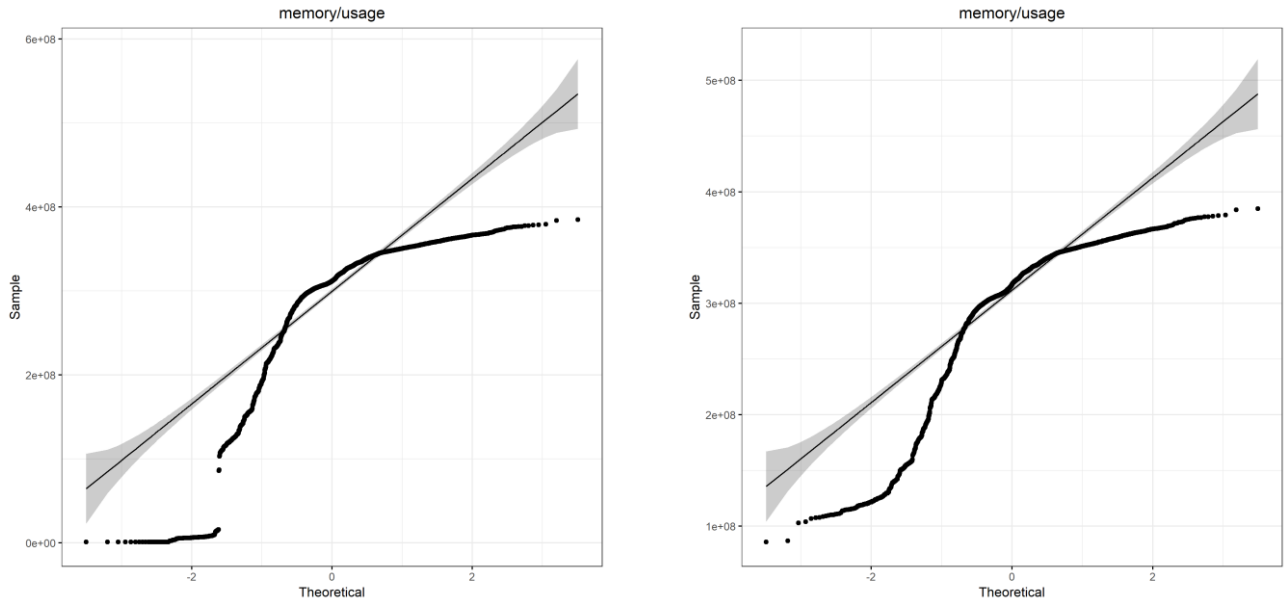


Рисунок 4.10 – Розподіл густини показників системної пам’яті

З графіку на Рисунку 4.10 можна бачити як значення показників продуктивності зліва від першого локального мінімуму було прибрано для отримання більш близького до нормального розподілу значень. На рисунку 4.11 зображено Q-Q графіки відповідності розподілу значень показників системної пам’яті до нормального розподілу до та після фільтрування.



а б

Рисунок 4.11 – Графік розподілу значень показників системної пам’яті
а) до фільтрування б) після фільтрування

З графіків на рисунку 4.11 можна зробити висновок про характер розподілу показників продуктивності показників системної пам’яті мікросервісу *carts*. Необхідно зауважити, що показники продуктивності системного процесора, мережі та файлової системи мають схожий характер розподілу значень. Таким чином метод контрольних карт працює з даними, що не мають нормальний характер розподілу.

Принцип роботи методу контрольних карт зображено на рисунку 4.12 який відповідає випадку відсутності регресії у новій версії сервісу та рисунку 4.13 на якому зображено розподіл значень показників продуктивності системи за якого відбувається реєстрація регресії.

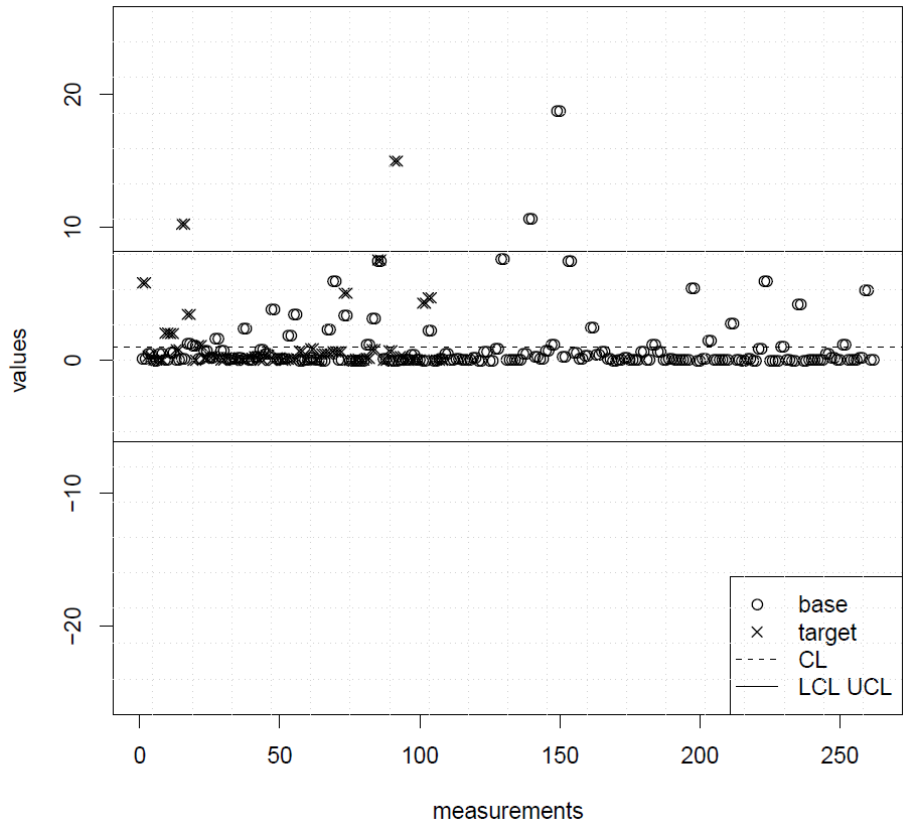


Рисунок 4.12 – Контрольна карта системних показників без регресії

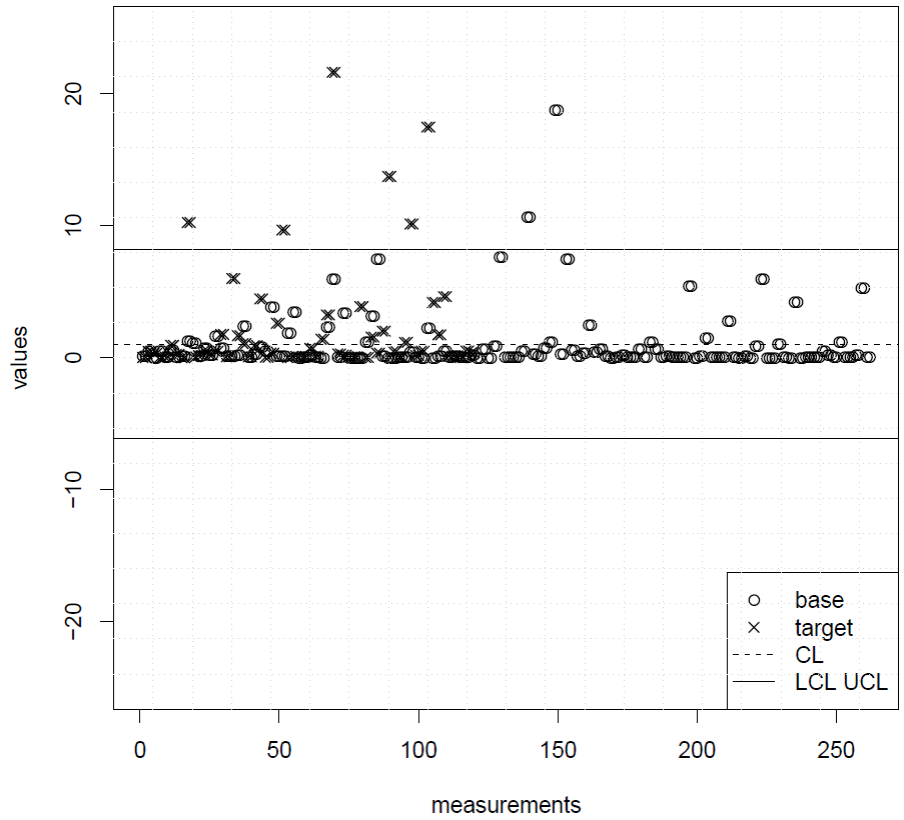


Рисунок 4.13 – Контрольна карта системних показників з регресією

Метод контрольних карт за результатами тестування виявився більш придатним до використання у мікросервісному середовищі за метод t-критерію Стьюдента. Проте, через низький рівень повноти метод контрольних карт також не може характеризувати поведінку продуктивності мікросервісного додатка в цілому. Метод контрольних карт за результатами тестування виявив 67% випадків штучних відхилень продуктивності з точністю 72%. Так само як і метод t-критерію Стьюдента метод контрольних карт потребує нормального розподілу вхідних даних для ефективної роботи.

Системні показники продуктивності мікросервісного додатка не відповідають нормальному розподілу, та як видно на рисунку 4.11 навіть методика фільтрування не допомогла привести дані до нормального розподілу, лише наблизити на певний відсоток. Наявність репозиторію історичних даних дозволила методу контрольних карт скоротити кількість хибно позитивних спрацювань у порівнянні з методом t-критерію Стьюдента, проте характер розподілу системних показників обумовив досить високу кількість хибно негативних результатів. За умови можливості фільтрації вхідних показників продуктивності до нормального розподілу метод контрольних карт може бути ефективно використаний для виявлення відхилень продуктивності додатків з мікросервісною архітектурою.

4.5.3 Результати методу асоціативних правил

В процесі тестування методу асоціативних правил було проведено порівняльний аналіз показників кожної пари 20 базових тестів без регресій, та порівняння показників одного базового тесту з показниками кожного з 24 тестів з штучними регресіями. Результати тестування наведено у таблиці 4.7.

Таблиця 4.7 – Результати тестування методу асоціативних правил

ХП	ІН	ІП	ХН	Точність	Повнота	F-міра
3	17	14	10	82%	58%	68%

З усіх трьох методів виявлення відхилень продуктивності метод асоціативних правил має найвищій показник точності – 82% та найнижчій показник повноти – 58%. Такий результат є наслідком низької кількості хибно позитивних спрацювань – 3 з 20 можливих та досить низького рівня виявлення штучних регресій 14 з 24 можливих.

Графік відношення десяти асоціативних правил для базового тесту зображено на рисунку 4.14. Розмір шарів на рисунку 4.14 відповідає значенню затребуваності окремого набору асоціативних правил, а колір шарів відповідає значенню ліфту, так наприклад, правило *memory/working_set low => memory/usage low* відповідає найвищім значенням затребуваності та ліфту у даному наборі.

Реєстрація регресії продуктивності відбувається за умови відсутності певних правил після нового тестування або при наявності значних відхилень в значеннях впевненості окремих асоціативних правил.

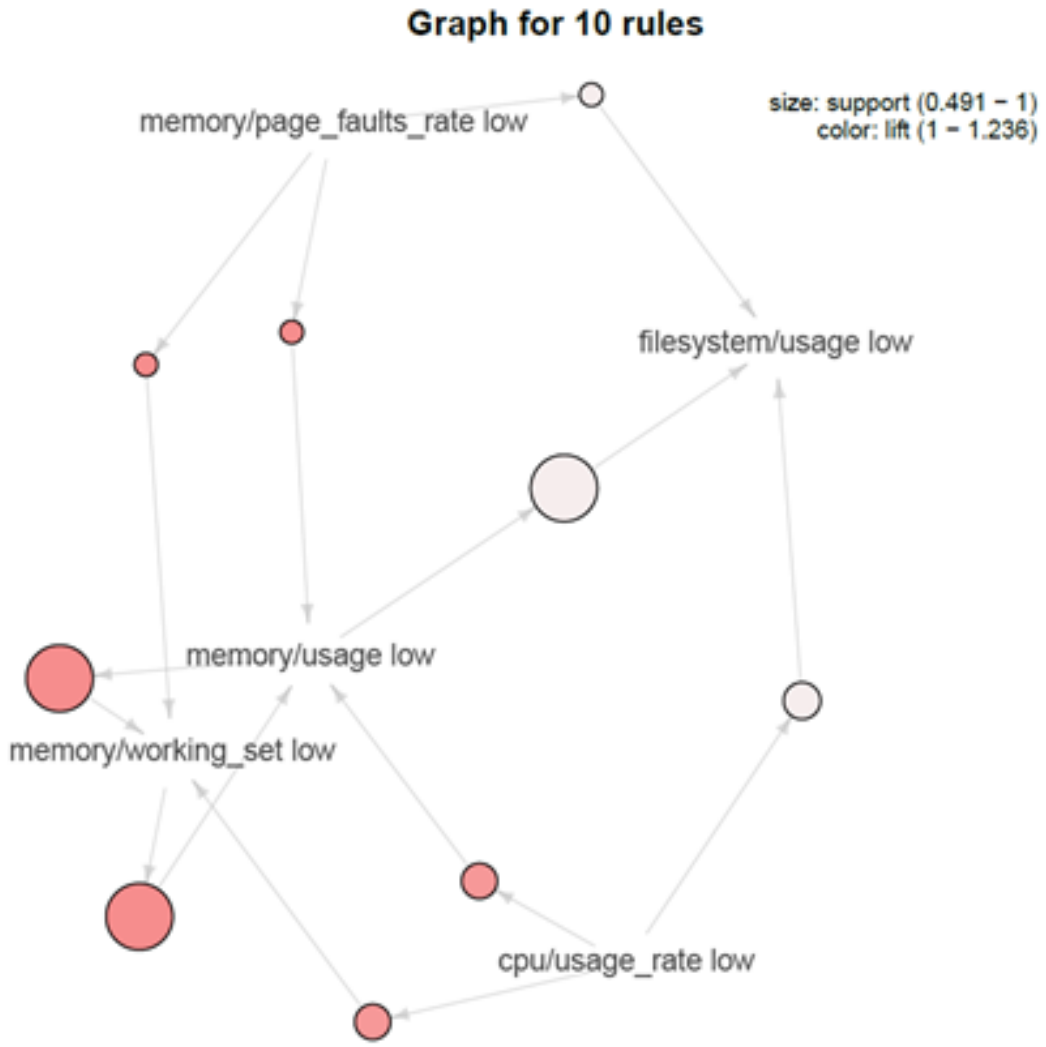


Рисунок 4.14 – Графік відношення асоціативних правил

Реалізація методу асоціативних правил використовує алгоритм Apriori [24] для побудови самих правил, цей алгоритм погано масштабується, тому час виконання сильно залежить від об’єму вхідних даних. Таким чином метод асоціативних правил в порівнянні з методами контрольних карт та t-критерію Стьюдента потребує набагато більше часу для виявлення потенційних регресій продуктивності. Доцільним є використання методу асоціативних правил для тестування окремих сервісів системи за умови визначеного часу навантажувального тестування, що не має перевищувати декілька годин.

4.6 Підсумки результатів тестування

Для оцінки роботи системи моніторингу продуктивності додатків з мікросервісною архітектурою було проведено 50 навантажувальних тестів загальною тривалістю 100 годин. В якості об’єкта тестування було використано додаток з відкритим кодом Sock-Shop [40]. Для кількісного визначення ефективності системи моніторингу продуктивності в мікросервіс додатка Sock-Shop було штучно введено 6 типів регресій продуктивності (розділ 4.3):

а) R1: надмірне логування;

б) R2: надмірні розрахунки;

в) R3: виділення додаткової пам’яті;

г) R4: некоректне налаштування бази даних;

д) R5: one lane bridge;

е) R6: ramp.

Статистику реєстрації різних типів регресій за допомогою методів виявлення відхилень продуктивності наведено у таблиці 4.8. Кожен з видів штучних відхилень було протестовано за 4 ітерації стресового навантаження загальною тривалістю 8 годин.

Таблиця 4.8 – Статистика реєстрації різних типів регресій продуктивності

Тип регресії / Метод	R1	R2	R3	R4	R5	R6
t-критерій Стьюдента	4	4	4	4	4	4
контрольних карт	4	2	4	2	2	2
асоціативних правил	3	2	4	3	2	0

За підсумками тестування метод виявлення відхилень продуктивності на основі статистичного аналізу схожості набору даних t-критерій Стьюдента зареєстрував 100% штучних регресій з точністю 62% та значенням F-міри 76%.

Метод контрольних карт, в основі якого лежить аналіз графіків зміни параметрів середнього значення та стандартного відхилу вибірки, зареєстрував 67% штучних регресій з точністю 72% та значенням F-міри 69%.

За допомогою побудови та аналізу правил відношення показників продуктивності в різних вибірках, метод асоціативних правил був здатен зареєструвати 58% штучних регресій з точністю 82% та значенням F-міри 68%.

Система моніторингу продуктивності бере до уваги результати аналізу кожного метода та у випадку реєстрації регресії в певному показнику продуктивності хоча б двома з трьох методів сигналізує про наявність відхилення продуктивності мікросервісного додатка. Таким чином тестування показало, що середнє значення точності виявлення відхилень продуктивності мікросервісних додатків за допомогою системи моніторингу лежить в межах від 60% до 70%.

5 СТАРТАП ПРОЕКТ

Стартап як форма малого ризикового (венчурного) підприємництва впродовж останнього десятиліття набула широкого розповсюдження у світі через зниження бар'єрів входу в ринок (із появою Інтернету як інструменту комунікацій та збуту стало простіше знаходити споживачів та інвесторів, займатись пошуком ресурсів, перетинати кордони між ринками різних країн), і вважається однією із наріжних складових інноваційної економіки, оскільки за рахунок мобільності, гнучкості та великої кількості стартап-проектів загальна маса інноваційних ідей зростає.

Проте створення та ринкове впровадження стартап-проектів відзначається підвищеною мірою ризику, ринково успішними стає лише невелика частка, що за різними оцінками складає від 10% до 20%. Ідея стартап-проекту, взята окремо, не вартує майже нічого: головним завданням керівника проекту на початковому етапі його існування є перетворення ідеї проекту у працюючу бізнес-модель, що починається із формування концепції товару (послуги) для визначеної клієнтської групи за наявних ринкових умов.

Розроблення та виведення стартап-проекту на ринок передбачає здійснення низки кроків, в межах яких визначають ринкові перспективи проекту, графік та принципи організації виробництва, фінансовий аналіз та аналіз ризиків і заходи з просування пропозиції для інвесторів. Узагальнено етапи розроблення стартап-проекту можна подати таким чином.

5.1 Опис ідеї проекту

Перші три пункти подаються у вигляді таблиці (таблиця 5.1) і дають цілісне уявлення про зміст ідеї та можливі базові потенційні ринки, в межах яких потрібно шукати групи потенційних клієнтів.

Таблиця 5.1 – Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
	1.Моніторинг продуктивності додатків з мікросервісною архітектурою.	Зменшення часу аналізу першопричин виникнення погіршення продуктивності додатка з випуском нової версії продукту.
	2. Інтеграція системи в існуючий цикл неперервної доставки програмного забезпечення.	Спрощення фази навантажувального тестування. Автоматизація процесу аналізу системних показників додатка.

Аналіз потенційних техніко-економічних переваг ідеї (чим відрізняється від існуючих аналогів та замінників) порівняно із пропозиціями конкурентів передбачає: визначення переліку техніко-економічних властивостей та характеристик ідеї, визначення попереднього кола конкурентів (проектів-конкурентів) або товарів-замінників чи товарів-аналогів, що вже існують на ринку. Проводиться порівняльний аналіз показників для власної ідеї (таблиця 5.2).

Таблиця 5.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів				W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Конкурент 1	Конкурент 2	Конкурент 3			
1	Підтримка мікро-сервісної архітектури							V
2	Зручність інтегрування до процесів неперервної доставки.							V
3	Гнучке налаштування						V	
4	Розповсюд-женість програмного рішення					V		

5.2 Технологічний аудит ідеї проекту

Визначення технологічної здійсненності ідеї проекту передбачає аналіз складових вказаних в таблиці 5.3.

Таблиця 5.3 – Технологічна здійсненність ідеї проекту

№ п/п	Ідея проекту	Технології її реалізації	Наявність технології	Доступність технології
		Мова програмування Java	Наявна	Доступна
		Мова програмування Python	Наявна	Доступна
		Мова програмування C#	Наявна	Доступна
		Мова програмування Python	Наявна	Доступна
		Мова програмування R	Наявна	Доступна
Обрані технології реалізації ідей проекту: Мови програмування Java та R.				

5.3 Аналіз ринкових можливостей запуску

Визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів.

Для початку проводиться аналіз попиту: наявність попиту, обсяг, динаміка розвитку ринку (таблиця 5.4).

Таблиця 5.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку (найменування)	Характеристика
1.	Кількість головних кравців, од	10
2.	Загальний обсяг продаж, грн/ум.од	10 тис. ум.од в місяць
3.	Динаміка ринку (якісна оцінка)	зростає
4.	Наявність обмежень для входу (вказати характер обмежень)	Початковий капітал до 50 тис. ум.од.
5.	Специфічні вимоги до стандартизації та сертифікації	Немає
6.	Середня норма рентабельності в галузі (або по ринку), %	40% в рік

Надалі визначаються потенційні групи клієнтів, їх характеристики, та формується орієнтовний перелік вимог до товару для кожної групи (таблиця 5.5).

Таблиця 5.5 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1.	Відсутність засобу моніторингу продуктивності систем з мікро-сервісною архітектурою.	Розробники програмного забезпечення.	Використання системи для розробки власного програмного продукту.	Простота використання системи, можливість інтеграції до існуючих процесів.
2.	Зменшення затрат часу на навантажувальне тестування програмного забезпечення.	Інженери з забезпечення якості програмного продукту.	Використання системи на етапі розробки програмного забезпечення.	Висока ефективність виявлення відхилень продуктивності.

Після визначення потенційних груп клієнтів проводиться аналіз ринкового середовища: складаються таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають (таблиці 5.6-5.7).

Таблиця 5.6 – Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1.	Різкий зріст кількості конкурентів розробленого програмного продукту.	Ринок засобів для роботи з мікросервісами розширюється, виникає все більше нових пропозицій.	Удосконалення функціоналу продукту, додавання нових унікальних засобів моніторингу продуктивності у розроблений продукт.

Таблиця 5.6 – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1.	Ріст популярності використання мікросервісної архітектури для розробки програмного забезпечення.	Мікросервісна архітектура набирає популярність у розробників які працюють у великих комерційних компаніях.	Введення частих циклів оновлення функціоналу розробленого продукту. Розширення існуючого функціоналу у відповідності до вимог ринку.

Надалі проводиться аналіз пропозиції: визначаються загальні риси конкуренції на ринку (таблиця 5.8).

Таблиця 5.8 – Ступеневий аналіз конкуренції на ринку.

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Вказати тип конкуренції - монополістична	На ринку існує багато пропозиції комерційного та некомерційного характеру.	Розробка кращого програмного продукту ніж в конкурентів.
2. За рівнем конкурентної боротьби - міжнародна	Користувачі продукту – міжнародні компанії по розробці програмного забезпечення.	Локалізація документації продукту на різні мови.
3. За галузевою ознакою - внутрішньогалузева	Програмний продукт фокусується на додатках з мікросервісною архітектурою.	Фокус на конкретному напрямку розвитку.
4. Конкуренція за видами товарів - товарно-видова	Конкуренція з іншими програмними продуктами.	Аналіз програмних рішень конкурентів.
5. За характером конкурентних переваг - нецінова	Автоматизований процес швидкого виявлення першопричин деградації продуктивності додатка.	Оновлення та покращення розробленого функціонала.
6. За інтенсивністю - марочна	Наявність унікальної назви розробленої системи.	Слідкувати за збереженням авторського права.

Після аналізу конкуренції проводиться більш детальний аналіз умов конкуренції в галузі (таблиця 5.9).

Таблиця 5.9 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі: компанії по розробці програмного забезпечення.	Потенційні конкуренти: Google, Facebook, IBM	Постачальники: сервер компанії виробника продукту.	Клієнти: Розробники додатків з мікро-сервісною архітектурою	Товари-замінники: WebLoad, LoadImpac, Httperf
	Перелік прямих конкурентів: WebLoad, LoadImpact, Httperf	Бар'єри входження в ринок: гнучкі ціни на пропозиції.	Фактори сили постачальників: ефективні канали збуту товару в інтернеті.	Фактори сили споживачів: чутливість до змін цінових пропозицій.	Фактори загрози з боку замінників: ціна, функці-ональність.
Висновки:	Середня інтенсив-ність боротьби з боку конкурентів.	Термін активізації потенцій-них конкурентів: 1рік.	Постачальники не диктують умови роботи на ринку.	Клієнти диктують ринкову ціну, зручність продукту.	Товари замінники не обмежують роботу на ринку.

На основі аналізу конкуренції, а також із урахуванням характеристик ідеї проекту, вимог споживачів до товару та факторів маркетингового середовища визначається та обґрунтовується перелік факторів конкурентоспроможності (таблиця 5.10).

Таблиця 5.10 – Обґрунтування факторів конкурентоспроможності.

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1.	Автоматизація процесу виявлення першопричин виникнення відхилень продуктивності в додатках з мікросервісною архітектурою.	Команда розробників отримує ефективний інструмент моніторингу продуктивності мікросервісів їх системи. Час на виявлення причин деградації продуктивності значно зменшується.
2.	Наявність засобів для інтеграції розробленої системи моніторингу продуктивності в процес неперервної доставки програмного забезпечення.	Інженери з забезпечення якості програмних систем отримують можливість автоматизувати процес навантажувального тестування на етапі випуску нової версії продукту.

За визначеними факторами конкурентоспроможності проводиться аналіз сильних та слабких сторін стартап-проекту (таблиця 5.11).

Таблиця 5.11 – Порівняльний аналіз сильних та слабких сторін продукту

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з розробленим програмним продуктом						
			-3	-2	-1	0	+1	+2	+3
1.	Автоматизація процесу виявлення першопричин виникнення відхилень продуктивності в додатках з мікросервісною архітектурою.	20	V						
2.	Наявність засобів для інтеграції розробленої системи моніторингу продуктивності в процес неперервної доставки програмного забезпечення.	20		V					

Фінальним етапом ринкового аналізу можливостей впровадження проекту є складання SWOT-аналізу (матриці аналізу сильних (Strength) та слабких (Weak) сторін, загроз (Troubles) та можливостей (Opportunities) (таблиця 5.12).

Таблиця 5.12 – SWOT – аналіз стартап-проекту.

Сильні сторони: – Автоматизованість виявлення відхилень продуктивності. – Інтеграція в процеси неперервної доставки.	Слабкі сторони: Популярність програмного рішення.
Можливості: Ріст попиту на розробку додатків з мікросервісною архітектурою.	Загрози: Збільшення кількості конкурентів, зменшення популярності.

Визначені альтернативи аналізуються з точки зору строків та ймовірності отримання ресурсів (таблиця 5.13).

Таблиця 5.13 – Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1.	Зменшення ціни програмного продукту.	менша	такі ж
2.	Збільшення ціни програмного продукту у відповідності до розширення функціоналу системи.	більша	такі ж
3.	Реалізація системи з відкритим кодом, залучення коштів за підтримку та обслуговування.	менша	менший

Обрана альтернатива – збільшення ціни програмного продукту у відповідності до розширення функціоналу системи.

5.4 Розроблення ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів (таблиця 5.14).

Таблиця 5.14 – Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1.	Розробники програмного забезпечення	висока	середній	середня	висока
2.	Інженери з забезпечення якості програмних продуктів.	висока	великий	середня	висока

В якості цільових груп було обрано: розробники програмного забезпечення та інженери з забезпечення якості програмних продуктів.

Для роботи в обраних сегментах ринку необхідно сформувати базову стратегію розвитку (таблиця 5.15).

Таблиця 6.15 – Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1.	Збільшення ціни програмного продукту у відповідності до розширення функціоналу системи.	Охоплення усього ринку.	Автоматизованість виявлення відхилень продуктивності та інтеграція в процеси неперервної доставки.	Стратегія спеціалізації.

Наступним кроком є вибір стратегії конкурентної поведінки (таблиця 5.16).

Таблиця 6.16 – Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конку-рентної поведінки
1.	Ні	Забирати існуючих.	Буде копіювати основні характеристики конкурентів.	Стратегія заняття конкурентної ніші.

На основі вимог споживачів з обраних сегментів до постачальника (стартап-компанії) та до продукту, а також в залежності від обраної базової стратегії розвитку та стратегії конкурентної поведінки розробляється стратегія позиціонування (таблиця 5.17).

Таблиця 5.17 – Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкуренто-спроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
1.	Легкість використання програмного продукту.	Конкурентна	Автоматизованість виявлення відхилень продуктивності та інтеграція в процеси неперервної доставки.	Швидкий, гнучкий, зрозумілий.

5.5 Розроблення маркетингової програми

Першим кроком є формування маркетингової концепції товару, який отримає споживач. Для цього у таблиці 5.18 потрібно підсумувати результати попереднього аналізу конкурентоспроможності товару.

Таблиця 5.18 – Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами
1.	Автоматизованість виявлення відхилень продуктивності.	Розробники мають засіб аналізу продук-тивності додатка.	Наявність функціоналу виявлення першопричин деградації продуктивності.
2.	Інтеграція в процеси неперервної доставки	Зменшення затрат часу на тестування.	Можливість легкої інтеграції в існуючи процеси.

Надалі розробляється трирівнева маркетингова модель товару: уточнюється ідея продукту та/або послуги, його фізичні складові, особливості процесу його надання (таблиця 5.19).

Таблиця 5.19 – Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові
1. Товар за задумом: Система моніторингу продуктивності додатків з мікросервісною архітектурою.	Опис базової потреби споживача, яку задовольняє товар (згідно концепції), її основної функціональної вигоди: Потреба в автоматизації процесу виявлення першопричин деградації продуктивності додатка за випуску нових версій.
2. Товар у реальному виконанні	Характеристики: – функціонал інтеграції навантажувального тестування в мікросервісне середовище, гнучка конфігурація навантаження; – функціонал аналізу системних показників мікросервісного додатка з використанням ефективних методів виявлення відхилень продуктивності.
3. Товар із підкріпленням	Підтримка реалізованого продукту з частими циклами оновлення функціоналу.

Наступним кроком є визначення цінових меж, якими необхідно керуватись при встановленні ціни на потенційний товар яке передбачає аналіз ціни на товари-аналоги

або товари субститути, а також аналіз рівня доходів цільової групи споживачів (таблиця 5.20).

Таблиця 5.20 – Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1.	-	200 ум.од.	1000 – 5000 ум.од.	200 – 500 ум.од.

Наступним кроком є визначення оптимальної системи збуту, в межах якого приймається рішення (таблиця 5.21).

Таблиця 5.21 – Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1.	Купівля ліцензії на використання програмного рішення розробниками.	Цілодобова доступ-ність продукту на ресурсах виробника.	Без посередників.	Збут власними силами.

Останньою складової маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо об-рану основу для позиціонування, визначену специфіку поведінки клієнтів (таблиця 5.22).

Таблиця 5.22 – Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідом-лення	Концепція рекламного звернення
1.	Пошук потрібних програмних рішень у мережі інтернет.	Мережа інтернет.	Автоматизованість виявлення відхилень продуктивності та інтеграція в процеси неперервної доставки.	Привернути увагу потенційних користувачів	Демонстрація переваг використання розробленого програмного продукту.

Висновки

За результатами проведеного аналізу можна зробити висновки, що стартап-проект на базі розробленого програмного рішення – системи моніторингу продуктивності додатків з мікросервісною архітектурою має великі шанси бути прибутковим та може успішно складати конкуренцію існуючим рішенням з автоматизації процесів навантажувального тестування на ринку.

ВИСНОВКИ

В даній дипломній роботі було спроектовано та розроблено систему моніторингу продуктивності додатків з мікросервісною архітектурою. Система має засоби для легкої інтеграції в процес безперервної доставки програмного забезпечення на етапі тестування нової версії продукту. Система моніторингу спроектована для оцінки змін ефективності додатків з мікросервісною архітектурою, має гнучкі засоби налаштування, та може бути інтегрована в мікросервісне середовище.

В результаті дослідження предметної області було виявлено, що компанії, які розробляють комерційне програмне забезпечення часто мають власні розробки засобів навантажувального тестування та моніторингу своїх систем, а також існує численна кількість програмних інструментів з відкритим кодом, які вирішують задачі оцінки продуктивності додатків.

Було проведено аналіз існуючих розробок в напрямку моніторингу продуктивності додатків та виявлено, що більшість таких інструментів спроектована для роботи з додатками які мають монолітну архітектуру, для яких значення кількості робочих компонент відомо на момент тестування. Тому виникає необхідність в розробці нових рішень оцінки продуктивності додатків з мікросервісною архітектурою.

Основні етапи розробки та тестування системи моніторингу продуктивності включають в себе: 1) Проектування архітектури та реалізація бізнес-логіки системи моніторингу продуктивності. 2) Налаштування та розгортка додатка з мікросервісною архітектурою в якості об’єкта тестування. 3) Генерація та введення штучних відхилень продуктивності в мікросервіс додатка для подальшого аналізу ефективності розробленої системи моніторингу. 4) Запуск системи моніторингу продуктивності для різних версій мікросервісного додатка. 5) Аналіз результатів тестування, оцінка методів виявлення відхилень продуктивності та оцінка ефективності системи моніторингу продуктивності.

Результати тестування показали, що розроблена система моніторингу продуктивності додатків з мікросервісною архітектурою при аналізі показників продуктивності одного мікросервісу за різних версій додатка змогла виявити усі види штучних відхилень продуктивності з точністю до 70%. Таким чином за допомогою системи моніторингу продуктивності можливо з досить великою вірогідністю визначити деградацію продуктивності окремих мікросервісів додатка. Система може бути налаштована на оцінку продуктивності додатка в цілому, для цього необхідно налаштувати паралельний аналіз відхилень продуктивності для кожного мікросервісу.

Спроектована система моніторингу продуктивності була протестована на достатньо малому об’ємі системних показників та з використанням досить простих в реалізації методів виявлення відхилень продуктивності. Тестування проводилось в кластері з однією нодою та досить обмеженими системними ресурсами. Для покращення показників ефективності розробленої системи моніторингу необхідно реалізувати ще декілька методів виявлення відхилень продуктивності та удосконалити існуючі реалізації з урахуванням надзвичайно великих об’ємів статистичних даних показників продуктивності мікросервісів.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Віртуалізація [Електронний ресурс] / Wikipedia – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Апаратна_віртуалізація.
2. Namespace [Електронний ресурс] / Wikipedia – Режим доступу до ресурсу: http://man7.org/linux/man-pages/man7/namespaces.7.html.
3. W. Felter, A. Ferreira, R. Rajamony, J. Rubio. “An updated performance comparison of virtual machines and linux containers.” In: Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on. IEEE. 2015, pp. 171–172 (cit. on pp. 5, 6).
4. Fowler M. Microservices [Електронний ресурс] / Martin Fowler – Режим доступу до ресурсу: https://martinfowler.com/articles/microservices.html.
5. Docker [Електронний ресурс] / Wikipedia – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Docker.
6. Microservices are hard [Електронний ресурс] – Режим доступу до ресурсу: https://hackernoon.com/microservices-are-hard-an-invaluable-guide-to-microservices-2d06bd7bcf5d.
7. What is Microservices Architecture? [Електронний ресурс] – Режим доступу до ресурсу: https://smartbear.com/learn/api-design/what-are-microservices/
8. Docker.io. Docker [Електронний ресурс] / Docker.io – Режим доступу до ресурсу: https://www.docker.com/.
9. Daemon [Електронний ресурс] / Wikipedia – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Daemon_(computing).
10. Kubernetes.io. Kubernetes [Електронний ресурс] / Kubernetes.io – Режим доступу до ресурсу: https://kubernetes.io.
11. Molyneaux I. The Art of Application Performance Testing / Ian Molyneaux.. – 209 с. – (O'Reilly Media).
12. Software Performance [Електронний ресурс] / Wikipedia – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Software_performance_testing.

13. Automated Verification of Load Tests Using Control Charts / T.H.D. Nguyen, A. Bram, J. Zhen Ming, H. Ahmed E.. // Asia-Pacific Software Engineering Conference. – 2011. – С. 282–289.
14. The httperf HTTP load generator [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/httperf/httperf.
15. Rational Performance Tester [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Rational_Performance_Tester.
16. System Performance Monitor [Електронний ресурс] – Режим доступу до ресурсу: https://www.techopedia.com/definition/12397/system-performance-monitor-spm.
17. Linux proc [Електронний ресурс] – Режим доступу до ресурсу: http://tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html.
18. Performance Testing Guidance for Web Applications by Microsoft Corporation / Microsoft Corporation., 2007. – 221 с. – (Microsoft Corporation).
19. Automated Detection of Performance Regressions Using Regression Models on Clustered Performance Counters / W. Shang, A.E. Hassan, M. Nasser, F. Parminder // Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering – Austin, 2015. – С. 15-26.
20. Student t-Test [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Student%27s_t-test.
21. Thanh H.D. Nguyen Automated detection of performance regressions using statistical process control techniques / T. H. Nguyen, B. Adams, Z. M. Jiang, A. E. Hassan, M. Nasser, P. Flora // Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering – Boston, 2012. – С. 299-310.
22. An Industrial Case Study on the Automated Detection of Performance Regressions in Heterogeneous Environments / A.Bram, M. Zhen, C. King, H. Ahmed E.. // IEEE/ACM 37th IEEE International Conference on Software Engineering. – 2015. – С. 49–58.
23. Association Rule [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Association_rule_learning.

24. Fast Algorithms for Mining Association Rules in Large Databases / R. Agrawal, R. Srikant // Proc. of the 20th VLDB Conference – San Francisco, 1994. – С. 487-499.
25. Locust: An open source load testing tool [Електронний ресурс] – Режим доступу до ресурсу: https://locust.io.
26. Master-slave. [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Master/slave_(technology)
27. Heapster: Compute Resource Usage Analysis and Monitoring of Container Clusters [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/kubernetes/heapster.
28. InfluxDB: Scalable datastore for metrics, events, and real-time analytics [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/influxdata/influxdb.
29. InfluxDB Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/InfluxDB
30. Неперервна інтеграція [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Continuous_integration
31. Continuous Delivery [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@Zaiku/continuous-delivery-in-a-nutshell-29f4213dabda
32. Influxdb-java - Java client for InfluxDB [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/influxdata/influxdb-java
33. Rserve — Binary R server [Електронний ресурс] – Режим доступу до ресурсу: http://www.rforge.net/Rserve/.
34. Jenkins An open source automation server written in Java. [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Jenkins_(software)
35. The R Project for Statistical Computing [Електронний ресурс] – Режим доступу до ресурсу: https://www.r-project.org/
36. R (мова програмування) Computing [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/R_(мова_програмування)
37. Student's t-Test [Електронний ресурс] – Режим доступу до ресурсу: https://stat.ethz.ch/R-manual/R-devel/library/stats/html/t.test.html

38. Repository [Електронний ресурс] – Режим доступу до ресурсу: https://microservices-demo.github.io/docs/index.html.
39. Weave [Електронний ресурс] – Режим доступу до ресурсу: https://www.weave.works/.
40. Sock-Shop [Електронний ресурс] – Режим доступу до ресурсу: https://www.infoq.com/articles/sock-shop.
41. Minikube [Електронний ресурс] – Режим доступу до ресурсу: https://kubernetes.io/docs/setup/minikube/.
42. An Industrial Case Study of Automatically Identifying Performance Regression-Causes / N.Thanh H. D., N. Meiyappan, H. Ahmed, E., P. Flora. // Proceedings of the 11th Working Conference on Mining Software Repositories. – 2014. – С. 232–241.
43. Antipattern-Based Problem Injection for Assessing Performance and Reliability Evaluation Techniques / K.Philipp, V. André, O. Dušan, P. Teerat. // IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). – 2016. – С. 64–70.
44. .Millicores [Електронний ресурс] – Режим доступу до ресурсу: http://www.noqcks.io/notes/2016/12/14/kubernetes-understanding-millicores/.

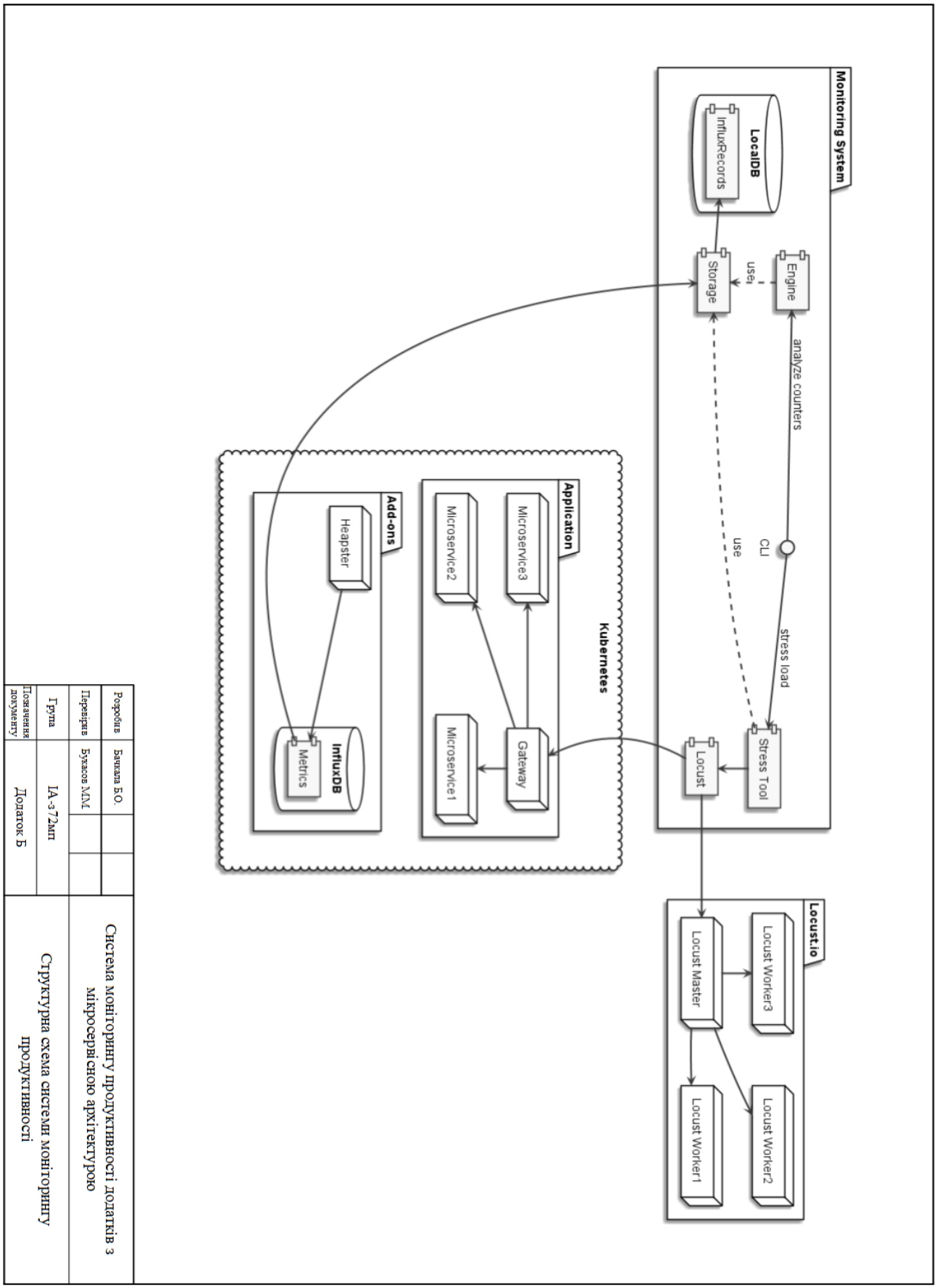
ДОДАТОК А

Приклад налаштування мікросервісу

```
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: catalogue
  labels:
    name: catalogue
  namespace: sock-shop
spec:
  replicas: 1
  template:
    metadata:
      labels:
        name: catalogue
    spec:
      containers:
      - name: catalogue
        image: weaveworksdemos/catalogue:0.3.5
        command: ["/app"]
        args:
        - -port=80
        resources:
          limits:
            cpu: 100m
            memory: 100Mi
          requests:
            cpu: 100m
            memory: 100Mi
```

ДОДАТОК Б

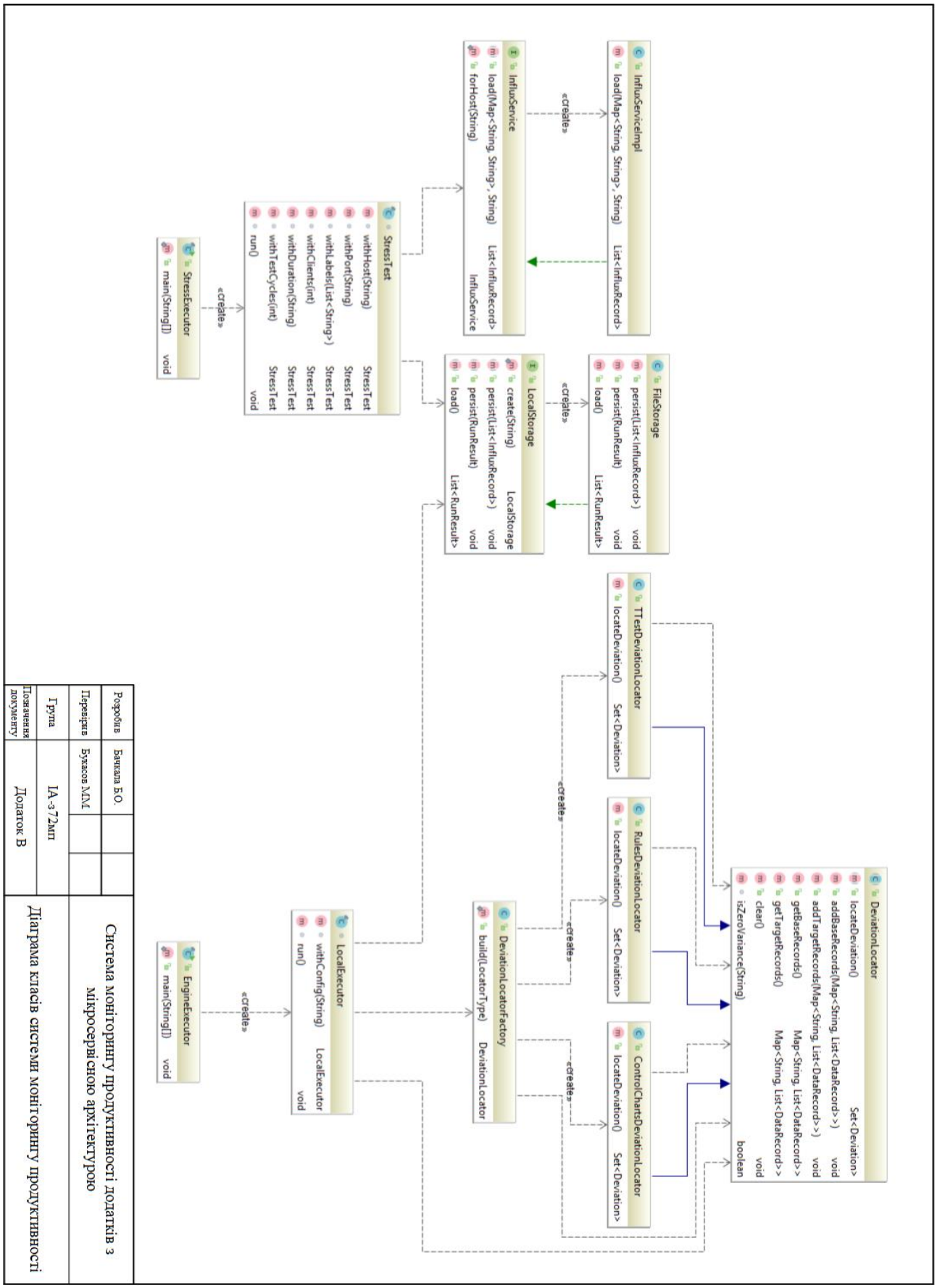
Структурна схема системи моніторингу продуктивності



Розробив	Бачкала Б.О.		Система моніторингу продуктивності додатків з мікросервісною архітектурою
Перевірив	Буласов М.М.		
Група	ІА-з72мп		Структурна схема системи моніторингу продуктивності
Позначення документу	Додаток Б		

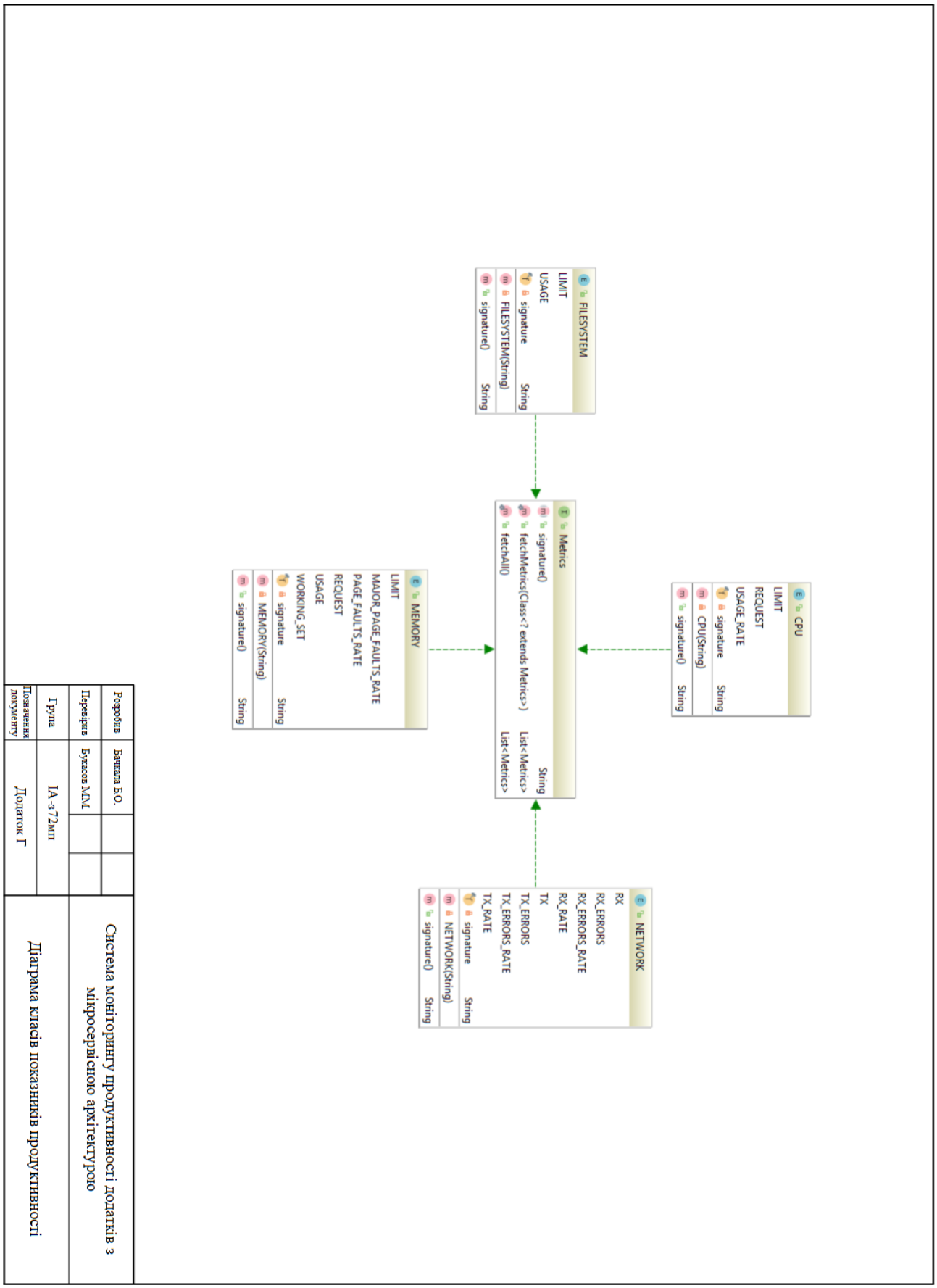
ДОДАТОК В

Діаграма класів системи моніторингу продуктивності



ДОДАТОК Г

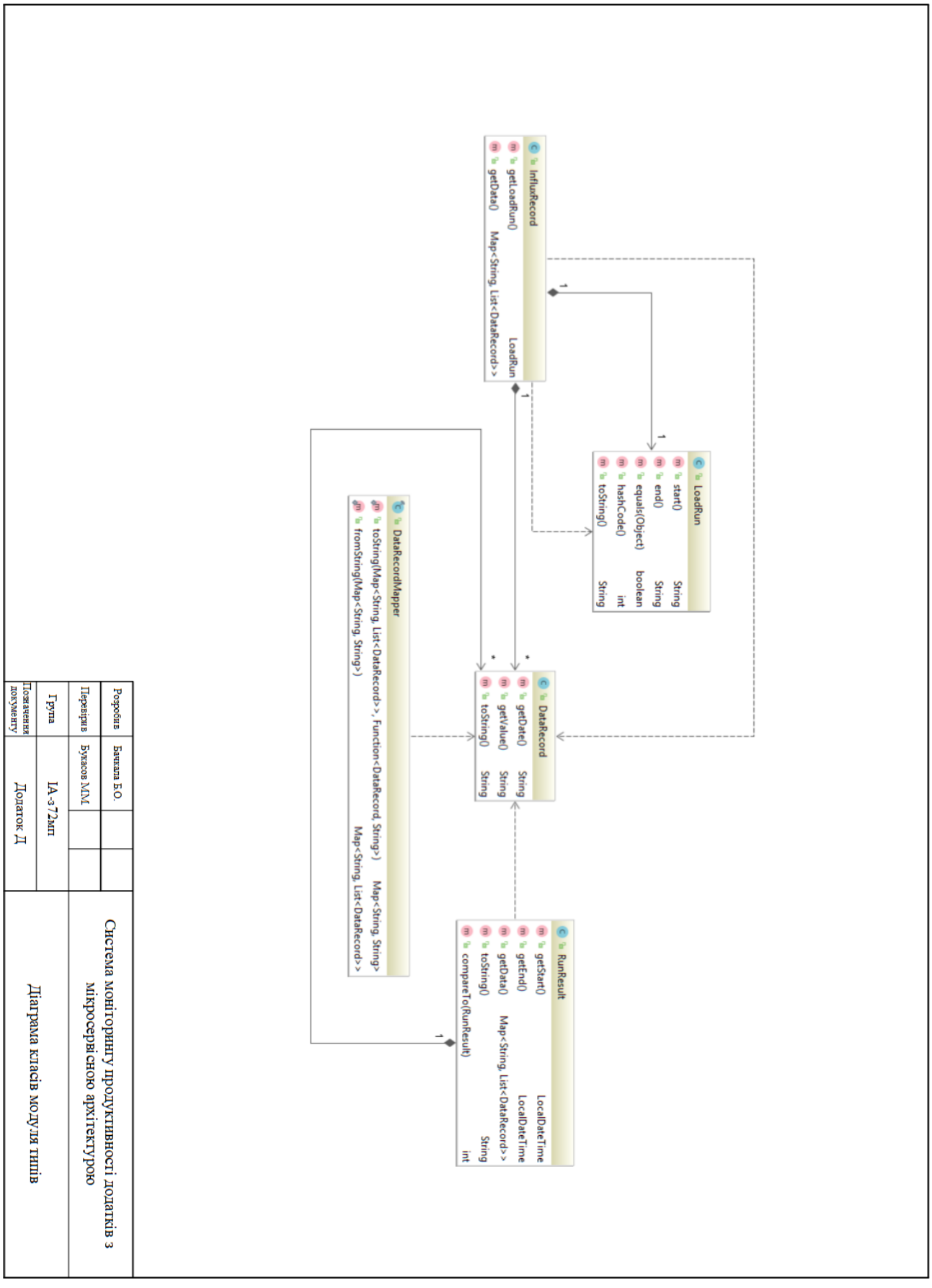
Діаграма класів показників продуктивності



Розробив	Бачкала Б.О.		Система моніторингу продуктивності додатків з мікросервісною архітектурою
Перевірив	Буказов М.М.		
Група	ІА-з72мп		
Позначення документу	Додаток Г		Діаграма класів показників продуктивності

ДОДАТОК Д

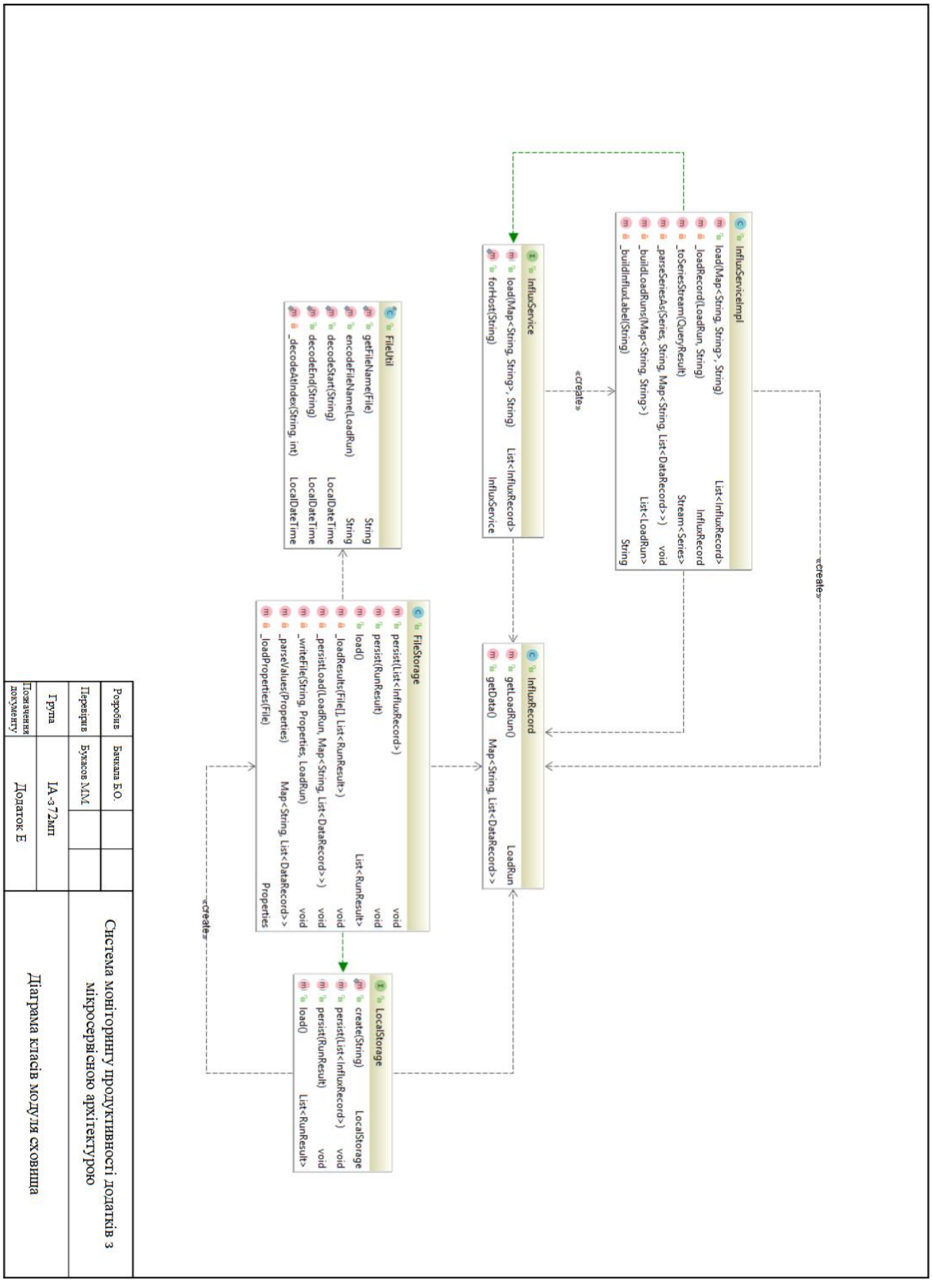
Діаграма класів модуля типів



Розробив	Бачкала Б.О.		Система моніторингу продуктивності додатків з мікросервісною архітектурою
Перевірив	Букасов М.М.		
Група	ІА-з72мп		
Позначення документу	Додаток Д		Діаграма класів модуля типів

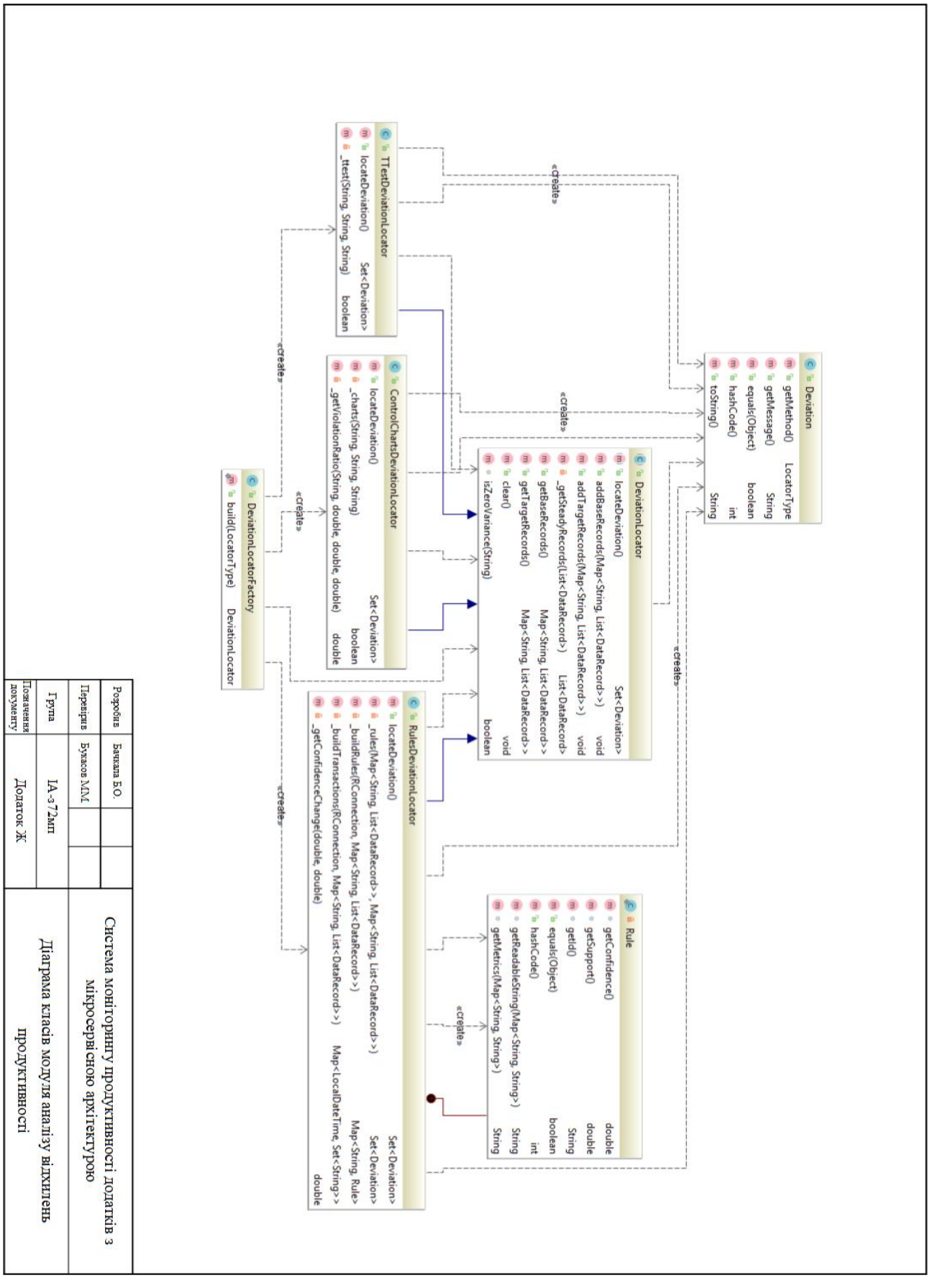
ДОДАТОК Е

Діаграма класів модуля сховища



ДОДАТОК Ж

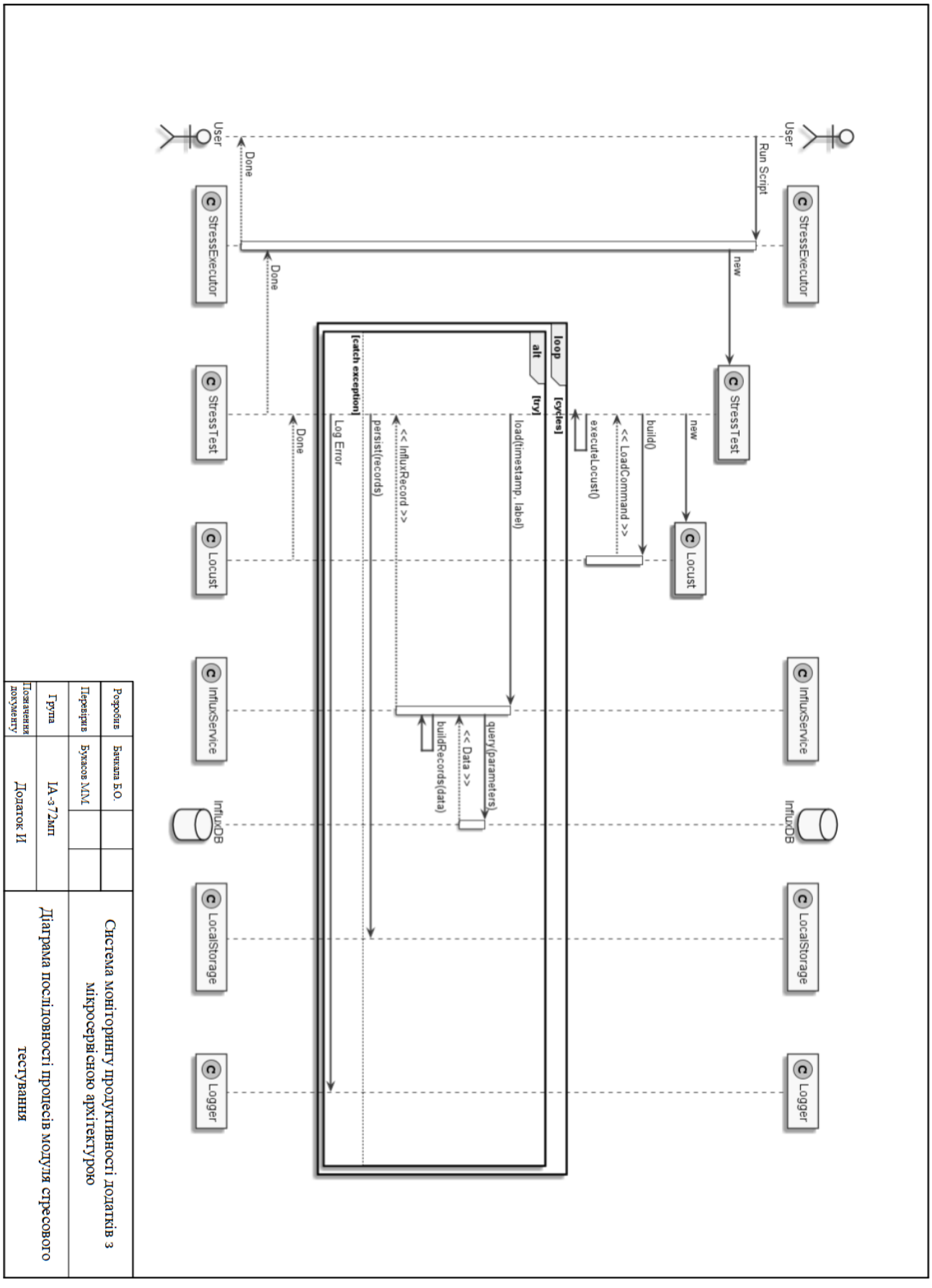
Діаграма класів модуля аналізу відхилень продуктивності



Розробив	Бачкала Б.О.		Система моніторингу продуктивності додатків з мікросервісною архітектурою
Перевірив	Буказов М.М.		
Група	ІА-з72мп		Діаграма класів модуля аналізу відхилень продуктивності
Позначення документу	Додаток Ж		

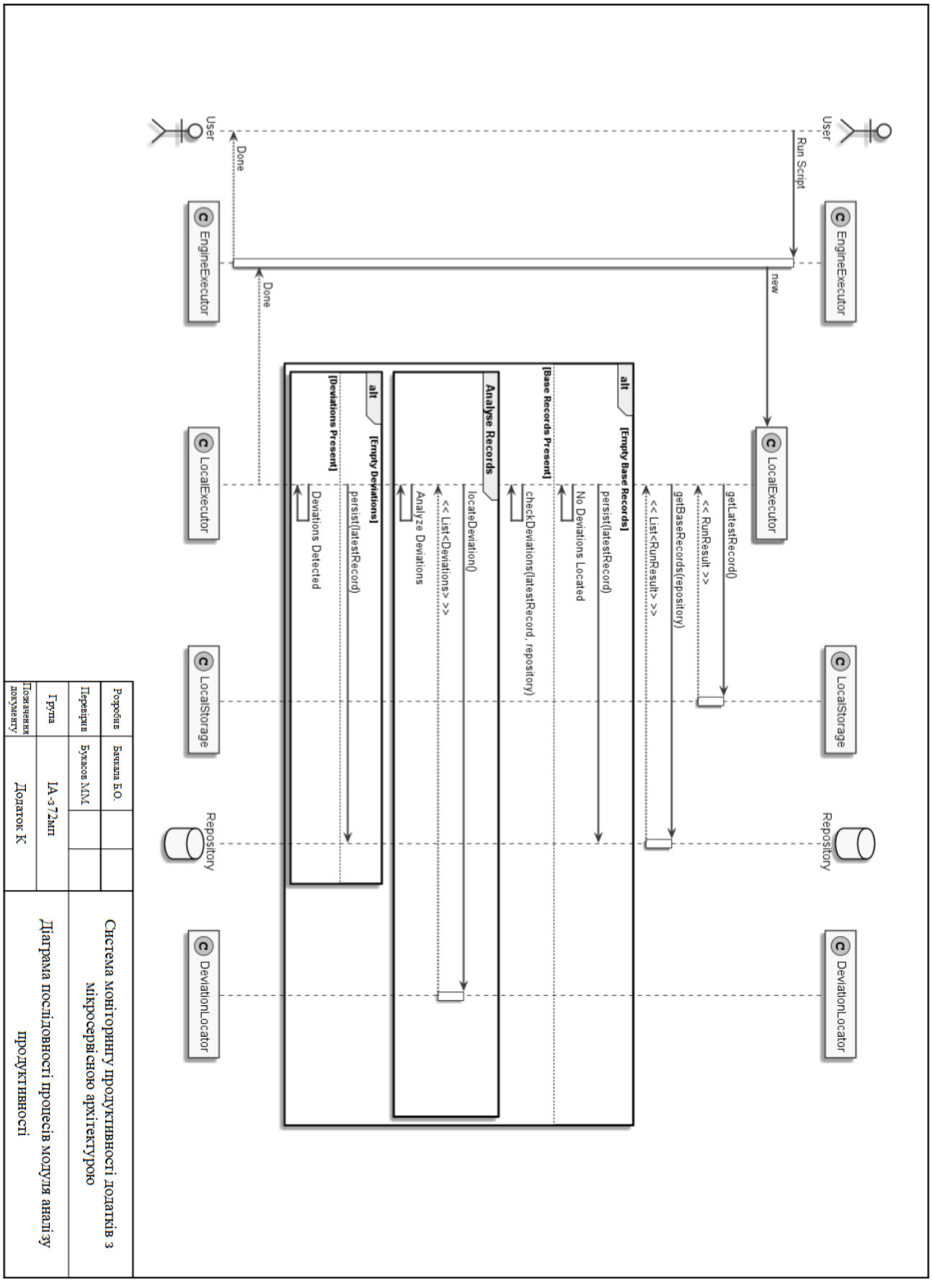
ДОДАТОК И

Діаграма послідовності процесів модуля стресового тестування



ДОДАТОК К

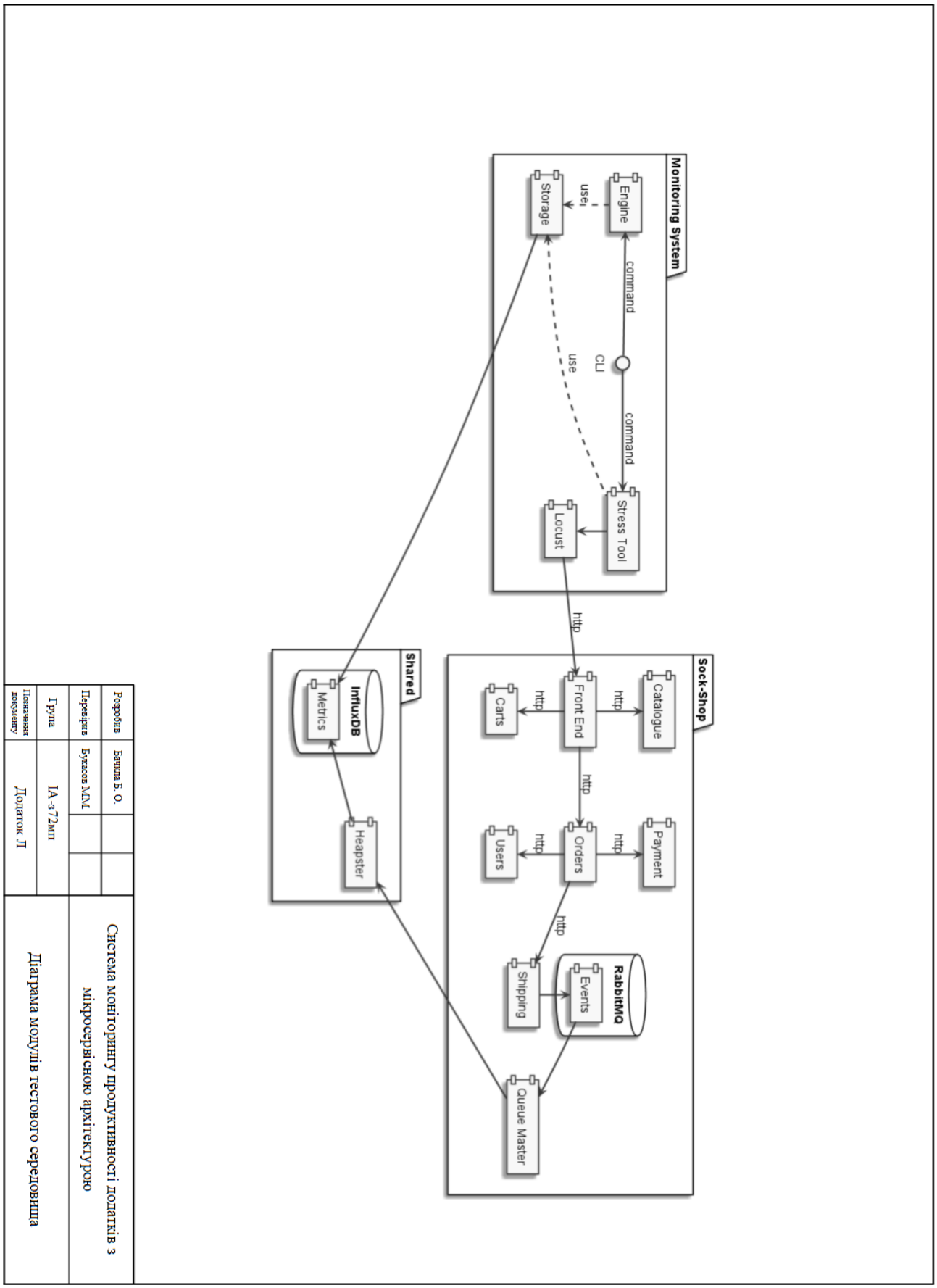
Діаграма послідовності процесів модуля аналізу продуктивності



Розробив	Бачкала Б.О.		Система моніторингу продуктивності додатків з мікросервісною архітектурою
Перевірив	Буказов М.М.		
Група	ІА-з72мп		Діаграма послідовності процесів модуля аналізу продуктивності
Позначення документу	Додаток К		

ДОДАТОК Л

Діаграма модулів тестового середовища



Розробив	Бачила Б. О.		Система моніторингу продуктивності додатків з мікросервісною архітектурою
Перевірив	Буласов М.М.		
Група	ІА-з72мп		Діаграма модулів тестового середовища
Позначення документу	Додаток Л		