Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

ДОСЛІДЖЕННЯ ТА ПОРІВНЯННЯ МЕТОДІВ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ПРОГРАМНИХ ЗАСТОСУНКІВ ШЛЯХОМ МАСШТАБУВАННЯ МІКРОСЕРВІСІВ
(тема)

Виконав:
студент 2 курсу, групи ІНФм-23-2
Авлякулов Т.Е.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Творошенко І.С.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Авлякулову Тимуру Елбарсовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження та порівняння методів підвищення ефективності програмних застосунків шляхом масштабування мікросервісів

затверджена наказом по університету від 25 листопада 2024 року № 1246Ст

2. Термін подання студентом роботи до екзаменаційної комісії 23 грудня 2024 р.

3. Вихідні дані до роботи алгоритми масштабування програмних застосунків, перелік використовуваних програмних засобів: Java Spring, Spring Boot, IntelliJ IDEA, Postman, теоретичні відомості про методи масштабування програмних застосунків: вертикальне масштабування, горизонтальне масштабування, масштабування через бази даних.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз існуючих методів підвищення ефективності програмних застосунків.
2. Особливості вибраних методів підвищення ефективності програмних застосунків шляхом масштабування мікросервісів.
3. Формування методики для підвищення ефективності програмних застосунків шляхом масштабування мікросервісів.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність проблеми підвищення ефективності програмних застосунків, постановка задачі дослідження масштабування мікросервісів, схема архітектури мікросервісного застосунку до і після масштабування, схеми процесів горизонтального та вертикального масштабування, графіки продуктивності програмного застосунку для різних методів масштабування, таблиці та діаграми з результатами порівняння ефективності різних методів масштабування.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	25.11.2024	
2	Аналіз завдання, підбір літератури	25.11.24-27.11.24	
3	Аналіз літератури з досліджуваної проблеми	27.11.24-28.11.24	
4	Аналіз методів масштабування	28.11.24-30.11.24	
5	Розробка методів масштабування	01.12.24-05.12.24	
6	Програмна реалізація	05.12.24-10.12.24	
7	Оформлення пояснювальної записки	10.12.24-15.12.24	
8	Перевірка на плагіат	17.12.2024	
9	Рецензування	20.12.2024	
10	Підготовка презентації та доповіді	25.12.2024	
11	Занесення роботи в електронний архів	02.01.2025	
12	Попередній захист кваліфікаційної роботи	02.01.2025	

Дата видачі завдання 25 листопада 2024 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Творошенко І.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 95 с., 1 табл., 27 рис., 41 джерело.

МАСШТАБУВАННЯ МІКРОСЕРВІСІВ, ГОРИЗОНТАЛЬНЕ МАСШТАБУВАННЯ, ВЕРТИКАЛЬНЕ МАСШТАБУВАННЯ, ЕФЕКТИВНІСТЬ ПРОГРАМНИХ ЗАСТОСУНКІВ, ПОРІВНЯННЯ МЕТОДІВ, ПРОДУКТИВНІСТЬ.

Об’єктом дослідження є методи масштабування мікросервісних архітектур.

Метою дослідження є розробка та порівняння методів підвищення ефективності програмних застосунків шляхом використання різних підходів до масштабування мікросервісів.

Використано методи моделювання, аналізу продуктивності та експериментального дослідження. Проведено дослідження методів масштабування мікросервісів, аналіз їхньої продуктивності в умовах різних навантажень, а також порівняння підходів до забезпечення стійкості та швидкодії систем. Досліджено різні стратегії балансування навантаження та управління ресурсами в мікросервісних архітектурах.

У результаті дослідження розроблено рекомендації щодо вибору методів масштабування для підвищення ефективності програмних застосунків на основі мікросервісів.

SCALE OF MICROSERVICES, HORIZONTAL SCALE, VERTICAL SCALING, EFFECTIVENESS OF SOFTWARE SYSTEMS, INTERVENTION OF METHODS, PRODUCTIVITY.

The object of the research is the methods of scaling microservice architectures.

The aim of the research is to develop and compare methods of improving the efficiency of software applications by using different approaches to scaling of microservices.

Methods of modeling, performance analysis, and experimental research were used. The research examined scaling methods for microservices, analyzed their performance under different load conditions, and compared approaches to ensuring system resilience and speed. Various load balancing and resource management strategies in microservice architectures were explored.

As a result of the research, recommendations were developed for choosing scaling methods to improve the efficiency of software applications based on microservices.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Interface (інтерфейс користувача)

RAM – Random Access Memory (оперативна пам’ять)

CPU – Central Processing Unit (центральний процесор)

SSD – Solid State Drive (твердотільний накопичувач)

HDD – Hard Disk Drive (жорсткий диск)

JDK – Java Development Kit (набір для розробки Java)

JRE – Java Runtime Environment (середовище запуску Java)

JVM – Java Virtual Machine (віртуальна машина Java)

LTS – Long Term Support (довгострокова підтримка)

CI CD – Continuous Integration and Continuous Delivery (безперервна інтеграція та безперервна доставка)

XML – Extensible Markup Language (розширювана мова розмітки)

SQL – Structured Query Language (структурована мова запитів)

CLI – Command Line Interface (інтерфейс командного рядка)

FTP – File Transfer Protocol (протокол передачі файлів)

JDBC – Java Database Connectivity (підключення до бази даних Java)

APM – Application Performance Management (управління продуктивністю програми)

HTTP – Hyper Text Transfer Protocol (протокол передачі тексту)

БД – бази даних

ВСТУП

Для ефективного функціонування сучасних програмних застосунків, особливо в умовах зростаючих вимог до продуктивності, масштабованості та надійності, розробники часто звертаються до мікросервісної архітектури. Мікросервіси – це незалежні компоненти програмного забезпечення, кожен з яких виконує окрему функцію і може працювати автономно. Важливою перевагою такої архітектури є можливість масштабування окремих компонентів замість всього застосунку.

Масштабування мікросервісів дозволяє підвищити ефективність програмних рішень, зменшити затримки та покращити витривалість до збоїв. Існують два основних підходи до масштабування мікросервісів:

– горизонтальне масштабування – додавання нових екземплярів сервісів для рівномірного розподілу навантаження;

– вертикальне масштабування – збільшення ресурсів на окремих серверах (процесори, пам’ять) для обробки більших обсягів даних.

У даній кваліфікаційній роботі досліджується та порівнюється різні методи масштабування мікросервісів, щоб знайти оптимальні рішення для підвищення продуктивності програмних застосунків, а також аналізується їх переваги та обмеження в різних умовах експлуатації.

1 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ПРОГРАМНИХ ЗАСТОСУНКІВ

1.1 Аналіз сучасних методів підвищення програмних застосунків

Аналіз сучасних методів підвищення ефективності програмних застосунків є однією з ключових задач в області розробки програмного забезпечення. В умовах стрімкого розвитку технологій та збільшення обсягу даних, які необхідно обробляти, питання ефективності програм стає все більш актуальним. Це стосується як обчислювальних ресурсів (центрального процесора, оперативної пам’яті, дискового простору), так і часу виконання операцій, що впливає на швидкодію програмних рішень. Постійне зростання складності систем та необхідність обробки дедалі більших обсягів інформації змушують розробників знаходити нові підходи до оптимізації програмного забезпечення. Це включає в себе ефективне використання ресурсів, таких як процесорні потужності, оперативна пам’ять, та дисковий простір, а також мінімізацію часу, необхідного для виконання певних операцій.

З розвитком інтернет технологій і хмарних рішень потреба у високопродуктивних і масштабованих системах зростає з кожним роком. Затримки у виконанні, високе споживання ресурсів та неефективне управління даними можуть призвести до зниження якості користувацького досвіду та збільшення витрат на інфраструктуру. Тому розробка нових методів і технік для оптимізації процесів, покращення продуктивності та зменшення витрат на обробку даних стає необхідністю для багатьох компаній і проєктів.

Крім того, зростаюча популярність мікросервісної архітектури стимулює розвиток нових підходів до побудови програмних систем. Це забезпечує можливість розробникам більш точно налаштовувати і оптимізувати окремі компоненти системи, що дозволяє досягти значного підвищення ефективності та масштабованості загалом.

Зокрема, підвищення ефективності дозволяє не тільки забезпечити стабільну роботу систем під високим навантаженням, але й зменшити витрати на обчислювальні ресурси. Сучасні програми повинні адаптуватися до змінних умов, таких як зростаюча кількість користувачів або обробка великих обсягів інформації в реальному часі. Це вимагає застосування новітніх методів, включаючи оптимізацію алгоритмів, використання хмарних сервісів для горизонтального масштабування та інтеграцію технологій машинного навчання для покращення продуктивності.

Крім того, важливою є можливість автоматичного моніторингу та аналізу продуктивності застосунків у реальному часі. Це дозволяє виявляти вузькі місця в системі та швидко усувати їх, тим самим забезпечуючи стабільність роботи та зниження ризику виникнення помилок або затримок. Сучасні методи профілювання, аналітики та управління ресурсами допомагають розробникам підтримувати баланс між швидкістю роботи, стабільністю та ефективністю використання ресурсів.

Ефективність програмних застосунків є основою їхньої успішної роботи в умовах постійно зростаючих вимог до продуктивності та масштабованості. Сучасні інформаційні системи повинні справлятися з великим навантаженням і забезпечувати високу швидкодію, водночас не збільшуючи витрати на інфраструктуру. Це особливо актуально для таких областей, як онлайн сервіси, де кожна затримка або збій може вплинути на досвід користувача та призвести до втрати прибутку [1].

Інновації у сфері програмування та архітектури систем дозволяють вирішувати ці завдання за допомогою розподілених обчислень, оптимізованих алгоритмів та ефективного використання сучасного апаратного забезпечення. Використання контейнерів та хмарних платформ, таких як Kubernetes, надає можливість автоматичного масштабування залежно від навантаження, що дозволяє системі залишатися гнучкою та адаптивною у мінливих умовах. Основні платформи можливо побачити на рисунку 1.1.

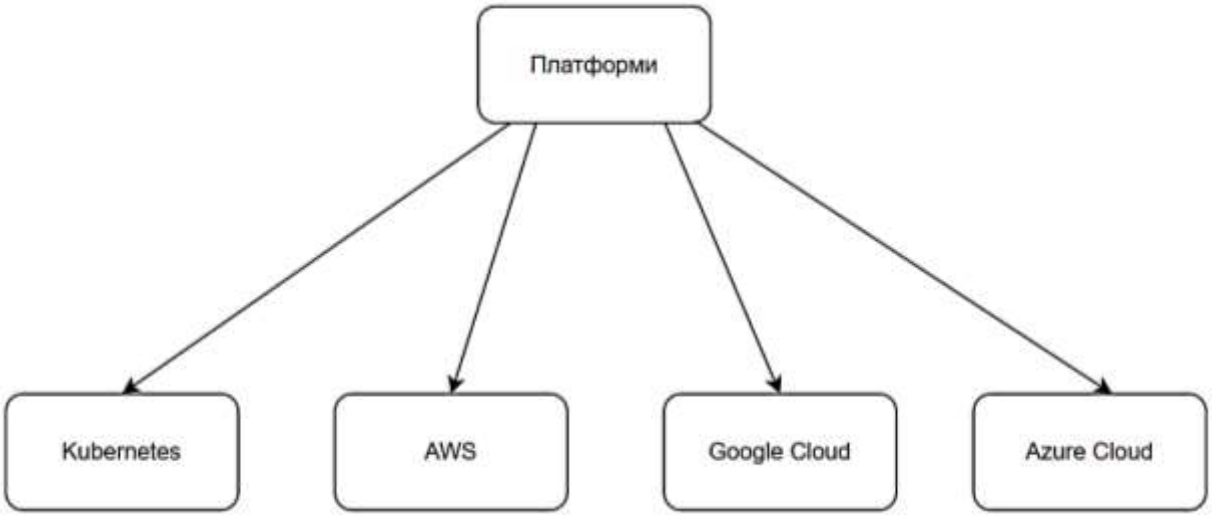

Рисунок 1.1 – Схема платформ масштабування сервісів

Важливою тенденцією є впровадження інтелектуальних систем управління ресурсами, що дозволяють прогнозувати та розподіляти ресурси більш раціонально, знижуючи енергоспоживання та фінансові витрати. Це не тільки підвищує ефективність програмних рішень, але й робить їх більш екологічними.

Таким чином, підвищення ефективності програмних застосунків є багатогранною задачею, яка охоплює як технічні аспекти, так і питання економічної доцільності, стабільності та якості користувацького досвіду. Розробники мають постійно впроваджувати нові технології та підходи, щоб відповідати сучасним вимогам і зберігати конкурентоспроможність на ринку.

Забезпечення ефективності програмних застосунків є важливим завданням на всіх етапах їх життєвого циклу – від проєктування до підтримки та розвитку. Зі зростанням обсягів даних і кількості користувачів, програми повинні бути здатні обробляти великі навантаження без зниження продуктивності. Це особливо актуально в умовах цифрової трансформації, коли кожна секунда простою або затримки може призвести до втрати клієнтів або важливих бізнес можливостей.

Хмарні технології є одним із ключових підходів до підвищення ефективності застосунків, які дозволяють динамічно змінювати

обчислювальні ресурси відповідно до потреб системи. Використання хмарних сервісів, таких як AWS, Google Cloud або Azure, дає можливість автоматично додавати чи зменшувати кількість серверів залежно від поточного навантаження, що дозволяє забезпечити оптимальну роботу без зайвих витрат.

Ще один важливий аспект – це вдосконалення роботи з базами даних. Оптимізація запитів, використання індексів, а також впровадження технік, таких як розподілення даних і копіювання поточних серверів, дозволяє значно зменшити час доступу до даних та підвищити швидкість обробки інформації. У великих системах, де дані постійно оновлюються, цей підхід є одним із ключових факторів для підтримки продуктивності на високому рівні.

Також варто зазначити, що ефективність програмного забезпечення залежить не лише від технічних рішень, але й від правильної організації процесів розробки. Використання методологій, таких як DevOps, дозволяє автоматизувати тестування, розгортання та моніторинг програм, що значно скорочує час на виявлення і виправлення проблем, забезпечуючи безперервну інтеграцію та доставку оновлень.

Таким чином, для досягнення високої ефективності необхідно поєднувати сучасні технічні рішення з гнучкими процесами розробки та управління ресурсами, що дозволить програмним застосункам залишатися продуктивними, надійними та масштабованими в умовах постійного зростання вимог до них.

Поєднання методів масштабування, показаних на рисунку 1.2, дає можливість забезпечити стабільну роботу систем навіть під великим навантаженням, мінімізуючи ризик збоїв та затримок.

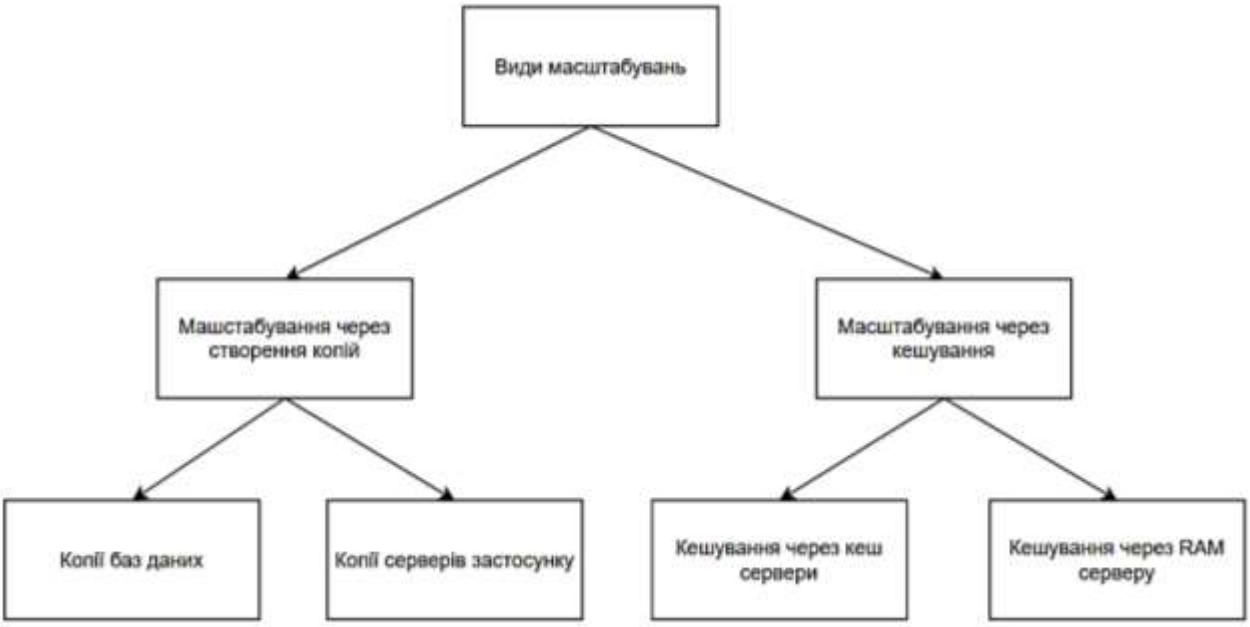

Рисунок 1.2 – Поєднання методів масштабування

Впровадження автоматизованих інструментів моніторингу та управління ресурсами допомагає вчасно виявляти потенційні проблеми і оптимізувати роботу системи. Таким чином, підприємства можуть ефективно адаптувати свої програмні рішення до змінних умов та вимог ринку.

1.1.1 Основні підходи до масштабування мікросервісних застосунків

Основними підходами до підвищення ефективності програмних застосунків є оптимізація коду, масштабування систем та використання кешування. Ці методи дозволяють зменшити час виконання операцій і покращити використання ресурсів.

Масштабування систем. Масштабування програмних застосунків є ще одним важливим підходом. Існують два основні підходи які зображено на рисунку 1.3 [2].

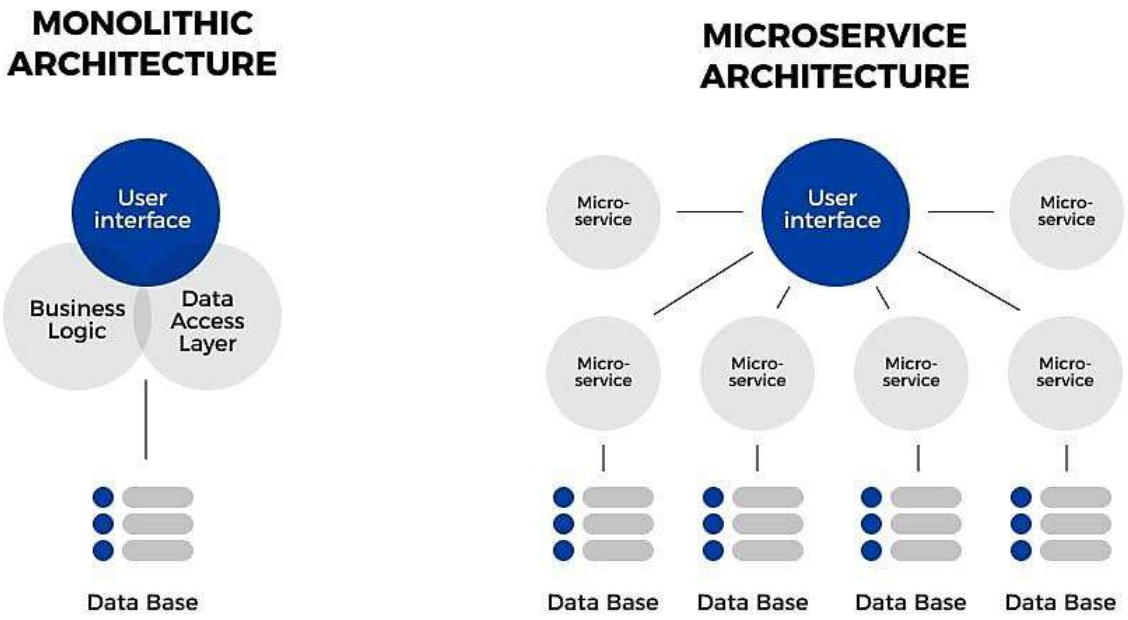

Рисунок 1.3 – Основні види масштабувань

Розглянемо кожен вид масштабування в деталях:

– горизонтальне масштабування – збільшення кількості серверів або екземплярів сервісів для обробки більших обсягів запитів або даних. Горизонтальне масштабування особливо актуальне для хмарних рішень та мікросервісних архітектур, де кожен мікросервіс може бути масштабований незалежно;

– вертикальне масштабування – збільшення потужності існуючих серверів шляхом додавання додаткових ресурсів, таких як процесори або оперативна пам’ять. Це дозволяє обробляти більші обсяги даних без необхідності змінювати архітектуру застосунку.

Кешування. Кешування є потужним інструментом для підвищення ефективності застосунків. Воно дозволяє зберігати результати обчислень або часто запитувані дані у швидкодіючих сховищах, таких як оперативна пам’ять, для повторного використання. Це значно скорочує час доступу до даних та зменшує навантаження на сервери баз даних. Найпоширенішими інструментами для реалізації кешування є Memcached та Redis.

Асинхронне та багатопоточне програмування. Використання асинхронних операцій та багатопоточності дозволяє ефективніше використовувати ресурси процесора. Завдяки цьому застосунки можуть виконувати кілька завдань паралельно, не блокуючи основний потік. Це особливо корисно для обробки великих обсягів даних або роботи з зовнішніми сервісами, де є значні затримки.

Мікросервісна архітектура. Мікросервісна архітектура дозволяє розбити великі, монолітні програми на невеликі, незалежні сервіси, кожен з яких виконує окремі функції. Це дозволяє масштабувати кожен мікросервіс незалежно, а також полегшує підтримку та оновлення окремих компонентів. Кожен мікросервіс можна реалізувати мовою програмування, яка найбільш підходить для виконання конкретних завдань, що підвищує загальну ефективність системи [3, 4].

Використання контейнеризації та керування ними. Технології контейнеризації, такі як Docker, дозволяють запускати програмні застосунки в ізольованих середовищах, що спрощує їх масштабування та розгортання. Інструменти керування, такі як Kubernetes, забезпечують автоматичне управління контейнерами, балансування навантаження та автоматичне відновлення у разі збоїв.

Оптимізація доступу до баз даних. Робота з базами даних є однією з найбільш ресурсномістких операцій для будь якого застосунку. Оптимізація запитів до баз даних, використання індексів, а також застосування технологій, таких як розподілення даних та створення копій серверу, дозволяє значно підвищити швидкодію програмного забезпечення.

1.1.2 Основні підходи до підвищення ефективності програмних застосунків програмним шляхом

Ефективність програмних застосунків є критично важливою у сучасному світі, де від програмних систем очікується висока продуктивність, швидке реагування та стійкість до великих навантажень. Щоб досягти цих вимог, розробники використовують різні методи та підходи, спрямовані на покращення роботи програмного забезпечення. Нижче описані основні з них:

– оптимізація алгоритмів та структур даних. Одним з перших кроків у підвищенні ефективності є вибір найбільш підходящих алгоритмів і структур даних. Наприклад, використання алгоритмів з кращою складністю, таких як бінарні дерева для швидкого пошуку або хеш таблиці для ефективного доступу до даних, може значно знизити час виконання завдань. Важливо також уникати неоптимальних рішень, таких як зайві цикли або дублювання обчислень, що можуть сповільнювати програму. Сучасні підходи, такі як алгоритми машинного навчання та обробка великих даних, також можуть бути корисними для аналізу та обробки складних структур даних;

– оптимізація коду. Один з перших і найбільш очевидних методів підвищення ефективності програмного забезпечення полягає в оптимізації коду. Це включає використання більш ефективних алгоритмів та структур даних, усунення зайвих операцій, зменшення кількості обчислень та повторень. Наприклад, перехід від алгоритмів із складністю $O(n^2)$ до $O(n \log n)$ може значно прискорити роботу програми. Важливим аспектом є також правильне управління пам’яттю та уникнення витоків пам’яті;

– балансування навантаження. Для забезпечення стабільної роботи застосунків з високим навантаженням використовуються системи балансування навантаження. Вони рівномірно розподіляють вхідні запити між кількома серверами або сервера застосунку, щоб уникнути перевантаження окремих компонентів. Це дозволяє підвищити стійкість системи до збоїв і збільшити її пропускну здатність;

– використання хмарних технологій. Хмарні обчислення надають можливості для масштабування ресурсів залежно від потреб застосунку. Наприклад, у разі збільшення навантаження можна динамічно додавати нові ресурси або сервери через провайдерів хмарних послуг (AWS, Google Cloud, Azure). Більш детально ознайомитись можливо на рисунку 1.4 [5].

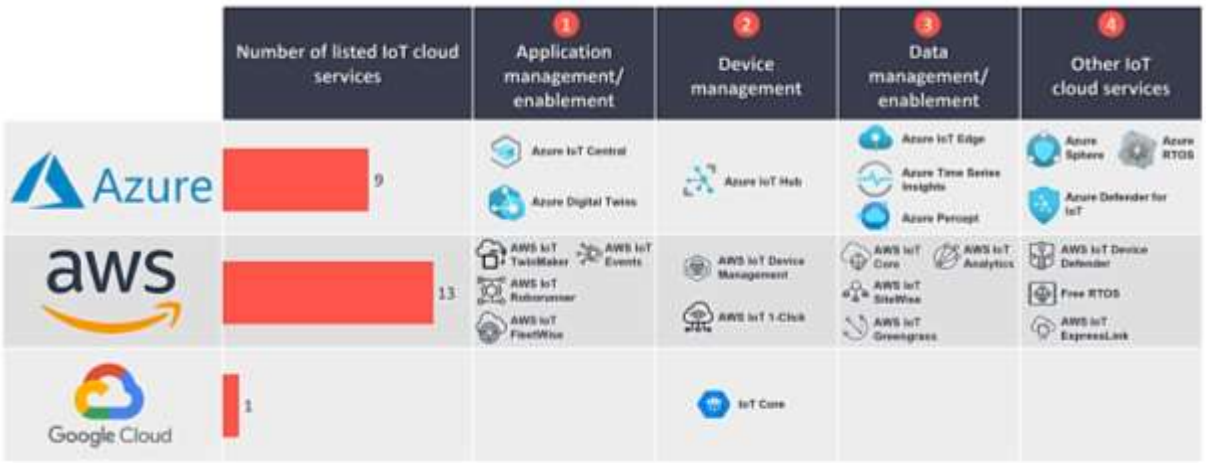

Рисунок 1.4 – Порівняння хмарних технологій

Крім того, хмарні платформи пропонують сервіси, що автоматично керують розподілом навантаження та забезпечують високу доступність застосунків навіть при значних коливаннях у трафіку;

– оптимізація використання оперативної пам’яті та процесорного часу. Часто програми споживають значні ресурси, що може призвести до перевантаження системи. Оптимізація використання пам’яті та процесорних ресурсів дозволяє зменшити витрати та збільшити продуктивність. Це може включати такі підходи, як мінімізація створення тимчасових об’єктів, використання копій об’єктів або збирання сміття. Додатково, техніка «відкладене завантаження» дозволяє завантажувати дані лише тоді, коли вони дійсно потрібні, що допомагає уникнути надмірного споживання пам’яті [6];

– профілювання та інструменти моніторингу. Важливим аспектом є безперервне профілювання і моніторинг програмного забезпечення.

Інструменти моніторингу (наприклад, Prometheus, Grafana) дозволяють відстежувати продуктивність системи в реальному часі та швидко реагувати на проблеми. Профілювання допомагає виявити «вузькі місця» в коді – ті частини програми, які споживають найбільше ресурсів або викликають затримки. Завдяки цим інструментам можна здійснювати точкові оптимізації, не змінюючи систему, принцип їх роботи зображено на рисунку 1.5 [7].

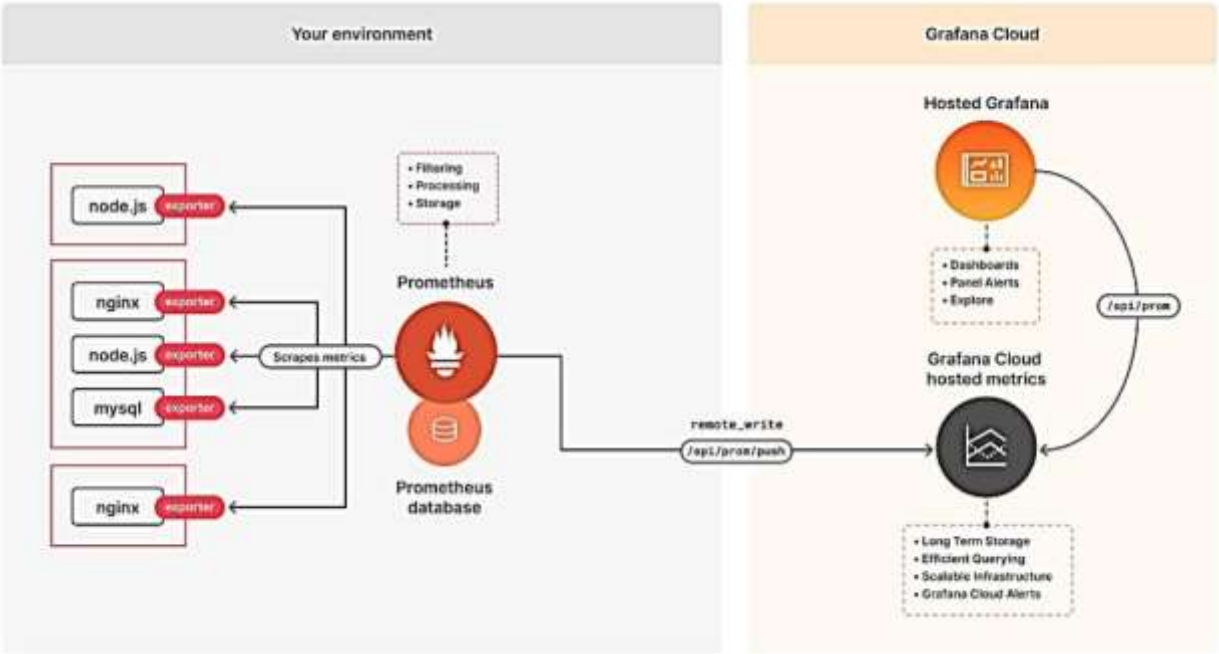

Рисунок 1.5 – Основний принцип роботи моніторингу

Таким чином, підвищення ефективності програмних застосунків є складним, багатофакторним процесом, який вимагає застосування різних підходів, залежно від специфіки програмного забезпечення та вимог до нього. Вибір правильного набору оптимізації може суттєво покращити продуктивність, масштабованість та надійність застосунку, що є особливо важливим у сучасних високонавантажених системах [8–10].

1.1.3 Використання кешування для оптимізації програмних застосунків

Кешування є однією з найважливіших і найбільш поширених технік оптимізації програмних застосунків, що дозволяє підвищити швидкість роботи, зменшити затримки та знизити навантаження на системні ресурси. Суть кешування полягає в тому, що часто використовувані дані тимчасово зберігаються в швидкодоступній пам’яті (кеші) також існують основні стратегії кешування рисунок 1.6 [11], що дозволяють уникнути повторних запитів до повільніших джерел даних, таких як бази даних або зовнішні API.

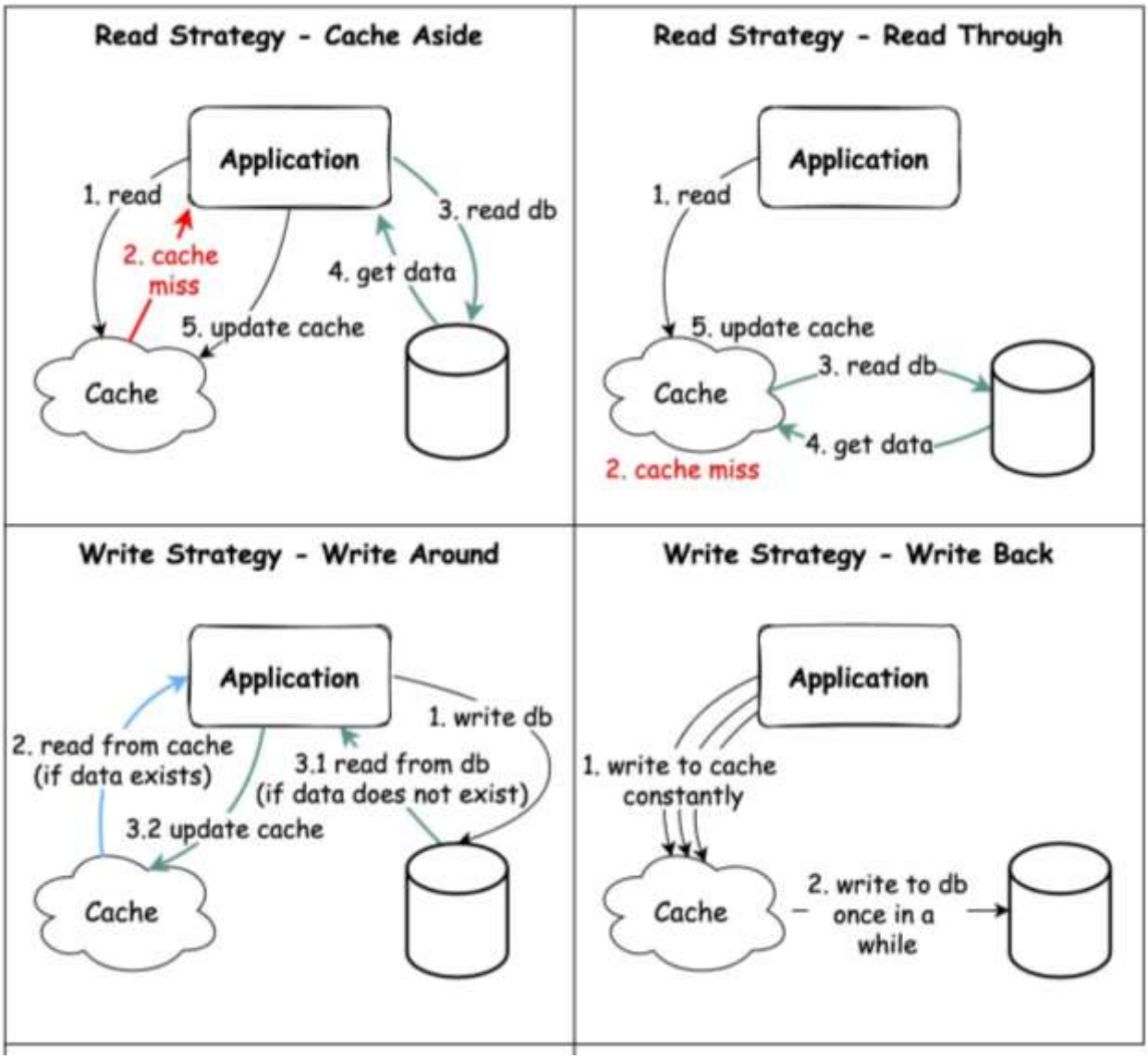

Рисунок 1.6 – Основні стратегії кешування

Основні переваги кешування включають зменшення часу відповіді застосунку, зниження навантаження на центральні компоненти системи, зокрема бази даних, і зменшення витрат на обчислення при повторному використанні результатів раніше виконаних операцій.

Існує кілька основних видів кешування, які використовуються для оптимізації програмних застосунків:

– кешування на рівні клієнта. Браузери можуть зберігати статичні ресурси, такі як зображення, CSS файли та JavaScript, у локальній пам’яті. Це дозволяє уникнути повторних запитів до серверу при кожному відвідуванні сторінки. Клієнтське кешування суттєво покращує продуктивність вебзастосунків, зменшуючи кількість мережевих запитів та час завантаження сторінок;

– кешування на рівні сервера. Серверне кешування передбачає зберігання результатів часто виконуваних запитів у пам’яті сервера. Це може бути реалізовано за допомогою таких технологій, як Memcached або Redis, які дозволяють зберігати ключі та їх значення у швидкодоступній оперативній пам’яті. Такий підхід дозволяє уникнути багаторазових запитів до бази даних або зовнішніх API, тим самим підвищуючи продуктивність застосунку [12];

– кешування результатів запитів до баз даних. При виконанні складних SQL запитів до бази даних, результати можуть бути збережені в кеші, щоб уникнути повторних обчислень під час наступних запитів. Це може бути реалізовано на рівні застосунку або безпосередньо у системі управління базами даних (наприклад, MySQL, PostgreSQL). Особливо ефективним цей підхід є у випадках, коли дані змінюються рідко, але запитуються часто;

– кешування на рівні CDN (Content Delivery Network). Використання мереж розподілу контенту дозволяє зберігати копії статичних ресурсів, таких як зображення, відео та документи, на серверах, розташованих географічно близько до кінцевих користувачів. Це дозволяє зменшити час завантаження ресурсів та забезпечити рівномірний розподіл навантаження на сервери;

– кешування об’єктів. Об’єктне кешування передбачає зберігання певних об’єктів або даних у пам’яті програми. Наприклад, це може бути корисно в об’єктно-орієнтованих системах, де певні об’єкти створюються лише раз і використовуються кілька разів протягом життєвого циклу програми. Це дозволяє уникнути надмірних операцій створення об’єктів або отримання даних.

Перевагами кешування є:

– зниження навантаження на базу даних. Використовуючи кешування, програма може зменшити кількість звернень до бази даних або інших джерел даних, зберігаючи результати попередніх запитів. Це особливо важливо для застосунків, які обслуговують велику кількість користувачів одночасно, оскільки зниження навантаження на базу даних дозволяє підвищити продуктивність та масштабованість системи;

– підвищення швидкості відповіді. Дані, які закешовані можуть бути отримані набагато швидше, ніж якщо їх кожного разу обробляти з нуля або отримувати з бази даних. Це дозволяє прискорити роботу застосунку та покращити користувацький досвід, особливо при повторних запитах до одних і тих самих даних;

– зменшення використання мережевих ресурсів. Використання кешу може знизити кількість мережевих запитів, оскільки локальні копії даних або статичних ресурсів можуть зберігатися на рівні клієнта або на проміжних серверах. Це знижує навантаження на сервери і мережеві канали та покращує масштабованість системи;

– гнучкість і масштабованість. Системи кешування, такі як Redis або Memcached, дозволяють налаштовувати складні стратегії збереження і видалення даних, що дозволяє ефективно управляти кешем навіть у великих розподілених системах. Наприклад, можна налаштувати TTL для кожного запису, щоб автоматично видаляти застарілі дані та оновлювати кеш відповідно до зміни даних.

Недоліками кешування є:

– незважаючи на очевидні переваги, кешування має свої недоліки та обмеження. Одним з ключових аспектів є узгодженість даних. Кеш може містити застарілі дані, особливо якщо база даних або джерело інформації змінилися після кешування. Це може призвести до показу невірної інформації кінцевим користувачам. Тому важливо використовувати правильні стратегії використовування кешу – коли і як оновлювати або видаляти кешовані дані;

– обмеження пам’яті. Кешування потребує додаткової пам’яті для зберігання даних. Тому важливо збалансувати обсяг кешу і необхідність швидкого доступу до даних, враховуючи можливості інфраструктури;

– складність конфігурації кешу. Залежно від потреб застосунку, може знадобитися налаштування різних рівнів і типів кешування, що може бути технічно складним і вимагати точного тестування та моніторингу.

Використання кешування є одним з найефективніших підходів до оптимізації програмних застосунків, що дозволяє досягти значного покращення продуктивності, знизити навантаження на центральні ресурси системи та покращити масштабованість. Вибір правильних стратегій кешування та їх ефективне впровадження можуть суттєво вплинути на загальну продуктивність застосунків і задовольнити вимоги до високошвидкісних систем, що обслуговують велику кількість користувачів.

Таким чином, кешування в мікросервісах відіграє важливу роль у підвищенні продуктивності та зменшенні затримок доступу до даних. Завдяки збереженню часто використовуваної інформації в кеші, мікросервіси здатні швидко обробляти запити, знижуючи навантаження на основні джерела даних і покращуючи загальну масштабованість системи. Правильне використання кешу дозволяє не тільки оптимізувати роботу кожного окремого мікросервісу, а й забезпечити ефективну роботу всієї системи при збільшенні кількості запитів. Це робить кешування ключовим інструментом для створення надійних та продуктивних мікросервісних архітектур.

1.2 Аналіз літературних джерел щодо апробації результатів застосування методів підвищення ефективності програмних застосунків, зокрема, шляхом масштабування мікросервісів

У джерелі [1] проведено детальний аналіз впровадження мікросервісної архітектури у сучасні вебзастосунки. Основна увага дослідження спрямована на виявлення переваг та складнощів, з якими стикаються команди розробників при переході на мікросервіси. Описані як позитивні сторони, такі як підвищення гнучкості та продуктивності систем, так і потенційні виклики, серед яких проблеми інтеграції, безпеки та управління. Крім того, дослідження розкриває питання масштабованості, стійкості до високих навантажень та можливості швидкої адаптації архітектури до змін вимог ринку. Розглядаються конкретні кейси впровадження мікросервісів у реальних умовах, що дозволяє оцінити практичну користь від застосування цієї архітектури.

У джерелі [3] проведено глибоке дослідження методів автоматизації процесу масштабування мікросервісних застосунків, які розгортаються у відкритих контейнеризованих середовищах. Основною метою є розробка методології, яка дозволяє автоматично управляти ресурсами мікросервісів в умовах динамічних навантажень та мінливих умов експлуатації. У статті представлено кілька підходів до моніторингу та регулювання масштабування в реальному часі, що забезпечує високу ефективність роботи системи. Досліджено також технічні аспекти, які стосуються розподілу навантаження, управління контейнерами та оптимізації використання серверних ресурсів для забезпечення стабільної роботи системи.

У джерелі [4] детально розглянуто різні архітектурні підходи до побудови та інтеграції мікросервісів у вебзастосунки. Особлива увага приділена забезпеченню сумісності між різними архітектурними стилями, що використовуються для побудови взаємодіючих мікросервісів. Важливою темою дослідження є інтероперабельність мікросервісів, а також методи

забезпечення їхньої стійкості до навантажень і можливості ефективної взаємодії з іншими системами у контексті складних вебінфраструктур. Також досліджується вплив різних архітектурних рішень на продуктивність і гнучкість вебзастосунків при зміні бізнес вимог.

У джерелі [6] висвітлено питання адаптивного масштабування мікросервісних архітектур, які використовуються для еластичних застосунків. Автори статті аналізують сучасні методи динамічного масштабування платформ, що побудовані на основі мікросервісів, та їхню здатність швидко підлаштовуватися до зростаючих навантажень. Дослідження фокусується на гнучкості мікросервісних платформ та їх ефективності у випадках різкої зміни трафіку. Представлено аналіз продуктивності та стабільності системи під час інтенсивної роботи, зокрема у пікові моменти навантаження.

У джерелі [8] проведено порівняння монолітних та мікросервісних архітектур з точки зору продуктивності, масштабованості та гнучкості. Основна увага приділяється швидкості розробки та впровадження, витратам на підтримку інфраструктури та можливості швидкої адаптації системи до зміни бізнес вимог. Автори також роблять акцент на тому, як кожна з архітектур відповідає певним бізнес потребам, та надають рекомендації щодо вибору оптимальної архітектури залежно від конкретного проєкту. Розглянуто також приклади впровадження цих архітектур у реальних проєктах, що дозволяє оцінити їх практичну цінність..

У джерелі [9] описано розробку автоматизованої системи Microscaler для масштабування мікросервісів, що використовує алгоритми онлайн навчання для адаптивного управління ресурсами. Основна увага приділяється забезпеченню стабільної роботи вебсервісів шляхом динамічного регулювання ресурсів у відповідь на зміну навантажень. Система здатна автоматично аналізувати поточний стан системи та приймати рішення щодо оптимального розподілу ресурсів, що дозволяє мінімізувати витрати та забезпечити стабільну продуктивність під час різких змін трафіку.

У джерелі [10] розглянуто використання мікросервісних архітектур для забезпечення масштабованості, гнучкості та надійності у сфері електронної комерції. Автори дослідження підкреслюють ключові переваги мікросервісів перед традиційними монолітними системами, зокрема в умовах динамічних ринкових змін. Описано підходи до побудови систем, які можуть швидко адаптуватися до нових вимог, забезпечуючи стабільну роботу під час збільшення кількості користувачів і транзакцій. Також розглянуто приклади успішного впровадження мікросервісних платформ у великих компаніях електронної комерції.

У джерелі [12] детально описано особливості проєктування масштабованої мікросервісної архітектури для вебсервісів. Основна увага приділяється питанням розподілу навантаження між компонентами системи та забезпеченню надійності під час збільшення трафіку. Досліджено також методи забезпечення стійкості мікросервісів до помилок та несправностей, що можуть виникати у вебсередовищі.

Автори [13] пропонують кілька підходів до оптимізації роботи системи, що базуються на досвіді впровадження масштабованих архітектур у комерційних проєктах.

У джерелі [14] проведено аналіз різних методів розробки та використання мікросервісів у сучасних програмних рішеннях. Особливу увагу приділено порівнянню різних підходів до розробки мікросервісних систем, включаючи використання DevOps підходу та CI CD (постійної інтеграції та безперервної доставки). Автори досліджують також ефективність використання мікросервісів у різних галузях промисловості та надають рекомендації щодо впровадження цієї архітектури у великих проєктах.

У джерелі [15] розглянуто основні характеристики та особливості мікросервісної архітектури. Дослідження зосереджується на перевагах мікросервісів, таких як гнучкість, можливість масштабування, та незалежне розгортання компонентів. Описано також потенційні недоліки, пов’язані з

ускладненням управління та підтримки такої архітектури, а також питаннями забезпечення безпеки. Автори надають рекомендації щодо вибору архітектурного підходу залежно від розміру та складності проєкту.

Таким чином, проаналізовані джерела [1–15], підтверджують значний потенціал мікросервісної архітектури у сучасних вебзастосунках. Зокрема, основними перевагами є підвищена гнучкість, продуктивність і масштабованість, що забезпечують швидке адаптування систем до змін ринкових вимог і динамічних навантажень. Водночас, у впровадженні мікросервісів виникають складнощі з інтеграцією, безпекою та управлінням. Окрему увагу дослідження приділяють питанням автоматизації процесу масштабування і стабільності роботи при пікових навантаженнях. Завдяки аналізу реальних кейсів та розроблених методик автоматизованого управління ресурсами, підтверджено практичну користь мікросервісної архітектури, особливо для великих проєктів та сфер, що потребують високої надійності і швидкої адаптивності.

1.3 Постановка задачі дослідження

Таким чином, підвищення ефективності програмних застосунків є актуальним завданням для забезпечення їх стабільної роботи та продуктивності в умовах високих навантажень. Тому ставиться завдання розробки та порівняння методів підвищення ефективності програмних застосунків шляхом масштабування мікросервісів.

Об’єктом дослідження є методи масштабування мікросервісних архітектур.

Метою дослідження є розробка та порівняння методів підвищення ефективності програмних застосунків шляхом використання різних підходів до масштабування мікросервісів.

Для досягнення мети необхідно вирішити такі завдання:

– провести аналіз існуючих методів масштабування та оптимізації мікросервісів;

– реалізувати порівняння методів масштабування системи на прикладі мікросервісної архітектури;

– реалізувати тестове середовище для апробації обраних методів масштабування на практичних прикладах;

– підготувати рекомендації щодо використання методів масштабування для оптимізації продуктивності програмних застосунків.

2 ОСОБЛИВОСТІ ВИБРАНИХ МЕТОДІВ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ПРОГРАМНИХ ЗАСТОСУНКІВ ШЛЯХОМ МАСШТАБУВАНЯ МІКРОСЕРВІСІВ

2.1 Аналіз методу вертикального масштабування

Метод вертикального масштабування є підходом до підвищення продуктивності програмного забезпечення, який передбачає збільшення обчислювальних потужностей на одному фізичному сервері. Замість того щоб додавати нові сервери для розподілу навантаження (як у випадку горизонтального масштабування), у вертикальному масштабуванні вдосконалюється апаратна конфігурація наявного обладнання – це можуть бути процесори, пам’ять, жорсткі диски, швидкість мережевих адаптерів тощо.

Основні способи вертикального масштабування показані на рисунку 2.1.

Рисунок 2.1 – Основні способи вертикального масштабування

З таким підходом у певних ситуаціях можна досягти значних покращень продуктивності, особливо для монолітних застосунків, які складно розбити на окремі модулі чи мікросервіси.

Також слід зазначити, що цей підхід підходить для організацій, які прагнуть уникнути складнощів із розподілом навантаження та управлінням великою кількістю серверів. Вертикальне масштабування дозволяє зосередити обчислювальні потужності в межах одного серверного вузла, що спрощує адміністрування і знижує ризики, пов’язані із синхронізацією даних між різними системами. У випадках, коли система не потребує надмірної складності, а навантаження є відносно стабільним, вертикальне масштабування стає ефективним і економічно виправданим рішенням.

Така стратегія особливо корисна для монолітних систем, в яких обробка даних відбувається в одному середовищі, і розподіл функцій на декілька серверів може створити додаткові труднощі. Крім того, вертикальне масштабування підходить для сценаріїв, де є суворі вимоги до продуктивності й затримки – такі, як обробка фінансових транзакцій у реальному часі або управління великими обсягами даних у базах даних. Підвищення потужності на окремому сервері допомагає зменшити затримки і підвищити швидкість обробки, що безпосередньо впливає на якість обслуговування кінцевих користувачів.

Однак, разом із цим слід враховувати, що вертикальне масштабування має фізичні обмеження. Коли сервер досягає своєї максимальної конфігурації, подальше підвищення продуктивності можливе лише через значне оновлення апаратного забезпечення або повну заміну обладнання на нове. Це може призвести до значних витрат, особливо якщо компанія вже інвестувала в потужне серверне обладнання. Більше того, вертикальне масштабування збільшує ризик виникнення єдиної точки відмови, коли збій на сервері призводить до відключення всіх залежних систем. Тому для критичних застосунків, які потребують високої доступності, вертикальне масштабування може бути недостатнім і вимагатиме додаткових заходів для забезпечення надійності, таких як резервне копіювання або застосування відмовостійких кластерів.

Метод вертикального масштабування також має велике значення в умовах, коли стабільність і передбачуваність навантаження на систему дозволяють обійтися без додаткових серверів. Завдяки збільшенню обчислювальних потужностей на одному сервері, можна отримати високий рівень продуктивності, не вдаючись до розподілу навантаження на різні вузли, що особливо важливо для застосунків, які потребують прямого і швидкого доступу до ресурсів без складних мережевих інтерфейсів. Це забезпечує меншу затримку в обробці даних і зменшує потребу в оптимізації міжсерверної взаємодії.

Крім того, вертикальне масштабування часто обирають у випадках, коли важливий контроль над системою, оскільки централізованість ресурсу на одному сервері спрощує моніторинг, контроль і аудит системи. У випадках, де зберігається конфіденційна інформація або працюють високочутливі дані, вертикальне масштабування може мати додаткову перевагу з точки зору безпеки. Менша кількість серверів знижує потенційні точки доступу до даних, полегшує управління правами доступу та контроль за інформаційними потоками.

Проте з часом зростаючі вимоги до обчислювальних ресурсів можуть перевищити потенціал навіть найпотужнішого сервера. У таких випадках компанії часто змушені шукати альтернативні шляхи, наприклад, переходити до мікросервісної архітектури або розподілених баз даних, де навантаження розподіляється між кількома вузлами. Це забезпечує системі більшу гнучкість, можливість зростання і адаптації до динамічних потреб, але потребує значних зусиль з реорганізації архітектури, що часто буває складним завданням для монолітних застосунків.

Отже, метод вертикального масштабування забезпечує простий і ефективний шлях до підвищення продуктивності, проте він обмежений фізичними можливостями і найкраще підходить для систем із стабільними вимогами до навантаження. У середовищах з високою динамічністю, які вимагають масштабованості та швидкої адаптації, вертикальне

масштабування може використовуватися на ранніх етапах розвитку системи, однак у подальшому часто доповнюється горизонтальним масштабуванням, щоб забезпечити стабільність і зростання продуктивності в довгостроковій перспективі.

2.1.1 Основні принципи вертикального масштабування

Основні принципи вертикального масштабування передбачають поступове збільшення обчислювальних потужностей одного сервера, щоб підвищити продуктивність системи без зміни її архітектури. Сюди належить розширення ключових компонентів, таких як процесор, оперативна пам’ять, сховища даних та мережеві інтерфейси. Збільшення кількості ядер та швидкості процесора дозволяє обробляти більше операцій за одиницю часу, що критично для ресурсоємних задач. Розширення оперативної пам’яті знижує частоту звернень до диска, що пришвидшує доступ до даних. Крім того, швидкісні SSD диски замість традиційних HDD допомагають зменшити затримки при зчитуванні й запису великих обсягів інформації (рис. 2.2 [13]).

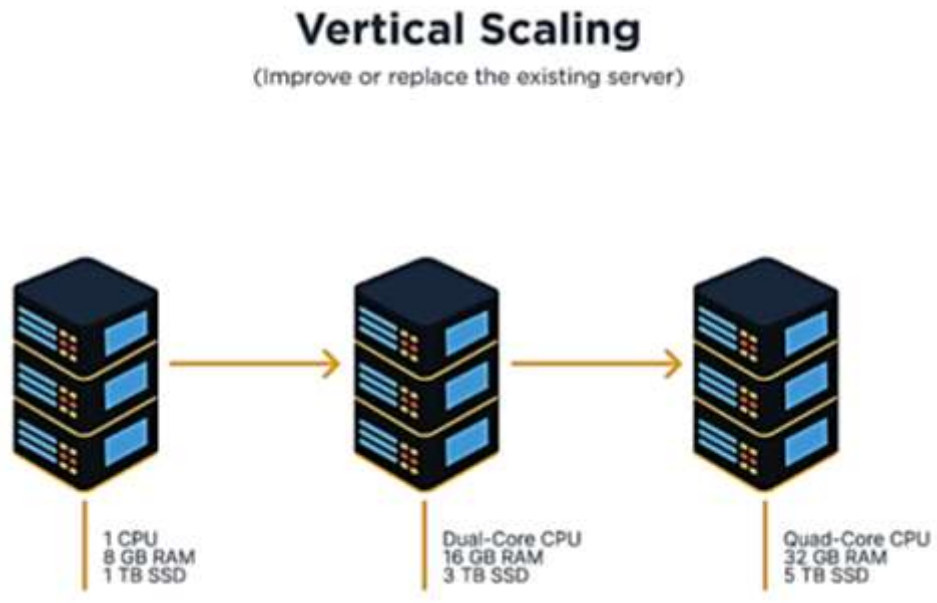

Рисунок 2.2 – Вертикальне масштабування системи

Оптимізація мережевих адаптерів зі збільшенням пропускної здатності сприяє більш швидкому обміну даними між сервісами, що особливо важливо для інтенсивних транзакційних застосунків.

Зазвичай вертикальне масштабування включає:

– збільшення обчислювальної потужності. Використання процесорів із більшою кількістю ядер або з вищою тактовою частотою дозволяє значно підвищити швидкість обробки даних. Потрібну потужність, яку треба встановити, можливо обчислити за таким виразом

$$p_{new} = p_{old} * \left(\frac{F_{new}}{F_{old}}\right), \quad (2.1)$$

де p_{new} та p_{old} – нова та стара продуктивність системи;

F_{new} та F_{old} – нова та стара тактова частота процесора.

Більша кількість ядер забезпечує можливість паралельної обробки завдань, що знижує час обробки складних операцій. Вища тактова частота збільшує кількість операцій, які процесор може виконати за одиницю часу, що позитивно позначається на продуктивності ресурсоємних застосунків;

– розширення оперативної пам’яті: велика кількість оперативної пам’яті (RAM) зменшує потребу в зчитуванні з диска, яке є значно повільнішим. Це дозволяє програмам зберігати більше даних і проміжних результатів обчислень безпосередньо в пам’яті, що значно прискорює доступ до них. Завдяки цьому системи можуть обробляти великі обсяги даних без частих звернень до дискових підсистем, що критично для аналітики даних і роботи з великими обсягами інформації;

– поліпшення сховищ: впровадження сучасних твердотільних накопичувачів (SSD) замість класичних жорстких дисків (HDD) знижує затримки доступу до даних, підвищуючи загальну швидкість завантаження системи і швидкість доступу до даних для інтенсивних застосунків. SSD дозволяють забезпечити високу швидкість зчитування та запису, що істотно

покращує продуктивність систем, де важлива швидкість доступу до інформації, зокрема у випадках обробки баз даних і великих обсягів медіафайлів;

– оптимізація мережевих з’єднань: використання мережевих адаптерів із високою пропускною здатністю та низькою затримкою дозволяє прискорити передачу даних між системами. Це особливо важливо для програм, які потребують частого обміну даними між серверами або обробляють великі обсяги інформації в реальному часі. Висока швидкість передачі даних, зменшені затримки і більш стабільні з’єднання сприяють ефективнішій обробці транзакцій, забезпечують плавне виконання операцій та підвищують швидкість роботи мережевих сервісів.

Таким чином можна зробити висновок, що розширення оперативної пам’яті є ключовим фактором у підвищенні продуктивності системи під час вертикального масштабування. Збільшення обсягу RAM знижує частоту звернень до диска, що значно прискорює доступ до даних, особливо для застосунків, які обробляють великі обсяги інформації. Це дозволяє зберігати більше проміжних результатів і даних безпосередньо в пам’яті, зменшуючи затримки та підвищуючи ефективність. Внаслідок цього системи стають здатними швидше виконувати важкі завдання, покращуючи продуктивність та забезпечуючи стабільнішу роботу застосунків навіть під значним навантаженням.

2.1.2 Переваги вертикального масштабування

Вертикальне масштабування є ефективним підходом для підвищення продуктивності системи, зберігаючи всі компоненти на одному сервері. Цей метод дає змогу оптимізувати обчислювальні ресурси без необхідності розподілу інфраструктури на кілька серверів або створення складних систем синхронізації.

Для компаній, яким не потрібна масштабована розподілена архітектура, вертикальне масштабування стає зручним рішенням, що забезпечує низку важливих переваг:

– спрощування адміністрування. Коли система функціонує на одному сервері, процес адміністрування значно полегшується. Не потрібно додатково обслуговувати численні мережеві інтерфейси або налагоджувати синхронізацію даних між серверами. Всі дані та сервіси зосереджені на одному сервері, що спрощує моніторинг, резервне копіювання та підтримку. Це знижує витрати на технічне обслуговування, оскільки менша кількість системних компонентів означає меншу ймовірність несправностей і збоїв. Такий підхід особливо зручний для підприємств, які прагнуть зосередити свої зусилля на оптимізації внутрішніх процесів замість управління складною IT-інфраструктурою;

– зниження затримок. Завдяки підвищенню обчислювальної потужності на одному сервері зменшуються затримки у передачі даних, оскільки немає потреби передавати інформацію через мережу між кількома серверами. Це стає значною перевагою для програм, які мають високі вимоги до часу відгуку і чутливі до затримок, таких як системи в реальному часі, фінансові застосунки, обробка транзакцій або аналітичні системи. Виконання операцій на одному сервері дозволяє мінімізувати кількість мережевих переходів, що знижує затримки та підвищує загальну продуктивність системи;

– економія ресурсів. Вертикальне масштабування часто обходиться дешевше для компаній, які працюють із завданнями, що не потребують розподілених обчислень або динамічного масштабування. Коли навантаження на сервер є стабільним, збільшення обчислювальних потужностей одного сервера дозволяє уникнути витрат на розгортання додаткових серверів та управління розподіленою інфраструктурою. Це економить не лише на обладнанні, але й на ліцензіях, енергоспоживанні та інших пов’язаних витратах. Крім того, відсутність потреби в мережевих

з’єднаннях між серверами знижує витрати на мережеве обладнання та його обслуговування.

Загалом, вертикальне масштабування забезпечує компаніям можливість нарощувати потужності, зберігаючи централізовану систему та мінімізуючи витрати на адміністрування, обслуговування та ресурси. Це особливо вигідно для застосунків, які потребують стабільної і швидкої обробки даних з мінімальними затримками. Таким чином, вертикальне масштабування стає економічно доцільним та технічно зручним рішенням, що дозволяє зберегти високу продуктивність без складних інфраструктурних змін.

2.1.3 Недоліки вертикального масштабування

Хоча вертикальне масштабування має численні переваги, воно також має певні недоліки, які можуть обмежити ефективність цього підходу для певних типів систем або завдань. Зокрема, існують фізичні обмеження, вартісні аспекти та ризики, пов’язані з відмовами і надмірними навантаженнями. Порівняно з горизонтальним масштабуванням, вертикальне масштабування може не завжди бути оптимальним для вирішення складних чи дуже швидко зростаючих вимог до системи.

Вертикальне масштабування має такі недоліки:

– фізичні обмеження. Кожен сервер має свої максимальні параметри, до яких можна досягти під час модернізації. Це означає, що після досягнення певної межі апаратних можливостей, подальше вдосконалення стає неможливим або надзвичайно складним. Наприклад, процесори, оперативна пам’ять і сховища мають фізичні обмеження за розмірами і характеристиками, які не можна перевищити. Коли система досягає цього порогу, можна лише замінити компоненти на новіші, але це не дає можливості забезпечити додаткові ресурси на рівні масштабування, як це

можна зробити за допомогою додавання нових серверів у горизонтальній архітектурі;

– висока вартість. Підвищення обчислювальної потужності одного сервера часто передбачає використання дорогих апаратних компонентів, таких як високопродуктивні процесори, великі обсяги оперативної пам’яті або швидкі SSD диски. Це може бути набагато дорожче порівняно з додаванням нових серверів до розподіленої системи. Придбання та обслуговування потужних компонентів також потребує значних витрат, що може бути економічно неефективно для середніх і малих компаній, особливо якщо існують можливості для горизонтального масштабування з використанням стандартного обладнання;

– єдина точка відмови. Оскільки система вертикального масштабування працює на одному сервері, вона завжди є потенційно уразливою до апаратних збоїв. У разі виходу з ладу цього сервера, вся система може припинити свою роботу, що призводить до значних простоїв і можливих втрат даних. Це створює додаткові ризики для бізнесу і потребує комплексних стратегій відновлення після збою, резервного копіювання та моніторингу для забезпечення безперебійної роботи. Відсутність резервних копій або потужної інфраструктури може призвести до значних фінансових втрат і збоїв у наданні послуг;

– невідповідність вимогам великих обчислювальних навантажень: Коли навантаження на систему швидко зростає, вертикальне масштабування може не встигати за цим зростанням. Хоча збільшення потужностей одного сервера може задовольняти помірковані вимоги, коли навантаження стає дуже великим, система може не витримати. У таких випадках вертикальне масштабування досягає своїх меж, і для забезпечення гнучкості та стійкості часто доводиться комбінувати його з горизонтальним масштабуванням. Це дозволяє масштабувати обчислювальні ресурси за рахунок додавання нових серверів або вузлів, що забезпечує можливість обробляти великі обсяги даних без обмежень, характерних для одного сервера.

Незважаючи на переваги вертикального масштабування, його недоліки, такі як фізичні обмеження, висока вартість, ризик єдиної точки відмови та обмеження у разі значного зростання навантажень, можуть бути суттєвими для великих і складних систем. Для деяких випадків вертикальне масштабування може бути недостатньо гнучким або економічно доцільним, і тоді доцільно використовувати комбінований підхід з горизонтальним масштабуванням для забезпечення більшої надійності і ефективності.

2.1.4 Вертикальне масштабування в контексті мікросервісів

Мікросервісна архітектура є широко поширеним підходом для розробки масштабованих і гнучких розподілених систем. Однак традиційно вона орієнтована на горизонтальне масштабування, де зростаюче навантаження обробляється шляхом додавання нових серверів або вузлів до системи. Це дозволяє розподіляти обчислювальні ресурси і зменшувати навантаження на окремі елементи. Водночас вертикальне масштабування, яке передбачає збільшення потужностей одного сервера, також залишається актуальним у деяких випадках, навіть у мікросервісних архітектурах. Наприклад, для деяких компонентів системи, таких як бази даних чи ресурсоємні сервіси, вертикальне масштабування може бути більш ефективним. У таких ситуаціях зростання обчислювальних потужностей одного сервера здатне значно підвищити продуктивність, знизити затримки й забезпечити кращу швидкість обробки запитів.

Мікросервіси часто використовують бази даних, які можуть мати високе навантаження або обробляти великий обсяг транзакцій. У таких випадках вертикальне масштабування може бути більш ефективним способом підвищення продуктивності баз даних, оскільки збільшення потужностей одного сервера дає можливість значно покращити час відповіді.

Підвищення обчислювальної потужності, зокрема через збільшення оперативної пам’яті, обчислювальних ядер або швидкості процесора, дозволяє обробляти більші обсяги запитів за менший час. Це важливо для високопродуктивних застосунків, де затримка при виконанні транзакцій може суттєво впливати на загальну ефективність системи. Також, у випадках, коли мікросервісна система використовує транзакційні бази даних, вертикальне масштабування дозволяє зберігати всі дані та транзакції на одному сервері, що робить обробку швидшою та більш узгодженою, зменшуючи ризик появи помилок, пов’язаних із синхронізацією між різними вузлами.

Мікросервіси, які виконують обчислювальні або інтелектуальні завдання, можуть мати високе навантаження на обчислювальні ресурси. Це може бути випадок для мікросервісів, що відповідають за обробку великих даних, аналітику або обробку машинного навчання, де швидкість обробки критична. Вертикальне масштабування дозволяє знизити затримки, оскільки обчислювальні ресурси одразу доступні на одному вузлі, а не на декількох. Це дає можливість зменшити час, необхідний для передачі даних між різними сервісами або вузлами, що особливо важливо для високопродуктивних або реального часу застосунків, де час відгуку є критичним. Збільшення потужностей одного сервера забезпечує більш стабільну і швидку обробку запитів без необхідності передачі даних між різними частинами системи, що знижує загальну затримку.

Використання вертикального масштабування може також бути вигідним у контексті спрощення управління системою. Коли система працює на одному сервері з більшою обчислювальною потужністю, зменшується кількість компонентів для моніторингу і підтримки. Це дозволяє знижувати складність адміністрування порівняно з горизонтальним масштабуванням, яке передбачає управління великою кількістю серверів або вузлів. Також, якщо сервіс має специфічні вимоги до потужностей, наприклад, для обробки великих баз даних чи спеціалізованих обчислень, вертикальне

масштабування може бути економічно вигіднішим у порівнянні з розгортанням окремих серверів для кожного додаткового процесу або компонента. Завдяки цьому вдається знизити витрати на інфраструктуру, зменшити складність інтеграції і зберегти більший контроль над ресурсами.

2.2 Аналіз методу горизонтального масштабування

Метод горизонтального масштабування є популярним і ефективним підходом до збільшення обчислювальних потужностей системи. Це досягається шляхом додавання нових серверів або вузлів до існуючої інфраструктури, що дозволяє значно підвищити здатність системи обробляти більші обсяги даних та запитів. Горизонтальне масштабування передбачає розподіл навантаження між кількома фізичними або віртуальними машинами, що дає можливість розширювати систему по мірі зростання потреб користувачів або вимог до продуктивності. Однією з головних переваг горизонтального масштабування є гнучкість і масштабованість системи.

Оскільки нові сервери можуть бути додані до мережі без значного втручання в архітектуру існуючої системи, це дозволяє досить просто реагувати на зростання навантаження і адаптувати інфраструктуру до змін. Кожен новий сервер або вузол може бути налаштований для виконання конкретних завдань, розподіляючи навантаження за певними критеріями, що дозволяє значно підвищити ефективність обробки запитів і знизити час очікування.

У порівнянні з вертикальним масштабуванням, де збільшення потужностей здійснюється на одному сервері, горизонтальне масштабування пропонує значно більшу стійкість і стійкість до збоїв. Оскільки навантаження розподіляється між кількома серверами, збій одного з них не призводить до втрати всього сервісу.

У випадку неполадки на одному з вузлів, інші сервери можуть продовжити виконувати свою роботу, що дозволяє забезпечити безперервність роботи системи. Це важливо для забезпечення високої доступності і надійності системи, особливо для критичних застосунків або в тих випадках, коли будь які простої можуть призвести до значних втрат.

Горизонтальне масштабування також може бути набагато більш економічно вигідним у порівнянні з вертикальним, оскільки воно дозволяє ефективно використовувати менш потужні сервери або вузли, що можуть бути дешевшими за окремі високопродуктивні машини.

Крім того, через можливість автоматизованого додавання нових серверів в процесі зростання навантаження, можна значно знизити витрати на початкову інфраструктуру, а також здійснювати оновлення без потреби в зупинці всієї системи.

У сучасних розподілених системах, таких як хмарні сервіси та інфраструктура, де масштаби можуть швидко змінюватися, горизонтальне масштабування є критично важливим для забезпечення адаптивності і здатності системи швидко реагувати на зміни в навантаженні.

Горизонтальне масштабування також часто використовують в поєднанні з іншими стратегіями, наприклад, з балансуванням навантаження і контейнеризацією, що дає ще більшу гнучкість в управлінні ресурсами і підвищує ефективність обробки великих обсягів даних.

2.2.1 Основні принципи горизонтального масштабування

Горизонтальне масштабування також часто використовують в поєднанні з іншими стратегіями, наприклад, з балансуванням навантаження і контейнеризацією, що дає ще більшу гнучкість в управлінні ресурсами і підвищує ефективність обробки великих обсягів даних. Горизонтальне масштабування є підходом, що полягає в додаванні нових серверів або вузлів

до існуючої інфраструктури з метою розподілу навантаження і збільшення обчислювальних ресурсів системи. Це дозволяє забезпечити більш високу продуктивність і стійкість системи, оскільки навантаження розподіляється між кількома машинами, а не концентрується на одному сервері.

Ключовими принципами горизонтального масштабування є:

– розподілення навантаження. Однією з основних переваг горизонтального масштабування є здатність розподіляти навантаження між кількома серверами. Це дозволяє значно зменшити навантаження на кожен окремий сервер, що, в свою чергу, підвищує ефективність обробки запитів або транзакцій. Розподілене навантаження дозволяє системі бути більш швидкою та чуйною на зростання обсягів даних або запитів, а також покращує стійкість системи до збоїв або перевантажень на окремих серверах;

– масштабування за рахунок додавання нових вузлів. Горизонтальне масштабування дає змогу системі розширюватися без необхідності модернізації або оновлення існуючих серверів. Для збільшення потужності достатньо додавати нові сервери, що робить систему надзвичайно гнучкою. Важливою перевагою такого підходу є можливість легко адаптувати систему до змінюваних навантажень, що особливо важливо для застосунків, які мають нестабільні або непередбачувані вимоги до ресурсів. У разі зростання навантаження можна просто додавати нові вузли, що дозволяє ефективно справлятися з великими обсягами оброблюваних даних або запитів [14, 15];

– автоматичне балансування навантаження. Важливим елементом горизонтального масштабування є механізми автоматичного балансування навантаження. Вони дозволяють рівномірно розподіляти запити між серверами, забезпечуючи ефективне використання обчислювальних ресурсів і запобігаючи перевантаженню окремих вузлів. Балансувальники навантаження можуть динамічно змінювати розподіл запитів на основі поточного стану серверів, що дає змогу підтримувати високу продуктивність навіть у випадку високих пікових навантажень. Це дозволяє системам з

горизонтальним масштабуванням забезпечити високу доступність і швидкий відгук навіть при великій кількості одночасних користувачів або запитів;

– розширення без простоїв. Однією з важливих особливостей горизонтального масштабування є те, що нові сервери можуть бути додані до інфраструктури без необхідності зупиняти існуючу систему. Це дозволяє масштабувати систему в реальному часі, що важливо для критичних застосунків, де будь яка зупинка або простої можуть призвести до значних втрат. Таким чином, система з горизонтальним масштабуванням здатна підтримувати безперервну роботу без негативного впливу на користувачів або операційні процеси;

– масштабування в хмарі. Горизонтальне масштабування ідеально підходить для хмарних середовищ, де додавання нових серверів може бути автоматизованим процесом. В хмарних платформах ресурси можуть бути швидко виділені або звільнені залежно від потреб бізнесу, що дозволяє оптимізувати витрати і підтримувати високий рівень доступності при мінімальних затратах на інфраструктуру. Крім того, хмарні провайдери часто пропонують вбудовані механізми балансування навантаження і автоматичного масштабування, що спрощує управління системою.

Горизонтальне масштабування є потужним інструментом для побудови масштабованих, надійних і гнучких систем. Його основні переваги включають можливість ефективного розподілу навантаження, гнучкість у додаванні нових вузлів без необхідності змінювати існуючі сервери, а також підтримку високої доступності та продуктивності. Це робить горизонтальне масштабування ідеальним вибором для сучасних застосунків, що потребують швидкої адаптації до змінюваних навантажень, високої стійкості до збоїв та оптимізації використання ресурсів.

2.2.2 Переваги горизонтального масштабування

Горизонтальне масштабування є одним із ключових підходів до збільшення обчислювальних потужностей системи, що передбачає розподіл навантаження між декількома серверами або вузлами.

Цей метод використовується для підтримки стабільної продуктивності, забезпечення високої доступності та надійності системи, особливо в умовах динамічного зростання даних і вимог до обчислювальних ресурсів. Основні переваги горизонтального масштабування, які зображено на рисунку 2.3 [16], також полягають у наступному:

– масштабованість. Горизонтальне масштабування дозволяє легко збільшувати обчислювальні потужності системи шляхом додавання нових серверів або вузлів, не вдаючись до модернізації кожного окремого компонента. Це забезпечує гнучку можливість підтримувати високі навантаження та ефективно працювати з великими обсягами даних, адаптуючи ресурси системи відповідно до поточних потреб. В умовах зростаючих бізнес вимог можливість простого додавання нових вузлів забезпечує постійний розвиток і стабільність системи без необхідності глобальної перебудови інфраструктури;

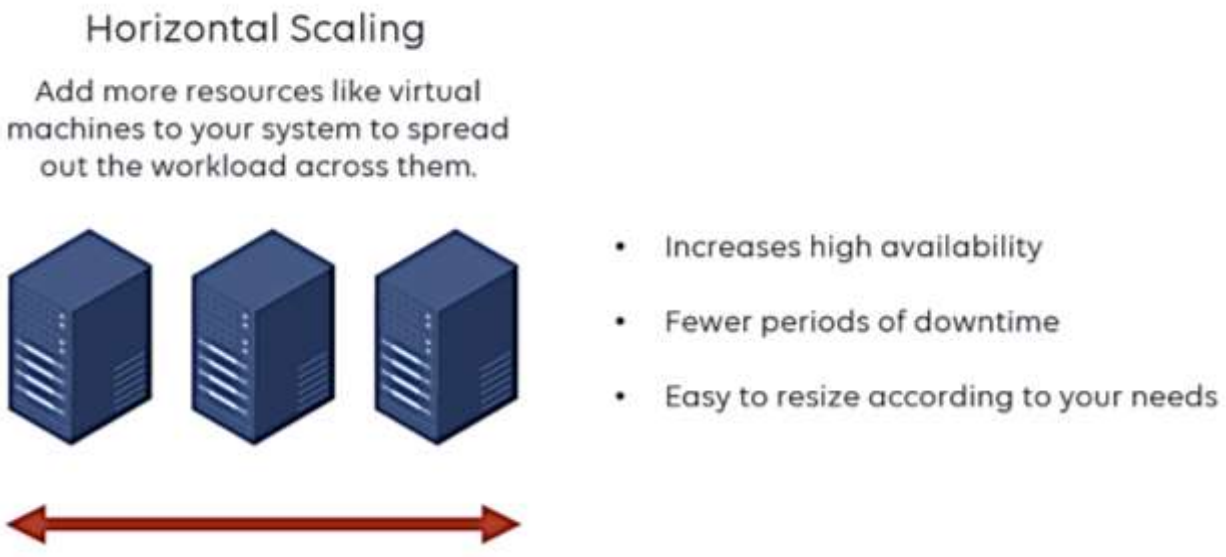

Рисунок 2.3 – Переваги горизонтального масштабування

– висока доступність. Однією з ключових переваг горизонтального масштабування є забезпечення високої доступності системи. У розподіленій архітектурі відмова одного вузла не призводить до зупинки всієї системи, оскільки інші сервери можуть продовжити обслуговувати запити. Це дозволяє уникнути простоїв, зберігати продуктивність системи на високому рівні навіть у випадку апаратних збоїв і забезпечує безперебійну роботу в будь яких обставинах. Такий підхід є особливо важливим для критичних застосунків, де простій або втрати даних можуть мати значні негативні наслідки;

– гнучкість. Горизонтальне масштабування дозволяє гнучко реагувати на зміни в навантаженні. Система може динамічно адаптуватися до пікових навантажень, додаючи нові вузли під час зростання активності користувачів і зменшуючи кількість вузлів, коли навантаження знижується. Це дає можливість оптимально використовувати ресурси та знижувати витрати на підтримку інфраструктури, підлаштовуючи її під поточні потреби бізнесу або рівень трафіку. Таким чином, система стає більш адаптивною, що є значною перевагою для бізнесів, які працюють у швидкозмінних умовах;

– підтримка географічно розподілених систем. Завдяки горизонтальному масштабуванню можна легко створювати географічно розподілені системи, розміщуючи сервери у різних дата центрах по всьому світу. Це забезпечує швидкий доступ до даних незалежно від місцезнаходження користувача, знижує затримки та покращує якість обслуговування для глобальних клієнтів. Така архітектура дозволяє легко розширювати присутність на нових ринках і покращує загальну стійкість системи;

– висока стійкість до збоїв і зменшення ризиків. Розподілена архітектура, яку забезпечує горизонтальне масштабування, робить систему більш стійкою до збоїв. У разі відмови одного або декількох вузлів інші вузли можуть продовжувати обслуговувати запити без втрати

продуктивності, що знижує ризик простоїв. Це забезпечує безперервну роботу системи і дає користувачам впевненість у надійності інфраструктури.

Горизонтальне масштабування є важливим підходом для створення високонавантажених, гнучких і надійних систем, що здатні адаптуватися до швидких змін у навантаженні. Завдяки можливості додавання нових вузлів система може масштабуватися без значних простоїв, забезпечуючи високу доступність, надійність і підтримуючи глобальну присутність. У результаті горизонтальне масштабування стає одним з найважливіших інструментів для побудови сучасних масштабованих інфраструктур, що здатні забезпечувати якісну роботу в умовах зростаючого обсягу даних і високих вимог до продуктивності.

2.2.3 Недоліки вертикального масштабування

Горизонтальне масштабування, хоч і надає значні переваги у підвищенні масштабованості та стійкості системи, має певні обмеження та недоліки, які можуть впливати на ефективність його впровадження та управління. Цей підхід вимагає більшої координації та часто додає складності в організацію роботи системи, особливо коли йдеться про забезпечення консистентності даних і ефективного управління ресурсами.

Розглянемо основні недоліки горизонтального масштабування:

– складність в управлінні. Горизонтальне масштабування значно збільшує кількість серверів та вузлів, що ускладнює управління інфраструктурою. Адміністратори мають координувати роботу декількох серверів, стежити за синхронізацією даних, оновлювати програмне забезпечення та налаштовувати безперебійне резервне копіювання для кожного з вузлів. Такий підхід потребує впровадження централізованих систем моніторингу та автоматизації, що забезпечує стабільну роботу інфраструктури. Часом вартість та складність таких рішень можуть

перевищувати вигоди, які дає горизонтальне масштабування, особливо для невеликих проєктів;

– необхідність в балансуванні навантаження. Ефективне горизонтальне масштабування неможливе без використання системи балансування навантаження. Балансування дозволяє рівномірно розподіляти запити між серверами для запобігання перевантаження одного з вузлів, однак це додає новий рівень складності. Потрібно встановити додаткове обладнання або програмні рішення, що регулюють цей процес, а також налаштувати алгоритми, які враховуватимуть поточне навантаження, доступність ресурсів та швидкість відповіді кожного з серверів. Така система балансування навантаження може стати додатковим джерелом витрат і ускладнити управління інфраструктурою, зокрема у випадках, коли кількість запитів різко зростає або змінюється протягом доби;

– проблеми з консистентністю даних. У розподілених системах, де дані зберігаються на кількох серверах, підтримка консистентності даних може стати серйозною проблемою. Оскільки інформація часто дублюється або синхронізується між вузлами, може виникати ризик розбіжностей у випадках одночасних змін або затримок у реплікації. Ця проблема стає ще більш критичною для застосунків, що працюють у режимі реального часу або мають високі вимоги до точності даних. Підтримка консистентності може вимагати додаткових рішень, таких як протоколи узгодження або технології баз даних, що забезпечують кінцевої консинстенції, але це може уповільнити роботу системи та вплинути на швидкість обробки запитів;

– витрати на обладнання та масштабування інфраструктури. Горизонтальне масштабування може потребувати значних інвестицій у додаткове обладнання, оскільки кожен новий сервер або вузол потребує окремих ресурсів. Такі витрати включають не лише фізичне обладнання, а й додаткове програмне забезпечення, налаштування мережевих підключень та забезпечення безперебійної роботи. Крім того, витрати на обслуговування і

підтримку кожного нового вузла також зростають із збільшенням інфраструктури, що може ускладнювати управління бюджетом проєкту;

– підвищені вимоги до масштабованих застосунків. Горизонтальне масштабування потребує, щоб застосунки були налаштовані для роботи у розподіленій інфраструктурі. Це означає, що розробники повинні передбачати можливість обробки даних на декількох вузлах одночасно, що може потребувати значних змін в архітектурі застосунку. Зокрема, необхідно впроваджувати механізми кешування, розподілу сесій і зберігання даних, а також враховувати, що кожен вузол може працювати незалежно і навіть у випадку часткової відмови інших компонентів.

Попри значні переваги горизонтального масштабування, такі як висока доступність і гнучкість, цей підхід має ряд обмежень, пов’язаних зі складністю управління, необхідністю балансування навантаження, підтримкою консистентності даних та витратами на інфраструктуру. Горизонтальне масштабування підходить для масштабованих систем з високими вимогами до продуктивності та стійкості, однак його впровадження може вимагати значних ресурсів і часу на налаштування та управління.

У зв’язку з цим, вибір між горизонтальним і вертикальним масштабуванням повинен враховувати специфіку проєкту, доступні ресурси та довгострокові цілі.

2.2.4 Горизонтальне масштабування в контексті мікросервісів

У мікросервісній архітектурі горизонтальне масштабування є ключовим підходом для підтримки гнучкості, продуктивності та надійності. Цей метод дозволяє масштабувати окремі сервіси незалежно один від одного, що є особливо важливим для складних застосунків, які обробляють великі обсяги запитів та даних. Кожен мікросервіс працює як окрема одиниця, яка

може бути розгорнута на власному сервері. У разі зростання навантаження на конкретний сервіс достатньо додати нові сервери саме цього сервісу, що дозволяє уникнути впливу на інші елементи системи. Така структура підвищує ефективність, дозволяючи гнучко адаптуватися до змін у потребах застосунка, зокрема під час пікових навантажень.

Горизонтальне масштабування також сприяє високій доступності мікросервісної архітектури, оскільки забезпечує роботу системи навіть у разі відмови окремих вузлів. Кожен сервіс, маючи кілька серверів, стає стійким до помилок, адже навантаження автоматично перенаправляється на інші доступні сервери, забезпечуючи безперервність роботи. Це робить горизонтальне масштабування критично важливим для застосунків із високими вимогами до стабільності, таких як електронна комерція, онлайн банкінг та інші сервіси, що обслуговують великий потік користувачів.

Ще однією перевагою горизонтального масштабування є можливість оптимального використання ресурсів. Оскільки сервіси масштабуються незалежно, можна виділяти ресурси пропорційно потребам кожного з них, що дозволяє мінімізувати витрати на інфраструктуру. Наприклад, сервіси, що потребують великих обчислювальних ресурсів, можуть мати більше сервері, тоді як для менш навантажених сервісів можна залишити менше серверів або, за потреби, тимчасово зменшити їхню кількість.

Попри всі переваги, горизонтальне масштабування в мікросервісній архітектурі вимагає ретельного налаштування. Одним з основних викликів є забезпечення ефективного балансування навантаження, що рівномірно розподіляє запити між серверами кожного сервісу. Для цього зазвичай застосовуються спеціалізовані рішення, такі як балансувальники навантаження або проксі-сервери. Також виникає потреба у синхронізації даних між серверами, щоб забезпечити узгодженість і консистентність даних у системі.

Збільшення кількості серверів окремих мікросервісів також може призвести до підвищених витрат на моніторинг і управління. Для кожного

нового серверу необхідно налаштувати системи моніторингу, які дозволять відстежувати стан та продуктивність. Це підвищує складність адміністративних задач і потребує використання централізованих рішень для управління.

Горизонтальне масштабування є потужним і гнучким методом для забезпечення продуктивності, стабільності та стійкості мікросервісної архітектури. Воно надає можливість ефективного використання ресурсів, підвищує доступність застосунків та дозволяє незалежно масштабувати кожен мікросервіс відповідно до навантаження. Проте його ефективне застосування вимагає ретельного планування, особливо у питаннях балансування навантаження, синхронізації даних і моніторингу інфраструктури.

Незважаючи на технічні виклики, горизонтальне масштабування є ідеальним рішенням для сучасних розподілених систем, надаючи їм потрібну гнучкість і надійність для роботи в умовах високого навантаження.

2.3 Аналіз методу масштабування через бази даних

Масштабування через бази даних є критично важливим підходом у забезпеченні високої продуктивності й надійності програмних систем, особливо у тих, що обробляють значні обсяги даних та підтримують багатокористувацькі середовища. На відміну від вертикального та горизонтального масштабування, які зосереджені на збільшенні апаратних або обчислювальних ресурсів, метод масштабування через бази даних передбачає оптимізацію та розподіл даних для забезпечення швидшого доступу і меншої затримки при обробці запитів. Це дозволяє ефективно обслуговувати велике навантаження на рівні зберігання і доступу до даних.

Основні принципи масштабування через бази даних. Метод масштабування баз даних зазвичай включає такі ключові принципи:

– розподіл даних. Розподілення даних полягає у розділі на логічні частини, які зберігаються на різних серверах баз даних. Це дозволяє паралельну обробку запитів до різних частин бази даних, зменшуючи навантаження на один сервер і прискорюючи доступ до даних;

– реплікація. Реплікація передбачає створення копій бази даних на кількох серверах. Це забезпечує стійкість і доступність системи, оскільки в разі відмови одного сервера інші репліки залишаються доступними, зберігаючи безперервність роботи;

– кешування. Використання кешування даних дозволяє зберігати часто запитувані дані в оперативній пам’яті або швидкодіючих SSD накопичувачах, що значно скорочує час доступу до інформації;

– індексація. Оптимізована індексація таблиць бази даних дозволяє швидше виконувати пошукові та фільтраційні операції. Це особливо корисно для великих баз даних, де без індексів обробка запитів може бути дуже повільною.

Масштабування через бази даних має низку значних переваг:

– висока продуктивність. Розподіл даних і їх паралельна обробка дозволяють швидше обробляти великий обсяг запитів, що критично важливо для систем з високими вимогами до часу відгуку;

– надійність і відмовостійкість. Реплікація даних забезпечує захист від втрат інформації і безперервну роботу в разі відмови одного з вузлів. Це підвищує стійкість системи до помилок і мінімізує ризик простоїв;

– гнучкість. Масштабування бази даних дає можливість налаштовувати архітектуру зберігання даних відповідно до специфіки запитів, що дозволяє адаптувати систему під потреби бізнесу та змінюваний трафік.

Попри свої численні переваги, масштабування через бази даних також має деякі обмеження та недоліки:

– складність управління. Розподілення даних і реплікація потребують додаткових налаштувань і координації, що ускладнює управління базою даних, особливо у великих системах;

– витрати на обчислювальні ресурси. Використання декількох серверів і реплікацій може бути дорогим, оскільки потребує більше ресурсів для підтримки всіх копій і обробки даних;

– консистентність даних. Підтримка консистентності між репліками може бути складним завданням, особливо у випадку, коли дані швидко змінюються. У розподілених системах може виникати проблема затримки в синхронізації даних, що призводить до можливих розбіжностей між репліками.

Масштабування через бази даних є ефективним методом для підтримки продуктивності, надійності та доступності сучасних інформаційних систем. Воно дозволяє зберігати та обробляти значні обсяги даних завдяки розподілу даних, реплікації, кешуванню та індексації, що дає змогу швидко адаптувати систему до змінюваних вимог. Проте, щоб максимально використовувати цей метод, необхідно враховувати складність налаштувань та витрати на управління інфраструктурою. У результаті правильного налаштування масштабування баз даних стає основою стабільної та масштабованої архітектури, особливо для мікросервісних систем із розподіленим зберіганням та обробкою інформації.

2.4 Формування методики для підвищення ефективності програмних застосунків шляхом масштабування мікросервісів.

Сьогоднішній світ інформаційних технологій характеризується інтенсивним ростом обсягу даних, що створюються, зберігаються та обробляються, а також постійно зростаючими вимогами до швидкості, надійності та масштабованості програмних застосунків. У цих умовах

питання ефективного функціонування програмних рішень виходить на перший план, адже саме здатність швидко та надійно обробляти великі обсяги інформації стає запорукою конкурентоспроможності й успіху компаній на ринку. Відповідно, все більше організацій звертаються до сучасних підходів, одним з яких є архітектура мікросервісів.

Архітектура мікросервісів дозволяє розділити складну систему на набір невеликих автономних компонентів, кожен з яких виконує окреме завдання та може працювати незалежно від інших. Така структура забезпечує гнучкість і надійність системи в цілому, полегшуючи її оновлення та модифікацію. Важливим аспектом при роботі з мікросервісами є можливість горизонтального масштабування, що дозволяє адаптувати систему до змінних навантажень шляхом збільшення або зменшення кількості інстанцій окремих сервісів залежно від поточного попиту.

Масштабування мікросервісів є ключовим для підвищення ефективності програмних застосунків. Воно надає можливість оперативно збільшувати потужність системи в моменти пікового навантаження, зберігаючи стабільну продуктивність і високу доступність. Досягти цього можна завдяки створенню методики, яка дозволить динамічно управляти ресурсами, адаптуючи кількість інстанцій мікросервісів під час пікового навантаження і, таким чином, оптимізуючи витрати на інфраструктуру.

Формування такої методики є важливим напрямом дослідження, оскільки ефективне управління мікросервісами сприяє підвищенню швидкості обробки даних і зменшенню часу простою системи. Крім того, така методика може бути адаптована до різних бізнес вимог і технічних умов, надаючи компаніям можливість налаштовувати свої програмні системи відповідно до потреб. У даному розділі кваліфікаційної роботи буде розглянуто основні підходи та стратегії масштабування мікросервісів, їх переваги та обмеження, а також запропоновано ефективну методику для досягнення максимальної продуктивності при оптимальних витратах на інфраструктуру.

На основі проведеного аналізу та розробки методики масштабування мікросервісів було визначено, що дана стратегія є ключовою для підвищення ефективності програмних застосунків у сучасних умовах. Використання горизонтального масштабування дозволяє швидко адаптувати інфраструктуру до змінних умов навантаження, забезпечуючи баланс між продуктивністю, надійністю та витратами. Завдяки масштабуванню мікросервісів системи здатні працювати стабільно навіть у моменти пікових навантажень, зберігаючи високу швидкість обробки запитів і запобігаючи перебоям у роботі.

Запропонована методика демонструє, як шляхом впровадження автоматичного масштабування можна забезпечити оптимальне використання обчислювальних ресурсів, що особливо актуально для великих та розподілених систем. Це сприяє не лише зниженню витрат на підтримку інфраструктури, але й підвищенню стійкості до потенційних відмов, адже система здатна автоматично перенаправляти навантаження у випадку збоїв окремих компонентів.

Висновки даного дослідження можуть бути корисними для організацій, що прагнуть впровадити чи вдосконалити архітектуру мікросервісів у своїх продуктах, забезпечуючи ефективне масштабування та підвищення продуктивності. Використання мікросервісної архітектури в поєднанні з правильною стратегією масштабування забезпечує значну гнучкість і дає змогу адаптуватися до швидко змінюваних бізнес вимог, що є вагомою перевагою для сучасних компаній.

2.5 Моделювання структури програмного застосунку для підвищення ефективності програмних застосунків шляхом масштабування мікросервісів

Застосунок буде складатися з такої структури, яка ідеально підходить для мікросервісної архітектури та легко розширюється.

Шар доступу до застосунку. API Gateway виступає єдиною точкою входу для всіх зовнішніх запитів. Він отримує запити від користувачів або зовнішніх систем і маршрутизує їх до відповідних мікросервісів.

Застосунок буде мати таки функції: аутентифікація та авторизація, маршрутизація запитів, обробка помилок, обмеження частоти запитів, кешування. Ці переваги дозволяють зменшити навантаження на мікросервіси і покращує безпеку та масштабованість системи.

Шар мікросервісу. Основний шар, що містить окремі мікросервіси, кожен з яких відповідає за певний функціонал. Кожен мікросервіс незалежно виконує бізнес логіку та має свою базу даних.

Основний шар мікросервісів буде складатись з таких мікросервісів:

– сервіс користувачів (керує профілями користувачів і забезпечує доступ до їхніх даних);

– сервіс банківських рахунків (відповідає за управління рахунками користувачів та перекази з одного рахунку на інший);

– сервіс переказів (обробляє перекази як транзакції, веде їх облік, та має доступ до них);

– сервіс нотифікацій (посилає повідомлення користувачу, коли був виконаний якийсь переказ).

Переваги: сервіси можуть масштабуватися незалежно один від одного, що знижує ризики перевантаження.

Шар управління даними (бази даних). Кожен мікросервіс має окрему базу даних або розподілену базу, що дозволяє уникати конкуренції за ресурси між мікросервісами.

Даний застосунок має таку структуру як показано на рисунку 2.4.

Буде використано реляційну базу даних – для управління даними, що вимагають транзакційної цілісності.

Переваги: швидкий доступ до даних і гнучкість управління, що дозволяє легко масштабувати систему.

Запропонована модель структури забезпечує гнучкість, незалежність і високу доступність кожного мікросервісу, що значно підвищує ефективність і масштабованість програмного застосунку.

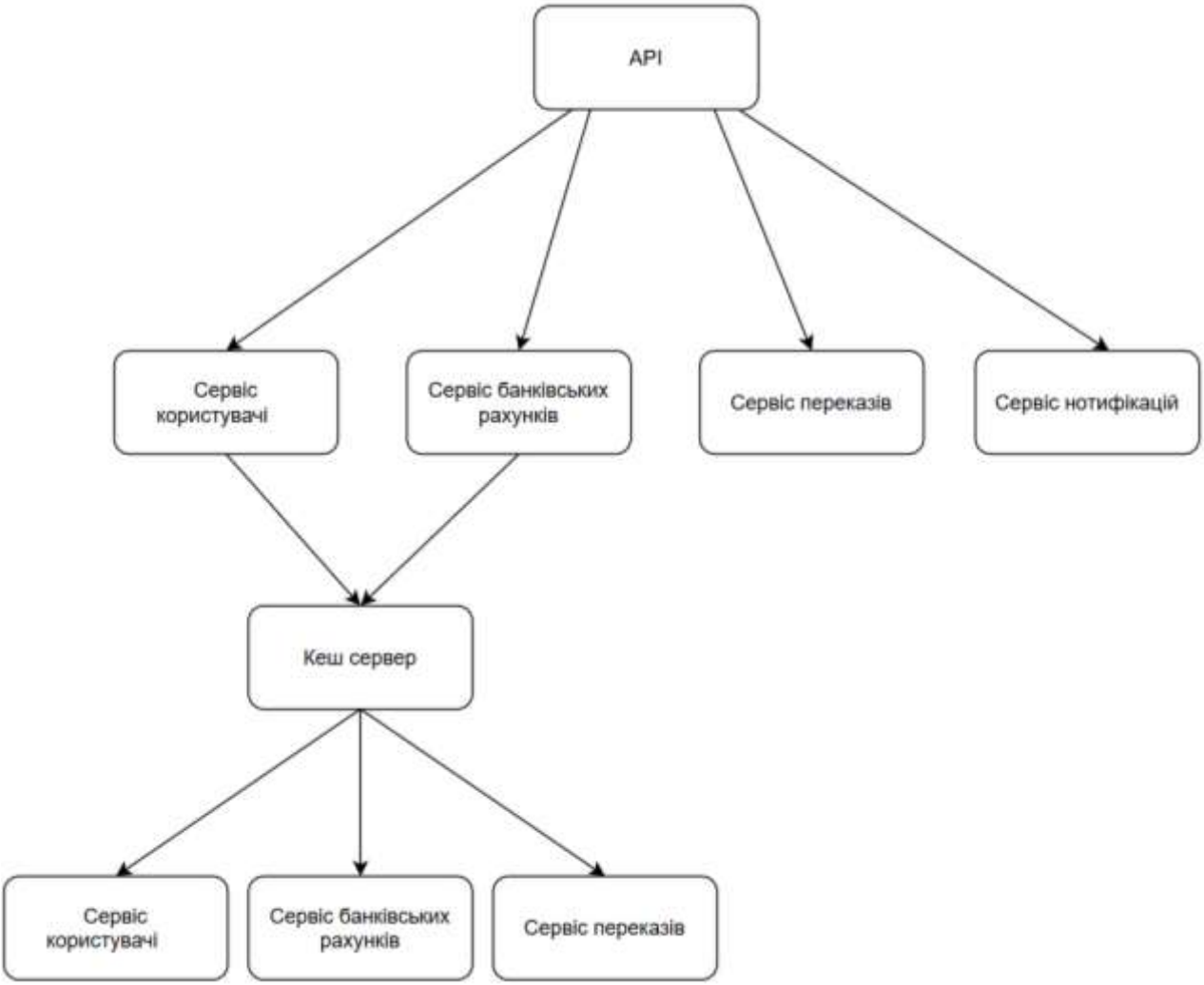

Рисунок 2.4 – Схема застосунку

3 ДОСЛІДЖЕННЯ ТА ПОРІВНЯННЯ МЕТОДІВ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ПРОГРАМНИХ ЗАСТОСУНКІВ ШЛЯХОМ МАСШТАБУВАННЯ МІКРОСЕРВІСІВ

3.1 Вибір інструментальних засобів для реалізації вибраних методів

Для реалізації даних методів, пов’язаних із масштабуванням мікросервісної архітектури, особливу увагу треба приділити вибору технологій, які відповідають сучасним вимогам до продуктивності, масштабованості та зручності інтеграції. Важливо, щоб обрані інструменти не тільки забезпечували швидкість розробки, але й дозволяли ефективно реалізовувати різні стратегії масштабування, зокрема вертикальне та горизонтальне масштабування, а також масштабування через бази даних.

Першочергово треба обрати середовище розробки де буде створено застосунок, найпопулярнішими є Eclipse та Intellij Idea. Вони мають свої переваги та недоліки.

Розглянемо Intellij Idea.

IntelliJ IDEA – це один із найкращих і найпопулярніших інструментів для розробки програмного забезпечення, який було обрано для роботи. Він був створений компанією JetBrains, відомою своїм фокусом на покращенні продуктивності розробників через інноваційні програмні рішення. IntelliJ IDEA вражає своїм функціоналом, зручним інтерфейсом і можливістю працювати з широким спектром технологій, що робить її ідеальним вибором для роботи над сучасними Java застосунками.

IntelliJ IDEA надає дуже багато інструментів, які роблять розробку Java застосунків максимально ефективною. Завдяки розширеному автозавершенню коду розробники можуть писати програми швидше, не витрачаючи час на запам’ятовування складних синтаксичних конструкцій. Потужний аналізатор коду допомагає знаходити потенційні помилки ще на етапі написання, до запуску компіляції, що зменшує кількість дефектів у

застосунку. Інструмент також пропонує рефакторинг на високому рівні – автоматичну зміну структури коду без втрати функціональності, що є критично важливим для підтримки великих проєктів.

Однією з найважливіших функцій IntelliJ IDEA є її інтеграція з екосистемою Java (рис. 3.1).

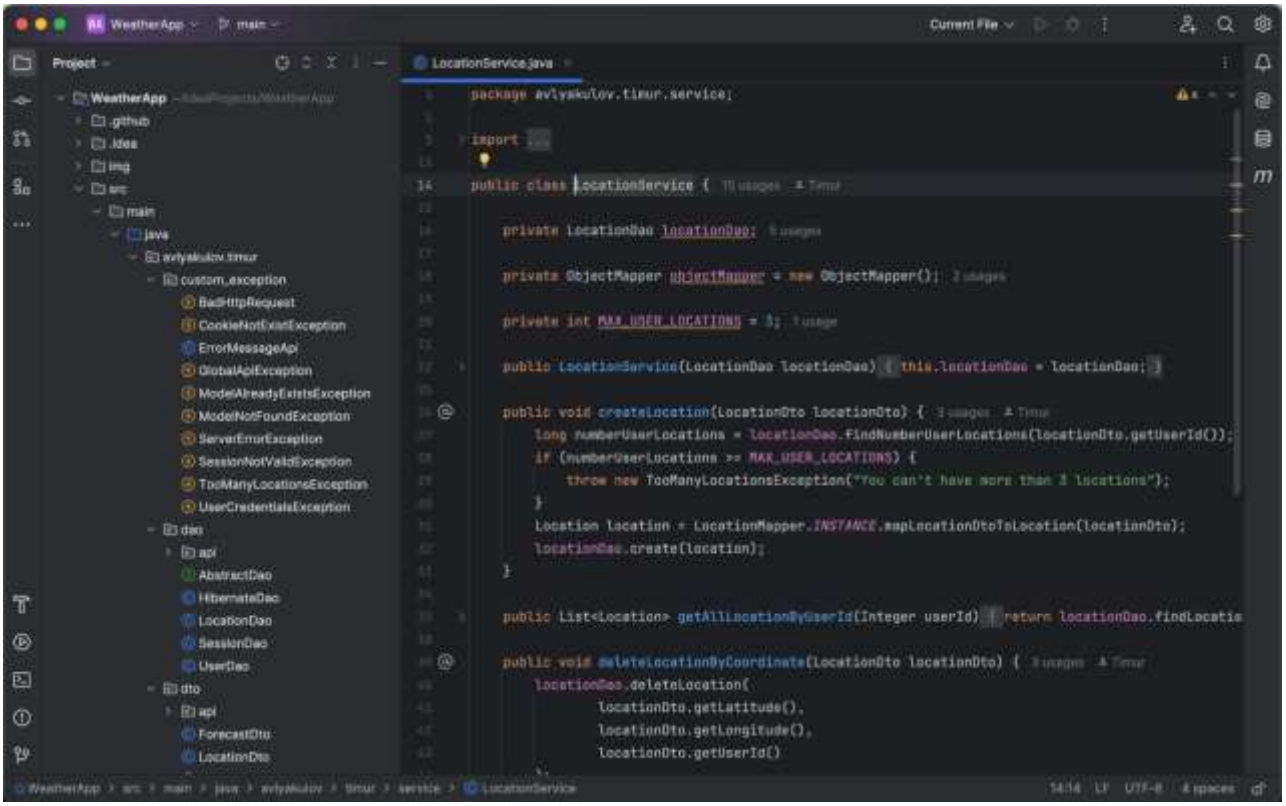

Рисунок 3.1 – Інтерфейс Intellij Idea

Вона має підтримку популярних інструментів і фреймворків, таких як Spring, Hibernate, Maven і Gradle, що дозволяє швидко налаштовувати проєкти. За допомогою IntelliJ IDEA можна легко працювати з базами даних, створювати Docker контейнери, інтегруватися із системами CI/CD і навіть запускати тестування всередині IDE. Це робить її універсальним інструментом, здатним охопити всі етапи життєвого циклу програмного забезпечення.

Ще однією перевагою IntelliJ IDEA є її мультиплатформенність. Вона однаково добре працює на Windows, macOS і Linux, що робить її доступною

для всіх розробників незалежно від операційної системи. Інструмент пропонує величезний набір гарячих клавіш, які дозволяють автоматизувати рутинні дії, та інтуїтивний інтерфейс, що дозволяє швидко освоїти середовище. IntelliJ IDEA також підтримує роботу з великою кількістю мов програмування, таких як Kotlin, Scala, Python, JavaScript, що робить її ще більш універсальною.

Було обрано IntelliJ IDEA для створення даного застосунку, оскільки цей застосунок забезпечує ідеальне поєднання потужності, зручності та універсальності. Цей інструмент дозволяє зосередитися на вирішенні складних бізнес завдань, мінімізуючи час на налаштування середовища чи пошук помилок у коді, завдяки своїй інтеграції з популярними інструментами та технологіями.

Крім того, IntelliJ IDEA пропонує чудову підтримку сучасних фреймворків, таких як Spring, Hibernate, і Maven, що забезпечує легке налаштування проєктів. Можливість інтеграції з Docker і Kubernetes відкриває перспективи для роботи з контейнеризацією та розподіленими системами. Завдяки цьому інструменту розробник отримує єдине середовище для виконання всіх етапів життєвого циклу програмного забезпечення – від написання коду до тестування та розгортання, що робить IntelliJ IDEA оптимальним вибором для розробки якісних і масштабованих застосунків.

Основою для розробки є стек технологій на базі Java, яка зараз є одну з найпопулярніших мов. До складу Java входять ключові компоненти JDK, JRE, JVM, які забезпечують повний цикл розробки, виконання та оптимізації Java програм (рис. 3.2 [17]). Буде використано останню LTS версію Java, а саме 21. Вона буде гарантувати правильне виконання функцій та довгу підтримку.

Також треба обрати веб фреймворк для того щоб була можливість отримувати та відправляти http запити по мережі. Одним із популярних фреймворків є Spring, він ідеально реалізовує цю задачу.

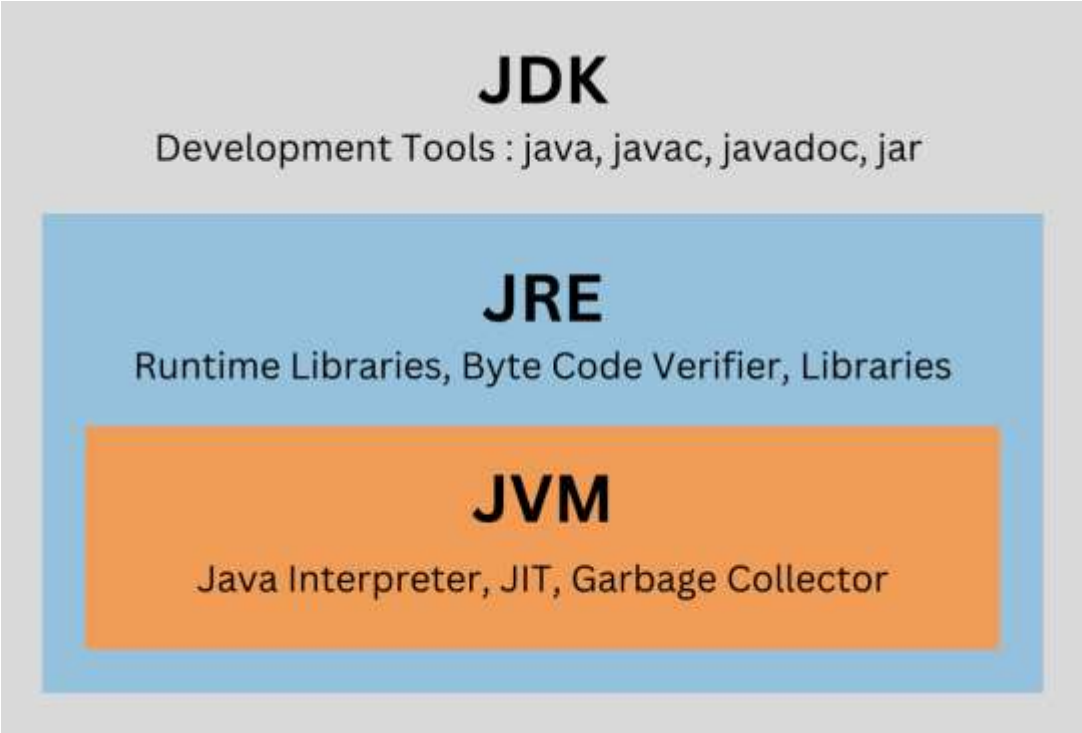

Рисунок 3.2 – Склад JDK

Spring є потужним фреймворком, який пропонує розробникам Java величезну екосистему для створення застосунків різної складності. Базові принципи, інверсія управління і впровадження залежностей, дозволяють зменшити зв’язність між компонентами застосунку. Це спрощує тестування, оновлення та масштабування програм, а також зменшує час на розробку завдяки можливості зосередитися на бізнес логіці замість технічних деталей.

Однією з найбільших переваг Spring є його модульна структура. Наприклад, Spring Core забезпечує основу для впровадження залежностей, Spring MVC полегшує створення застосунків за допомогою шаблону model veiew controller, а Spring Boot дозволяє створювати готові застосунки за лічені хвилини завдяки вбудованій конфігурації системи. Це означає, що розробник може обрати лише ті модулі, які потрібні для конкретного проєкту, і легко інтегрувати їх один з одним.

Spring також має широкі можливості для роботи з базами даних. Зокрема, Spring Data надає інструменти для роботи з реляційними та нереляційними базами, такими як MySQL, PostgreSQL, MongoDB або Redis.

Крім того, фреймворк підтримує написання складних запитів через ORM інструменти, такі як Hibernate або JPA, що робить взаємодію з даними максимально зручною.

Ще одним важливим аспектом є забезпечення безпеки. Завдяки Spring Security можна легко додати автентифікацію, авторизацію та інші механізми захисту даних. Це критично важливо для сучасних застосунків, де безпека даних користувачів є пріоритетом [18, 19]. Крім того, Spring Security інтегрується з популярними сервісами автентифікації, такими як OAuth2 або OpenID Connect, що спрощує створення застосунків із сучасними механізмами логіну.

Екосистема Spring (рис. 3.3 [20]) також підтримує побудову складних розподілених систем. Завдяки Spring Cloud можна легко інтегрувати такі функції, як балансування навантаження, централізоване управління конфігураціями, система моніторингу та інші інструменти для роботи з мікросервісами. Це робить Spring ідеальним для розробки застосунків, які повинні витримувати високі навантаження і легко масштабуватися.

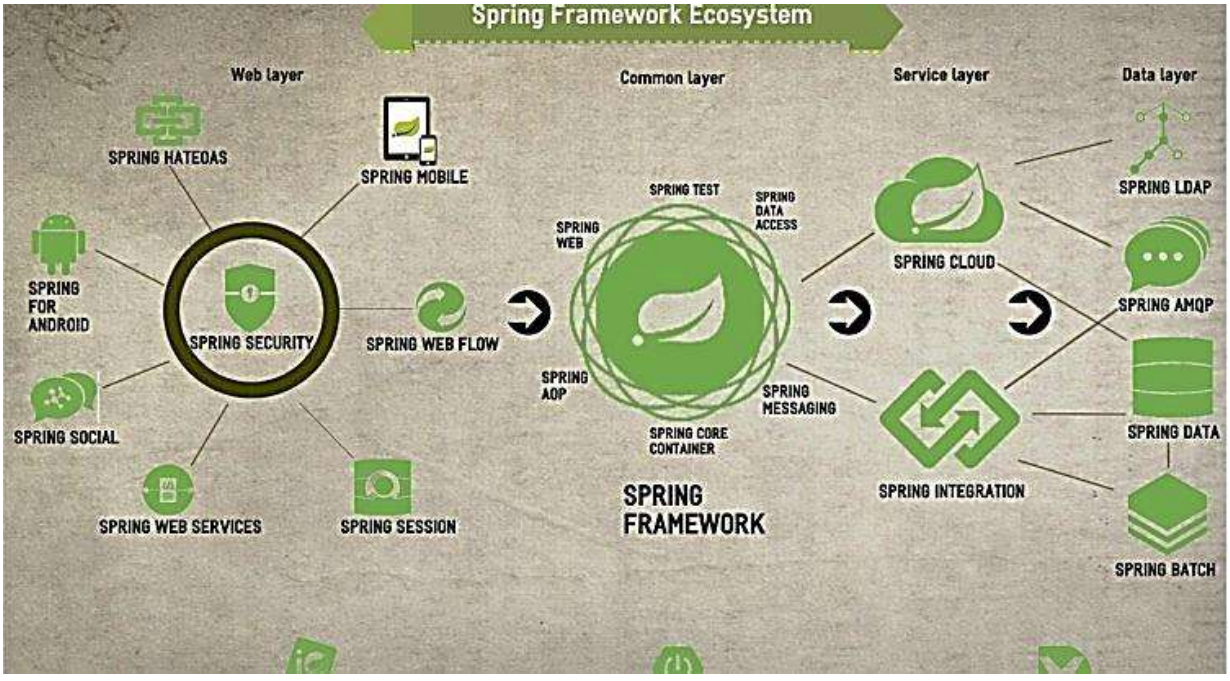

Рисунок 3.3 – Екосистема Spring

Сучасні тенденції розробки також враховані в Spring. Наприклад, підтримка реактивного програмування через Spring WebFlux дозволяє створювати застосунки, які можуть обробляти тисячі одночасних запитів без втрати продуктивності. Це особливо актуально для систем, які працюють із потоками даних у реальному часі або обслуговують велику кількість користувачів.

Spring було обрано як основний фреймворк для даного проєкту, оскільки він забезпечує поєднання простоти, гнучкості та потужності. Його модульна архітектура дозволяє зібрати екосистему, яка відповідає потребам проєкту, а велика кількість готових рішень значно скорочує час розробки.

Після того як буде створений застосунок, треба буде ще додати підтримку зборки проєкту. Є два популярних інструменти (рис. 3.4 [21]) для зборки проєктів Java застосунків. А саме maven та gradle. Для роботи обираємо maven. Після того як структура проєкту готова треба обрати інструмент для тестування. Розглянемо найпопулярніші інструменти для тестування даного застосунку.

Рисунок 3.4 – Порівняння maven та gradle

3.1.1 Тестування застосунку за допомогою Postman

Postman – це сучасний інструмент, який завоював популярність серед розробників, тестувальників і DevOps інженерів завдяки своїй простоті у використанні та широкому набору функцій. Він призначений для роботи з API, що є основою сучасних програмних систем, побудованих на мікросервісній архітектурі.

Використання API стало невід'ємною частиною створення веб і мобільних застосунків, оскільки забезпечує інтеграцію між різними сервісами, платформами та технологіями.

Головною перевагою Postman є його інтуїтивно зрозумілий інтерфейс, який робить роботу з ним доступною навіть для тих, хто не має глибоких технічних знань. У ньому зручно створювати та відправляти HTTP запити (рис. 3.5), переглядати відповіді від серверів і аналізувати їх. Це значно спрощує процес тестування API як для новачків, так і для досвідчених фахівців.

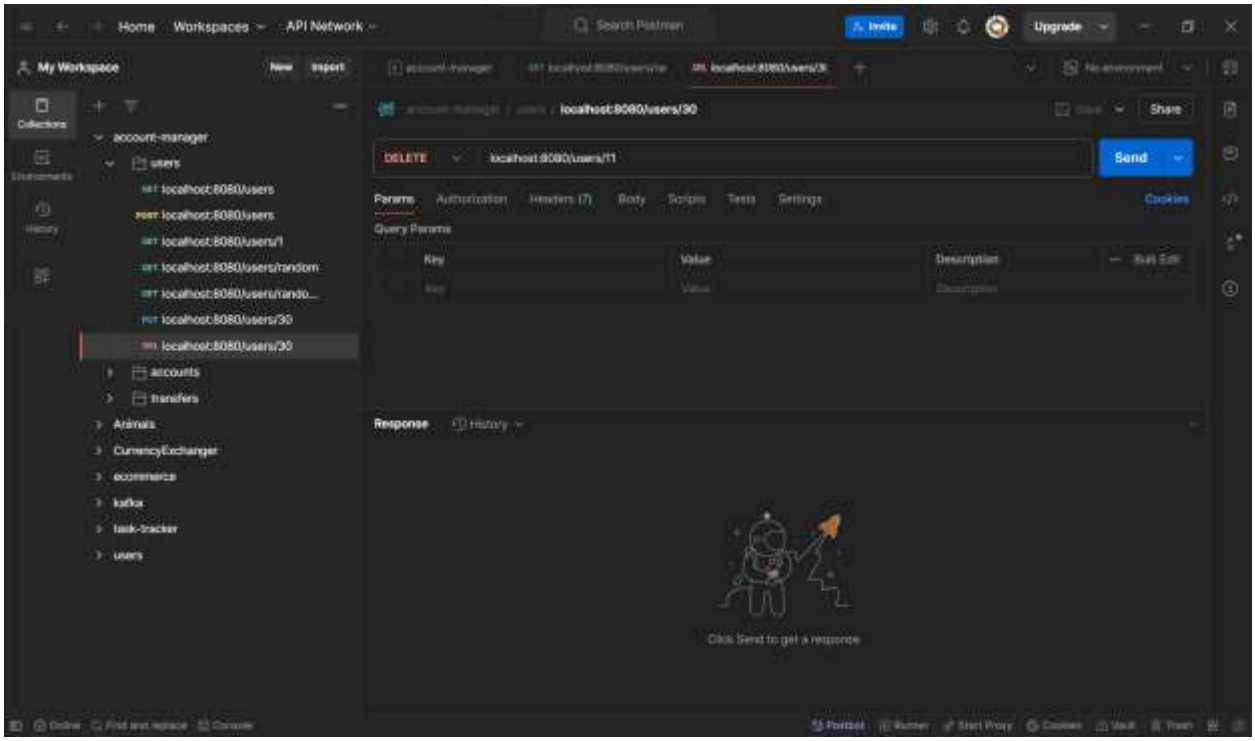

Рисунок 3.5 – Інтерфейс Postman

Однією з важливих функцій є збереження колекцій запитів, що дозволяє організувати роботу з великими наборами даних. Колекції – це набір структурованих запитів, які можна запускати послідовно, створюючи сценарії для автоматизації. Наприклад, тестувальник може зберегти десятки запитів до різних частин API, а потім одним натисканням запустити перевірку їх коректності [22].

У Postman є можливість автоматизації тестування. Це реалізується за допомогою спеціальних сценаріїв на мові JavaScript, які дозволяють перевіряти, чи відповідає отриманий результат очікуваному. Наприклад, можна написати тест, який перевіряє, що статус відповіді – 200 (успішний запит), а дані у відповіді містять певні значення. Це робить Postman зручним для регулярного функціонального тестування.

Для інтеграції у великі проєкти з розвиненими DevOps процесами Postman пропонує CLI інструмент Newman. Завдяки Newman колекції запитів можна запускати прямо в командному рядку або інтегрувати їх у CI CD конвеєри. Це означає, що API може автоматично перевірятися під час кожного оновлення коду, забезпечуючи стабільність і якість роботи всієї системи (рис. 3.6 [23]).

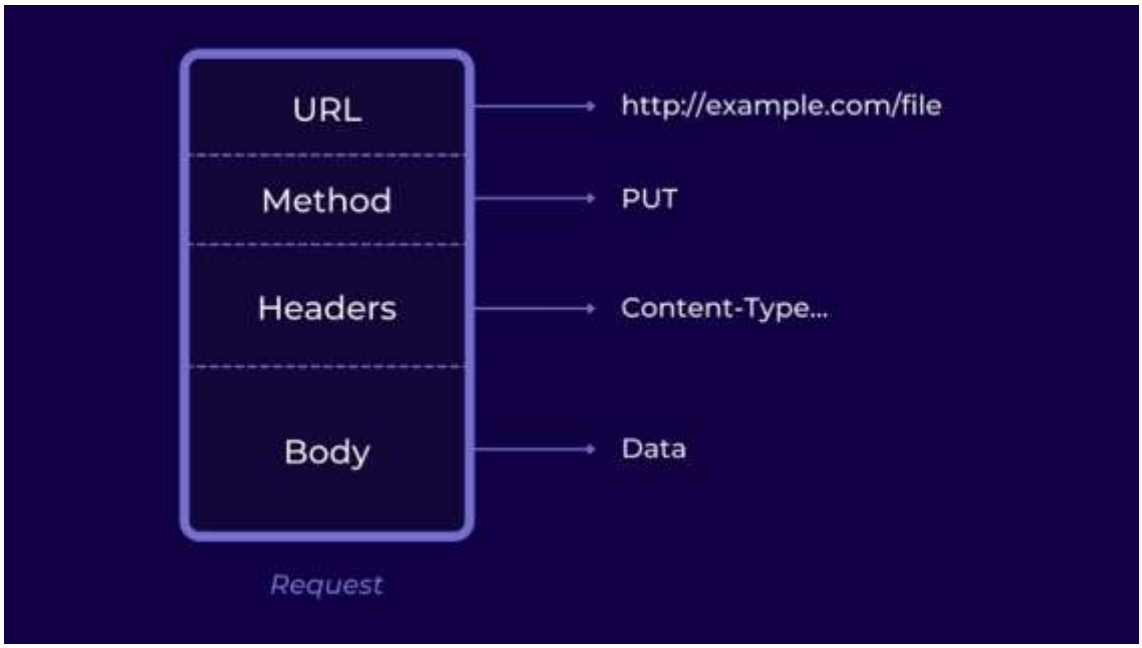

Рисунок 3.6 – Структура HTTP запиту

Postman підтримує структуру та основні HTTP методи:

– GET для отримання інформації;

– POST для створення нових даних;

– PUT для оновлення існуючих даних;

– DELETE для видалення.

Це робить його універсальним інструментом, який підходить для роботи з будь якими RESTful або GraphQL API.

Окрім функціонального тестування, Postman має функцію генерації документації API. Ця документація створюється автоматично на основі запитів у колекції, і що важливо – вона автоматично оновлюється. Це корисно для командної роботи, оскільки дозволяє всім учасникам проєкту бути в курсі актуальної структури API.

Postman також дозволяє працювати з змінними середовища, що спрощує тестування на різних серверах – наприклад, розробницькому, тестовому і продуктивному. Замість того, щоб вручну змінювати URL чи інші параметри в запитах, можна використовувати змінні, які змінюються залежно від обраного середовища.

Завдяки інтеграції з застосунком та додатковими функціями, Postman стає ще більш потужним. Він дозволяє експортувати запити у різні формати, інтегруватися з такими інструментами, як Jenkins чи GitLab, та навіть створювати mock сервери для тестування, коли реальний сервер недоступний.

Загалом, Postman є важливим інструментом для всіх, хто працює з API. Він не лише спрощує роботу, але й робить її більш ефективною завдяки автоматизації, організації та інтеграції. Незалежно від того, чи працюєте ви в невеликій команді, чи в компанії з глобальними проєктами, Postman допоможе підвищити продуктивність і знизити ризик помилок.

3.1.2 Тестування застосунку за допомогою Insomnia

Insomnia – це сучасний інструмент(рис. 3.7), що став одним із найзручніших рішень для розробників, які працюють із API. Його простота, естетичний дизайн та функціональність роблять його ідеальним вибором як для новачків, так і для досвідчених програмістів. Insomnia підтримує тестування RESTful та GraphQL API, що дозволяє охоплювати широкий спектр задач у сучасній розробці програмного забезпечення.

Однією з ключових особливостей Insomnia є легкість налаштування. Завдяки інтуїтивному інтерфейсу, розробники можуть швидко створювати та тестувати запити без необхідності глибокого вивчення документації. Інструмент дозволяє працювати з усіма основними HTTP методами, такими як GET, POST, PUT, DELETE та інші, забезпечуючи гнучкість у тестуванні різних сценаріїв взаємодії з API.

Insomnia має розширені можливості для роботи з авторизацією. Вбудована підтримка OAuth, JWT, а також інших типів токенів дає змогу легко налаштовувати доступ до захищених ресурсів.

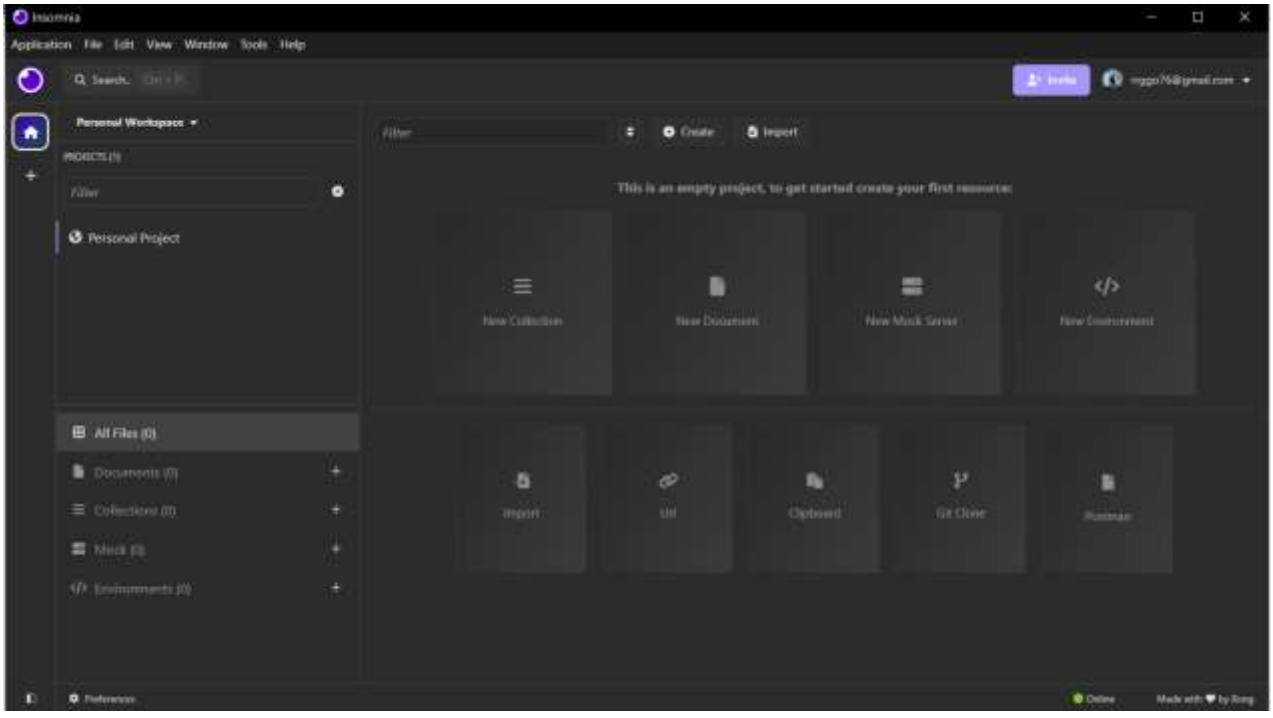

Рисунок 3.7 – Застосунок Insomnia

Наприклад, розробники можуть додавати токени авторизації до запитів лише за кілька кліків, що значно скорочує час на рутинні задачі.

Ще однією зручністю є робота зі середовищами. Insomnia дозволяє створювати окремі конфігурації для різних середовищ, таких як розробницьке, тестове чи продуктивне. Використовуючи змінні середовища, розробники можуть легко змінювати URL адреси, токени чи інші параметри залежно від обраного оточення. Це особливо корисно для команд, які працюють над великими проєктами з багатоступеневими процесами.

Для складних сценаріїв, що вимагають послідовного виконання запитів, Insomnia пропонує простий і зрозумілий підхід до роботи з змінними. Змінні можуть використовуватися для передачі даних між запитами, що спрощує тестування багатоступеневих API запитів. Наприклад, результат одного запиту може автоматично підставлятися як параметр для наступного, що дозволяє легко налаштовувати та перевіряти цілі ланцюги взаємодії з API [24]. Insomnia також дозволяє створювати та ділитися шаблонами запитів, що є дуже корисним для командної роботи. Розробники можуть створювати стандартизовані шаблони для певних типів запитів, які інші члени команди можуть використовувати у своїй роботі. Це забезпечує консистентність у тестуванні та спрощує навчання нових співробітників.

Окрім цього, інструмент підтримує експорт і імпорт даних, що дозволяє легко переносити конфігурації між різними машинами або передавати їх іншим членам команди. Це особливо зручно, якщо потрібно налаштувати однакові колекції запитів на кількох робочих станціях [25].

У підсумку, Insomnia є потужним і зручним інструментом для тестування API, який поєднує простоту використання, розширені функціональні можливості та високу гнучкість. Завдяки підтримці RESTful і GraphQL API, інтеграції з іншими інструментами та можливості автоматизації, Insomnia підходить як для індивідуальних розробників, так і для великих команд, забезпечуючи ефективність і продуктивність у роботі з API.

3.1.3 Тестування застосунку за допомогою JMeter

JMeter – це один із найпопулярніших і найпотужніших інструментів для навантажувального тестування та оцінки продуктивності програмних систем (рис. 3.8).

Розроблений компанією Apache, цей інструмент став незамінним для розробників, тестувальників і DevOps інженерів, які прагнуть забезпечити надійність і стабільність своїх застосунків навіть за умов високого навантаження.

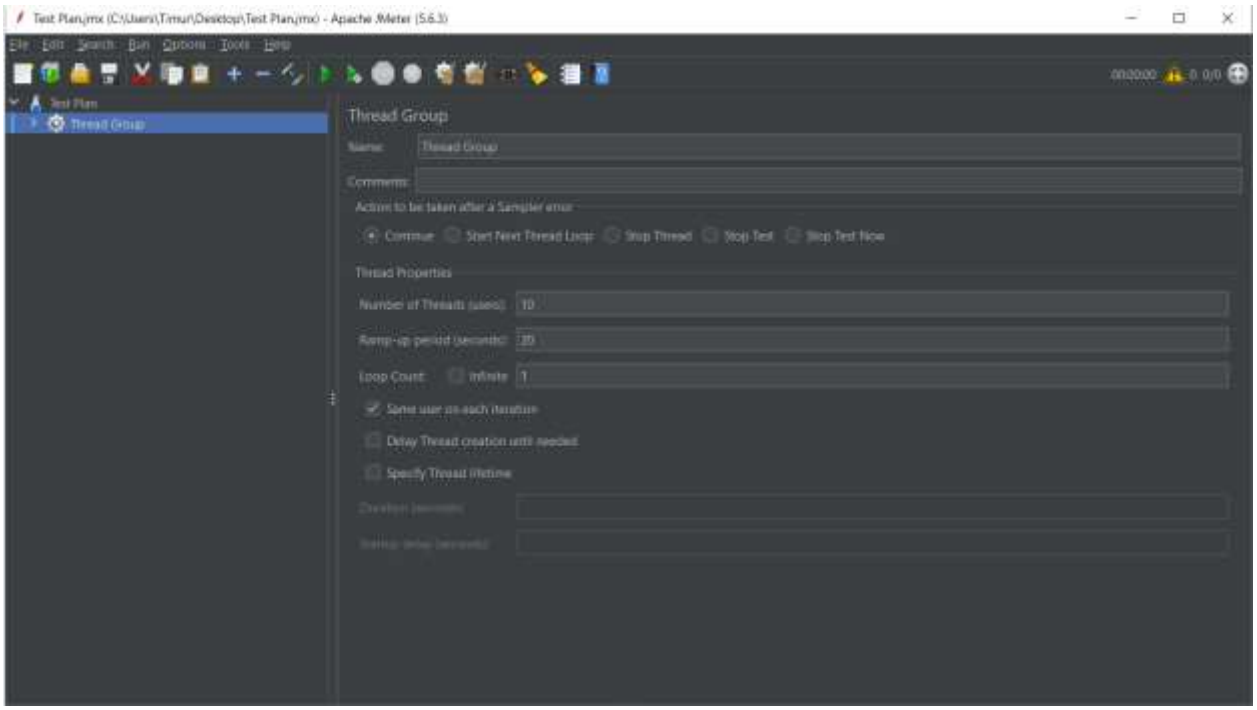

Рисунок 3.8 – Інтерфейс JMeter

Основна функція JMeter полягає в симуляції поведінки великої кількості користувачів, які одночасно взаємодіють із системою. Це дозволяє перевірити, як веб застосунок, сервер, база даних або інша система буде працювати під час пікових навантажень. Наприклад, можна симулювати десятки, сотні або навіть тисячі користувачів, які надсилають запити до сервера, та аналізувати, чи здатна система впоратися із цим трафіком без затримок або збоїв.

Однією з ключових переваг JMeter є його підтримка різноманітних протоколів, включаючи:

– HTTP для вебзастосунків;

– FTP для перевірки файлових серверів;

– JDBC для тестування баз даних;

– SOAP та REST для API.

Ця гнучкість робить JMeter універсальним інструментом, який підходить для роботи з різними технологіями та платформами. Графічний інтерфейс JMeter є інтуїтивно зрозумілим і дозволяє легко створювати складні сценарії тестування навіть новачкам.

Завдяки структурованому підходу до побудови тестів, користувачі можуть організовувати сценарії у вигляді деревоподібної структури, додаючи запити, таймери, логічні контролери та інші компоненти. Наприклад, можна створити сценарій, у якому запити надсилаються з певними затримками або відповідно до умовних переходів, що дозволяє моделювати реальні сценарії використання системи [26–28].

Для досвідчених користувачів JMeter пропонує можливість працювати з файлами конфігурації у форматі XML. Це дозволяє створювати або змінювати сценарії вручну, інтегрувати їх із зовнішніми інструментами або автоматизувати тестування в CI CD конвеєрах.

Ще однією важливою особливістю JMeter є його здатність генерувати детальні звіти. Після завершення тестування користувач отримує вичерпну інформацію про продуктивність системи:

– час відгуку сервера;

– пропускну здатність;

– відсоток помилок;

– середній час обробки запитів.

Ці дані візуалізуються у вигляді графіків, таблиць і діаграм, що допомагає виявляти вузькі місця в системі, наприклад, сервери, які не встигають обробляти запити, або бази даних, що працюють із затримками.

JMeter активно використовується в різних галузях:

– фінансова сфера, щоб перевірити стабільність банківських платформ;

– електронна комерція, для оцінки здатності обробляти пікові навантаження під час розпродажів;

– ігрова індустрія, щоб гарантувати безперебійний доступ до серверів.

У підсумку, JMeter – це не лише інструмент, але й універсальна платформа для тестування, яка дозволяє забезпечити високу якість і надійність програмних систем. Завдяки широким можливостям, активній спільноті розробників і постійним оновленням, JMeter залишається одним із провідних рішень у сфері тестування продуктивності та навантаження. Postman і Insomnia забезпечують ефективну роботу на етапі функціонального тестування, дозволяючи перевіряти коректність роботи API та перевіряти бізнес-логіку.

Використання Postman, Insomnia та JMeter забезпечує гнучкість і ефективність у процесі тестування. Ці інструменти дозволяють швидко та зручно створювати, виконувати й аналізувати тестові запити, що значно прискорює роботу з API. Кожен із цих інструментів має свої сильні сторони, які доповнюють один одного, створюючи міцну основу для забезпечення якості програмних застосунків.

Наприклад, Postman пропонує зручний інтерфейс для роботи з API, Insomnia підходить для створення складних сценаріїв, а JMeter дозволяє моделювати високі навантаження на систему. Завдяки цим інструментам можна досягти впевненості в тому, що система працює коректно, відповідає вимогам і здатна витримувати навантаження в реальних умовах експлуатації.

Крім того, використання таких засобів сприяє виявленню потенційних вузьких місць у системі ще до її впровадження у робочу систему. У результаті це знижує ризики збоїв та підвищує стабільність і продуктивність програмного забезпечення.

3.2 Етапи програмної реалізації вибраних методів підвищення програмних застосунків шляхом масштабування мікросервісів

Реалізація методів масштабування мікросервісів є ключовим етапом у забезпеченні ефективності програмних застосунків. Буде представлено покроковий підхід до впровадження кожного з методів масштабування: вертикального, горизонтального та масштабування через бази даних. Вибрані підходи охоплюють налаштування середовища, конфігурацію інструментів, розробку та тестування функціоналу, що дозволяє досягти оптимальної продуктивності [29–31].

Описані етапи базуються на сучасних підходах до розробки програмного забезпечення, зокрема використанні таких технологій, як Java, Spring Boot, а також інструментів для тестування Postman.

Метою цих кроків є створення масштабованої архітектури, здатної адаптуватися до динамічних змін навантаження та забезпечувати стабільність роботи системи навіть у пікові моменти.

3.2.1 Вертикальне масштабування

Вертикальне масштабування є одним із ключових методів підвищення продуктивності програмних застосунків. Воно передбачає заміну існуючої серверної машини на більш потужну, що дозволяє системі обробляти більше запитів, працювати швидше та стабільніше. Цей підхід особливо ефективний у випадках, коли потрібно забезпечити покращення продуктивності без суттєвих змін у архітектурі програмного забезпечення.

Перший етап реалізації вертикального масштабування передбачає детальний аналіз поточного навантаження на систему. Для цього використовуються сучасні моніторингові інструменти, такі як Prometheus або Grafana, які дозволяють відстежувати ключові метрики продуктивності:

використання процесора, пам’яті, дискових ресурсів та мережі. Завдяки цьому аналізу можна визначити «вузькі місця» системи, тобто ті компоненти, які стають причиною зниження продуктивності.

На наступному етапі здійснюється вибір нової серверної машини, характеристики якої відповідають потребам застосунку. При виборі серверу звертається увага на такі параметри, як обсяг оперативної пам’яті, кількість і потужність ядер процесора, швидкість накопичувачів (наприклад, SSD із підтримкою прискорення диску), а також пропускна здатність мережевих інтерфейсів [31–34]. У застосунку до цього, розглядається можливість вибору серверу з додатковими функціями, наприклад, апаратною підтримкою віртуалізації або спеціалізованими картами для обробки великих обсягів даних (GPU або TPU).

Особливу увагу приділяють оптимізації мережевої інфраструктури. Сервер підключається безпосередньо до локальної мережі за допомогою Ethernet кабелю з високою пропускною здатністю (1 Gbps, 10 Gbps або більше). Це дозволяє значно знизити затримки в передачі даних і забезпечити стабільність мережевих з’єднань. У той час як Wi Fi може бути зручним рішенням, він не гарантує стабільності й швидкості, необхідних для роботи продуктивних серверів.

Схема вертикального масштабування існуючого сервера (рис. 3.9).

Рисунок 3.9 – Схема вертикального масштабування

Необхідно виконати оптимізацію конфігурацій системного програмного забезпечення. Для цього проводяться налаштування операційної системи, що дозволяють максимально ефективно використовувати ресурси машини. Наприклад, оптимізуються параметри кешування пам’яті, управління потоками даних, а також конфігурації ядра операційної системи.

Крім того, якщо програмне забезпечення використовує Docker контейнери, необхідно адаптувати конфігурацію контейнерів для роботи на новій апаратній платформі.

Зокрема, налаштовуються обмеження використання CPU та пам’яті, забезпечується правильне управління кешем і оптимізуються параметри JVM для ефективного виконання Java застосунків [34–36].

Завершальним етапом є тестування системи під навантаженням. Для цього використовується інструмент Postman, який дозволяє створити сценарії моделювання реального навантаження. Наприклад, можна задати одночасну обробку сотень або навіть тисяч запитів, оцінити швидкість їх виконання та виявити можливі проблеми. Після тестування результати аналізуються для підтвердження ефективності змін. Якщо результати свідчать про значне покращення продуктивності, система вважається готовою до подальшого використання.

Отже, вертикальне масштабування є поетапним процесом, який включає аналіз, підбір обладнання, оптимізацію та тестування. Такий підхід дозволяє не лише покращити продуктивність, але й забезпечити стабільну роботу програмного забезпечення в умовах зростаючого навантаження.

3.2.2 Горизонтальне масштабування

Горизонтальне масштабування є одним із основних підходів для покращення продуктивності програмних застосунків у випадках, коли вертикальне масштабування стає недостатнім.

Воно включає створення декількох серверів серверної програми, що дозволяє ефективно розподіляти навантаження між ними та підвищити загальну доступність і стабільність системи. Завдяки цьому підходу можна обробляти більший обсяг запитів, а також забезпечити безперервну роботу сервісів у разі збою окремих серверів.

Перше, що необхідно зробити для впровадження горизонтального масштабування – це налаштувати балансувальник навантаження. Одним із популярних інструментів для цієї мети є NGINX, який є високопродуктивним веб сервером і зворотнім сервером. Завдання NGINX полягає в тому, щоб рівномірно перенаправляти вхідні запити на різні сервери серверної програми, тим самим забезпечуючи баланс навантаження. Завдяки цьому методові можна запобігти перевантаженню окремих серверів, а також забезпечити високу доступність, оскільки у разі збою одного серверу запити будуть автоматично перенаправлятися на інші доступні сервери [37–39].

Після налаштування балансувальника необхідно створити додаткові сервери серверної програми. Це зазвичай відбувається за допомогою Docker контейнерів, які дозволяють ізолювати програми та їх залежності в окремих середовищах (рис. 3.10). Кожен сервіс може працювати в окремому контейнері, що забезпечує мобільність і спрощує управління сервісами. Docker дозволяє створити стандартизоване середовище, яке буде однаковим на всіх серверах, незалежно від конфігурації головної системи. Такі сервери можуть бути створені швидко і в автоматизованому режимі.

Рисунок 3.10 – Докер кластер сервісів

Лістинг 3.1 Докер компоуз файл для запуску сервісів:

```
version: ‘3’

services:
  account_service_1:
    build:
      context: ./
      dockerfile: Dockerfile
    container_name: account-service-1
    environment:
      DB_URL: jdbc:postgresql://localhost:5432/transfers
    ports:
      - "8081:8081"
    depends_on:
      postgres:
        condition: service_healthy
      redis:
        condition: service_completed_successfully

  account_service_2:
    build:
      context: ./
      dockerfile: Dockerfile
    container_name: account-service-2
    environment:
      DB_URL: jdbc:postgresql://localhost:5432/transfers
    ports:
      - "8082:8081"
    depends_on:
```

```
    postgres:
      condition: service_healthy
    redis:
      condition: service_completed_successfully

  account_service_3:
    build:
      context: ./
      dockerfile: Dockerfile
    container_name: account-service-3
    environment:
      DB_URL: jdbc:postgresql://localhost:5432/transfers
    ports:
      - "8083:8081"
    depends_on:
      postgres:
        condition: service_healthy
      redis:
        condition: service_completed_successfully

  nginx:
    image: nginx:latest
    container_name: nginx
    ports:
      - "80:80"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
    depends_on:
      - account_service_1
      - account_service_2
```

```
    - account_service_3

  postgres:
    image: postgres:14-alpine
    container_name: postgres_db_accounts
    volumes:
      - postgres_volume:/var/lib/postgresql/data
    environment:
      POSTGRES_DB: transfers
      PGUSER: postgres
      POSTGRES_PASSWORD: postgres
    ports:
      - "5432:5432"
    healthcheck:
      test: [ "CMD-SHELL", "pg_isready", "-U", "docker_app" ]
      interval: 5s
      timeout: 5s
      retries: 3

volumes:
  postgres_volume:
```

Наступним кроком є тестування налаштованої системи. Для цього проводиться перевірка ефективності балансування навантаження, а також можливості системи обробляти збільшений обсяг запитів.

Тестування допомагає виявити потенційні проблеми, такі як погане розподілення навантаження або неефективне використання ресурсів. Для цього можуть бути використані інструменти для навантажувального тестування, такі як JMeter або Gatling, які дозволяють змоделювати високі навантаження та оцінити, як система справляється з великими обсягами

запитів. Крім того, перевіряється, чи правильно налаштовані механізми автоматичного розширення, тобто чи система правильно реагує на зміни навантаження, додаючи або видаляючи сервіси за потреби.

3.2.3 Масштабування через бази даних

Масштабування бази даних також потребує впровадження механізмів розподілу навантаження. Для цього використаємо реплікацію баз даних. Для цього спочатку налаштовується механізм реплікації, при якому основний сервер відповідає за операції запису, а кілька реплік – за обробку запитів на читання. Реплікація дозволяє розподіляти навантаження між кількома серверами, покращуючи ефективність роботи застосунка та знижуючи навантаження на основний сервер.

Водночас, для забезпечення ефективної роботи системи використовується балансувальник навантаження, який перенаправляє запити на читання до найбільш доступних реплік. Це дозволяє оптимізувати використання ресурсів і зменшити навантаження на основний сервер.

Для налаштування реплікації в PostgreSQL потрібно виконати кілька кроків на основному сервері та на сервері репліці. Основна мета реплікації – це створити точну копію даних на декількох серверах для забезпечення відмовостійкості та масштабованості [40].

На першому етапі потрібно налаштувати основний сервер. Для цього відредагуйте файл postgresql.conf на основному сервері. Потрібно встановити параметри, які дозволяють створювати копії серверів.

Лістинг 3.2 Налаштування postgres:

```
//postgres.conf
listen_addresses = ‘*’
wal_level = replica
```

max_wal_senders = 3

wal_keep_segments = 64

hot_standby = on

//Launch

pg_basebackup -h master_ip_address -D /var/lib/postgresql/data -U replica_user -P --wal-method=stream

//Final configure

standby_mode = ‘on’

primary_conninfo = ‘host=master_ip_address port=5432 user=replica_user password=password’

trigger_file = ‘/tmp/postgresql.trigger.5432’

Застосування описаних підходів до масштабування дозволяє значно підвищити ефективність програмних застосунків, забезпечити стабільну роботу навіть під великим навантаженням і оптимізувати використання ресурсів. Кожен з методів має свої сильні сторони та може використовуватися окремо або в комбінації залежно від вимог до системи. Реалізація цих рішень закладає основу для гнучкості та масштабованості сучасних програмних систем.

3.3 Дослідження методів підвищення ефективності програмних застосунків шляхом масштабування мікросервісів

Першочергово потрібно запустити навантажувальне тестування на сервіс коли він ще не масштабований. Запустимо навантажувальне тестування в Postman (рис. 3.11).

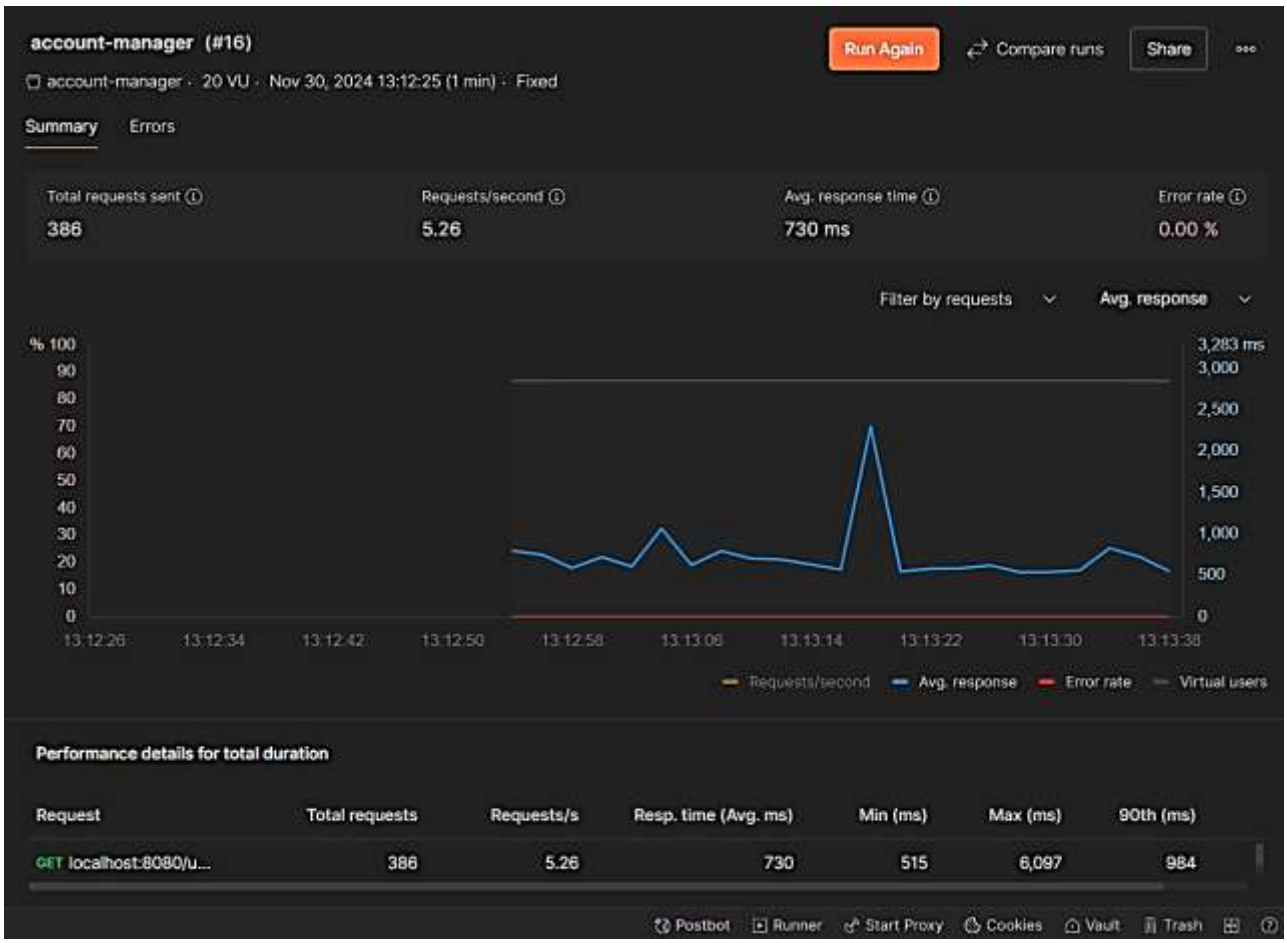

Рисунок 3.11 – Навантажувальне тестування в Postman

Одразу бачимо, що середній час відповіді складає 730 мс це тільки зі сторони бекенду, отже ще можлива затримка зі сторони візуальної частини сайту.

Можемо припустити що затримка буде складати більше 1 секунди, що можемо дуже дратувати користувачів даного застосунку, тепер виконаємо вертикальне масштабування та перевіримо тестом скільки зараз складає затримка.

За результатами тестування (рис. 3.12) затримка складає 268 мс, що є вже кращим результатом.

Тепер можна провести навантажувальне тестування для методу горизонтального масштабування (рис. 3.13).

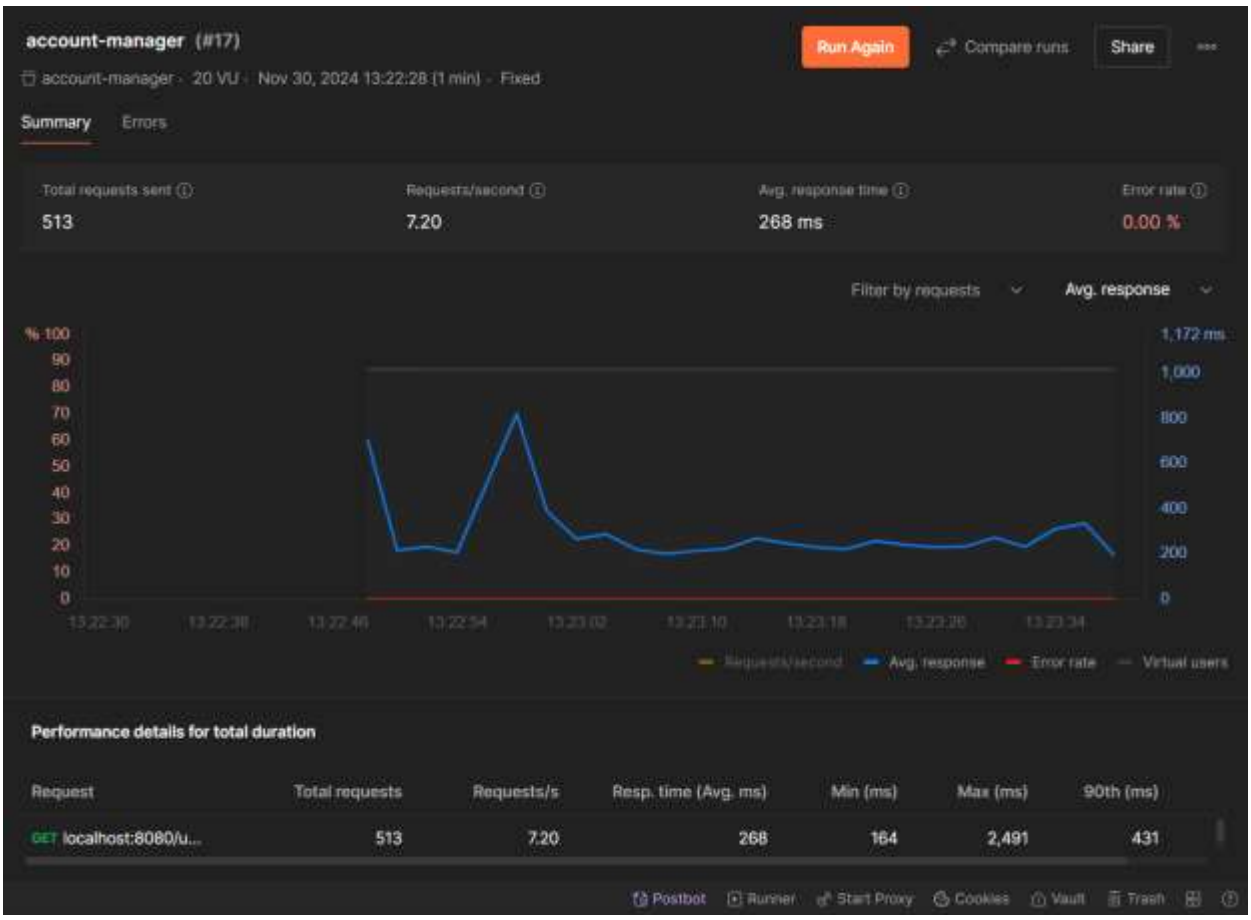

Рисунок 3.12 – Навантажувальне тестування вертикального методу

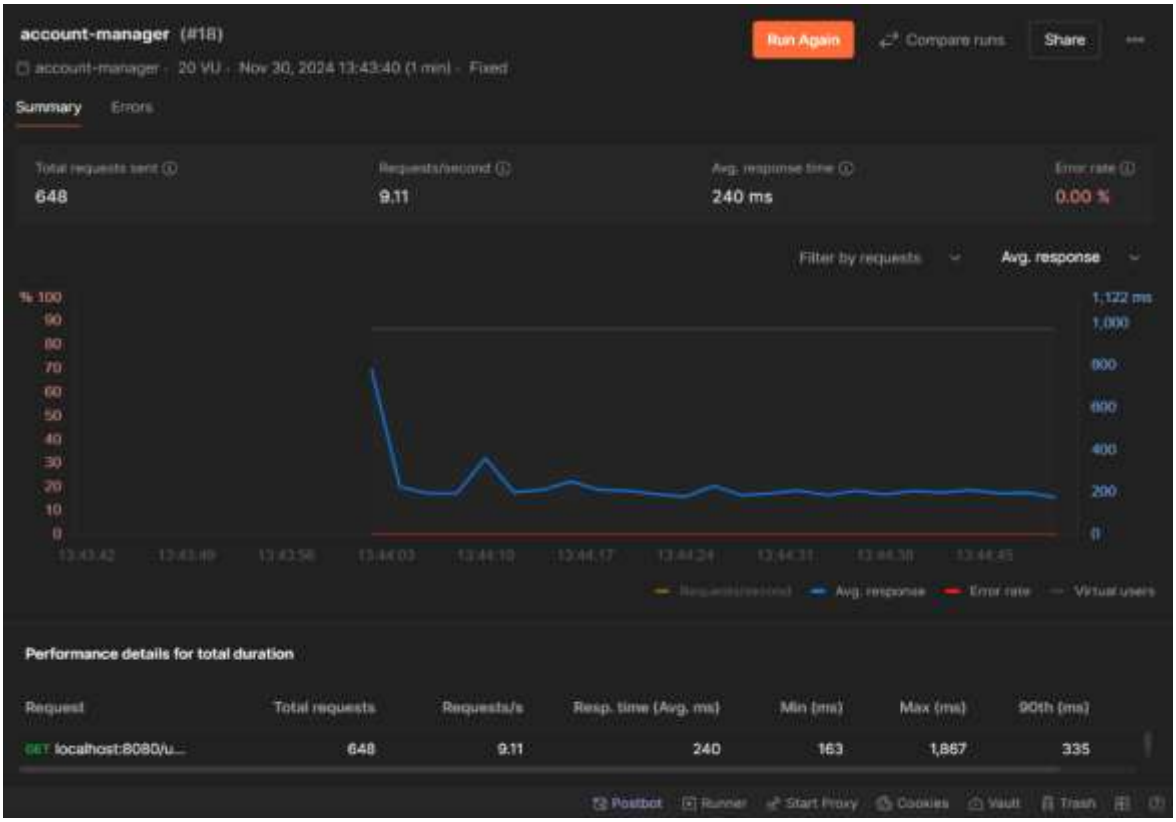

Рисунок 3.13 – Навантажувальне тестування горизонтального методу

Бачимо що затримка склала 240 мс. Трішки краще, ніж при вертикальному методі, можливо припустити, що маємо дуже схожі результати через невеликий розмір мікросервісу, на більших даних та запитах різниця між вертикальним методом та горизонтальним має бути більш суттєвою.

Залишилось перевірити навантажувальне тестування на методі через бази даних (рис. 3.14).

Проаналізувавши результати тестування, можливо дійти висновку, що метод масштабування через бази даних, має найбільшу затримку, це зумовлено ти що так само залишаються активні сервери, які просто розподіляють навантаження на бази даних. Якби об’єднати горизонтальне масштабування та масштабування через баз даних, ми б дійшли набагато кращих результатів. Після того як було реалізовано тестування всіх методів, переходимо до критеріального аналізу всіх методів де буде порівняно їх всіх між собою.

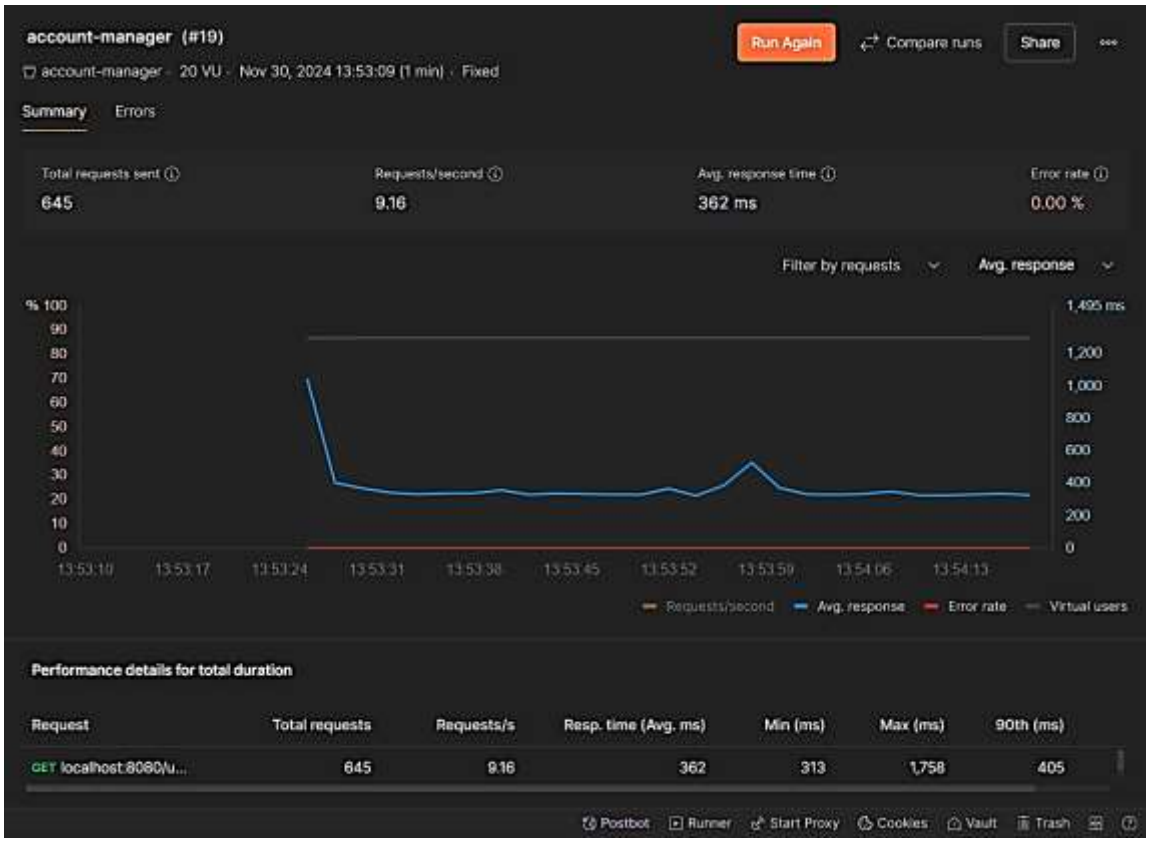

Рисунок 3.14 – Навантажувальне тестування методу через бази даних

3.4 Критеріальний порівняльний аналіз методів підвищення ефективності програмних застосунків шляхом масштабування мікросервісів

Для проведення ефективного навантажувального тестування були застосовані три різні підходи до масштабування: вертикальне масштабування, горизонтальне масштабування та масштабування через бази даних. Кожен із цих методів має свої особливості, переваги й недоліки, які були проаналізовані на основі отриманих метрик.

На основі тестування з використанням 100 віртуальних користувачів були отримані такі результати які показано у таблиці 3.1.

Таблиця 3.1 – Порівняння результатів

Параметр	**Вертикальний метод (№ 17)**	**Горизонтальний метод (№ 18)**	**Масштабування через БД (№ 19)**
Загальна кількість запитів	513	648	645
Кількість запитів на секунду	7,22	9,21	9,21
Середній час відповіді	268	240	362
Мінімальний час відповіді	164	163	313
Максимальний час відповіді	2491	1867	1758

Вертикальне масштабування, представлене в тесті №17, передбачає збільшення потужності одного сервера, наприклад, за рахунок додавання оперативної пам’яті чи потужнішого процесора. Цей метод показав найнижчу пропускну здатність 7,22 запитів/с серед усіх трьох підходів, але при цьому середній час відповіді становив 268 мс, що є відносно прийнятним для простих сценаріїв. Проте максимальний час відповіді досяг 2491 мс, що свідчить про потенційні затримки при пікових навантаженнях. Такий підхід може бути ефективним у випадках, коли розширення інфраструктури неможливе, але він має свої фізичні обмеження.

Горизонтальне масштабування, яке тестувалося у сценарії №18, показало найкращі результати за всіма ключовими показниками. Пропускна здатність системи зросла до 9,21 запитів/с, а середній час відповіді становив 240 мс. Максимальний час відповіді, був вищим, ніж очікувалося (1867 мс), але залишається в межах прийнятного для масштабованих систем.

Також буде показано порівняння усіх графіків між собою. Порівняння горизонтального та вертикального показано на рисунку 3.15.

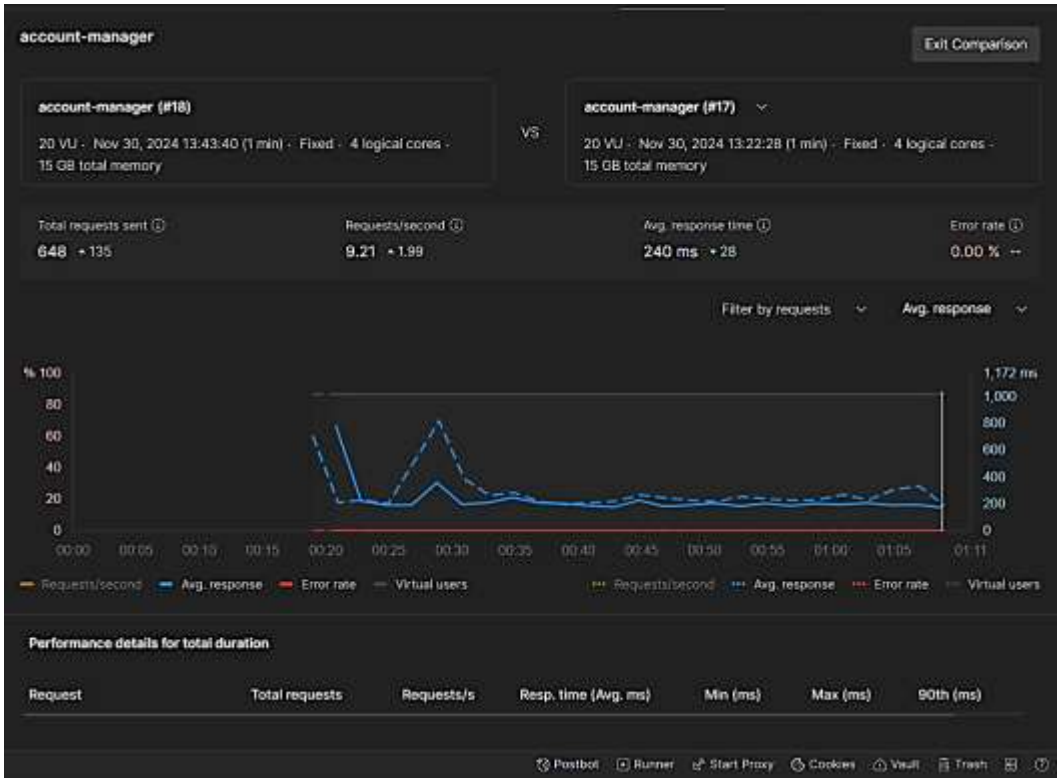

Рисунок 3.15 – Порівняння горизонтального та вертикального

Масштабування через бази даних, представлене у тесті №19, передбачає використання реплікації для розподілу запитів на читання між декількома репліками бази.

Пропускна здатність склала 9,21 запитів/с, що є найвищим серед усіх трьох підходів. Проте середній час відповіді зріс до 362 мс через додаткову затримку, викликану реплікацією даних. Мінімальний час відповіді становив 313 мс, а максимальний – 1758 мс. Цей підхід ідеально підходить для застосунків, у яких переважають запити на читання, оскільки реплікація значно знижує навантаження на основний сервер бази даних.

Порівняння масштабування через бази даних та горизонтального способу показано на рисунку 3.16.

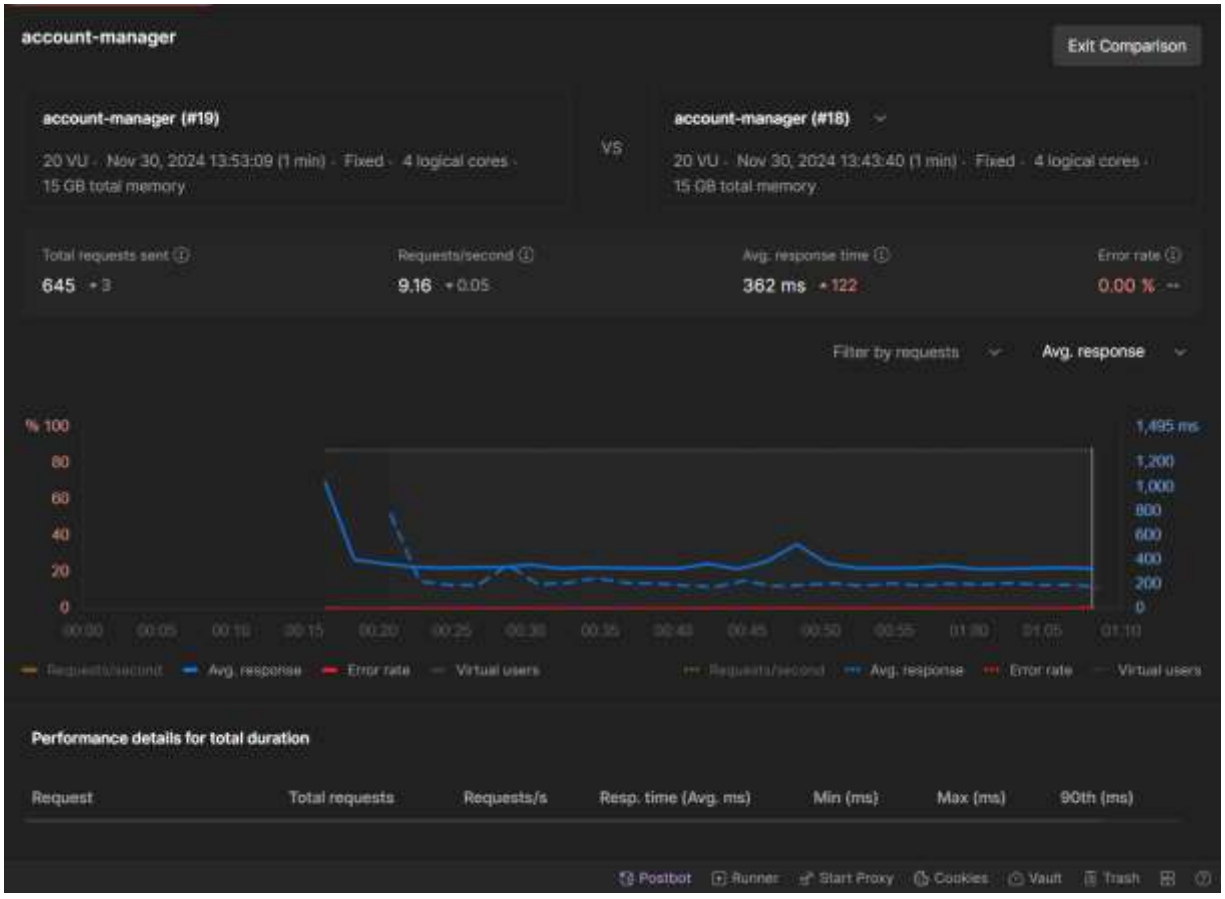

Рисунок 3.16 – Порівняння через БД та горизонтального способу

Порівняння масштабування через бази даних та вертикального способу показано на рисунку 3.17.

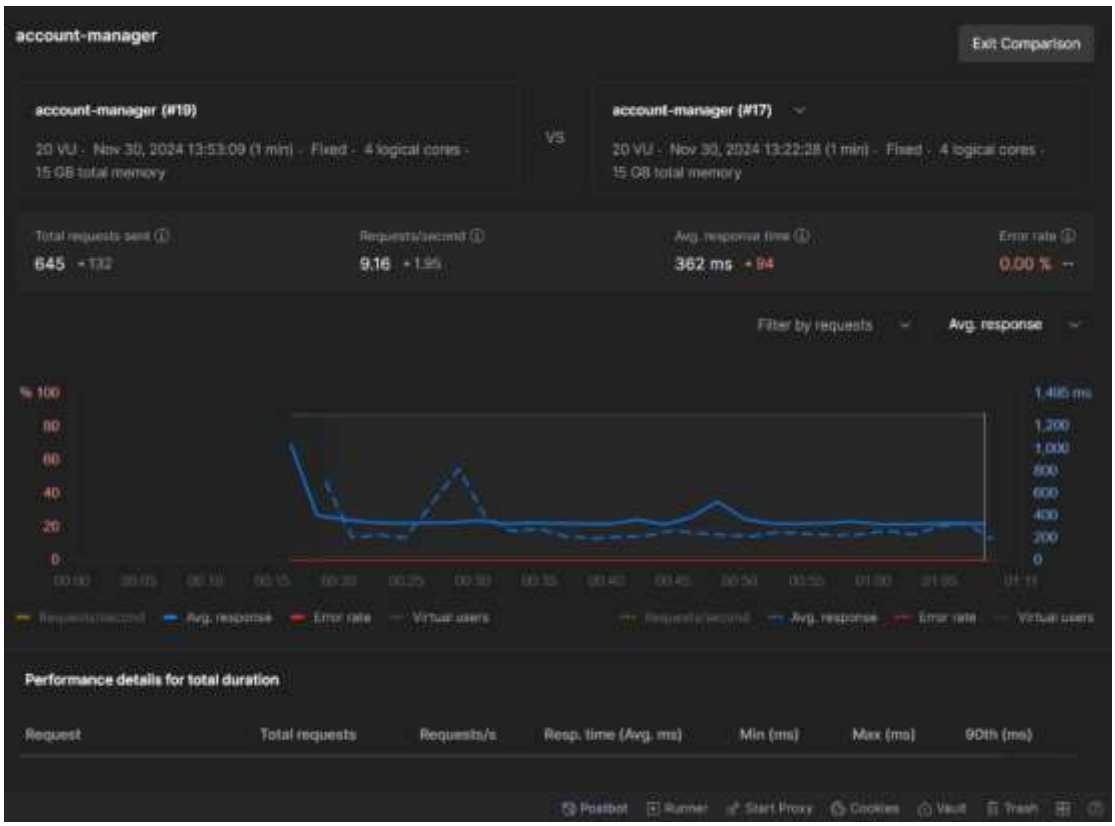

Рисунок 3.17 – Порівняння через БД та вертикального способу

Проведене тестування продемонструвало, що кожен із методів масштабування має свої унікальні переваги та обмеження, які слід враховувати залежно від конкретних вимог системи й очікуваного навантаження. Кожен підхід розроблений для оптимізації роботи застосунку за певних умов і дозволяє забезпечити стабільну та ефективну роботу програмного забезпечення.

Вертикальне масштабування, хоч і є простим у реалізації, найкраще підходить для систем, що не вимагають обробки великої кількості запитів. Цей підхід забезпечує додавання обчислювальних ресурсів (процесорної потужності, оперативної пам’яті чи дискового простору) до одного сервера, що дозволяє значно підвищити його продуктивність у короткостроковій перспективі. Однак цей метод має свої обмеження, оскільки апаратні ресурси завжди мають максимальні межі, після досягнення яких подальше масштабування стає неможливим. Це робить вертикальне масштабування ефективним лише для простих або невеликих застосунків [41].

Горизонтальне масштабування виявилося найбільш універсальним і потужним рішенням, оскільки воно дозволяє системі залишатися продуктивною навіть при різкому зростанні кількості користувачів або запитів. Цей метод передбачає розподіл навантаження між кількома серверами, що працюють паралельно, що не тільки підвищує пропускну здатність системи, але й забезпечує її стійкість до збоїв. У випадках виходу з ладу одного з серверів навантаження автоматично перерозподіляється на інші. Завдяки своїй гнучкості горизонтальне масштабування є ідеальним вибором для сучасних хмарних сервісів і застосунків, які потребують високої доступності та продуктивності.

Масштабування через бази даних продемонструвало свою ефективність у сценаріях, де переважають запити на читання. Цей метод базується на реплікації даних між кількома серверами бази даних, що дозволяє розподілити навантаження між основним сервером і його репліками. Завдяки цьому основний сервер може зосередитися на виконанні запитів запису, тоді як репліки обробляють запити читання. Такий підхід є ідеальним для застосунків, де основний обсяг операцій пов’язаний з читанням даних, наприклад, інформаційних порталів або систем аналітики. Цей метод має деякі виклики, пов’язані з налаштуванням і підтримкою синхронізації між серверами, що може вимагати додаткових ресурсів та технічної експертизи.

Таким чином, обраний метод масштабування повинен відповідати конкретним вимогам і особливостям проєкту. Вертикальне масштабування забезпечує швидке та просте рішення для невеликих систем. Горизонтальне масштабування ідеально підходить для застосунків з великим і непередбачуваним навантаженням. Масштабування через бази даних дозволяє оптимізувати роботу застосунків із великою кількістю операцій читання, забезпечуючи їх ефективність і стабільність. Успішна реалізація цих підходів залежить від розуміння потреб системи, прогнозованих навантажень і технічних можливостей інфраструктури.

3.5 Перспективи подальшої роботи

Подальша робота в напрямку масштабування програмних застосунків відкриває значні перспективи для досліджень і впроваджень нових підходів. Одним із ключових напрямків є вивчення можливостей масштабування у хмарних середовищах, які сьогодні є основою для багатьох сучасних систем. Хмарні сервіси надають великий спектр інструментів для автоматичного горизонтального і вертикального масштабування, які дозволяють не тільки розширювати ресурси за потреби, але й оптимізувати витрати, сплачуючи лише за фактично використані ресурси. Розгляд таких платформ, як Amazon Web Services (AWS), Microsoft Azure чи Google Cloud Platform, може стати важливим етапом подальших досліджень.

Окрім цього, варто звернути увагу на впровадження контейнеризації та оркестрації за допомогою таких інструментів, як Docker та Kubernetes. Контейнеризація дозволяє створювати легковагі, ізольовані середовища для застосунків, що спрощує їх розгортання і масштабування. Kubernetes, у свою чергу, автоматизує процес управління контейнерами, забезпечуючи балансування навантаження, розподіл ресурсів і відмовостійкість. Використання цих технологій у поєднанні з хмарними сервісами створює потужний інструментарій для побудови масштабованих і ефективних систем.

Ще одним важливим аспектом є дослідження гібридних підходів до масштабування, які комбінують переваги вертикального, горизонтального масштабування і масштабування через бази даних. Наприклад, можна розробити систему, яка залежно від типу і обсягу навантаження динамічно обирає найбільш підходящий метод масштабування. Такий підхід може підвищити гнучкість і продуктивність системи, забезпечуючи при цьому оптимальне використання ресурсів.

Варто розглянути питання оптимізації витрат, пов’язаних із масштабуванням. Впровадження механізмів моніторингу і прогнозування навантажень, які дозволяють заздалегідь планувати розширення ресурсів.

Інструменти аналітики та штучного інтелекту можуть відігравати ключову роль у цьому процесі, допомагаючи автоматизувати ухвалення рішень щодо масштабування.

Зрештою, подальша робота може також зосередитися на підвищенні безпеки масштабованих систем. Зі зростанням кількості компонентів і складності системи зростають і ризики, пов’язані з кібератаками, втратою даних чи неправильною конфігурацією. Тому впровадження комплексних підходів до захисту даних і забезпечення безперебійної роботи є критично важливим завданням.

ВИСНОВКИ

У рамках кваліфікаційної роботи було проведено дослідження методів масштабування мікросервісів, що дало змогу проаналізувати їх ефективність, переваги та обмеження. Було розглянуто три основні підходи до масштабування: вертикальне масштабування, горизонтальне масштабування та масштабування через бази даних. Для кожного з методів було реалізовано навантажувальне тестування, що дозволило оцінити їх вплив на продуктивність системи за різних умов.

Вертикальне масштабування продемонструвало високу ефективність при низьких навантаженнях, однак виявило свої обмеження у випадках, коли система потребує значного збільшення ресурсів. Горизонтальне масштабування показало себе як найбільш універсальний підхід, здатний забезпечити стабільну роботу навіть при високих навантаженнях завдяки розподілу запитів між кількома серверами мікросервісів. Масштабування через бази даних продемонструвало ефективність у сценаріях із високою кількістю запитів на читання, дозволяючи оптимізувати навантаження на основний сервер за допомогою реплікації даних і балансування запитів.

Наукова новизна роботи полягає у розробці комплексної методики вибору та реалізації методів масштабування мікросервісів. Під час дослідження вперше поєднано на практиці три підходи із їхньою адаптацією до різних умов та типів навантажень, що дозволило підвищити ефективність програмного застосунку із врахуванням специфіки інфраструктури.

Проведене дослідження також показало, що вибір методу масштабування залежить від специфіки застосунку, типу навантаження та обмежень інфраструктури.

Таким чином, виконана робота не лише підтвердила важливість і актуальність масштабування для підвищення ефективності програмних застосунків, але й надала інструменти для адаптації масштабування під конкретні проєкти.

Дане дослідження створює основу для подальших розробок у сфері оптимізації систем, зокрема із використанням хмарних технологій, автоматизованого моніторингу та адаптивних підходів до масштабування.

Результати роботи апробовано у вигляді тези доповіді під час Міжнародного молодіжного форуму «РАДІОЕЛЕКТРОНІКА І МОЛОДЬ У XXI СТОЛІТТІ» [39].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Кіяшко, Д. (2023, September). Аналіз використання мікросервісів у вебзастосунках. In The 36th International scientific and practical conference “Current trends in the development of youth theories”(September 12–15, 2023) Ankara, Turkey. International Science Group. 2023. 212 p. (p. 185).

2. Building Microservices with JHipster. URL: https://www.transdigitaltech.com/post/building-microservices-with-jhipster (дата звернення 15.11.2024).

3. Маєвський, Я. Ю. (2022). Метод підвищення ефективності автоматизації масштабування мікросервісів у відкритій системі автоматичного розгортання і управління контейнеризованими застосунками.

4. Кіяшко, Д. Г. (2024). Вивчення засобів та архітектури для взаємодії мікросервісів у вебзастосунках.

5. Cloud security. URL: https://www.slideshare.net/slideshow/cloud-security-unit-2-notes-ppt-unit-2-pptpptx/266562082 (дата звернення 12.11.2024).

6. Coulson, N. C., Sotiriadis, S., & Bessis, N. (2020). Adaptive microservice scaling for elastic applications. IEEE Internet of Things Journal, 7(5), 4195-4202.

7. Demonstrates-Grafana setup and logs monitoring. URL: https://medium.com/@harshBlogs/demonstrates-grafana-setup-and-logs-monitoring-351d5d735e33 (дата звернення 10.11.2024).

8. Blinowski, G., Ojdowska, A., & Przybyłek, A. (2022). Monolithic vs. microservice architecture: A performance and scalability evaluation. IEEE Access, 10, 20357-20374.

9. Yu, G., Chen, P., & Zheng, Z. (2019, July). Microscaler: Automatic scaling for microservices with an online learning approach. In 2019 IEEE International Conference on Web Services (ICWS) (pp. 68-75). IEEE.

10. Hasselbring, W., & Steinacker, G. (2017, April). Microservice architectures for scalability, agility and reliability in e-commerce. In 2017 IEEE International Conference on Software Architecture Workshops (ICSAW) (pp. 243-246). IEEE.

11. Top caching strategies. URL: https://blog.bytebytego.com/p/top-caching-strategies (дата звернення 29.10.2024).

12. СЕЛІВЬОРСТОВА, Т., & КРАСНОШАПКА, Н. (2023). ОСОБЛИВОСТІ ПРОЄКТУВАННЯ МАСШТАБОВАНОЇ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ ДЛЯ ВЕБСЕРВІСІВ. Information Technology: Computer Science, Software Engineering and Cyber Security, (4), 58-66.

13. Creating a Scalable SaaS Platform: Architectural Strategies, Scalability, and Technology Stack Essentials. URL: https://medium.com/@zainkamran29/creating-a-scalable-saas-platform-architectural-strategies-scalability-and-technology-stack-d78de1e9911e (дата звернення 17.10.2024).

14. Семенюк, В. В. (2019). Мікросервіси. Аналіз і порівняння методів розробки і способів використання. Міжнародні наукові дослідження: інтеграція науки та практики як механізм ефективного розвитку, 171.

15. Daradkeh, Y.I., Gorokhovatskyi, V., Tvoroshenko, I., Gadetska, S., and Al-Dhaifallah, M. (2021) Methods of Classification of Images on the Basis of the Values of Statistical Distributions for the Composition of Structural Description Components, *IEEE Access*, 9, pp. 92964-92973.

16. Scalability in Cloud Computing: Horizontal vs. Vertical Scaling. URL: https://www.stormit.cloud/blog/scalability-in-cloud-computing-horizontal-vs-vertical-scaling (дата звернення 27.10.2024).

17. JDK, JRE and JVM in Java. URL: https://medium.com/@heeah/jdk-jre-and-jvm-in-java-8882128f9ff7 (дата звернення 28.10.2024).

18. Xiao, Z., & Hu, S. (2022, September). Dscaler: A horizontal autoscaler of microservice based on deep reinforcement learning. In 2022 23rd Asia-Pacific Network Operations and Management Symposium (APNOMS) (pp. 1-6). IEEE.

19. Gorokhovatskyi, V.O., Tvoroshenko, I.S., and Peredrii O.O. (2020) Image classification method modification based on model of logic processing of bit description weights vector, *Telecommunications and Radio Engineering*, 79(1), pp. 59-69.

20. Гороховатський В.О., Творошенко І.С., Чмутов Ю.В. (2022) Застосування систем ортогональних функцій для формування простору ознак у методах класифікації зображень, *Сучасні інформаційні системи*, 6(3), С. 5-12.

21. Maven and Gradle in-depth comparison. URL: https://tomgregory.com/gradle/maven-vs-gradle-comparison (дата звернення 05.10.2024).

22. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., and Al-Dhaifallah M. (2022) Classification of Images Based on a System of Hierarchical Features, *Computers, Materials & Continua*, 72(1), pp. 1785-1797.

23. Understanding HTTP Requests: Ways to Monitor and Troubleshoot. URL: https://nestify.io/blog/http-requests-ways-to-monitor-and-troubleshoot/ (дата звернення 11.10.2024).

24. Pomazan, V., Tvoroshenko, I., & Gorokhovatskyi, V. (2023). Development of an application for recognizing emotions using convolutional neural networks, *International Journal of Academic Information Systems Research*, 7(7), pp. 25-36.

25. Spring Framework Introduction. URL: https://www.cloudtechtwitter.com/2022/10/spring-framework-introduction.html (дата звернення 25.10.2024).

26. Pomazan V., Tvoroshenko I., and Gorokhovatskyi V. (2023) Handwritten character recognition models based on convolutional neural networks, International Journal of Academic Engineering Research, 7(9), pp. 64-72.

27. Gorokhovatskyi, V., Tvoroshenko, I., Kobylin, O., & Vlasenko, N. (2023). Search for visual objects by request in the form of a cluster representation for the structural image description, *Advances in Electrical and Electronic Engineering*, 21(1), pp. 19-27.

28. Tvoroshenko I., Gorokhovatskyi V., Kobylin O., and Tvoroshenko A. (2023) Application of deep learning methods for recognizing and classifying culinary dishes in images, International Journal of Academic and Applied Research, 7(9), pp. 57-70.

29. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., and Zeghid M. (2024) Improving the effectiveness of image classification structural methods by compressing the description according to the information content criterion, Computers, Materials & Continua, vol. 80, no. 2, pp. 3085-3106.

30. Гороховатський В., Передрій О., Творошенко І., Марков Т. (2023) Матриця відстаней для множини компонентів структурного опису як інструмент для створення класифікатора зображень, Сучасні інформаційні системи, 7(1), С. 5-13.

31. Tvoroshenko I., and Gorokhovatskyi V. (2022) The Application of Hybrid Intelligence Systems for Dynamic Data Analysis, International Journal of Engineering and Information Systems, 6(2), pp. 40-48.

32. Tvoroshenko I., Pomazan V., Gorokhovatskyi V., and Kobylin O. (2023) Application of video data classification models using convolutional neural networks, International Journal of Academic and Applied Research, 7(11), pp. 134-145.

33. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., Gadetska S., and Al-Dhaifallah M. (2023) Statistical data analysis models for determining the relevance of structural image descriptions, IEEE Access, vol. 11, pp. 126938-126949.

34. Gorokhovatskyi V., Tvoroshenko I., and Yakovleva O. (2024) Transforming image descriptions as a set of descriptors to construct classification features, Indonesian Journal of Electrical Engineering and Computer Science, vol. 33, no. 1, pp. 113-125.

35. Gorokhovatskyi V., Tvoroshenko I. (2023) Identification of visual objects by the search request. International scientific symposium «INTELLIGENT SOLUTIONS-S». Computational intelligence (results, problems and perspectives). Decision making theory: proceedings of the international symposium, September 28, 2023, Kyiv-Uzhorod, Ukraine, pp. 25-27.

36. Gorokhovatskyi V., Tvoroshenko I., Yakovleva O., Hudáková M., and Gorokhovatskyi O. (2024) Application a committee of Kohonen neural networks to training of image classifier based on description of descriptors set, IEEE Access, vol. 12, pp. 73376-73385.

37. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., and Zeghid M. (2022) Cluster representation of the structural description of images for effective classification, Computers, Materials & Continua, 73(3), pp. 6069-6084.

38. Yakovleva O., Matúšová S., Tvoroshenko I., and Isaiev Y. (2024) Visitor counting based on video stream analysis from surveillance cameras to solve various business problems, Verejná správa a regionálny rozvoj ekonómia, manažment a marketing, XX(1), pp. 67-87.

39. Авлякулов Т.Е. Аналіз деяких видів тестувань бекенд застосунків. *Радіоелектроніка та молодь у XXI столітті: тези доповідей 28-го Міжнародного молодіжного форуму (Харків, 16–18 квітня 2024 р.). Харків: ХНУРЕ*, 2024. Т. 7. С. 11-12.

40. Choi, B., Park, J., Lee, C., & Han, D. (2021, June). pHPA: A proactive autoscaling framework for microservice chain. In Proceedings of the 5th Asia-Pacific Workshop on Networking (pp. 65-71).

41. Li, S., Zhang, H., Jia, Z., Zhong, C., Zhang, C., Shan, Z., ... & Babar, M. A. (2021). Understanding and addressing quality attributes of microservices architecture: A Systematic literature review. Information and software technology, 131, 106449.