Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп’ютерних наук та штучного інтелекту
Кафедра комп’ютерних систем та робототехніки

«Затверджую»
в.о. завідуючого кафедри
комп’ютерних систем та робототехніки
_____ к. ф.-м. н., доцент Максим Хруслов
«___» грудня 2024 р.

Пояснювальна записка

до кваліфікаційної роботи
магістра

на тему: «**Дослідження шаблонів стійкості до відмов у мікросервісних архітектурах та імплементація динамічного налаштування параметрів на основі ретроспективних даних**»

Спеціальність 174 – Автоматизація, комп’ютерно-інтегровані технології та робототехніка
Галузь знань 17 – Електроніка, автоматизація та електронні комунікації.
Освітня програма «Комп’ютеризовані системи управління та автоматика»

Захищено на засіданні
Екзаменаційної комісії № 44
протокол № __ від __.12.2024 р.
Оцінка ____/ _____
Голова Екзаменаційної комісії
_____ **СКОБ Ю. О.**

Виконав:
Студент групи КУ– 61
ШЕВЧЕНКО Андрій Романович

Керівник: доцент ЗВО кафедри Інтелектуальних програмних систем і технологій, кандидат технічних наук, доцент
ГАМЗАЄВ Рустам Олександрович

Рецензент: доцент ЗВО кафедри системного аналізу та інформаційно-аналітичних технологій Націоального технічного університету, «Харківський політехнічний інститут», к.т.н., доцент СИДОРЕНКО Ганна Юріївна

Харків – 2024

АНОТАЦІЯ

Пояснювальна записка до магістерської атестаційної роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел і трьох додатків. Загальний обсяг роботи складає 68 сторінки, із яких 51 сторінок основної частини з 22 рисунками, 8 таблицями, списку використаних джерел із 19 найменувань та трьома додатками на 12 сторінках.

Метою кваліфікаційної роботи є розробка моделі динамічного налаштування параметрів шаблонів стійкості, що дозволить підвищити надійність, продуктивність та адаптивність мікросервісних систем в умовах змінного навантаження.

Об’єкт дослідження – процеси управління відмовостійкістю в мікросервісних архітектурних системах, зокрема механізми налаштування параметрів шаблонів стійкості.

Предмет дослідження – методи, алгоритми моніторингу навантаження, оцінки параметрів продуктивності та адаптивного налаштування параметрів шаблонів стійкості у мікросервісних архітектурах з урахуванням поточного стану системи, навантаження.

Сучасні рішення для управління міжсервісною комунікацією не враховують змінне навантаження, що потребує переналаштування. Але дослідження механізмів динамічного налаштування параметрів стійкості дозволить підвищити ефективність використання ресурсів та пропускну здатність у мікросервісних системах.

Ключові слова: МОДЕЛЬ, МІКРОСЕРВІСИ, ШАБЛОНИ СТІЙКОСТІ, CIRCUIT BREAKER, НАВАНТАЖЕННЯ, МЕРЕЖА, МЕТРИКИ, МОНІТОРИНГ.

ABSTRACT

The explanatory note for the master's thesis consists of an introduction, three chapters, conclusions, a list of references, and two appendices. The total volume of the work is 68 pages, of which 51 pages are the main text, containing 22 figures, 8 tables, a list of 19 references, and three appendices totaling 12 pages.

The aim of the qualification work is to develop and theoretically justify a model for dynamically adjusting the parameters of resilience patterns, which will enhance the reliability, performance, and adaptability of microservices systems under varying loads.

The object of the research is the processes of fault tolerance management in microservice architectural systems, specifically the mechanisms for configuring the parameters of resilience patterns.

The subject of the research is the methods and algorithms for monitoring load, assessing performance parameters, and adaptively adjusting the parameters of resilience patterns in microservice architectures, taking into account the current system state and load.

Current solutions for managing inter-service communication do not account for changing loads, which require reconfiguration. However, research into the mechanisms for dynamically adjusting resilience parameters will improve resources and throughput in microservices systems.

Keywords: MODEL, MICROSERVICES, RESILIENCE PATTERNS, CIRCUIT BREAKER, LOAD, NETWORK, METRICS, MONITORING.

ЗМІСТ

ВСТУП

Робота присвячена дослідженню та розробці моделі динамічного налаштування параметрів стійкості в мікросервісних архітектурах. Особливу увагу приділено автоматизації конфігурації шаблону Circuit Breaker (CB), який забезпечує управління відмовостійкістю сервісів. Метою роботи є підвищення надійності, продуктивності та адаптивності таких систем у динамічних умовах зміни навантаження.

Забезпечення стабільної роботи є критичним для систем мікросервісної архітектури. Оскільки програмні системи піддаються коливанням навантаження, здатність до адаптації в реальному часі дозволяє підвищити стійкість системи, зменшуючи ризик відмов. Це також оптимізує використання ресурсів і сприяє покращенню загальної продуктивності системи.

Об’єктом дослідження є процеси управління відмовостійкістю в мікросервісних системах, зокрема механізми адаптивного налаштування параметрів стійкості.

Предметом дослідження є методи, алгоритми моніторингу навантаження, оцінки параметрів продуктивності та адаптивного налаштування параметрів шаблонів стійкості з урахуванням поточного стану системи.

Метою роботи є розробка та теоретичне обґрунтування моделі динамічного налаштування параметрів шаблонів стійкості для підвищення надійності, продуктивності та адаптивності мікросервісних систем у змінних умовах навантаження.Запропоноване рішення відіграє роль у сучасних DevOps-практиках, сприяючи інтеграції принципів автоматизації, надійності та швидкої адаптації до змін середовища. Завдяки можливості автоматичного налаштування параметрів Circuit Breaker, система легко інтегрується у процеси CI/CD, забезпечуючи постійний контроль і оновлення параметрів конфігурації без необхідності ручного втручання.

Для досягнення цієї мети поставлено такі **задачі**:

1. Провести комплексний аналіз сучасних стратегій та практик налаштування параметрів стійкості у мікросервісних системах.
2. Розробити та впровадити програмне забезпечення для автоматизованого налаштування стійкості в мікросервісних системах.
3. Оцінити ефективність розробленої системи на практичних прикладах у реальному середовищі.

Методи дослідження. У роботі був проведений *аналіз* сучасних стратегій та практик налаштування параметрів стійкості в мікросервісних системах. Це дозволило визначити ключові недоліки існуючих рішень і сформулювати вимоги до нової моделі динамічного налаштування. Запропонована *модель* динамічного налаштування параметрів стійкості була реалізована та протестована за допомогою *комп'ютерного моделювання*.

Ці методи забезпечили комплексний підхід до дослідження. Теоретичний аналіз дозволив обґрунтувати необхідність розробки нової моделі, комп'ютерне моделювання — створити та протестувати її.

Актуальність роботи зумовлена зростанням популярності мікросервісної архітектури у сучасних інформаційних системах, що потребує високої гнучкості, масштабованості та стійкості. Існуючі рішення для управління міжсервісною комунікацією часто не відповідають реальним потребам, оскільки не враховують змінне навантаження та вимагають ручного втручання для коригування параметрів. Дослідження та впровадження механізмів динамічного налаштування дозволять усунути ці недоліки.

Очікувані результати роботи включають розробку та впровадження моделі автоматизованого налаштування параметрів стійкості, що забезпечить підвищення продуктивності, надійності та адаптивності мікросервісних систем. Результати дослідження можуть бути інтегровані в існуючі програмні рішення для управління мікросервісною архітектурою, підвищуючи їхню ефективність та стабільність у реальних умовах експлуатації.

РОЗДІЛ 1
АНАЛІЗ КОНЦЕПЦІЙ І ПІДХОДІВ ДО ПОБУДОВИ СТІЙКИХ МІКРОСЕРВІСНИХ СИСТЕМ

1.1 Аналіз архітектурних стилів. Мікросервісна архітектура

У сучасній сфері інформаційних технологій мікросервісна архітектура (Microservices Architecture, MSA) набула значної популярності як одна з провідних парадигм розробки програмного забезпечення. Попри відсутність формального визначення, мікросервісна архітектура зазвичай трактується як підхід до побудови програмних систем, що базується на використанні невеликих, автономних компонентів, кожен з яких спеціалізується на виконанні окремих функцій або обробці конкретних запитів. Ці компоненти, відомі як мікросервіси, можуть бути розроблені, протестовані та впроваджені незалежно один від одного.

1.1.1 Визначення, концепція та характеристики мікросервісної архітектури

Мікросервісна архітектура відрізняється від традиційної монолітної архітектури, у якій всі компоненти системи розташовуються в єдиному процесі. Порівнюючи архітектурні стилі, можна зазначити, що монолітні системи масштабуються шляхом реплікації цілого додатка на кількох серверах. Натомість у мікросервісній архітектурі кожен функціональний елемент виділяється в окремий сервіс, що дозволяє масштабувати лише ті компоненти, які цього потребують. Таким чином, замість дублювання всього додатка, розробники можуть сконцентрувати ресурси на масштабуванні лише критичних сервісів [1]. Візуально різницю між стилями зображено на рис. 1.1.

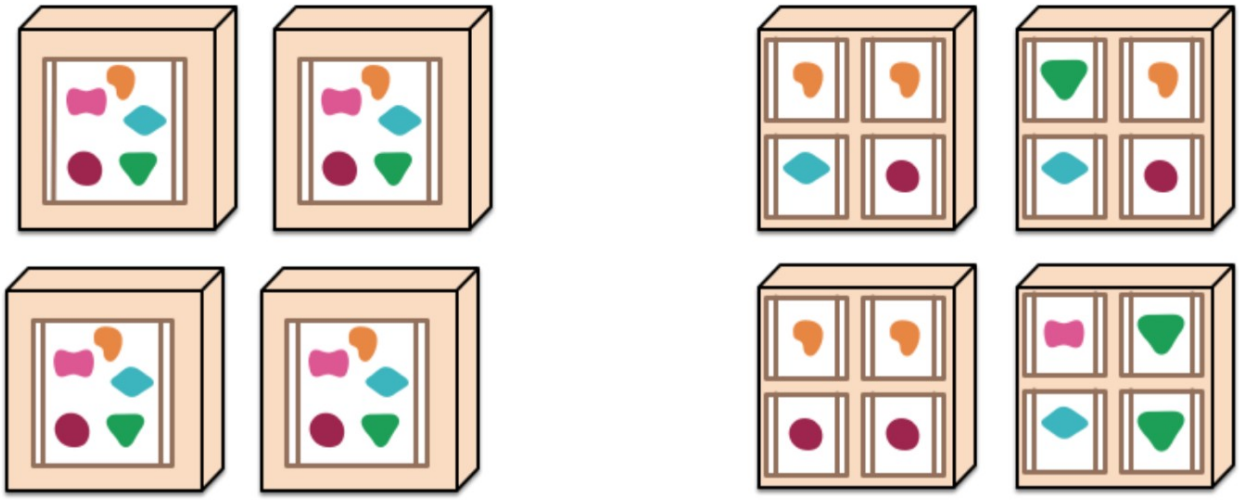


Рисунок 1.1 — Маштабування монолітних додатків та мікросервісних систем [1].

Основні характеристики мікросервісної архітектури [2]:

1. Незалежність: Кожен мікросервіс є автономним і може функціонувати окремо від інших. Це дозволяє проводити оновлення, виправлення помилок або навіть замінювати окремі сервіси без впливу на всю систему.
2. Гнучкість: Розробники мають можливість використовувати різні мови програмування, фреймворки та бази даних для кожного окремого мікросервісу. Це дає змогу оптимізувати кожен компонент під конкретні задачі.
3. Масштабованість: Мікросервісна архітектура дозволяє масштабувати окремі сервіси незалежно один від одного, що забезпечує ефективніше використання обчислювальних ресурсів.
4. Комунікація: Мікросервіси взаємодіють між собою через легковагі протоколи, найчастіше HTTP, REST, gRPC або AMQP, що забезпечує гнучкість та інтеграцію з іншими системами.
5. Надійність: Вихід з ладу одного мікросервісу зазвичай не впливає на роботу інших компонентів, що підвищує стійкість системи до відмов.

Основні принципи побудови мікросервісної архітектури можуть бути

описані наступними положеннями [3]:

1. Індивідуальність компонентів – кожен сервіс виконує одну конкретну функцію.
2. Висока модульність – сервіси незалежні один від одного і можуть бути розгорнуті окремо.
3. Автономія розробки – кожна команда відповідає за розробку, тестування і розгортання свого сервісу.
4. Децентралізоване управління даними – кожен мікросервіс може мати власну базу даних або доступ до окремого джерела даних.
5. Легка комунікація між компонентами – сервіси обмінюються даними за допомогою стандартних протоколів.
6. Автоматизація процесів – Continuos Integration/Continuos Delivery (CI/CD) забезпечує швидке оновлення сервісів з мінімальним ризиком збоїв.

Тож, була розглянута загальна структура систем мікросервісної архітектури, її принципи та характеристики

1.1.2 Актуальність дослідження мікросервісних систем

У сучасних умовах інтенсивного розвитку інформаційних технологій мікросервісна архітектура набуває дедалі більшої популярності завдяки своїй гнучкості, масштабованості та відмовостійкості. Однак ефективне функціонування мікросервісів значною мірою залежить від здатності системи адаптуватися до змінного навантаження. Існуючі рішення для управління міжсервісною комунікацією часто не відповідають реальним вимогам, що зумовлює необхідність дослідження нових підходів до динамічного налаштування параметрів стійкості.

Аналіз, моделювання та впровадження оптимальних параметрів управління мікросервісами сприятимуть покращенню їхньої продуктивності,

підвищенню надійності та зменшенню простоїв. Це, у свою чергу, дозволить ефективніше використовувати ресурси системи та забезпечить її стабільність навіть за значних коливань навантаження.

1.2 Основні шаблони стійкості. Обґрунтування вибору СВ

Шаблони стійкості є важливими інструментами для забезпечення безперервної роботи програмних систем, особливо в умовах високих навантажень та непередбачених помилок. Шаблон стійкості в контексті математичного моделювання та автоматичного управління визначає, як система реагує на зміни в початкових умовах або параметрах. Це поняття важливе для аналізу динамічних систем, оскільки дозволяє оцінити, чи зможе система повернутися до рівноваги після зовнішніх впливів.

1.2.1 Визначення та призначення шаблонів стійкості

Стійкість системи — це її здатність повертатися до стабільного стану після відхилень. У математичних моделях, таких як різницеві схеми для рівнянь переносу, стійкість визначається через спектр власних чисел, що виникають при дискретизації диференціальних рівнянь. Якщо всі власні числа лежать всередині одиничного кола в комплексній площині, схема вважається стійкою [4].

Шаблон стійкості (ШС) — це шаблон проєктування, який надає стандартизований підхід до забезпечення безперервної роботи системи при виникненні непередбачених помилок або перевантажень. ШС дозволяють автоматизувати процеси відновлення системи після збоїв, мінімізуючи вплив помилок на інші компоненти і сервісні взаємодії [4].

1. Шаблони стійкості потрібні для:
2. Підвищення надійності системи шляхом автоматизації реагування на помилки.

3. Зниження ризику повного відмовлення сервісу через несподівані помилки.
4. Підтримки високої доступності та безперервної роботи системи в умовах високого навантаження.

Стійкість у мікросервісній архітектурі є критично важливою характеристикою, яка забезпечує надійність і доступність системи. Використання сучасних методів та інструментів дозволяє створювати гнучкі рішення, здатні адаптуватися до змінних умов роботи та забезпечувати безперебійну роботу навіть у разі часткових відмов.

1.2.2 Огляд основних шаблонів стійкості

У цьому розділі описані лише деякі шаблони стійкості, оскільки вони є найбільш поширеними. В цілях роботи немає повний розгляд та аналіз шаблонів та паттернів що існують, деякі з них потребують згадки, але не розглянуті в роботі [5].

Шаблон «Автоматичний запобіжник» (Circuit Breaker)

Circuit Breaker (автоматичний запобіжник, CB) — це шаблон, який дозволяє системі реагувати на невдачі в окремих компонентах, тимчасово ізолюючи їх для запобігання подальшим помилкам і перевантаженням. Коли кількість помилок перевищує встановлений поріг, ланцюг розривається, і запити більше не надсилаються до цього компонента, доки система не відновиться.

Шаблон «Перегородка» (Bulkhead)

Bulkhead ізолює окремі частини системи, дозволяючи їм працювати незалежно одна від одної. Це дозволяє зберігати стабільність і надійність роботи одних частин системи, навіть якщо інші частини зазнають відмов.

Шаблон «Обмежувач швидкості» (Rate Limiter)

Rate Limiter обмежує кількість запитів або дій, що можуть бути виконані в певний проміжок часу. Це допомагає зменшити навантаження на систему, попереджаючи її перевантаження в разі надмірного запиту.

Шаблон «Повтор запиту» (Retry)

Retry дозволяє автоматично повторювати операцію після невдачі. Це може бути корисним для тимчасових помилок або збоїв, які можуть бути вирішені після кількох спроб.

Між вказаними шаблонами, в рамках кваліфікаційної роботи, був досліджений шаблон «Автоматичний запобіжник», а саме були розглянуті особливості роботи, конфігурації та

1.2.3 Обґрунтування дослідження Circuit Breaker

Circuit Breaker є потужним інструментом для підвищення надійності і стійкості системи. Він дозволяє значно зменшити ризик аварійного відключення сервісів за рахунок своєчасного виявлення і ізоляції проблемних компонентів. Вибір Circuit Breaker для цього дослідження обґрунтований його здатністю до автоматичного реагування на перевантаження та помилки, що особливо важливо для мікросервісних систем, де кожен окремий сервіс може стати слабким ланцюгом у загальній системі.

Також, потрібно зазначити, що CB широко досліджується, і має велику кількість областей. На рис 1.2 приведено дерево областей дослідження.

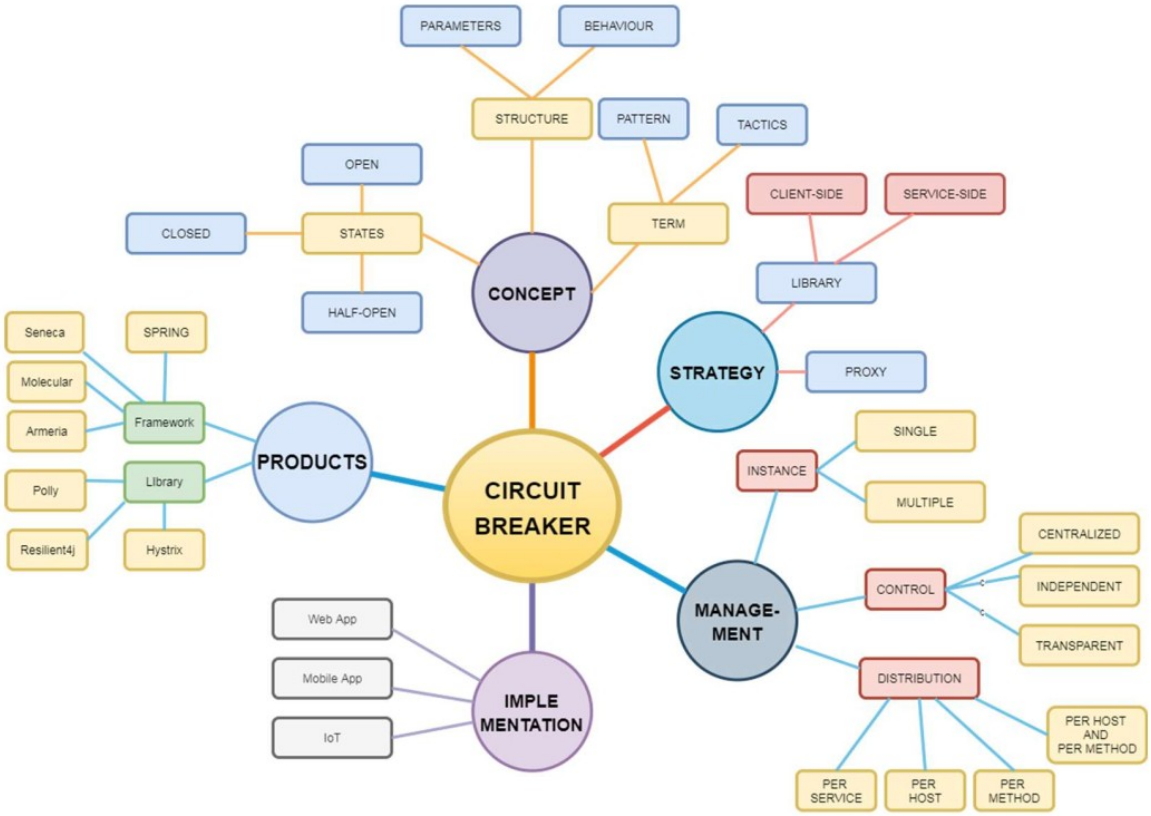


Рисунок 1.2 — Області дослідження Circuit Breaker в системах мікросервісної архітектури [4].

Дипломна робота спрямована на поглиблення розуміння параметрів та особливостей функціонування структури СВ, зокрема автоматизацію процесу визначення оптимальних параметрів і аналіз подальшої поведінки сервісу.

1.3 Детальний розгляд Circuit Breaker

Circuit Breaker є ключовим елементом шаблонів стійкості і може мати різні ролі в системі, включаючи паттерн, тактику та елемент безпеки. У межах цієї роботи ми розглядаємо Circuit Breaker як паттерн, який допомагає забезпечити стійкість мікросервісних архітектур через ефективне управління відмовами та перевантаженнями.

Переваги використання СВ можна описати наступним чином [5]:

1. Запобігання доступу до несправного компонента.
2. Швидке та елегантне оброблення помилок.
3. Запобігання зависанню викликів на несправних сервісах.
4. Можливість налаштування дій на випадок відмови.
5. Запобігання перевантаженню сервісів шляхом обмеження кількості запитів.

Ці переваги є основними в налаштуванні CB і визначають ситуації його використання в системах.

1.3.1 Основні компоненти та стани

Більшість статей узгоджуються в тому, що CB є "містком" між клієнтом і цільовим сервісом, який запобігає зверненню до сервісу, коли він недоступний [4]. Як місток, CB може перебувати в трьох станах: відкритий, закритий та напіввідкритий. У закритому стані всі запити передаються до сервісу, в відкритому — усі запити блокуються. Візуалізація взаємодії між станами приведена на Рис 1.3:

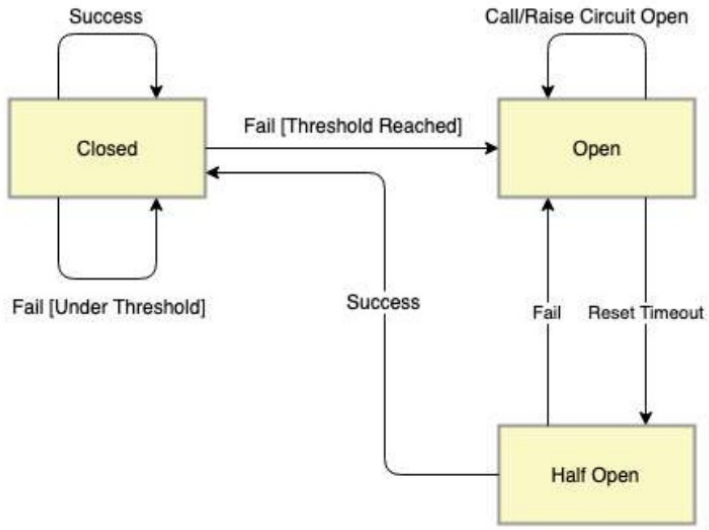


Рисунок 1.3 — Механізм роботи Circuit Breaker.

Опис трьох основних станів Circuit Breaker:

1. *Закритий (Closed):* Усі запити передаються до цільового сервісу. Це триває до того часу, поки система не виявить помилку, яка перевищує певний поріг (наприклад, лічильник тайм-аутів або кількість помилок). Після цього СВ переходить в стан відкритий. Поріг можна налаштувати або визначити системою.
2. *Відкритий (Open):* Усі запити блокуються СВ, і він або інший компонент, як правило, надсилає повідомлення про помилку запитувачу або виконує процедуру обробки відмов.
3. *Напіввідкритий (Half-open):* У цьому стані, коли сервіс відмовив, і СВ перебуває у відкритому стані, Circuit Breaker намагається періодично зв'язатися з сервісом, щоб перевірити його працездатність. Коли сервіс стає доступним, СВ знову переходить в закритий стан, і запити знову передаються до сервісу.

Більшість сучасних авторів використовують саме ці три концепції стану для роботи шаблону, але існують інші ідеї які в роботі ігноруються.

1.3.2 Механізм роботи Circuit Breaker

Механізм роботи Circuit Breaker можна описати наступним чином, зображеним на рисунку 1.4:

```
[Ініціалізація]
Параметри (тайм-аут, поріг помилок)
Initial_state = closed
[Стани]

    State = Closed
        Почати
        Переслати повідомлення до сервісу
        Якщо помилка, обробка помилки
        Встановити стан = open
        Кінець

    State = Open
        Почати
        Обробка помилки
        Встановити тайм-аут
        Встановити стан = half-open
        Кінець

    State = Half-open
        Почати
        Перевірка сервісу (через певні періоди)
        Якщо відповідь = true, встановити стан = closed
        Кінець
```

Рисунок 1.4 — Псевдокод що описує механізм роботи.

З цієї структури можна виділити три основні елементи Circuit Breaker: параметри, фрагменти коду, що описують дії для кожного стану, обробка помилок та альтернативні дії.

Через те що не було знайдено жодного дослідження, що описує принципи та підходи до визначення параметрів в залежності від навантаження системи, нас найбільше цікавить саме цей елемент шаблону — параметри. Також, з іншої сторони, немає жодного сенсу розробляти особисту реалізацію шаблону. В спільноті уже існують рішення, які неформально вважаються «стандартами» для кожного типу стратегії реалізації.

Робота Circuit Breaker залежить від кількох ключових параметрів, які визначають, коли і як буде активовано захист від перевантаження або помилок. Основні параметри:

1. Тайм-аут

2. Максимальна кількість одночасних запитів
3. Поріг обсягу запитів
4. Часове вікно сну (Sleep Window)
5. Поріг відсотка помилок

Механізми пошуку оптимальних параметрів, що будуть вдало описувати навантаження на сервіс і захищати його від збоїв у системі наразі не описані в науковій спільноті і визначаються адміністраторами систем емпірично, відповідно до реакції системи на параметри.

1.3.3 Стратегії реалізації CB

Існують три основні стратегії реалізації Circuit Breaker в мікросервісах [4]:

Client-side CB

У цьому випадку Circuit Breaker вбудований безпосередньо в клієнтську частину компонента. Це дозволяє легко і гнучко реалізувати CB на рівні кожного окремого клієнта. Проте є важливий недолік: інформація про доступність сервісів доступна лише локально для клієнта. Якщо кілька клієнтів звертаються до одного сервісу, кожен з них повинен мати окремо реалізований Circuit Breaker.

Server-side CB

У цій стратегії Circuit Breaker реалізується на рівні сервісу, що дозволяє публікувати стан доступності сервісу та читати його іншим компонентам. Однак, часто провайдери сервісів не дозволяють іншим сторонам змінювати їхній код або додавати механізми управління відмовами. Це обмежує можливості застосування цього підходу, особливо в умовах великої кількості мікросервісів.

Proxy-side CB

Proxy-side CB передбачає реалізацію Circuit Breaker в окремому проксі-

сервісі, який розміщується між клієнтами та сервісами. Цей підхід дозволяє реалізувати Circuit Breaker без необхідності модифікації клієнтських або сервісних компонентів. Проксі-сервер може централізовано керувати станами Circuit Breaker та здійснювати розподілене управління на рівні хостів або методів.

Проксі-сервер можна розглядати як альтернативну стратегію для впровадження СВ, оскільки він дає нам більш гнучкіший спосіб реалізації СВ без модифікації клієнта або сервісу. Можливість впровадження КБ на стороні проксі-сервісу може бути пов'язана з конкретною проблемою в іншій дослідницькій галузі. Але в сучасних системах зазвичай достатньо перших двух стратегій.

Тож, дослідження механізмів та стратегій реалізації Circuit Breaker є актуальним у сучасних мікросервісних архітектурах, оскільки вони дозволяють підвищити надійність і стійкість систем, запобігаючи каскадним відмовам при високих навантаженнях. Оскільки мікросервіси часто взаємодіють через нестабільні мережі або зазнають змін у навантаженнях, використання СВ стає критичним для забезпечення безперервної роботи систем. Визначення оптимальних параметрів для налаштування СВ, таких як тайм-аут, поріг помилок та максимальна кількість запитів, дозволяє адаптувати систему до динамічних змін у навантаженні, мінімізуючи час простою та покращуючи ефективність обробки запитів. Тому дослідження в цій сфері є важливим для розробки більш гнучких та адаптивних підходів до управління відмовами в складних розподілених системах.

1.4 Системи моніторингу та метрики

Системи моніторингу є невіддільною частиною сучасних мікросервісних архітектур. Вони забезпечують збір, зберігання та аналіз даних про стан

компонентів системи, дозволяючи оперативно виявляти проблеми, оцінювати ефективність роботи сервісів.

1.4.1 Причини моніторингу систем

Моніторинг мікросервісних систем необхідний для забезпечення їхньої надійності, продуктивності та доступності. Основні причини моніторингу включають [5]:

1. Виявлення збоїв — моніторинг дозволяє оперативно виявити відмови окремих компонентів та мінімізувати їх вплив на загальну роботу системи.
2. Оптимізація продуктивності — відстеження метрик допомагає виявити вузькі місця, що сповільнюють роботу сервісів, та забезпечити їх оптимізацію.
3. Підтримка стійкості системи — системи, що використовують шаблони стійкості, такі як Circuit Breaker, Bulkhead, Rate Limiter тощо, повинні відстежувати свою реакцію на зміни в середовищі. Наприклад:
    1. Circuit Breaker змінює свій стан між Closed, Open та Half-Open залежно від рівня помилок та доступності сервісу.
    2. Bulkhead обмежує кількість одночасних запитів до певних ресурсів, і його стан може змінюватись під час пікових навантажень.
    3. Rate Limiter контролює частоту запитів, адаптуючись до встановлених обмежень.

Таким чином, інтеграція шаблонів стійкості з системами моніторингу дозволяє оцінювати їхню ефективність та вчасно реагувати на зміну стану системи.

1.4.2 Механізми збереження метрик

Збір та зберігання метрик є важливою частиною систем моніторингу. Основні механізми збереження метрик включають [5]:

1. Централізоване зберігання — всі метрики зберігаються в єдиному сховищі даних для спрощення доступу та аналізу.
2. Інструменти для візуалізації — для кращого розуміння даних використовуються такі платформи, як Grafana, Kibana, які забезпечують графічне відображення метрик у режимі реального часу.

1.5 Аналіз сучасних рішень та обґрунтування роботи

Сучасні мікросервісні архітектури використовують різні підходи для забезпечення відмовостійкості. Основним рішенням є Circuit Breaker, який ізолює несправні сервіси, однак його параметри часто налаштовуються статично, що обмежує адаптивність системи до змінних умов.

CB зазвичай має фіксовані параметри (тайм-аут, поріг помилок), які визначаються як константні значення. У випадках зміни навантаження на сервіс шаблон потрібно реконфігурувати. Але наразі це потребує зупинки і повторного запуску сервісу.

Для досягнення цілей дослідження та розробки системи динамічного налаштування параметрів Circuit Breaker у мікросервісній архітектурі сформульовано наступну робочу гіпотезу:

Інтеграція системи для динамічного налаштування параметрів Circuit Breaker на основі актуальних та історичних даних про навантаження дозволить підвищити надійність, адаптивність та продуктивність мікросервісної системи, мінімізуючи час простою та знижуючи ризик відмов.

Для перевірки цієї гіпотези визначено ключові етапи реалізації в таблиці 2.1.

Таблиця 1.1

Етапи реалізації

1.	Реалізація програмного рішення:	Розробка програмного забезпечення, яке керує параметрами шаблону стійкості та інтегрується в мікросервісну систему для динамічного налаштування параметрів Circuit Breaker.
2.	Імплементація програмного забезпечення у тестове середовище	Розгортання та налаштування розробленого рішення у тестовому середовищі для оцінки його роботи в реальних умовах.
3.	Оцінка ефективності отриманої системи	Вимірювання ефективності запропонованого рішення за ключовими показниками, такими як зменшення часу простою, покращення продуктивності системи та стабільності роботи сервісів при різних рівнях навантаження

Ці етапи забезпечують системний підхід до перевірки гіпотези та дозволяють визначити, наскільки інтеграція системи покращує відмовостійкість мікросервісної архітектури.

Висновки до розділу 1

У першому розділі було проаналізовано основні аспекти мікросервісної архітектури, її ключові переваги та виклики. Мікросервісна архітектура забезпечує високу гнучкість, масштабованість і незалежність компонентів, що дозволяє створювати стійкі до відмов системи. Водночас такі системи потребують спеціальних механізмів для управління помилками та підтримання

стабільності, особливо в умовах високого взаємозв'язку між сервісами.

Шаблони стійкості, такі як Circuit Breaker, Bulkhead та Rate Limiter, є критично важливими для запобігання каскадним відмовам у системах із численними залежностями. Однак статичні конфігурації параметрів, які часто застосовуються для цих шаблонів, обмежують їхню здатність адаптуватися до змінних умов роботи, що може знижувати ефективність системи під час пікових навантажень.

Системи моніторингу та логування відіграють важливу роль у забезпеченні відмовостійкості, оскільки вони надають актуальні дані про стан системи. Ці дані можуть бути використані для аналізу ефективності роботи сервісів, а також для адаптації параметрів шаблонів стійкості відповідно до поточного стану системи.

Таким чином, метою даного дослідження є розробка та оцінка адаптивної конфігурації шаблону стійкості Circuit Breaker в мікросервісних системах з динамічним управлінням параметрами, що забезпечить підвищену надійність і ефективність роботи таких систем під час змінних навантажень і помилок.

РОЗДІЛ 2
РОЗРОБКА МОДЕЛІ АДАПТИВНОГО НАЛАШТУВАННЯ CIRCUIT BREAKER

2.1 Визначення вимог до програмного забезпечення

У цьому підрозділі визначено функціональні та нефункціональні вимоги до двох основних компонентів системи: моделі пошуку оптимальних значень параметрів Circuit Breaker та додатку для інтеграції моделі в мікросервісну архітектуру, перелік вимог зазначений в таблиці 2.1.

Таблиця 2.1

Вимоги до програмного забезпечення.

Категорія	Вимоги
Функціональні	1. Додавання нового типу сервісу. 2. Додавання залежних сервісів. 3. Редагування інформації про сервіси. 4. Автоматичне надсилання параметрів до відповідних сервісів. 5. Веб-інтерфейс для управління додатком. 6. Забезпечення безпеки (аутентифікація та захист даних). 7. Збереження та перегляд історії змін параметрів. 8. Інтеграція з сервісами для обробки конфігурацій.
Нефункціональні	1. Легка інтеграція з існуючою інфраструктурою. 2. Оптимальне використання ресурсів. 3. Всебічна безпека (захист каналів зв’язку, безпечне зберігання токенів тощо).

Ці вимоги забезпечують функціональність, надійність, безпеку та адаптивність системи, яка здатна ефективно працювати в умовах змінного навантаження та забезпечувати стабільну роботу мікросервісної архітектури.

2.2 Моделювання роботи мікросервісної системи

2.2.1 Симуляційна модель мікросервісної системи

Цей підрозділ описує побудову симуляційної моделі, яка відображає взаємодію мікросервісів у системі, враховуючи затримки відповідей та аномальні стани.

Система складається з каскадно залежних мікросервісів, де кожен виконує визначену роль (рис 2.1):

1. Сервіс A — ініціює запити до сервісів B і C.
2. Сервіс B — відповідає на запити від A.
3. Сервіс C — залежить від сервісу D для обробки запитів від A.
4. Сервіс D — забезпечує підтримуючу функцію для сервісу C.

Всі сервіси працюють за заданими параметрами часу відповіді, а також можуть імітувати аномальні стани для аналізу стійкості системи.

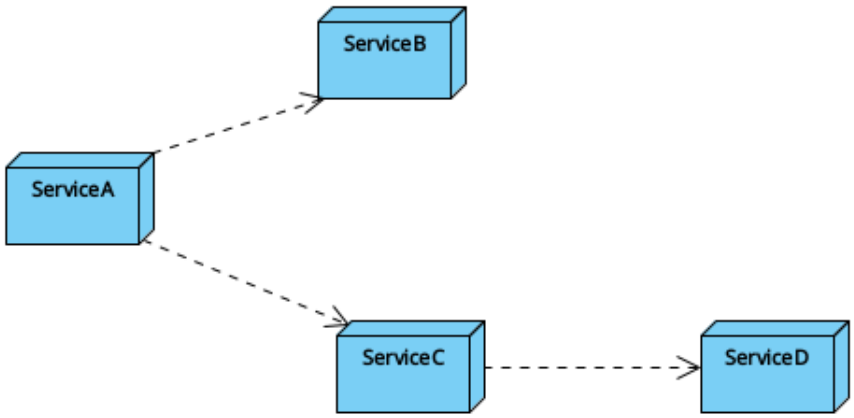


Рисунок 2.1 — Схематичне зображення каскадно залежних сервісів

Кожен сервіс (B, C, D) відповідає на запити з певною *затримкою*, яка визначається базовим часом відповіді, модифікованим випадковим фактором:

1. Базовий час відповіді (T_{base}) — встановлюється під час ініціалізації сервісу.
2. Випадковий фактор ($N(0,2)$) — нормально розподілене випадкове значення зі стандартним відхиленням *2* секунди.

Формула часу відповіді:

$$T_{response} = T_{base} + N(0,2)$$

Цей алгоритм дозволяє симулювати умови роботи сервісів з урахуванням випадкових затримок.

Аномальні стани кожного сервісу моделюються періодично:

1. Період розрахунку аномалій (T_{period}) — визначається випадковим чином з рівномірного розподілу в діапазоні від 0 до 120 секунд.
2. Зсув періоду (T_{shift}) — рівномірний розподіл між 0 і 60 секундами.
3. Момент початку аномалії (T_{anom}) — визначається за формулою:

$$T_{anom} = T_{period} + U(0,5)$$

Під час аномального стану час відповіді сервісу значно збільшується:

$$T_{anom\_response} = T_{response} \times K$$

де K — випадкове значення в діапазоні [10, 15].

Графік, що відображає часи відповіді сервісів A, B, C та D з аномаліями, розподіленими по частоті. що відображає, як час відповіді для кожного сервісу групується в певні інтервали, включаючи запити з аномальними затримками (рис. 2.2).

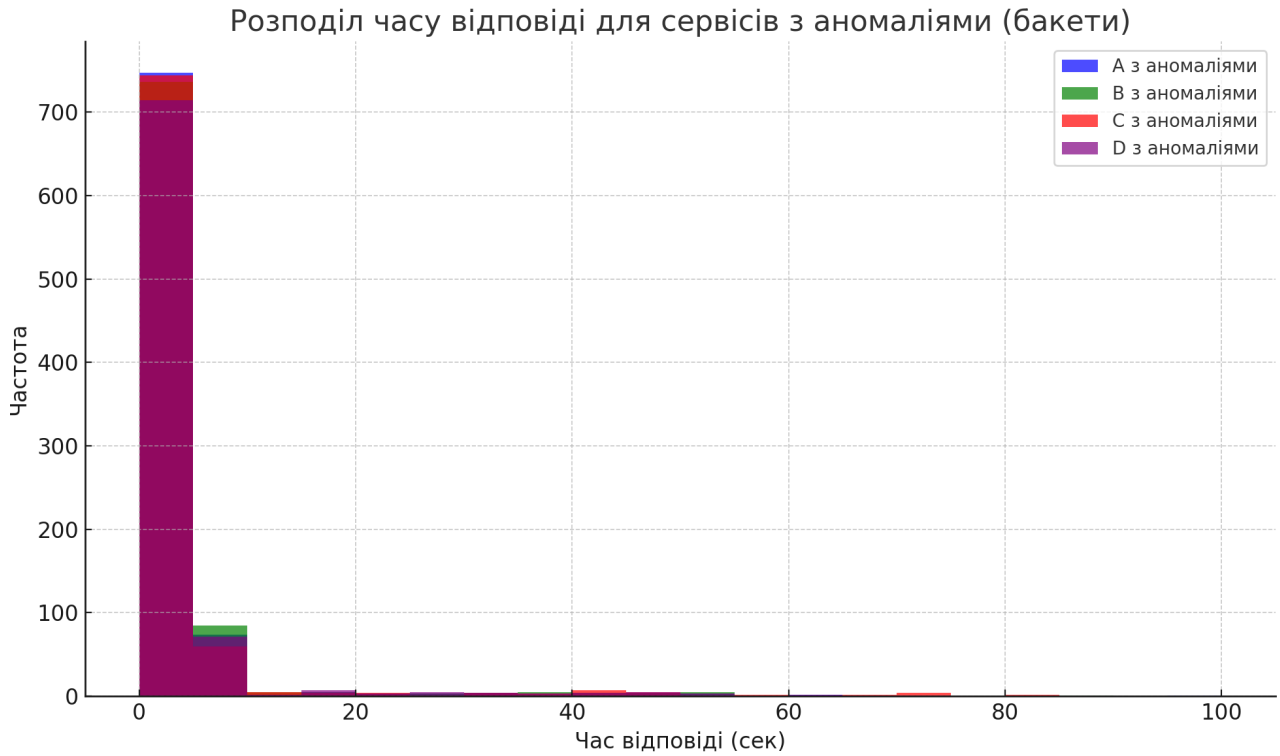


Рисунок 2.2 — Розподіл часу відповіді для сервісів з урахуванням аномальних станів.

Описані алгоритми та вірогідності дозволяють оцінити вплив аномалій на продуктивність системи та її здатність до відновлення.

2.2.2 Моніторинг і аналіз продуктивності системи

Цей підрозділ описує підхід до моніторингу роботи мікросервісів у реальному часі, а також метрики, які використовуються для аналізу продуктивності системи.

Кожен сервіс (A, B, C, D) здійснює постійний *моніторинг часу відповіді*, збираючи та передаючи дані до централізованої системи моніторингу. Основні аспекти моніторингу:

1. Середній час відповіді (ART) — використовується для оцінки загальної продуктивності.

2. Перцентилі часу відповіді (95-й і 99-й) — дозволяють визначити максимальні затримки в екстремальних випадках.
3. Кількість запитів із перевищенням порогового значення затримки — використовується для виявлення критичних проблем.

Зібрані дані агрегуються централізованою системою моніторингу для подальшого аналізу та візуалізації.

Система відстежує стан шаблону автоматичного вимикача для кожного сервісу. Моніторинг охоплює:

1. Поточний стан СВ:
   - Закритий стан — сервіс працює нормально, обробляючи всі запити.
   - Напіввідкритий стан — виконується обмежена кількість запитів для перевірки стабільності.
   - Відкритий стан — запити блокуються для уникнення подальших збоїв.
2. Контекстні метрики:
   - Кількість невдалих запитів, які призвели до зміни стану СВ.
   - Час переходу між станами.
   - Частка успішних запитів після повернення з напіввідкритого стану.

Ці дані передаються до системи моніторингу для оцінки ефективності налаштувань Circuit Breaker і загальної стійкості системи.

2.2.3 Шаблон стійкості Circuit Breaker та налаштування

Для забезпечення стійкості мікросервісної системи до збоїв і затримок у відповідях використовується шаблон Circuit Breaker. Цей підхід дозволяє сервісам автоматично адаптуватися до проблем, що виникають, зберігаючи загальну стабільність системи.

Кожен сервіс має власний Circuit Breaker, який *працює у трьох станах*:

1. *Закритий стан:* Усі запити обробляються нормально.

2. *Відкритий стан:* Запити не обробляються, щоб уникнути додаткових збоїв. Сервіс очікує закінчення періоду відновлення (Recovery Timeout).
3. *Напіввідкритий стан:* Обмежена кількість запитів дозволяється для перевірки стабільності системи. Якщо запити виконуються успішно, CB переходить у закритий стан.

Основні параметри налаштування Circuit Breaker

1. *Threshold (Порогове значення):* Визначає, коли CB переходить у відкритий або напіввідкритий стан.
2. *Failure Rate Threshold (Порогова частка помилок):* Частка невдалих запитів, після досягнення якої CB відкривається.
3. *Recovery Timeout (Час відновлення):* Час, після якого CB намагається повернутися в напіввідкритий стан для перевірки стабільності.
4. *Max Requests (Максимальна кількість запитів у напіввідкритому стані):* Кількість дозволених запитів для оцінки працездатності сервісу.

Зміна параметрів Circuit Breaker *впливає на стійкість і продуктивність* системи:

1. Зменшення порогу помилок прискорює перехід до відкритого стану, зменшуючи ризик перевантаження.
2. Подовження часу відновлення дозволяє уникнути передчасного повернення до нормального стану.
3. Збільшення кількості запитів у напіввідкритому стані дозволяє гнучкіше перевіряти стабільність, але може створювати додаткове навантаження.

Шаблон Circuit Breaker є критично важливим для підтримання стійкості системи, особливо в умовах аномальних станів сервісів.

2.2.4 Очікувані результати тестування системи

Після впровадження симуляційної моделі очікується, що тестування

дозволить оцінити продуктивність і стійкість мікросервісної системи за різних умов. Зокрема, буде отримано метрики часу відповіді (середній час, перцентилі, частка запитів із високою затримкою) та проаналізовано поведінку системи під час аномальних станів. Моніторинг шаблону Circuit Breaker забезпечить розуміння ефективності параметрів конфігурації для запобігання збоїв і швидкого відновлення сервісів.

Зміна параметрів системи, таких як час відповіді, рівень аномальності та конфігурація Circuit Breaker, дозволить виявити їхній вплив на показники стійкості. Очікується, що результати тестування продемонструють зменшення часу простою сервісів, стабільну роботу за умов високого навантаження та покращення продуктивності завдяки адаптивному налаштуванню.

2.3 Вибір програмних засобів для реалізації тестового стенду та системи динамічного конфігурування

Для реалізації тестового стенду та динамічного налаштування параметрів Circuit Breaker обрано програмні засоби, які забезпечують надійність, масштабованість, та зручність інтеграції. Нижче наведено перелік інструментів із обґрунтуванням їх використання.

2.3.1 Java Spring Boot + Spring Boot Actuator

Spring Boot забезпечує платформу для створення мікросервісів на базі Java. Він дозволяє швидко налаштувати сервіс із вбудованими конфігураціями для підключення, безпеки та управління [9].

Таблиця 2.2

Огляд технології Java Spring Boot

Причини	1. Висока продуктивність і надійність.
	2. Підтримка масштабованості та інтеграції з іншими інструментами.
	3. Інтеграція з бібліотекою Resilience4j для реалізації шаблону Circuit Breaker.

Java Spring Boot Actuator надає готові ендпоінти для моніторингу та управління мікросервісами Spring Boot, що дозволяє отримувати інформацію про метрики продуктивності, статус сервісу та стан Circuit Breaker.

2.3.2 Resilience4j

Resilience4j реалізує наступні шаблони стійкості: Circuit Breaker, Rate Limiter, Bulkhead і Retry, що дозволяє управляти поведінкою сервісів під час збоїв [10, 11].

Таблиця 2.3

Огляд технології Resilience4j

Причини	1. Підтримка асинхронних операцій і реактивного програмування	
	2. Гнучка конфігурація Circuit Breaker та інших шаблонів	
	3. Висока сумісність із Spring Boot.	
Аналоги	1. Hystrix	застарілий
	2. Sentinel	краще підходить для великих систем на базі Alibaba Cloud

2.3.3 Ruby on Rails

Ruby on Rails використовується для створення REST API та допоміжних сервісів, які підтримують основні мікросервіси. Rails забезпечує швидку розробку та інтеграцію з іншими компонентами системи [12].

Таблиця 2.4

Огляд технології Ruby on Rails

Причини	1. Швидка розробка CRUD-сервісів	
	2. Зручний синтаксис для інтеграції з базами даних.	
	3. Легка інтеграція з Docker для контейнеризації.	
Аналоги	1. Django (Python)	Не визначено
	2. Express.js	Не визначено

2.3.4 Prometheus

Prometheus забезпечує збір, зберігання та обробку метрик у реальному часі, зокрема метрик часу відповіді, статусу сервісів та стану Circuit Breaker [9] [13].

Таблиця 2.5

Огляд технології Prometheus

Причини	1. Висока продуктивність для збору великого обсягу метрик	
	2. Простота інтеграції з Docker, Spring Boot Actuator та Resilience4j.	
Аналоги	1. Grafana Loki	зосереджений на логах, а не на метриках
	2. Zabbix	важча конфігурація, не орієнтований на мікросервіси

2.3.5 Docker і Docker Compose

Docker та Docker Compose забезпечують контейнеризацію сервісів, що дозволяє легко розгортати, тестувати та масштабувати систему [14, 15].

Таблиця 2.6

Огляд технології Docker

Причини	1. Швидке розгортання ізольованих контейнерів.
	2. Підтримка різних середовищ для тестування та розробки.
Аналоги	Не розглянуті

Також був розглянутий Kubernetes на роль оркестратора системи, але це потребувало значного ускладнення системи. Мається на увазі, потреба у визначенні адрес сервісів, які потребують оновлення параметрів за рахунок хуків [16].

2.4 Методика оцінки ефективності запропонованого рішення

Основними критеріями ефективності системи є зменшення часу простою та зниження кількості відмов під час обробки запитів. Ці показники відображають стабільність і надійність роботи сервісів у реальних умовах. Відповідно, оптимізація параметрів конфігурації повинна сприяти мінімізації впливу навантажень і аномалій на загальну продуктивність системи.

Висновки до розділу 2

У цьому розділі було визначено етапи реалізації та вимоги до системи, розроблено модель тестового стенду та обґрунтовано вибір програмних засобів.

Система повинна забезпечувати динамічне управління параметрами Circuit Breaker, підтримку веб-інтерфейсу для зручного адміністрування, а також гарантувати високий рівень безпеки та легкість інтеграції з існуючою інфраструктурою. Важливим аспектом є забезпечення гнучкості налаштувань для адаптації до змінних умов навантаження та поведінки системи.

Модель тестового стенду передбачає створення середовища для симуляції навантаження та затримок між сервісами, що дозволяє ефективно моніторити метрики продуктивності та тестувати параметри Circuit Breaker в умовах реальних сценаріїв. Це дає змогу оцінити стійкість та надійність системи за різних умов експлуатації.

Для реалізації мікросервісної архітектури було обрано технології Java Spring Boot з бібліотекою Resilience4j для реалізації паттерну Circuit Breaker, Prometheus для моніторингу системи та Docker для контейнеризації всіх компонентів. Методика оцінки ефективності включає критерії, такі як час простою, кількість відмов та стабільність роботи, що дозволяють провести порівняльний аналіз до і після застосування динамічного налаштування параметрів Circuit Breaker.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

У цьому розділі розглянуто практичну реалізацію тестового стенду для оптимізації параметрів Circuit Breaker в системах, що включають кілька взаємозалежних сервісів. Описано архітектуру, використані технології та програмні засоби для створення тестового середовища. Всі компоненти були реалізовані з використанням Docker та Docker Compose для контейнеризації сервісів, а також Java Spring Boot для розробки серверних додатків. Крім того, було впроваджено систему моніторингу з Prometheus для збору метрик та аналізу продуктивності, що дозволило реалізувати адаптивне налаштування параметрів Circuit Breaker на основі даних, отриманих в реальному часі.

Аналіз результатів тестування включає порівняння ефективності роботи системи в умовах статичних та динамічних налаштувань параметрів Circuit Breaker. Результати показали значне зниження часу відповіді сервісів, зменшення кількості відмов та збоїв, а також покращення загальної стійкості системи до аномалій. Оцінка ефективності нових параметрів конфігурації дозволила визначити оптимальні налаштування для кожного сервісу, що забезпечило стабільну роботу системи навіть в умовах підвищеного навантаження.

3.1 Реалізація тестового стенду та програмного забезпечення

Тестовий стенд був реалізований за допомогою Docker та Docker Compose, що забезпечує легке розгортання й управління середовищем. Сервіси розроблені з використанням Java Spring Boot і виконуються в контейнерах, які базуються на образі openjdk:17-jdk-alpine. Це дозволяє отримати компактні, оптимізовані для роботи в контейнерах сервіси. На рисунку 3.1 схематично наведено структура тестового стенду, що був розроблений в рамках поточної

рооти [17].

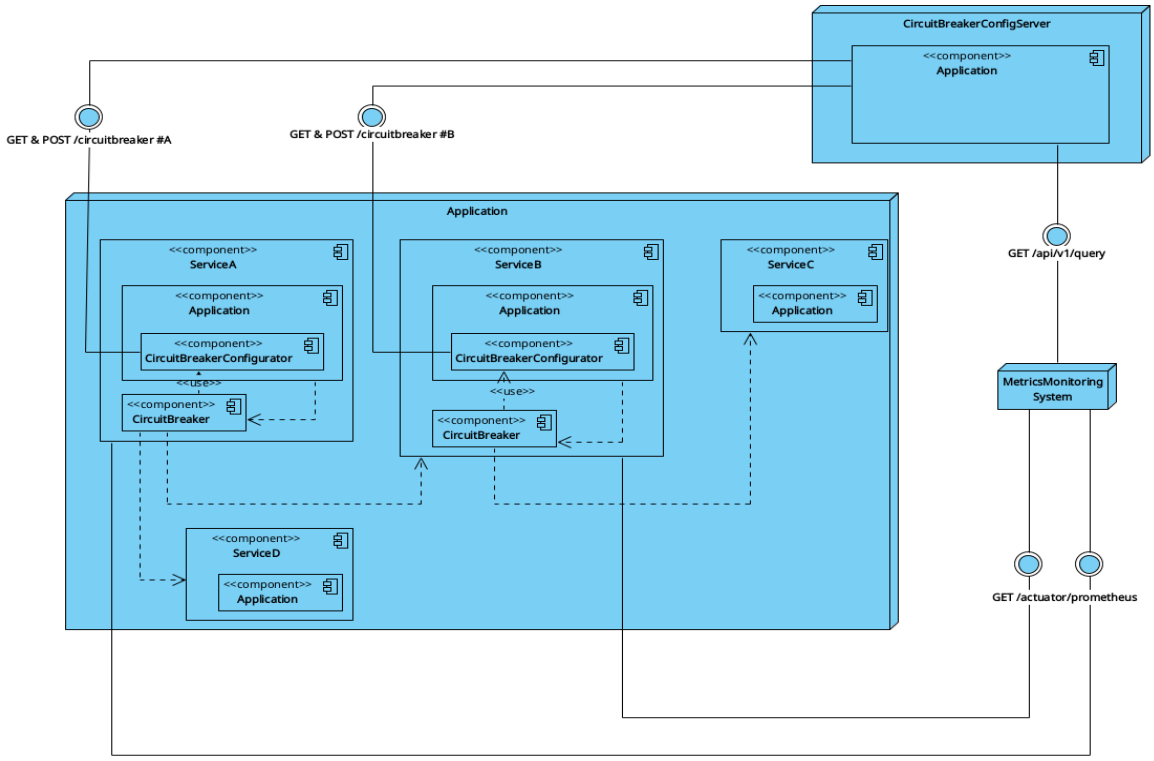


Рисунок 3.1 —Діаграма компонентів для тестового стенду.

Кожен сервіс функціонує без використання баз даних для збереження інформації, що мінімізує затримки й підвищує простоту реалізації. Контейнери експортують наступні порти для доступу: *serviceA:3005*, *serviceB:3006*, *serviceC:3007*, *serviceD:3008*. Всі сервіси, окрім сервісу A, реалізують просте API із запитом *GET /process*, яке виконує таймер із затримкою, визначеною у відповідному розділі документа.

ServiceA є ініціатором запитів, і оцінка результатів проводиться відповідно до його метрик. Запити ініціюються відповідно до розділу «2.3.2 Затримка відповіді сервісів».

Крім того, до складу системи контейнерів входить Prometheus, який відповідає за збір метрик і забезпечує моніторинг продуктивності.

3.1.1 Налаштування середовища розробки

Нижче наведені фрагменти коду для налаштування тестового середовища, включаючи конфігурацію образів (рис. 3.3), контейнерів (рис. 3.4) і моніторинг за допомогою Prometheus (рис. 3.5).

```
FROM openjdk:17-jdk-alpine
WORKDIR /app
COPY target/service.jar /app/service.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "/app/service.jar"]
```

Рисунок 3.3 —Dockerfile для SpringBoot додатку.

```
version: '3.8'
services:
  serviceA:
build:
  context: .
  dockerfile: Dockerfile
container_name: serviceA
ports:
  - "3005:8080"
environment:
  - SERVICE_NAME=ServiceA
networks:
  - monitoring
  ...

prometheus:
image: prom/prometheus:latest
container_name: prometheus
volumes:
  - ./prometheus.yml:/etc/prometheus/prometheus.yml
ports:
  - "9090:9090"
networks:
  - monitoring

networks:
  monitoring:
driver: bridge
```

Рисунок 3.4 — docker-compose.yml (скорочено).

```
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'serviceA'
static_configs:
  - targets: ['serviceA:8080']
  ...
```

Рисунок 3.5 — prometheus.yml (скорочено).

Ці конфігураційні файли створюють середовище, в якому кожен сервіс працює у своєму контейнері та доступний через визначений порт. Prometheus забезпечує моніторинг, виконуючи регулярний збір метрик із кожного сервісу.

3.2 Розробка додатку

Для інтеграції шаблону стійкості (Circuit Breaker) у сервісну архітектуру було створено Java-бібліотеку, яка надає інтерфейс для управління конфігураціями вимикачів. Основною метою бібліотеки є забезпечення гнучкості в налаштуванні параметрів Circuit Breaker для кожного сервісу, а також інтеграція цих параметрів у робочий процес сервісів без необхідності перезапуску або втручання у внутрішню логіку.

Бібліотека реалізує два маршрути API: *GET /circuitbreaker/:service_name* та *POST /circuitbreaker/:service_name*. Перший маршрут повертає поточну конфігурацію Circuit Breaker у форматі JSON для конкретного сервісу, що дозволяє адміністратору або системі моніторингу отримати актуальні параметри. Другий маршрут дає змогу оновлювати ці параметри шляхом надсилання нового JSON-об'єкта, який зберігається за допомогою механізму @ConfigurationProperties, забезпечуючи динамічне застосування змін.

Клас CircuitBreakerProperties відіграє ключову роль у реалізації цієї функціональності. Він побудований як вкладена структура, яка підтримує множинні екземпляри вимикачів. Це дозволяє незалежно конфігурувати

параметри для кожного сервісу (наприклад, serviceB, serviceC) у межах одного додатку

На Рис. 3.6 наведено листинг класу CircuitBreakerProperties, який демонструє його структуру та використання в контексті обробки запитів. Реалізація кінцевих точок GET і POST, представлена на Рис. 3.7, ілюструє, як параметри отримуються й оновлюються для конкретних сервісів. Ці кінцеві точки забезпечують інтерактивну взаємодію між сервісами та системою моніторингу, дозволяючи зберігати стійкість архітектури навіть за умов високого навантаження чи аномалій.

```
    @ConfigurationProperties(prefix =
"resilience4j.circuitbreaker")
    @Data
    public class CircuitBreakerProperties {
    private Map<String, CircuitBreakerInstanceConfig> instances;

    @Data
    public static class CircuitBreakerInstanceConfig {
    private float failureRateThreshold;
    private float slowCallRateThreshold;
    private Duration slowCallDurationThreshold;
    private int permittedNumberOfCallsInHalfOpenState;
    private Duration maxWaitDurationInHalfOpenState;
    private String slidingWindowType;
    private int slidingWindowSize;
    private int minimumNumberOfCalls;
    private Duration waitDurationInOpenState;
    private boolean automaticTransitionFromOpenToHalfOpenEnabled;
    private List<Class<? extends Throwable>> recordExceptions;
    private List<Class<? extends Throwable>> ignoreExceptions;
    private String recordFailurePredicate;
    private String ignoreExceptionPredicate;
    }
    }
```

Рисунок 3.6 —Класс CircuitBreakerProperties.

```
    @RestController
    @RequestMapping("/circuitbreaker")
    public class CircuitBreakerController {

    @Autowired
    private CircuitBreakerProperties circuitBreakerProperties;

    @GetMapping("/{service}")
    public
ResponseEntity<CircuitBreakerProperties.CircuitBreakerInstanceConfi
g> getServiceConfig(
    @PathVariable String service) {
    var config =
circuitBreakerProperties.getInstances().get(service);
    if (config == null) {
    return ResponseEntity.notFound().build();
    }
    return ResponseEntity.ok(config);
    }

    @PostMapping("/{service}")
    public ResponseEntity<String> updateServiceConfig(
    @PathVariable String service,
    @RequestBody
CircuitBreakerProperties.CircuitBreakerInstanceConfig request) {
    circuitBreakerProperties.getInstances().put(service, request);
    return ResponseEntity.ok("Configuration for " + service + "
updated successfully");
    }
    }
```

Рисунок 3.7— Реалізація контроллеру.

Наприклад, щоб оновити характеристики serviceB, потрібно надіслати POST /circuitbreaker/serviceb запит, з Json-тілом приведеним на рисунці 3.8.

```
    {
      "failureRateThreshold": 50.0,
      "slowCallRateThreshold": 50.0,
      "slowCallDurationThreshold": "2s",
      "permittedNumberOfCallsInHalfOpenState": 5,
      "maxWaitDurationInHalfOpenState": "10s",
      "slidingWindowType": "COUNT_BASED",
      "slidingWindowSize": 10,
      "minimumNumberOfCalls": 10,
      "waitDurationInOpenState": "30s",
      "automaticTransitionFromOpenToHalfOpenEnabled": true,
      "recordExceptions": ["java.io.IOException",
"java.util.concurrent.TimeoutException"],
      "ignoreExceptions": ["java.lang.IllegalArgumentException"],
      "recordFailurePredicate":
"com.example.RecordFailurePredicate",
      "ignoreExceptionPredicate":
"com.example.IgnoreExceptionPredicate"
    }
```

Рисунок 3.8 — Приклад JSON-даних для оновлення serviceB.

Вище приведені ключові фрагменти бібліотеки resielence4j-circuitbreaker-config, що була зкомпільована. Ця бібліотека додається до існуючих сервісів додаванням залежності до Maven або Gradle

Розроблена бібліотека додає до сервісу окремі кінцеві точки, які забезпечують можливість роботи з параметрами Circuit Breaker у динамічному режимі Її функціонал відкриває важливі можливості для управління сервісами в умовах продакшену. Він дозволяє змінювати характеристики Circuit Breaker без необхідності зупинки сервісу чи його перезапуску для застосування нових конфігурацій. Це значно підвищує стійкість системи, спрощує адміністрування та зменшує потенційний час простою, спричинений редеплоєм із новими параметрами.

Такий підхід є важливим кроком у напрямку створення сучасних

високодоступних і динамічно налаштовуваних систем, де продуктивність і стабільність сервісів мають вирішальне значення.

Для управління характеристиками Circuit Breaker був розроблений додаток на базі Ruby on Rails, який використовує базу даних SQLite3. Цей додаток дозволяє зберігати та маніпулювати параметрами конфігурації, а також забезпечує збереження історії змін і взаємозалежностей між сервісами:

1. service — ця модель зберігає необхідну інформацію про кожний сервіс, включаючи його унікальний ідентифікатор, назву, ,порт через який можливо надсилати запити до сервісу.
2. dependency — модель, що описує залежності між сервісами, тобто, від яких сервісів залежить робота поточного сервісу. Ця модель дозволяє відслідковувати взаємозв'язки та впливи між компонентами системи.
3. configuration — ця модель зберігає інформацію про параметри конфігурації, а також час, коли вони були застосовані до відповідних сервісів. Вона дозволяє відслідковувати історію змін конфігурацій і здійснювати аудит змін у параметрах. Останнє додане значення до списку є актуальним для заданого сервісу.

Залежність компонентів графічно відображено на рисунку 3.9 [18].

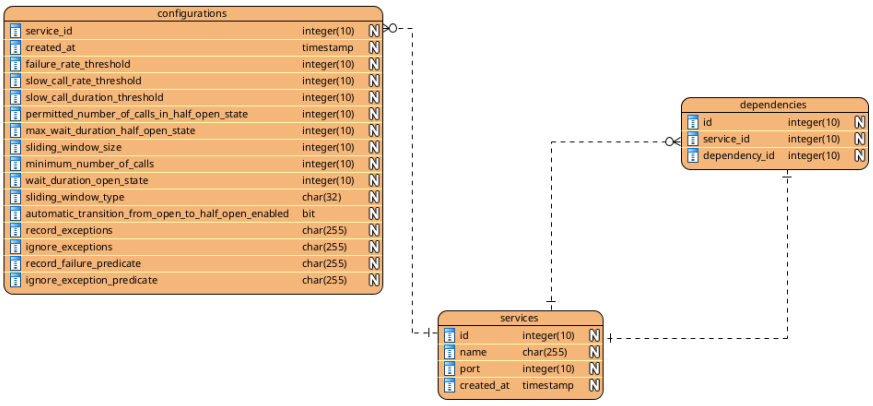


Рисунок 3.9 — ERD діаграма розробленого додатку для маніпуляції характеристиками CircuitBreaker.

Додаток таким чином забезпечує зручний інтерфейс для маніпулювання конфігураціями Circuit Breaker, зберігаючи всю необхідну інформацію в базі даних для подальшого аналізу і оптимізації.

Інтерфейси взаємодії з додатком були реалізовані через браузер, що дозволяє зручно керувати параметрами Circuit Breaker через веб-інтерфейс. Додаток надає користувачам можливість виконувати операції, такі як перегляд поточних налаштувань, зміна конфігурацій, а також моніторинг історії змін у реальному часі. Такий підхід забезпечує надійність та стабільність роботи системи, дозволяючи оперативно реагувати на можливі технічні проблеми. На відповідних рисунках нижче наведено приклади функціонування програми, зокрема форми авторизації (рис. 3.10), домашньої сторінки (рис. 3.11) та інтерфейсу для створення або редагування сервісу (рис. 3.12).

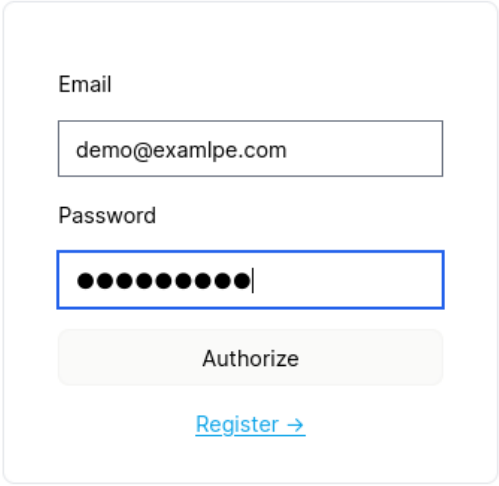


Рисунок 3.10 — Форма авторизації в систему.

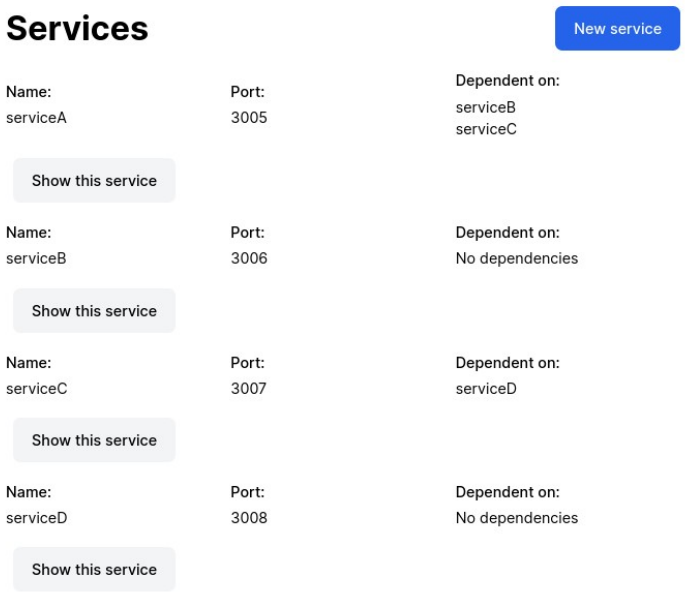


Рисунок 3.11 — Домашня сторінка.

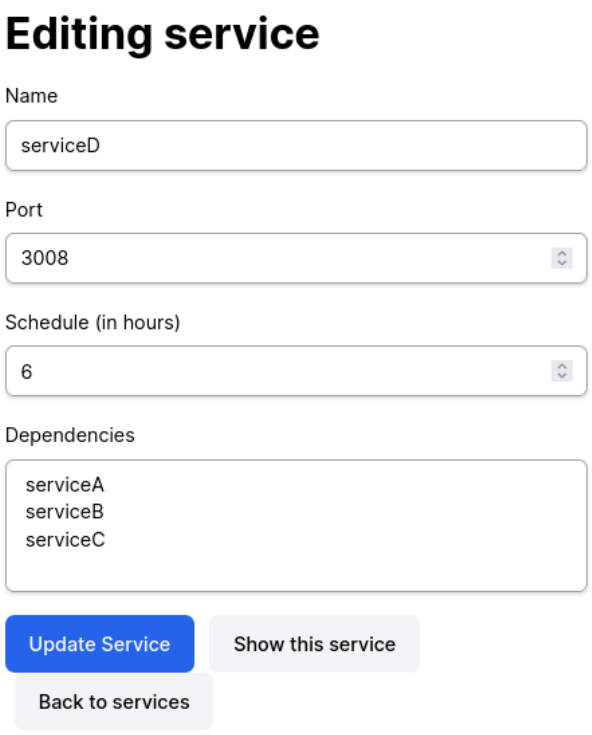


Рисунок 3.12 — Створення/редагування сервісу.

Реалізовані форми для редагування характеристик Circuit Breaker дозволяють адміністраторам зручно та ефективно змінювати ключові параметри, такі як порогові значення помилок, тривалість очікування перед повторним відкриттям або інші критичні показники. Після внесення змін система автоматично ініціює процес передачі оновлених даних до відповідного сервісу для їх застосування. У разі виникнення помилки під час оновлення на стороні сервісу, система оперативно скасовує внесені зміни, гарантуючи збереження попередньої конфігурації, та повідомляє адміністратора про причину збою.

Інтерфейси взаємодії з даними конфігурації шаблону приведенні нижче, а саме форма зміни характеристик (рис. 3.13), приклад відповіді серверу, якому вдалося змінити параметри конфігурації на вказані (рис. 3.14) та обробка помилок при проблемах з оновленням параметрів (рис. 3.15).

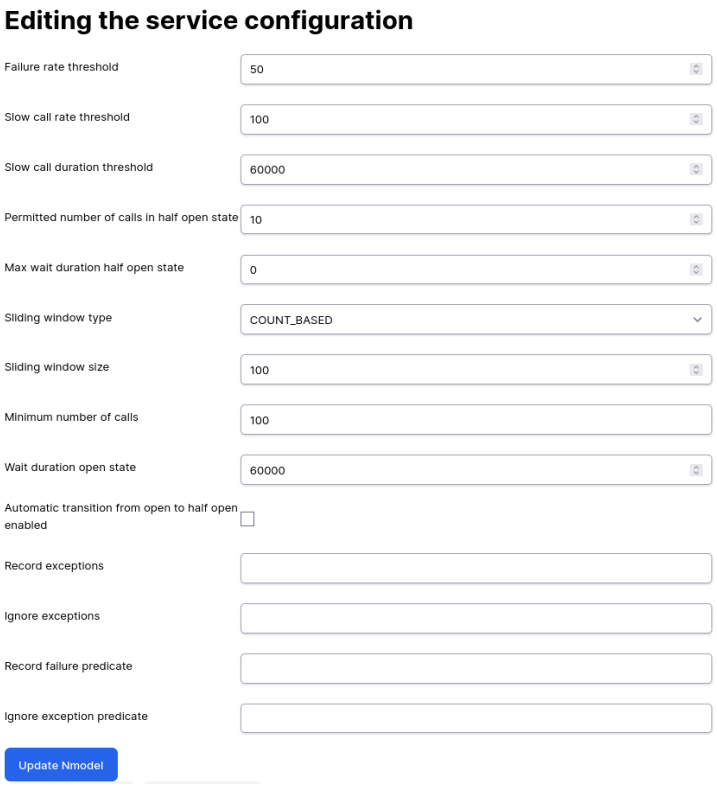


Рисунок 3.13 — Редагування характеристик CB.

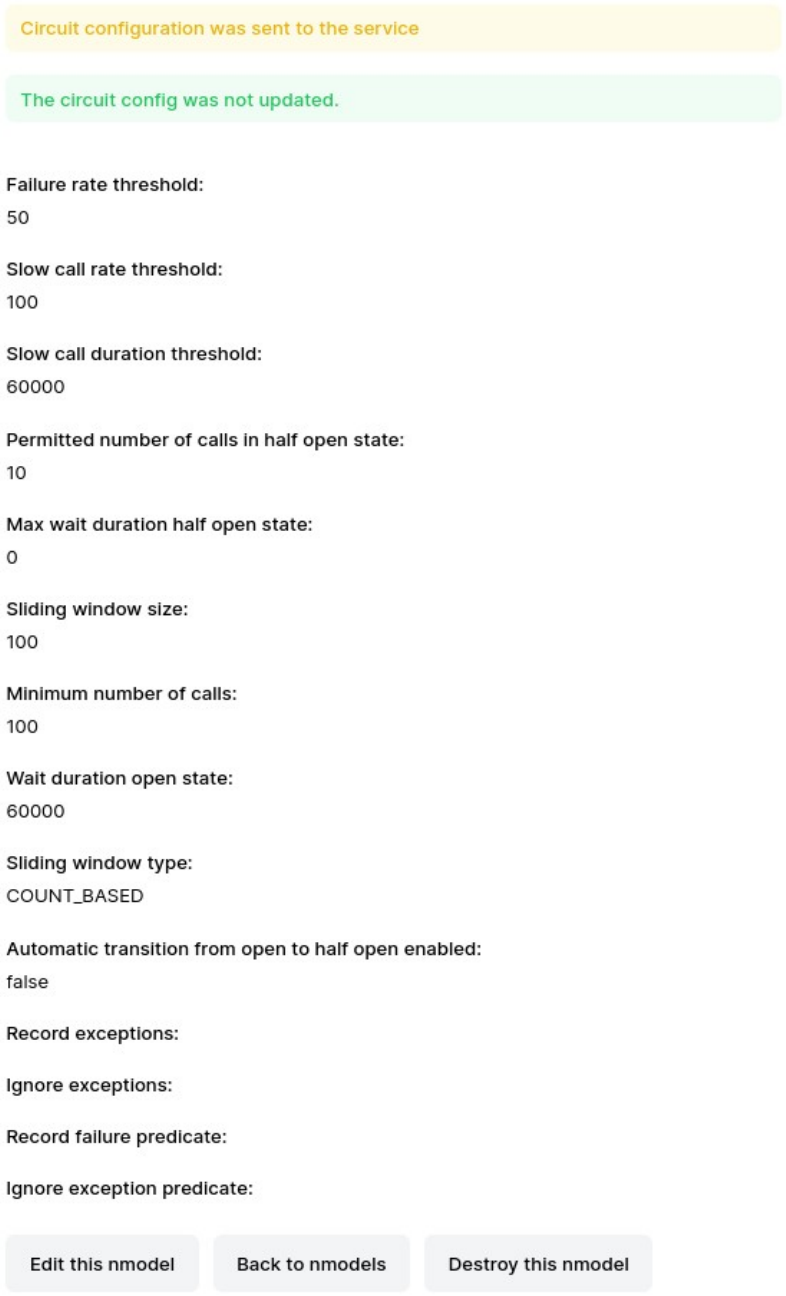


Рисунок 3.14 —Позитивний результат виконання запиту.

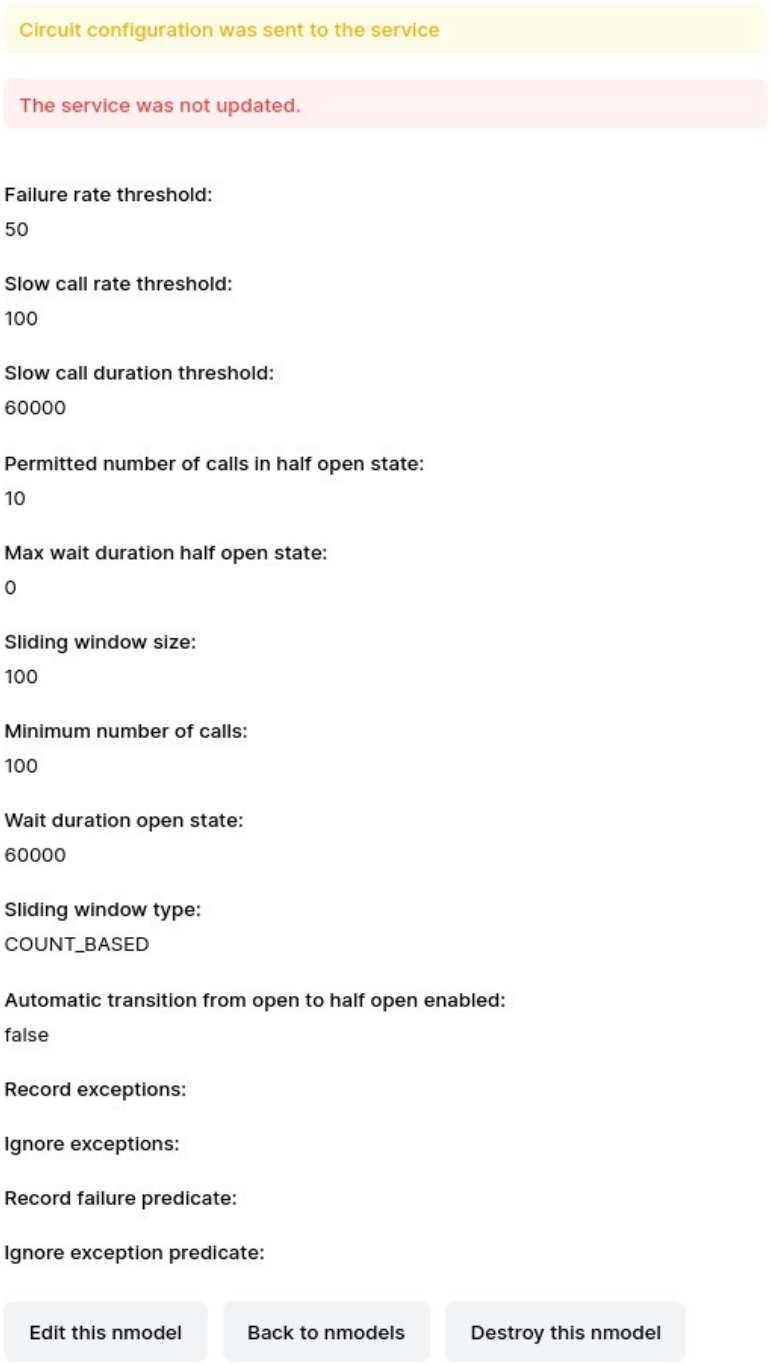


Рисунок 3.15 — Негативний результат виконання запиту.

З безпекової точки зору на серверній стороні було реалізовано функціонал авторизації та аутентифікації користувачів. Це включає в себе механізм перевірки користувачів перед тим, як дозволити доступ до інтерфейсу для зміни

параметрів або перегляду конфігурацій. Процес авторизації забезпечує, що тільки уповноважені користувачі мають доступ до конфіденційних налаштувань системи, а аутентифікація гарантує, що кожен користувач є тим, за кого себе видає.

3.3 Інтеграція додатку до тестового стенду

Для інтеграції Ruby on Rails додатку до тестового стенду можна використовувати автоматично згенерований Dockerfile, створений під час ініціалізації проекту. Цей Dockerfile додається до конфігурації docker-compose.yml, що дозволяє легко налаштувати контейнер для додатку та інтегрувати його з іншими сервісами в системі. Початок Dockerfile де зазначено конфігурацію образу наведено на рис 3.16. Також контейнер був доданий у вищеописану конфігурацію docker-compose.yml, фрагмент приведено на рисунку 3.17.

```
ARG RUBY_VERSION=3.3.0
FROM docker.io/library/ruby:$RUBY_VERSION-slim AS base

# Rails app lives here
WORKDIR /rails

# Install base packages
RUN apt-get update -qq && \
apt-get install --no-install-recommends -y curl libjemalloc2
libvips sqlite3 && \
rm -rf /var/lib/apt/lists /var/cache/apt/archives
...
```

Рисунок 3.16 — Фрагмент Dockerfile.

```
rails-app:
build:
  context: .
  dockerfile: Dockerfile-rails
container_name: rails-app
ports:
  - "3000:3000"
volumes:
  - ".:/app
environment:
  - RAILS_ENV=development
networks:
  - monitoring
```

Рисунок 3.17 — Фрагмент docker-compose.yml.

Розроблений додаток успішно інтегрований в загальну систему, проте для повної функціональності необхідно виконати додаткове налаштування. Це включає в себе додавання інформації про сервіси до бази даних, що забезпечить правильне зберігання даних про кожен сервіс, такі як його унікальний ідентифікатор, ім’я та порт, через який можна здійснювати запити до сервісу. Таке налаштування дозволить не тільки зберігати дані про сервіси, а й забезпечить зручний доступ до них для подальшого аналізу та моніторингу. Важливо також, щоб кожен сервіс був коректно представлений у базі даних, що дозволить інтерфейсу додатку здійснювати динамічне оновлення та моніторинг параметрів конфігурації для кожного сервісу.

Крім того, необхідно встановити залежності між сервісами для правильного управління їх взаємодією. Це дозволить відслідковувати, які сервіси залежать від інших, що дасть змогу оптимізувати обробку запитів і забезпечити ефективну роботу всієї системи. Встановлення залежностей між сервісами також відкриває можливості для подальшої автоматизації управління їх конфігураціями та моніторингом, що підвищує ефективність адміністрування і знижує ризик помилок при взаємодії між різними компонентами системи. Таким чином, налаштування бази даних і встановлення залежностей є

ключовими етапами в інтеграції додатку до тестового стенду, що забезпечує стабільну та ефективну роботу всієї системи.

3.4 Аналіз отриманих даних

Для оцінки ефективності динамічного налаштування параметрів Circuit Breaker було проведено тестування з симуляцією навантаження, як описано раніше.

Спочатку було розгорнуто тестове середовище, включаючи сервіси, налаштування контейнерів і моніторинг через Prometheus. Стенд був готовий до збору метрик з моменту запуску, тожбуло здійснено моніторинг початкових показників системи, включаючи час відповіді, частоту відмов та інші ключові метрики.

На основі отриманих даних було проведено коригування параметрів Circuit Breaker через веб-інтерфейс додатку, що забезпечує динамічну настройку в реальному часі. Ці зміни дозволили адаптувати поведінку системи під змінювані умови навантаження. Після змін у конфігурації параметрів було повторно проведено моніторинг системи, щоб зібрати дані про вплив нових налаштувань на час відповіді, кількість відмов і загальну стабільність.

Далі були порівняні результати до і після зміни параметрів. Основними критеріями порівняння стали час відгуку сервісів, кількість відмов і стабільність роботи при високому навантаженні. На основі отриманих даних були побудовані графіки (рис. 3.18), які наочно демонструють зміни показників системи до і після налаштування параметрів Circuit Breaker.

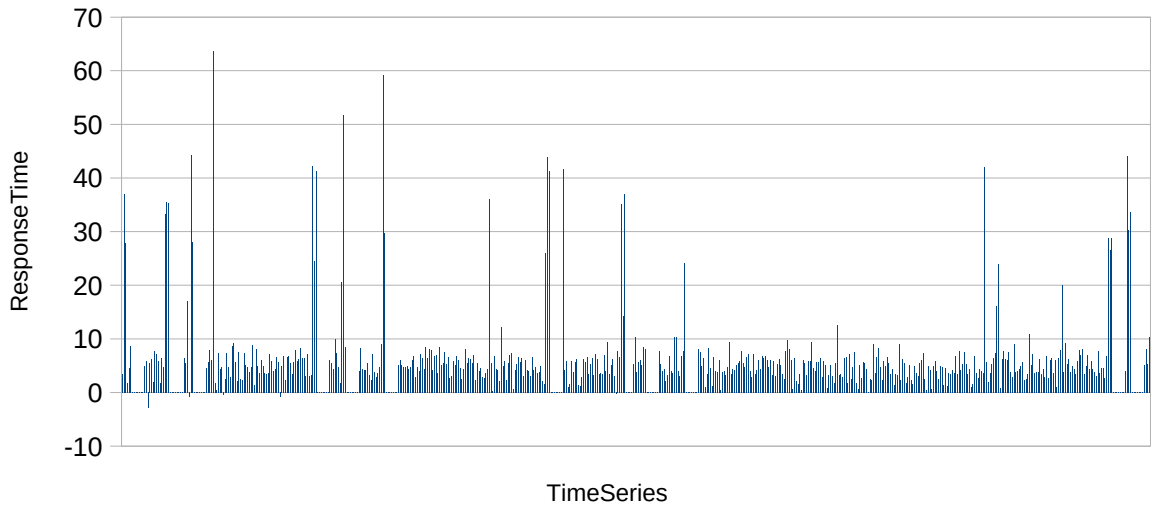


Рисунок 3.18 —Графік зміни навантаження

Вимірювання проводилися протягом десяти хвилин (600 секунд). У процесі замірів, через 300 секунд від початку експерименту, параметри шаблону автоматичного вимикача були динамічно змінені за допомогою розробленого програмного забезпечення. Ця зміна параметрів сприяла оптимізації роботи системи, зокрема, скороченню часу її простою. До внесення змін час простою системи становив 54 секунди, тоді як після адаптації цей показник зменшився до 28 секунд, що майже вдвічі покращило ефективність роботи.

Отже, результати тестування показали значне покращення у роботі системи з динамічними параметрами. Зміна налаштувань дозволила зменшити час відповіді та підвищити стійкість до аномальних ситуацій, таких як підвищене навантаження. Порівняння з результатами тестування із статичними параметрами підтвердило ефективність динамічного налаштування, що забезпечує більшу адаптивність і продуктивність системи.

Висновки до розділу 3

Запропоноване рішення для автоматизації роботи адміністратора та

динамічного налаштування параметрів Circuit Breaker значно покращує ефективність управління сервісами. Завдяки можливості змінювати конфігурацію без необхідності перезавантаження сервісів, час реконфігурації зменшується до мінімуму, що дозволяє швидко адаптувати систему до змінюваних умов навантаження.

Автоматизація процесу налаштування через веб-інтерфейс додатку також спрощує роботу адміністратора, дозволяючи зручно та оперативно вносити необхідні зміни. Це знижує ймовірність помилок, спричинених вручну налаштуванням, і дає змогу швидко реагувати на можливі проблеми. В результаті, запропоноване рішення дозволяє досягти високої стабільності та продуктивності системи, зменшуючи час на реконфігурацію та підвищуючи гнучкість управління.

ВИСНОВОК

У процесі виконання роботи була розроблена та обґрунтована модель динамічного налаштування параметрів стійкості для мікросервісних систем, яка дозволяє автоматизувати управління відмовостійкістю через адаптацію параметрів шаблону Circuit Breaker в умовах змінного навантаження.

Проведений аналіз сучасних підходів до забезпечення стійкості в мікросервісних архітектурах дозволив визначити основні обмеження існуючих рішень, зокрема відсутність динамічної адаптації та необхідність ручного втручання. Запропонована система усуває ці недоліки, забезпечуючи автоматичне налаштування ключових параметрів, таких як порогові значення відмов та час повторного відкриття, на основі поточних показників продуктивності.

Практична реалізація програмного забезпечення та проведення симуляційного моделювання підтвердили ефективність розробленого рішення. Система дозволяє зменшити час простоїв, оптимізувати використання ресурсів, а також підвищити продуктивність і надійність мікросервісної архітектури.

Запропоноване рішення відіграє роль у сучасних DevOps-практиках, сприяючи інтеграції принципів автоматизації, надійності та швидкої адаптації до змін середовища. Завдяки можливості автоматичного налаштування параметрів Circuit Breaker, система легко інтегрується у процеси CI/CD, забезпечуючи постійний контроль і оновлення параметрів конфігурації без необхідності ручного втручання.

Окрім цього, розроблене рішення має значний потенціал для подальшої автоматизації. Зокрема, параметри конфігурації можуть визначатися за допомогою сучасних методів машинного навчання. Використання алгоритмів машинного навчання для аналізу історичних даних і реального навантаження системи може забезпечити ще більш точне і швидке налаштування параметрів, що підвищить ефективність роботи системи навіть у складних і динамічних

умовах.

Таким чином, результати роботи демонструють перспективність запропонованого підходу до динамічного управління стійкістю, що може бути інтегрований у сучасні системи для підвищення їхньої стабільності та адаптивності. Впровадження розроблених механізмів у реальні умови експлуатації, доповнене можливостями машинного навчання, сприятиме покращенню якості сервісів, що особливо важливо в умовах високих вимог до масштабованості та надійності інформаційних систем.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. James Lewis, Martin Fowler. Microservices a definition of this new architectural term. URL: https://web.archive.org/web/20241002115303/https://martinfowler.com/articles/microservices.html (Дата звернення 02.10.2024)
2. Newman, S. "Building Microservices: Designing Fine-Grained Systems". O'Reilly Media, 2021.
3. Guidance. Government Design Principles. URL: https://web.archive.org/web/20241119040419/https://www.gov.uk/guidance/government-design-principles (Дата звернення 19.11.2024)
4. Falahah et al. Circuit Breaker in Microservices: State of the Art and Future Prospects. 2021 IOP Conf. Ser.: Mater. Sci. Eng. 1077 012065
5. Fowler, M. "Microservices: Principles and Practices". Addison-Wesley, 2019.
6. Shore, J. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley, 2010.
7. Liu, Z., & Ma, J. "Building Scalable and Fault-Tolerant Microservices." Springer Handbook of Cloud Computing, Springer, 2011.
8. Nygard, M. T. "Release It!: Design and Deploy Production-Ready Software". Pragmatic Bookshelf, 2018.
9. Spring Boot Authors. "Spring Boot Documentation". URL: https://spring.io/projects/spring-boot (Дата звернення 19.11.2024)
10. Resilience4j Authors. "Resilience4j Documentation". URL: https://resilience4j.readme.io (Дата звернення 19.11.2024)
11. A. Hlybovets & I. Paprotskyi.Increasing the Fault Tolerance in Microservice Architecture. DOI 10.1007/s10559-024-00689-0, Springer Nature Link, 2024.
12. Ruby on Rails Core Team. "Ruby on Rails Guides". URL: https://guides.rubyonrails.org (Дата звернення 19.11.2024)
13. Prometheus Authors. "Prometheus Documentation". URL:

https://prometheus.io (Дата звернення 19.11.2024)

14. Divio. "A Guide to Using Multiple Dockerfiles". URL: https://www.divio.com/blog/guide-using-multiple-dockerfiles/ (Дата звернення 19.11.2024).
15. Docker Inc. "Docker Documentation". URL: https://docs.docker.com (Дата звернення 19.11.2024)
16. Kim, G., Debois, P., Willis, J., Humble, J. "The DevOps Handbook". IT Revolution Press, 2016.
17. Visual Paradigm. "UML Component Diagram Guide". URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-component-diagram (Дата звернення 19.11.2024).
18. Visual Paradigm. "Entity Relationship Diagram (ERD) Guide". URL: https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/ (Дата звернення 19.11.2024).

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) **Магістр**
Галузь знань: 17 – Електроніка, автоматизація та електронні комунікації
Спеціальність: 174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка
Освітня програма «Комп'ютеризовані системи управління та автоматика»

ЗАТВЕРДЖУЮ
в.о. завідувача комп'ютерних
систем та робототехніки
к. ф.-м. н., доц. ХРУСЛОВ М. М.
«12» вересня 2024 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

ШЕВЧЕНКА АНДРІЯ РОМАНОВИЧА
(прізвище, ім'я, по батькові студента)

1. Тема роботи: **«Дослідження шаблонів стійкості до відмов у мікросервісних архітектурах та імплементація динамічного налаштування параметрів на основі ретроспективних даних»**

керівник роботи: **Гамзаєв Рустам Олександрович**, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету ***№ 4101-5/3657 від 12 листопада 2024 року***

2. Строк подання студентом роботи ***30 листопада 2024 року***

3. Перелік питань, які потрібно розробити
1. Проаналізувати, виявити ключові функції та особливості існуючих програмних продуктів для динамічного конфігурування системи.
2. Розробити стратегії прогнозування навантаження на сервіс та визначити критерії успішного прогнозування.
3. Розробити програмне забезпечення, що генерує конфігураційні файли відповідно до навантаження та відповідає критеріям якості.
4. Змоделювати систему, що буде симулювати навантаження на сервіс, та протестувати отримані результати в цих умовах.
5. Оптимізувати та внести необхідні зміни для досягнення більшої точності результатів.
6. Порівняти результати розробленого застосунку, з існуючими рішеннями в рамках змодельованої системи.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Проведення літературного огляду з проблематики роботи	05.09.2024 — 15.09.2024
2	Вивчення сучасного стану проблеми. Дослідження літератури за подібною тематикою.	16.09.2024 — 30.09.2024
3	Розробка інструментарію для автоматизації конфігурування	01.10.2024 — 10.10.2024
5	Аналіз результатів	01.11.2024 — 10.11.2024
6	Розробка рекомендацій щодо застосування моделей прогнозування характеристик динамічних систем у соціально-економічній сфері	11.11.2024 — 18.11.2024
7	Підготовка пояснювальної записки, доповіді та презентації, оформлення документів до захисту, подача кваліфікаційної роботи на наявність запозичень	19.09.2024 — 23.10.2024
8	Представлення кваліфікаційної роботи керівнику та рецензенту	24.11.2024 - 30.11.2024

5. Дата видачі завдання ***05 вересня 2024року.***

Студент А. Р. Шевченко _____
ініціали, прізвище підпис

Керівник роботи Р. О. Гамзаєв _____
ініціали, прізвище підпис

Додаток Б

ІНДИВІДУАЛЬНЕ ТЕХНІЧНЕ ЗАВДАННЯ

Технічне завдання
на розробку програмного виробу
«Дослідження шаблонів стійкості до відмов у мікросервісних архітектурах та імплементація динамічного налаштування параметрів на основі ретроспективних даних»

Назва розділу	Назва і зміст підрозділу
1. Введення	1.1. Назва програмного виробу “Модель динамічного налаштування параметрів шаблонів стійкості у мікросервісних архітектурах “ 1.2. Галузь застосування “17 – Електроніка, автоматизація та електронні комунікації”
2. Підстава для розробки	2.1. Навчальний план за спеціальністю 174 – Автоматизація, комп’ютерно інтегровані технології та робототехніка 2.2. Завдання на кваліфікаційну роботу від 12 листопада 2024 року №4101-5/3657 (Представити як Додаток А до пояснювальної записки до кваліфікаційної роботи)
3. Призначення розробки	3.1. Мета розробки програмного виробу є розробка та теоретичне обґрунтування моделі динамічного налаштування параметрів шаблонів стійкості, яка дозволить підвищити надійність, продуктивність та адаптивність мікросервісних систем в умовах змінного навантаження. 3.2. Призначенням програмного виробу є зменшення часу простою сервісів з шаблоном стійкості Circuit Breaker. 3.3. Вихідні дані для розробки є існуюча практика налаштування параметрів шаблонів стійкості.
4. Технічні вимоги до програмного виробу	4.1 Вимоги до функціональних характеристик є можливість забезпечувати динамічне налаштування параметрів шаблону стійкості Circuit Breaker для сервісу в залежності від прогнозованого навантаження на цей сервіс в мікросервісній системі 4.2 Вимоги до надійності є стабільна робота системи без відмов та помилок.

	4.3 Вимоги до умов експлуатації є розгортання додатку за допомогою Docker в системі мікросервісної архітектури 4.4 Вимоги до складу і параметрів технічних засобів є можливість виконання на будь якому GPU 4.5 Вимоги до інформаційної та програмної сумісності є бібліотека Resilience4j (Java) 4.6 Вимоги до маркування та упаковки немає 4.7 Вимоги до транспортування і зберігання немає 4.8 Спеціальні вимоги немає.
5. Вимоги до програмної документації	1. Справжнє Технічне завдання на розробку програмного вироби (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи). 2. Програму і методику випробувань розробленого програмного вироби (представити у вигляді Додатку В до пояснювальної записки до кваліфікаційної роботи).
6. Техніко-економічні показники	Оцінка економічної ефективності непотрібна.
7. Стадії та етапи розробки	1. Аналіз літератури за темою. 2. Аналіз та порівняння аналогічних існуючих систем. 3. Створення датасету для навчання моделі 4. Розробка та навчання моделі 5. Розробка ПЗ, що обслуговує отриману модель 6. Представлення пояснювальної записки 7. Представлення кваліфікаційної роботи
8. Порядок контролю і приймання	1. Випробування програмного виробу відповідно до Програми і методики випробувань провести на базі комп'ютерного класу. 2. Захист розробленого програмного виробу провести на засіданні атестаційної комісії

Виконавець

студент групи КУ-61

Шевченко А. Р.

Замовник

к. т. н, доц

Гамзаєв Р. О.

Додаток В

ПРОГРАМА І МЕТОДИКА ВИПРОБУВАНЬ ПРОГРАМНОГО ВИРОБУ

1 Об'єкт випробувань

Найменування випробуваного програмного виробу: «Модель динамічного налаштування параметрів шаблонів стійкості у мікросервісних архітектурах»

Область застосування: Комп’ютерні системи

2. Мета випробувань

2.1 Підтвердження відповідності функціональних характеристик розробленої моделі вимогам, зазначеним у Технічному завданні.

3. Загальні положення

3.1 Підстави для проведення випробувань

Підставою для проведення випробувань є наказ про призначення атестаційної комісії.

3.2 Місце і тривалість випробувань

Приймальні випробування проводяться на базі комп'ютерного класу кафедри в період роботи атестаційної комісії.

3.3 Обсяг випробувань

Приймальні випробування програмного вироби проводяться в обсязі відповідному цієї Програми і методики випробувань.

3.4 Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться атестаційною комісією напередодні засідання (або в процесі засідання) за участю Замовника, Виконавця та інших осіб, присутніх на засіданні.

4. Вимоги до програми або програмного виробу

Модель повинна задовольняти наступні вимоги:

7. Інтегруватися в існуючі системи
8. Система повинна надавати інтерфейси взаємодії для користувача.
9. Система повинна працювати без відмов та помилок
10. Вимоги до маркування та упаковки немає.
11. Система повинна транспортуватися в Docker контейнері.
12. Спеціальні вимоги немає.

5. Вимоги до програмної документації

Програмною документацією до виробу вважати:

Справжнє Технічне завдання на розробку програмного вироби

5. (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи).
6. Програму і методику випробувань розробленого програмного вироби (представити у вигляді Додатку В до пояснювальної записки до кваліфікаційної роботи).

6. Засоби та порядок випробувань

6.1 Засоби випробувань

Для виконання програми потрібно запустити тестове середовище, і програмний виріб, сконфігурований відповідно до цього середовища. Програма сумісна з платформами що надають доступ до Nvidia CUDA.

6.2 Порядок проведення випробувань

Як правило, випробування проводяться в два етапи:

6. ознайомчий (1-й етап);
7. випробування програмного виробу (2-й етап).

Перелік перевірок, що проводяться на 1 етапі випробувань, включає в себе:

4. Перевірку комплектності програмної документації.
5. Перевірка комплектності складу програмної документації здійснюється за критерієм наявності зазначеної в ТЗ документації.
6. Перевірку комплектності складу технічних і програмних засобів.
7. Методику проведення перевірок на 1 етапі випробувань.
8. Якість програмної документації перевіряється на відповідність вимогам стандартів ЕСПД.

Перелік перевірок, що проводяться на 2 етапі випробувань, включає в себе:

6. перевірку відповідності технічних характеристик програми вимогам технічного завдання;
7. перевірку ступеня виконання функціональних вимог до програми;
8. методику проведення перевірок, що входять до переліку по 2 етапу випробувань.

3. Порядок проведення випробувань:

3.1 Запуск програми здійснюється при наявності програмного коду викликом команди docker run production.

3.2 Реєстрація, а після цього авторизація у системі.

3.3 Тестування кожної з наявних можливостей. Тестування коректності відображення інформації.

Для проведення випробувань пропонується тест 1, тест 2.

Тест №1: Функціональність облікового запису та налаштувань сервісів

Мета тесту: перевірити можливість входу в обліковий запис, взаємодії з інтерфейсом для перегляду та налаштування серверів і сервісів, а також коректність змін статичних і динамічних параметрів.

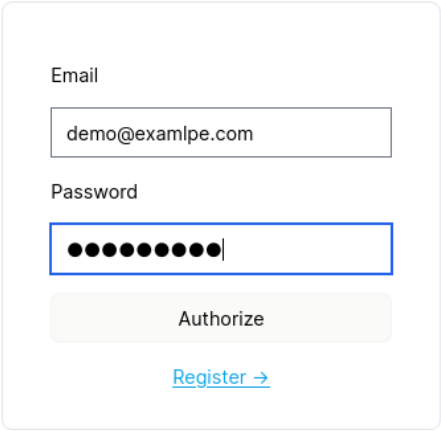
1. Вхід у систему:
    1. Увійти до облікового запису з обліковими даними demo@example.com.
    2. Переконатися, що система завантажує інтерфейс без помилок.
2. Перегляд списку серверів:
    1. Відкрити розділ зі списком серверів.
    2. Переконатися, що кожен сервер відображається коректно із зазначенням основної інформації (ім'я, порт, статус).
3. Зміна параметрів сервісів:
    1. Для кожного сервісу перевірити можливість:
    2. Змінити ім’я.
    3. Змінити порт.
    4. Налаштувати період, у який ініціюється адаптація.
    5. Змінити список сервісів, від яких залежить робота поточного.
    6. Переконатися, що після збереження змін нові параметри відображаються коректно.
4. Робота із залежностями:
    1. Переглянути список сервісів, від яких залежить робота поточного сервісу.
    2. Перевірити можливість змінити статичні параметри конфігурації цих залежних сервісів (ім’я, порт, тощо).

5. Конфігурація спілкування з Prometheus:
    1. Перевірити можливість перегляду та зміни налаштувань інтеграції з Prometheus (адреса, порт, налаштування доступу).
    2. Переконатися, що після змін система успішно підключається до Prometheus.

    Очікуваний результат:

19. Користувач може успішно увійти до системи.
20. Всі сервери та сервіси відображаються коректно.
21. Зміни параметрів сервісів і залежностей виконуються успішно.
22. Динамічні параметри оновлюються, і рефіт моделі ініціюється.
23. Інтеграція з Prometheus працює стабільно після налаштування.

    Тест вважається пройденим, якщо відбуваються вказані операції і їх відображення у програмному продукті



Services

New service

Name:	Port:	Dependent on:
serviceA	3005	serviceB serviceC

Show this service

Name:	Port:	Dependent on:
serviceB	3006	No dependencies

Show this service

Name:	Port:	Dependent on:
serviceC	3007	serviceD

Show this service

Name:	Port:	Dependent on:
serviceD	3008	No dependencies

Show this service

Editing service

Name

serviceD

Port

3008

Schedule (in hours)

6

Dependencies

serviceA
serviceB
serviceC

Update Service

Show this service

Back to services

Тест №2: Додавання Java бібліотеки до сервісу та тестування його функціональності

Мета тесту: перевірити правильність інтеграції розробленої Java бібліотеки з існуючим сервісом, а також оцінити його здатність відповідати на запити та реконфігурувати систему на основі отриманих характеристик.

Кроки тестування:

4. Додавання Java бібліотеки до сервісу:
    1. Включити розроблену Java бібліотеку до існуючого сервісу.
    2. Переконатися, що бібліотека успішно інтегрована та сервер компілюється без помилок.
5. Перевірка відповіді на запит GET /circuitbreaker:
    1. Виконати запит GET /circuitbreaker до сервісу.
    2. Переконатися, що сервіс відповідає на запит статусом "200 OK", та JSON тілом з інформацією про актуальні характеристики.
6. Перевірка отримання та обробки JSON файлу з характеристиками:
    1. Надіслати на сервіс за шляхом POST /circuitbreaker JSON файл з характеристиками для реконфігурації.
    2. Переконатися, що сервіс отримує файл і коректно обробляє його вміст.
7. Перевірити, що після отримання JSON файлу система виконує реконфігурацію відповідно до нових характеристик

Очікуваний результат:

6. Сервіс успішно інтегрує бібліотеку без помилок.
7. Запит GET /circuitbreacker повертає актуальні значення параметрів
8. Запит POST /circuitbreaker отримує JSON файл з характеристиками успішно приймається і коректно обробляється, реконфігурація системи відбувається без помилок.

Висновки: тест 1 успішно пройшов випробування, тест 2 успішно пройшов випробування. Тож , випробування пройшло успішно

Виконавець: студент групи КУ-61, Шевченко А. Р.