Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів навантажувального тестування для перевірки швидкості запитів до БД

Виконав:
студент (ка) 2 курсу, групи ІПЗм-22-5

Шуміла А.І.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-наукова

Керівник доц. Бабій А.С
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___»_____20__ р.

ЗАВДАННЯ
НА КВАЛІФАКАЦІЙНУ РОБОТУ

студентові Шумілі Андрію Івановичу

1. Тема роботи *«Дослідження методів навантажувального тестування для перевірки швидкості запитів до БД»*

затверджена наказом університету від 29 березня 2024 р. № 250Ст

2. Термін подання студентом роботи до екзаменаційної комісії 19 червня 2024 р.

3. Вихідні дані до роботи *дослідження підходів та методів навантаженого тестування (load Testing) для перевірки швидкості бд та запитів до неї.*

4. Перелік питань, що потрібно опрацювати у роботі *вступ, аналіз предметної галузі, дослідження існуючих алгоритмів прогнозування, підготовка даних та встановлення метрик, проведення експериментального дослідження, проектування архітектури ансамблю моделей, тренування моделей, розробка програмної системи, висновки.*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі та постановка задачі	22.01 – 14.02.24	*виконано*
2	Аналіз та вибір методик для дослідження	15.02 – 24.02.24	*виконано*
3	Аналіз та моделювання предметної області	17.02 – 28.02.24	*виконано*
4	Планування експериментів	25.02 – 28.02.24	*виконано*
5	Програмна реалізація кожного з обраних для дослідження методик	25.02 – 01.04.24	*виконано*
6	Експериментальні дослідження	02.04 – 20.04.24	*виконано*
7	Аналіз результатів експериментальних досліджень та розробка рекомендацій	20.04 – 23.04.24	*виконано*
8	Написання та оформлення статті та тез доповіді	17.04 – 23.04.24	*виконано*
9	Підготовка пояснювальної записки	01.04 – 26.04.24	*виконано*
10	Підготовка презентації та доповіді	26.04 – 02.05.24	*виконано*
11	Нормоконтроль	03.05 – 08.05.24	*виконано*
12	Рецензування	08.05 – 14.05.24	*виконано*
13	Занесення диплома в електронний архів	15.05.2024	*виконано*
14	Попередній захист	15.05.2024	*виконано*
15	Допуск до захисту у зав. кафедри	18.05.2024	*виконано*

Дата видачі завдання 22 січня 2024 р.

Студент _____ Шуміла А.І.
(підпис)

Керівник кваліфікаційної роботи_____ доц. Бабій А.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Робота містить: 66 с., 11 рис., 1 табл., 12 джер.

АНАЛІТИКА, НАВАНТАЖУВАЛЬНЕ ТЕСТУВАННЯ, НОРМАЛЬНИЙ РОЗПОДІЛ, LOAD TESTING.

Об’єктом дослідження є підхід до оцінки швидкодії запитів.

Метою роботи є дослідження предметної області для дослідження та вдосконалення підходів до навантажувального тестування.

В результаті роботи запропоновано найоптимальніше рішення для оцінки швидкодії запитів до баз даних.

ANALYTICS, LOAD TESTING, LOAD TESTING, NORMAL DISTRIBUTION.

The object of the study is an approach to evaluating the speed of requests.

The purpose of the work is to study the subject area for research and improvement of load testing approaches.

As a result, the most optimal solution for evaluating the speed of requests to databases is proposed.

Я, Шуміла Андрій Іванович, студент групи ІПЗм-22-5 здобувач вищої освіти на другому (магістерському)рівні кафедра програмної інженерії, заявляю: моя кваліфікаційна робота на тему Дослідження методів навантажувального тестування для перевірки швидкості запитів до БД, що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному

плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

У сучасному світі програмування, де швидкість і надійність систем є ключовими для задоволення потреб користувачів, навантажувальне тестування стає невід'ємною частиною розробки програмних застосунків.

Цей процес не тільки допомагає забезпечити оптимальну продуктивність та ефективність баз даних і обробки запитів але й виявляє потенційні проблеми до їх виникнення у реальних умовах.

Вступаючи в еру великих даних і складних обчислювальних завдань, розуміння і впровадження ефективних методів навантажувального тестування стає вирішальним для успіху будь-якої сучасної інформаційної системи.

Відповідно у розробників зростає необхідність організовувати безперебійну, швидку та стабільну комунікацію із сервером та базою даних та клієнтами, але не завжди можна зрозуміти причину затримок запитів у різних користувачів, особливо в дуже розгалуженій архітектурі і тоді необхідно провести ряд навантажувальних тестувань для розуміння природи затримок.

Навантажувальне тестування, або Load Testing, надає змогу не тільки оцінити здатність системи витримувати великі обсяги даних і запитів, але й гарантувати, що ці системи зможуть ефективно функціонувати під час критичних навантажень.

Таким чином, ми розуміємо що для веб-застосунків та програмних систем цей процес є ключовим для забезпечення високого рівня користувацького досвіду та задоволення потреб сучасного ринку програмного забезпечення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Загальна характеристика навантажувального тестування

На початку епохи інформаційних технологій, використання великих мейнфрейм-комп'ютерів було звичайною практикою, комп'ютери були дорогими і важливими для бізнес-операцій, тому забезпечення їхньої здатності справлятися з очікуваними навантаженнями було критично важливим.

Спочатку навантажувальне тестування було досить простим, воно мало за мету перевірку здатності системи працювати під трохи вищим, ніж звичайно, навантаженням.

З розвитком персональних комп'ютерів та пізніше з появою інтернету потреба в більш складному навантажувальному тестуванні стала очевидною. Епоха інтернету, зокрема, відіграла важливу роль у цьому процесі. Веб-сайти та онлайн-платформи почали обслуговувати глобальну аудиторію, що призвело до викликів у забезпеченні стабільної роботи під час непередбачуваних та високих навантажень. Навантажувальне тестування розвивалось, щоб вирішувати ці виклики.

Основна мета вже не просто забезпечення здатності системи обробляти задане навантаження, а розуміння поведінки системи при різних умовах, особливо під часпікових навантажень або стресових ситуацій.

Ця зміна була критично важливою в епоху, коли веб-сайт міг мати нормальний трафік в один проміжок часу, а потім витримати масивні навантаження у інший проміжок, часто під впливом вірусного контенту, рекламних акцій або новин.

Також цей час позначився на розвитку електронної комерції, онлайн-ігор та різних інших сервісів, які вимагали не лише постійної роботи, а й стабільної продуктивності під час різних навантажень. Недоступність або погана продуктивність могли призвести до значних фінансових втрат та пошкодження репутації [1].

З розвитком інтернету та складності програмних систем методи та інструменти для навантажувального тестування також еволюціонували. Спочатку навантажувальне тестування могло включати в себе прості скрипти для імітації дій

користувачів.

Однак, по мірі зростання складності взаємодій користувачів та архітектури систем, інструменти та методики тестування стали більш складними.

Сучасне навантажувальне тестування включає в себе складні симуляції за допомогою спеціалізованого програмного забезпечення. Ці інструменти можуть імітувати дії тисяч або навіть мільйонів користувачів, які взаємодіють із системою різними способами. Ця симуляція є ключовою для розуміння не тільки того, як система справляється з навантаженням, але й того, як вона відновлюється після пікового використання, як масштабується та як різні компоненти системи взаємодіють під час стресу.

Однією з унікальних особливостей цього тестування є потреба у глибокому розумінні як самої системи, так і потенційних шаблонів її використання.

Тестувальникам потрібно передбачати безліч способів, якими користувачі можуть взаємодіяти з системою, різні шляхи, якими вони можуть піти, та непередбачувані способи, якими вони можуть вивести систему з ладу.

Особливо коли навантажувальне тестування не обмежується лише програмним забезпеченням. Воно також охоплює базову інфраструктуру - сервери, бази даних, мережі. Усі ці компоненти відіграють важливу роль у тому, як система працює під час навантаження, тому вони всі аналізуються під час навантажувального тестування.

В майбутньому можна очікувати, що цей підхід буде продовжувати еволюціонувати, особливо з огляду на розвиток штучного інтелекту та машинного навчання. Ці технології можуть допомогти автоматизувати та оптимізувати процеси тестування, роблячи їх більш точними та ефективними.

Підводячи підсумок, навантажувальне тестування залишається ключовою частиною розробки програмного забезпечення, гарантуючи, що продукти не тільки виконують свої функції, але й витримують реальні умови використання. Це невід'ємна частина забезпечення якості, яка допомагає компаніям та сервісам уникнути втрати доходів, репутації та довіри користувачів через проблеми з продуктивністю.

Культура навантажувального тестування в IT-індустрії є фундаментальною складовою процесу розробки та підтримки стабільних та ефективних систем. Це дозволяє компаніям не тільки визначити продуктивність програмного забезпечення під час пікових навантажень, але й гарантувати, що нові оновлення або системні зміни не призведуть до непередбачених збоїв чи деградації послуг.

Навантажувальне тестування передбачає симуляцію реальних або гіпотетичних умов для оцінки, як система впорається з великою кількістю запитів. Це охоплює не тільки вимірювання швидкості відповіді серверів, але й здатність інфраструктури витримувати високе навантаження [2]. Основна мета полягає у забезпеченні того, щоб системи були оптимізовані та стійкі до високих навантажень, які можуть виникнути під час критичних операційних періодів, таких як онлайн-розпродажі, запуски продуктів чи інші ключові бізнес-події.

Культура навантажувального тестування включає ряд кращих практик. Планування та аналіз. Ефективне навантажувальне тестування починається з детального планування, що включає визначення ключових показників продуктивності (KPIs), які необхідно виміряти, та сценаріїв тестування, які імітують реальне використання системи.

Використання автоматизованих інструментів. Автоматизація тестування дозволяє забезпечити постійність та повторюваність процесу, а також збільшує швидкість отримання результатів.

Моніторинг та оптимізація. Регулярний моніторинг системи під час тестування дає можливість швидко виявляти проблеми і вживати заходів для їх усунення.

Документування та аналіз результатів. Збір та аналіз даних після тестування критично важливі для розуміння поведінки системи і впливу змін у конфігурації чи коді.

Навантажувальне тестування не просто виявляє слабкі сторони чи вузькі місця в системі, але й допомагає застосування змін, які можуть поліпшити загальну продуктивність та стійкість. Це забезпечує, що будь-які нововведення чиоовлення системи можуть бути впроваджені з мінімальним ризиком для бізнес- операцій, а

також допомагає у підготовці системи до реальних навантажень, які можуть виникнути під час роботи. Таким чином, культура навантажувального тестування стає ключовою частиною процесу розробки і підтримки високопродуктивних та надійних ІТ-систем, гарантуючи їх готовність до будь-якихвикликів майбутнього.

Тестування швидкості роботи запитів є критично важливим в багатьох аспектах розробки та експлуатації систем. Ось основні сценарії, коли таке тестування стаєнеобхідним:

- запуск нових продуктів або функцій: перед впровадженням нових додатків або оновлень до існуючих систем, важливо переконатися, що запити обробляються ефективно і не викликають неочікуване зниження продуктивності;
- оптимізація продуктивності: якщо існуючі системи демонструють повільну відповідь або не відповідають очікуванням користувачів, систематичне тестування швидкості обробки запитів може допомогти ідентифікувати проблемні аспекти та оптимізувати їх [3];
- скалинг або масштабування системи: перед тим як збільшувати кількість користувачів або обсяг оброблюваних даних, важливо зрозуміти, як зміна навантаження вплине на швидкість запитів. тестування допомагає гарантувати, що система зможе витримати збільшення навантаження;
- гарантія високої доступності та надійності: у критичних системах, таких як фінансові, медичні або екстрені служби, де високий час реакції може призвести до серйозних наслідків, регулярне тестування і моніторинг швидкості обробки запитів є обов'язковим;
- відповідність стандартам продуктивності та регулятивним вимогам: деякі індустрії мають конкретні вимоги до швидкості системних відповідей. тестування швидкості дозволяє перевірити, чи відповідають системи цим стандартам;
- планування інфраструктурних оновлень: аналіз швидкості відповіді запитів може виявити потребу в модернізації апаратного забезпечення, оновленні серверів або оптимізації баз даних, що в свою чергу допоможе

планувати бюджет і ресурси.

Ці сценарії відіграють ключову роль у підтриманні і покращенні якості іт-сервісів, а також у забезпеченні високого рівня задоволення користувачів і бізнесу загалом.

1.2 Актуальність проблеми

На сьогоднішній день, коли обчислювальна потужність і обсяги даних набувають все більшого значення в цифровому світі, вдосконалення методів навантажувального тестування стає вкрай необхідною майже для кожної компанії або продукту.

Аналізуючи запити користувачів, можна вивести певні ключові проблеми в цій області, користувачі сьогодні очікують надзвичайної швидкодії та ефективності в інтернет-сервісах та застосунках.

Майже усі користувачі дуже чутливо ставляться до швидкодії того чи іншого сервісу. Відсутність високої швидкості може призвести до втрати користувачів та негативно вплинути на репутацію компанії або сервісу.

Далі варто відмітити що зараз та останніми роками спостерігається зріст обсягів даних та трафіку. Сучасні компанії обробляють величезні обсяги інформації та взаємодіють з великою кількістю користувачів. Виконання запитів швидко та ефективно стає складнішим завданням, коли мова йде про масштабність. Зростання складності програмних рішень та впровадження нових технологій, таких як хмарні обчислення, штучний інтелект, розподілені системи тощо, також створює великий попит на тестування швидкості і навантажувальне тестування.

Важливо визначити, як ці нові технології впливають на продуктивність та забезпечити їхню оптимальну роботу.

Змінність завдань та навантажень викликає потребу у тестуванні для різних сценаріїв. Система повинна вміти ефективно реагувати на зміни навантаження та виконувати різноманітні завдання. Тестування швидкості дозволяє визначити, наскільки добре система впорається з цими змінами [4].

Усе це свідчить про те, що дослідження та вдосконалення методів

навантажувального тестування є актуальним завданням, яке допоможе забезпечити надійну та швидку роботу сучасних інформаційних систем та додатків.

1.3 Проблема визначення швидкодії запитів

Далі нам треба розглянути ключовий елемент навантаженого тестування, а саме – вимірювання швидкодії запитів до бази даних.

Повільні запити до бази даних – це проблема, яка може серйозно впливати на продуктивність і ефективність системи. Вони можуть виникати з різних причин і сигналізувати про можливі недоліки в дизайні бази даних, неоптимальних запитах або проблемах із налаштуванням сервера баз даних. Розглянемо детальніше цю проблему.

Повільні запити можуть бути викликані неоптимальними операціями в базі даних, неправильними індексами або недостатньою оптимізацією SQL-запитів. Коли дані великі, або таблиці мають складні структури, неоптимізовані запити можуть призводити до значного збільшення часу виконання.

Проблема повільних запитів може виникнути із зростанням обсягу даних в базі або зі змінами в структурі даних. Наприклад, додавання нових колонок без належного індексування може призвести до погіршення продуктивності.

Також, слід звертати увагу на недооптимізовані JOIN-операції, які можуть витрачати значний час, особливо при об'єднанні великих таблиць. Написання запитів, які використовують "SELECT *", також може призводити до зайвогообсягу даних, що передається від сервера до додатку.

Додатковими факторами можуть бути неоптимальні налаштування сервера баз даних, відсутність необхідних індексів або відсутність кешування результатів запитів.

У випадку великих та розподілених систем також може виникнути проблема мережевої затримки.

Щоб вирішити проблеми повільних запитів, слід провести аналіз SQL-запитів та їх виконання, використовуючи та модифікуючи інструменти для профілювання, моніторингу та швидкісних метрик для аналізу запитів.

1.3.1 Актуальність роботи із .Net платформою

Розробка інструменту для навантажувального тестування на платформі .NET є актуальною з кількох причин, які випливають з унікальних переваг і можливостей цієї платформи. У світі програмного забезпечення .NET здобула широке визнання завдяки своїй гнучкості, продуктивності та підтримці різних типів додатків, від веб-додатків до хмарних рішень і мобільних додатків.

По-перше, платформа .NET відома своєю продуктивністю і ефективністю. Вона забезпечує високу швидкість виконання коду, що є критичним для тестування продуктивності додатків. Завдяки підтримці компіляції Just-In-Time (JIT) та передчасної компіляції (Ahead-Of-Time, AOT), .NET дозволяє оптимізувати виконання додатків для досягнення максимальної швидкості. Це важливо для навантажувального тестування, оскільки дозволяє точно моделювати реальні умови роботи системи і отримувати достовірні результати.

По-друге, .NET забезпечує високу масштабованість. Це дозволяє створювати тестові сценарії, які можуть симулювати велику кількість одночасних користувачів і запитів до системи. Такі сценарії важливі для визначення здатності системи витримувати пікові навантаження. Багатоплатформенна підтримка .NET (Windows, Linux, MacOS) робить цей інструмент доступним для широкого кола користувачів і дозволяє використовувати його в різних середовищах розробки.

Крім того, .NET має розвинену екосистему бібліотек і інструментів, що спрощує розробку складних тестових сценаріїв і обробку результатів тестування. Завдяки великій кількості доступних бібліотек можна легко розширювати функціонал інструменту, додаючи нові можливості для аналізу даних та звітності. Інтероперабельність .NET також є значною перевагою. Платформа підтримує роботу з різними базами даних, такими як SQL Server, Postgres, MySQL, MariaDB, CosmosDB, MongoDB та Redis. Це забезпечує універсальність інструменту та дозволяє тестувати продуктивність різних систем з однаковими критеріями, що спрощує порівняння та вибір найкращого рішення для конкретного завдання.

Ще однією важливою перевагою є підтримка сучасних стандартів безпеки та надійності. Завдяки регулярним оновленням та вдосконаленням, .NET забезпечує

високу безпеку додатків, що є критичним для тестування в умовах реального світу, де безпека даних є пріоритетом.

Нарешті, варто відзначити, що на ринку існує обмежена кількість інструментів для навантажувального тестування на платформі .NET, які мають широкий функціонал. Наприклад, проект BenchmarkDotNet, хоча і є потужним інструментом для мікробенчмаркінгу, не надає всього необхідного функціоналу для комплексного навантажувального тестування. Розробка нового інструменту з підтримкою багатоплатформенності, різних типів баз даних, розширеного аналізу даних і інтеграції з сучасними CI/CD системами заповнить цю прогалину і надасть розробникам потужний засіб для забезпечення високої продуктивності та надійності своїх систем.

Таким чином, розробка інструменту для навантажувального тестування на платформі .NET є актуальною і необхідною для забезпечення високої продуктивності, стабільності та надійності сучасних програмних систем.

1.4 Постановка задачі

У ході вивчення предметної області та аналізу наявних підходів та методів аналізу результатів навантажувального тестування, будуть сформульовані конкретні вимоги до дослідження. Основний акцент роботи буде зосереджений на виявленні оптимальних технік виміру швидкості запитів для перевірки швидкості реакції баз даних та ефективності опрацювання складних запитів.

Вивчення наявних методів вимірювання швидкодії API та їхнього впливу на продуктивність системи. Оцінка сучасних технологій тестування для визначення їхньої придатності до вирішення конкретних завдань. Проведення тестів для виведення результатів навантажувального тестування. Створення програмної системи для візуалізації результатів тестування та надання можливості демонстрації найефективніших шляхів та підходів до виміру швидкості.

Ця робота спрямована на розкриття та оптимізацію методів тестування ефективності запитів. Результати дослідження допоможуть впроваджувати ефективні та оптимальні методи у реальних проектах, покращуючи при цьому

швидкодію та надійність API.

Реалізація наукового дослідження складається з наступних етапів:

- проведення аналізу існуючих рішень;
- виявлення недоліків існуючих підходів;
- модифікація та додавання більш детальних метрик;
- розробка платформи для візуального виведення результатів тестування;
- аналіз результатів.

2 АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ТА МЕТОДІВ

2.1 Аналіз особливостей навантажувального тестування

Load Testing – це ключовий етап в процесі тестування програмного забезпечення, спрямований на оцінку та визначення продуктивності системи, програми або сервісу під реальним або припущеним завантаженням. Цей вид тестування дозволяє визначити, як система реагує на різні рівні навантаження, включаючи пікові навантаження та періоди великої активності користувачів.

Аналітичний підхід цього тестування допомагає визначити максимальну кількість одночасних користувачів або запитів, яку система може обробити без втрати продуктивності. Це важливо для гарантії того, що система здатна обслуговувати очікуване навантаження в реальних умовах експлуатації.

Цей підхід допомагає ідентифікувати елементи системи, які можуть стати вузьким місцем під великим навантаженням. Це може включати бази даних, сервери, мережеві ресурси або інші компоненти, які можуть впливати на продуктивність системи.

Під час тестування імітується реальна діяльність користувачів для оцінки того, наскільки надійно система працює в реальних умовах.

Важливо визначити, як система може відновлюватися після перевищення меж її максимальної пропускної здатності. Таке тестування допомагає визначити час відновлення та робить прогнози стосовно деградації продуктивності.

На рисунку 2.1, можна побачити архітектурну структуру процесу навантажувального тестування.

На діаграмі ми можемо побачити, як різні користувачі отримують доступ допрограми. Тут кожен користувач робить запит через інтернет, який пізніше пропускається через брандмауер.

Після брандмауера у нас є балансувальник навантаження, який розподіляє навантаження на будь-який із веб-серверів, а потім передає на сервер додатків, а потім на сервер бази даних, де отримує необхідну інформацію на основі запиту користувача.

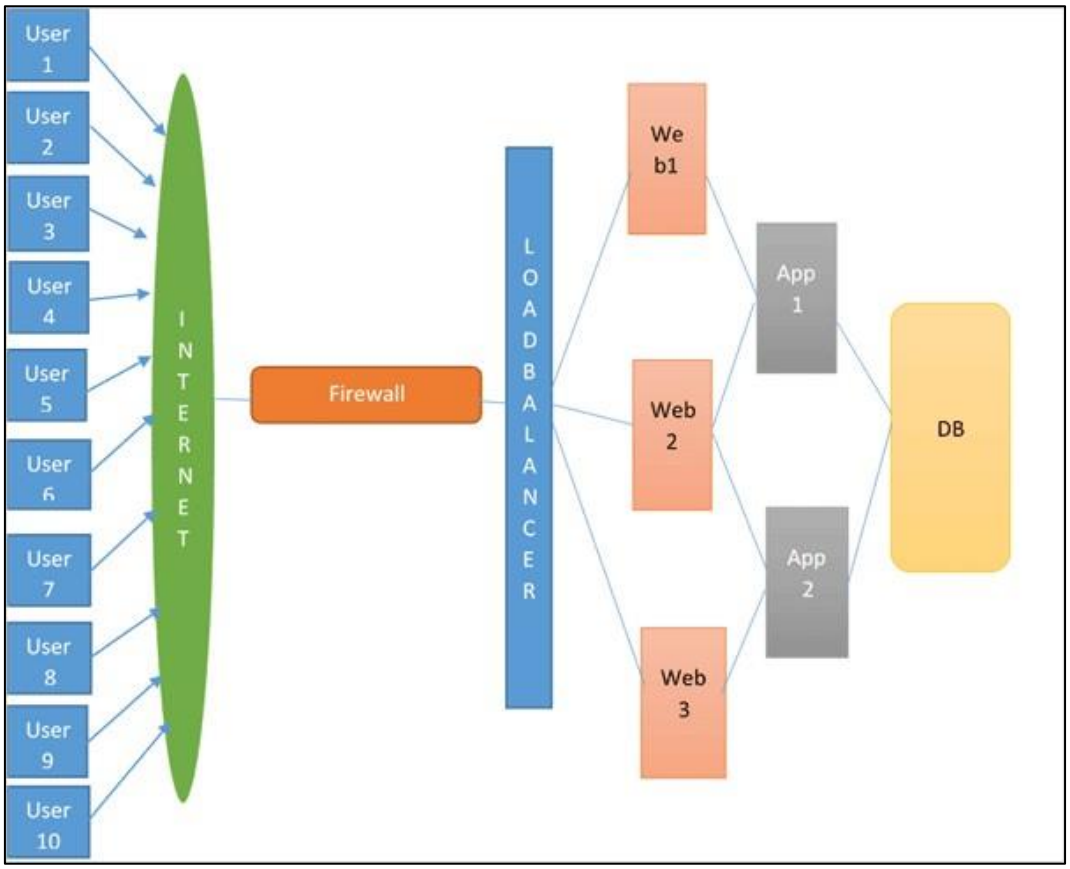

Рисунок 2.1 – Архітектура процесу навантажувального тестування

Процес навантажувального тестування:

- планування: розпочинається планування тестового навантаження, включаючи обрання сценаріїв використання, визначення очікуваного обсягу користувачів та вибір інструментів для тестування;
- створення сценаріїв: розробляються сценарії, які відображають реальні умови використання системи. Це може включати в себе реєстрацію, авторизацію, навігацію та інші дії користувачів;
- конфігурація тестового середовища: підготовлене тестове середовище, аналогічне реальному, де тестуються фактичні архітектурні компоненти;
- виконання тестів: сценарії виконуються в умовах навантаження, а потім результати аналізуються для визначення продуктивності та ідентифікації можливих проблем;
- аналіз та оптимізація: після виконання тестів проводиться аналіз результатів. Виявлені проблеми можуть бути виправлені шляхом

оптимізації коду, налаштування інфраструктури або вдосконалення архітектури систем.

Розглянемо модель навантаженого тестування на прикладі магазину одягу.

Припустимо, ми використовуємо програму для онлайн-покупок, де користувачі входять в систему та знаходять великий вибір суконь.

Переходячи між різними товарами, вони можуть переглядати деталі кожного продукту.

Для отримання докладної інформації про конкретний товар їм необхідно лише натискати на відповідний елемент. Якщо вартість та бренд продукту їм підходять, вони можуть додати його в кошик та здійснити покупку, перевіривши та виконавши оплату.

Кожна дія відповідає певному запиту, та запити виконуються певну кількість раз, тому на рисунку 2.2, наведено модель яка демонструє результати навантаженого тестування запитів.

S.No	Бізнес-потік	Кількість транзакцій	Віртуальне завантаження користувача	Час відгуку (с)	Дозволений % відмов	Операцій за годину
1	переглядати	17	1600	3	Менше 2%	96000
2	Огляд, перегляд продукту, додавання в кошик	17	200	3	Менше 2%	12000
3	Огляд, перегляд продукту, додавання в кошик і перевірка	18	120	3	Менше 2%	7200
4	Огляд, перегляд товару, додавання в кошик Оформлення та здійснення оплати	20	80	3	Менше 2%	4800

Рисунок 2.2 – Модель результатів тестування

Навантажувальне тестування є необхідним етапом у впровадженні стійкої та продуктивної програмної системи. Воно дозволяє розглядати систему в умовах

реального навантаження, забезпечуючи надійність та ефективність в її роботі.

Правильно проведене тестування є гарантією того, що програмне забезпечення буде здатне витримувати навантаження в реальних умовах використання, також розробники будуть знати “виключні” та слабкі місця системи.

2.1.1 Порівняння мануального перегляду та навантажувальних результатів

Дуже важливо розібрати чому не підійде просто переглядати результати запитів, їх швидкість роботи мануальним способом в середі розробки проекту.

Навантажувальне тестування проти перегляду окремих запитів:

- обсяг даних: перегляд індивідуальної швидкості кожного запиту може бути ефективним у системах з дуже малим обсягом даних. однак, у великих системах, де обробляються мільйони запитів на день, такий підхід стає непрактичним. статистичний аналіз дозволяє обробляти великі масиви даних, виділяючи загальні тенденції та виявляючи аномалії без необхідності ручного перегляду кожного запиту;
- репрезентативність даних: навантажувальне тестування дозволяє оцінити, як система поводиться під тиском різних сценаріїв, які можуть не виникати під час звичайної роботи. це включає піки навантаження, поступовезбільшення користувацької активності або одночасний доступ багатьох користувачів;
- виявлення вузьких місць та оптимізація: статистичні методи дозволяють не тільки знайти повільні запити, а й визначити їхній вплив на загальну продуктивність системи. це важливо для визначення пріоритетів оптимізації. наприклад, зниження часу відповіді на 10% для запитів, які становлять 80% загального навантаження, може мати набагато більший вплив, ніж оптимізація рідкісних, але дуже повільних запитів;
- прогнозування майбутньої поведінки: статистичні аналізи дозволяють моделювати, як може змінюватися продуктивність системи зі зміною обсягів даних або користувацької поведінки. це допомагає планувати масштабування системи та здійснювати превентивні заходи, замість

реагування на проблемипісля їх виникнення.

Суть навантажувального тестування полягає не просто у визначенні швидкості відповідей, а у здатності системи витримувати різні типи навантаження та виявлення її меж продуктивності. Ось кілька ключових аспектів, які роблять навантажувальне тестування критично важливим:

- симуляція реальних умов: навантажувальне тестування дозволяє симулювати реальні умови роботи системи, включаючи пікові навантаження, які можуть бути важко передбачити або відтворити за допомогою простого перегляду запитів;
- виявлення та розуміння аномалій: статистичні методи допомагають виявити аномалії, які можуть бути втрачені при простому перегляді індивідуальних запитів. це можуть бути рідкісні події, які викликають значне зниження продуктивності, але які важко помітити без аналізу великої кількості даних;
- планування масштабування: на основі результатів навантажувального тестування можна робити обґрунтовані рішення про необхідність додавання нових ресурсів або оптимізації існуючих для забезпечення стабільної роботисистеми при зростанні навантаження;
- підтримка стабільності та надійності: розуміння, як система поводитьсяпід великим навантаженням, допомагає вжити заходів для запобігання збоїв та підтримання високого рівня надійності та доступності;
- оптимізація запитів: ідентифікувавши найповільніші запити, ви можете сфокусувати зусилля на їх оптимізації, що значно покращить загальну продуктивність системи;
- масштабування інфраструктури: результати навантажувального тестування показали, що під час пікових навантажень система не встигає обробляти всі запити вчасно. це може бути сигналом для масштабування інфраструктури (наприклад, додавання нових серверів або збільшення ресурсів).
- У результаті можна відмітити що навантажувальне тестування та

використання статистичних метрик є критично важливими для забезпечення стабільної та ефективної роботи системи.

Вони допомагають не лише виявити повільні запити, а й зрозуміти загальні тенденції продуктивності, ідентифікувати вузькі місця та прийняти обґрунтовані рішення щодо оптимізації та масштабування системи.

2.2 Методи навантажувального тестування

2.2.1 Метод середнього значення (Average response time)

Метод середнього часу виконання запиту є важливим інструментом в області навантажувального тестування, спрямованим на вимірювання та аналіз продуктивності системи під час великої кількості одночасних запитів. Цей метод дозволяє оцінити, наскільки швидко система реагує на навантаження та визначити середній час, необхідний для обробки окремого запиту.

Основною метою навантажувального тестування є визначення меж системи, її стійкості та здатності долати навантаження в умовах реального використання. Метод середнього часу виконання запиту стає ключовим елементом в цьому процесі, оскільки він дозволяє отримати кілька важливих метрик ефективності.

Процес вимірювання середнього часу виконання запиту починається з визначення набору тестових сценаріїв, які репрезентують типові дії користувачів чи інших системних агентів. Ці сценарії потім виконуються в умовах навантаження, щоб спостерігати за реакцією системи.

Одним з ключових кроків є генерація навантаження, яке може включати в себе одночасні запити, велику кількість користувачів або інші фактори, які можуть впливати на роботу системи.

Під час виконання тестових сценаріїв вимірюються часові показники для кожного окремого запиту [5]. Таким чином математична формула має наступний вигляд (див. ф. 2.1):

$$Z = \frac{\sum_{i=1}^{n} t_i}{n} \tag{2.1}$$

де t_i – це час виконання кожного окремого запиту;

n – кількість запитів.

Таким чином ми визначаємо середній вплив кожного запиту на систему.

Ця формула надає детальне розуміння того, як різні фактори впливають на загальний час виконання запитів, і допомагає розробникам і адміністраторам баз даних виявляти проблеми продуктивності та приймати обґрунтовані рішення щодо оптимізації системи.

2.2.2 Вимірювання медіанного часу

Медіана відгуку є важливим показником при вимірюванні продуктивності системи та використовується у сфері тестування на навантаження. Цей метод надає інформацію про центральний тренд часу відгуку і дозволяє врахувати розподіл значень серед часів виконання окремих запитів.

У тестуванні на навантаження час відгуку може бути дуже різноманітним через фактори, такі як коливання мережі, різні операційні умови або нестабільність ресурсів. У таких умовах середній час виконання може не завжди точно відображати реальні умови використання.

Медіана пропонує більш стабільний погляд на розподіл часу відгуку, оскільки вона враховує значення, що розташовані посередині послідовності часів виконання. Це означає, що 50% запитів виконуються швидше, ніж медіана, і 50% - повільніше.

У порівнянні з середнім часом відгуку, медіана менше схильна до впливу великих варіацій у часі виконання. Це дозволяє отримати більш реалістичну картину про те, як швидко користувачі отримують відповідь в середньому.

Також, медіана дозволяє виявити екстремальні випадки, які можуть впливати на користувачів, такі як запити з надмірно довгим часом відгуку. Це дозволяє команді розробників та тестувальників аналізувати та виправляти проблеми, пов'язані з часом відгуку, що мають значний вплив на користувачів.

Математична формула для обчислення медіани відгуку може бути виражена

наступним чином – нехай X – масив часів відгуку, впорядкований в порядку зростання. Якщо n – кількість елементів у масиві, тоді медіана M обчислюється як (див. ф. 2.2):

$$M = \begin{cases} X[\frac{n}{2}], & \text{якщо } n \text{ непарне} \\ \frac{X[\frac{n}{2}-1]+X[\frac{n}{2}]}{2}, & \text{якщо } n \text{ парне} \end{cases} \quad (2.2)$$

де *X[n/2]* – значення розташоване посередині впорядкованого масиву длянепарної кількості елементів [6];

$\frac{X[\frac{n}{2}-1]+X[\frac{n}{2}]}{2}$ – значення розташоване посередині впорядкованого масиву для парної кількості елементів.

2.2.3 Перцентиль 90-й/95-й/99-й в тестуванні на навантаження

Перцентиль – це статистичні метрики, які вказують на значення, нижче яких певний відсоток спостережень чи даних. У тестуванні на навантаження 90-й, 95-й та 99-й Перцентиль використовуються для аналізу часів відгуку системи під навантаженням, щоб отримати детальніше розуміння розподілу часу відгуку та виявити екстремальні випадки. Перцентиль 90-й – це значення, нижче якого потрапляє 90% усіх часів відгуку. Іншими словами, лише 10% запитів мають час відгуку, більший за значення 90-го перцентиль. Це дає змогу зрозуміти, як веде себе система для більшості користувачів. Перцентиль 95-й – по аналогії із минулим, це значення, нижче якого потрапляє 95% усіх часів відгуку. Тобто лише 5% запитів мають час відгуку, більший за значення 95-го. Використання 95-го дозволяє врахувати більш екстремальні сценарії, що можуть впливати на менший відсоток користувачів. Перцентиль 99-й – це значення, нижче якого потрапляє 99% усіх часів відгуку. Тобто лише 1% запитів мають час відгуку, більший за значення 99-го перцентилю. Використання 99-го перцентилью дозволяє виявити найбільш екстремальні сценарії, які можуть виникати для невеликого відсотка користувачів, але важливі для забезпечення надійності та ефективності системи.

Системи можуть мати природжену варіабельність у часах відгуку, і перцентилі допомагають виявити цю варіабельність та забезпечити детальний

огляд часових показників для різних відсотков користувачів. Розглядаючи значення, які вищі за середні та медіани, команди розробників можуть виявити та виправити найбільш екстремальні сценарії, що можуть виникати в реальних умовах експлуатації. Даний підхід дозволяє бізнесам та командам IT забезпечити відповідність SLA(Service Level Argument) та забезпечити, що велика більшість користувачів отримує задовільний час відгуку [7]. Використання 90-го, 95-го та 99-го перцентилі у тестуванні на навантаження є важливим елементом для забезпечення ефективності та надійності систем. Ці метрики дозволяють виявити екстремальні сценарії, враховуючи варіабельність часів відгуку для різних відсотків користувачів.

2.3 Порівняння методів навантажувального тестування

Переваги методу знаходження середнього:

- простота та зрозумілість. обчислення середнього значення – це простий інструктивний процес, легкий для розуміння;
- чутливість до кожного значення. враховує кожне значення вибірки, що може бути корисним для отримання загальної "середньої" характеристики.

Недоліки методу знаходження середнього:

- чутливість до викидів. великі викиди або аномалії вибірки можуть суттєво вплинути на середнє значення, зробивши його менш репрезентативним;
- неінваріантність до розподілу. час від часу, середнє може бути використане для характеристики розподілу, але воно не завжди точно відображає форму розподілу.

Переваги методу знаходження медіани:

- стійкість до викидів. медіана не чутлива до значень вибірки, які виходять за межі "типового" розподілу;
- підходить для несиметричних розподілів. ефективно використовуєтьсядля характеристики розподілів з суттєвими асиметріями.

Недоліки методу знаходження медіани:

- чутливість до пари великих значень. якщо вибірка складається з пари

великих значень, медіана може бути менш репрезентативною;

– складніше обчислення. обчислення медіани може бути складніше, особливо для великих наборів даних.

Обидва методи є корисними в різних контекстах. Середнє значення використовується там, де важлива "середнє" представлення даних, але воно вимагає обережності при наявності великих викидів. Медіана, з іншого боку, корисна для отримання "типового" значення, яке не піддається впливу великих викидів. Часто вони використовуються в поєднанні для більш повного розуміння розподілу даних.

Переваги методу перцентилів:

– врахування розподілу. перцентилі дозволяють отримати детальне представлення про розподіл даних, і не обмежуються лише центральною тенденцією;

– менше чутливий до викидів. в порівнянні з середнім значенням, відсутність обчислення середнього може робити метод менш чутливим до великихвикидів або аномалій.

Недоліки методу перцентилю: не завжди інформативний. деякі перцентиль можуть бути менш інформативними, особливо якщо дані дуже асиметричні чи мають велику концентрацію значень навколо одного піку.

Після проведеного аналізу отримали методи які так чи інакше знаходять компромісне значення швидкості запиту і намагаються звести його до зручного значення яке б визначало характер усього тесту, але цей спосіб працює не завжди, інколи потрібно визначити більш детально – чи є дуже повільні чи швидкізапити, та до якого запиту відповідають, чи можемо визначити тенденції груп запитів для більш детальної оцінки їх роботи та виявлення закономірностей, такожне підійде метод логування та сортування за повільними запитами адже об’ємиможуть бути дуже великі та з такого масиву неможливо оцінити загальну картину роботи запитів у застосунку.

3 ПРОЕКТУВАННЯ НОВИИЗНИ ТА МУЛЬТИМОДАЛЬНОГОПІДХОДУ НАВАНТАЖЕНОГО ТЕСТУВАННЯ

3.1 Імплементація нормального розподілу та його модифікація

Аналізуючи математичні методи та моделі по аналізу вибірки даних, неможливо оминути “Гаусову криву” (див. рис. 3.1)

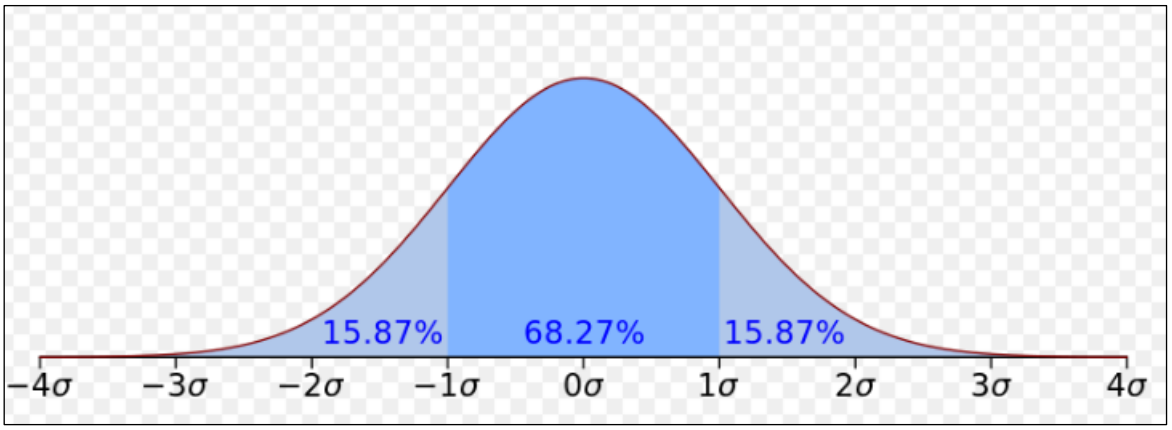

Рисунок 3.1 – Гаусова крива у розрізі Load testing [8]

Вважаючи, що мета дослідження полягає в глибокому розумінні динаміки швидкості виконання запитів до інтерфейсів бази даних, ідея використання нормального розподілу є логічним кроком у спробі отримати повніший обсяг інформації.

Основними перевагами методу нормального розподілу в цьому контексті є його здатність забезпечувати статистично обґрунтоване представлення часів відповіді, виявлення аномалій та моніторинг змін у системі.

Розуміння стандартного відхилення та середнього значення часів відповіді дозволяє отримати ясну картину того, наскільки різними можуть бути часи виконання різних запитів.Пригадуючи основи нормального розподілу, відомого як "крива Гаусса", розуміємо, що він ідеально підходить для виявлення різниці між часами відповіді, особливо коли спостерігається симетричний та купольний характер розподілу. Це важливо, оскільки ми хочемо знати не лише середнє значення, але й те, наскільки це значення може відхилятися.

Також важливою властивістю є можливість виявлення аномалій чи викидів в розподілі часів відповіді. Поєднуючи цю інформацію зі статистикою викидів,

можна легко визначити та вивчати незвичайні патерни, які можуть вказувати на проблеми в системі або неочікувані зміни.

Застосування методу нормального розподілу також дає змогу виконати прогнозування та моніторинг ефективності. Враховуючи динаміку розподілу в часі, можна спрогнозувати, як система поведе себе під впливом різних умов навантаження та оптимізації.

Також є можливість виявлення аномалій чи викидів в розподілі часів відповіді. Поєднуючи цю інформацію зі статистикою викидів, можна легко визначити та вивчати незвичайні патерни, які можуть вказувати на проблеми в системі або неочікувані зміни.

Застосування методу нормального розподілу також дає змогу виконати прогнозування та моніторинг ефективності. Враховуючи динаміку розподілу в часі, можна спрогнозувати, як система поведе себе під впливом різних умов навантаження та оптимізацій.

Виходячи із зазначеного – до нашого методу нормального розподілу варто додати метод міжкварального розмаху. Міжквартальний розмах (IQR) – це міра розмаху, яка визначається як різниця між верхнім (Q3) та нижнім (Q1) квартилями набору даних.

Цей показник дозволяє виявити різницю між центральними 50% значень в наборі даних, і він широко використовується для виявлення викидів та визначення ступеня варіабельності в даних.

Застосування IQR в нашій виборці, може бути вкрай корисним. IQR дозволяє визначити "типовий" діапазон значень, які лежать в межах між Q1 та Q3.

Значення, що виходять за межі цього діапазону, можуть вважатися викидами, що може бути корисно при виявленні аномалій у швидкодії окремих запитів чи функціональних блоків системи.

За допомогою IQR можна визначити, наскільки рівномірно розподілені значення швидкості у центральних 50%.

Це дозволяє оцінити ступінь стабільності та неперервності швидкісного режиму системи.

IQR може бути використаний для визначення асиметрії розподілу швидкості, але важливо також розглядати відхилення від середнього за допомогою методу нормального розподілу.

Загальна ідея полягає в тому, що IQR дозволяє визначити та відокремити центральні 50% значень у наборі даних, що може бути корисним при виявленні аномалій та змін в швидкодії системи. Поєднання цього методу з методом нормального розподілу надає більш комплексний підхід для розуміння та оптимізації швидкодії запитів.

Хоча розглянуті метод є доволі самостійними, але необхідно бачити загальний вигляд розподілення та метрик швидкостей та усі параметри часу нашої вибірки, тому варто застосувати мультимодальний підхід який буде реалізовано програмно а візуальна складова буде відображена у консольному режимі, також варто відобразити криву розподілу по у консольному вигляді, згодом застосувати бібліотеку для відображення кривої.

3.2 Проектування мультимодального підходу

Трансформуючи наш метод розподілу із квартильним розподілом – варто також спроектувати загальну структуру мультимодального підходу а згодом розглянути реалізацію. На рисунку 3.2 зображено візуальна імплементація мультимодального підходу.

На рисунку вісь *x* – представлена часом запиту, а *y* – представлена кількістю запитів.

На рисунку представлено мультимодальний підхід до аналізу розподілу часу виконання запитів, який включає різні метрики для глибокого розуміння продуктивності системи [9].

Медіана – це центральне значення у впорядкованому списку часів відповіді. На графіку медіана відображена вертикальною зеленою лінією, яка показує точку, в якій 50% запитів виконуються швидше, а 50% – повільніше. Це значення корисне для розуміння типового часу відповіді, менш чутливого до екстремальних значень порівняно із середнім (див.рис.3.2).

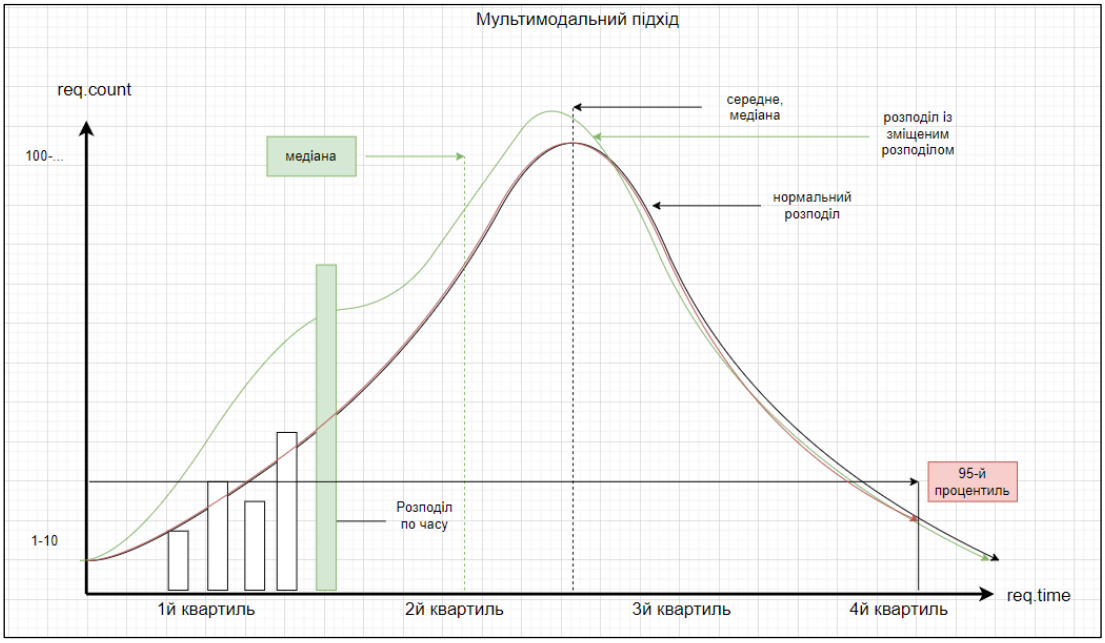

Рисунок 3.2 – Мультимодальний метод вимірювання

Нормальний розподіл показаний червоною кривою, що має дзвоноподібну форму. Нормальний розподіл використовується для відображення розподілу значень часів відповіді, де більшість значень згруповані навколо середнього. Він допомагає виявити типову продуктивність системи та зрозуміти варіацію значень.

Середнє показане як точка на осі X, де зосереджена більшість значень. Середнє значення є арифметичною серединою всіх часів відповіді. Хоча це корисна метрика для загального уявлення, вона може бути спотворена викидами.

Наприклад, 95-й перцентиль, показаний на графіку праворуч. Перцентиль показує, що певний відсоток запитів (в даному випадку 95%)виконується швидше за це значення, а решта 5% – повільніше. Це корисно для ідентифікації найповільніших запитів і оптимізації продуктивності.

Міжквартильний розмах показаний як діапазон між першими (Q1) і третіми (Q3) квартилями. Він відображає розподіл центральних 50% значень і дозволяє виявляти аномалії. Значення за межами IQR можуть бути викидами, що вказує на потенційні проблеми.

Розподіл по часу – це гістограма, яка показує кількість запитів (вісь Y) за певні інтервали часу відповіді (вісь X). Вона допомагає візуально оцінити, як часто зустрічаються різні значення часу відповіді.

На графіку також показаний "розподіл із зміщеним розподілом", який демонструє, як може змінюватися розподіл часу відповіді в залежності від умов навантаження.

Це зображення ілюструє, як різні статистичні метрики допомагають комплексно оцінити продуктивність системи, виявляти вузькі місця та аномалії, що дозволяє приймати обґрунтовані рішення щодо оптимізації [10].

3.3 Проектування значення кожної метрики у мультимодальності

Далі дуже важливо встановити назначення кожної метрики, ми вжерозбирали загальне назначення кожної метрики в попередніх розділах, але це булозагальний, математичний та доволі абстрактне визначення.

Адаптуючи ці метрики та поєднуючи їх у один підхід варто чітко розмежувати задачі кожної із них, отже у таблиці 3.1 буде проаналізовано користь конкретної метрики для певної системи.

Також у таблиці наведено більш практичне застосування методів у конкретному сценарії використання.

Таблиця 3.1 – Прикладна користь методів

Назва методу	Практична користь	Сценарій використання
Середнє (Average)	Середнє значення часу відповіді дозволяє оцінити загальну продуктивність системи та зрозуміти основний тренд у часах відповіді.	Середнє найкраще використовувати, коли дані мають нормальний розподіл без великих викидів. Це допомагає у визначенні базової лінії продуктивності для стандартних умов. Проте, якщо викиди присутні, середнє може бути спотвореним.

Продовження таблиці 3.1

Медіана (Median)	Медіана вказує на те, де знаходиться центральне значення в даних, і є більш стійкою метрикою до викидів порівняно з середнім.	Ідеально підходить для визначення "типового" досвіду користувача, особливо коли дистрибуція даних нерівномірна. Наприклад, у випадках, коли додаток має інтенсивні піки обробки, які спотворюють середнє значення.
Перцентиль (Percentiles)	Перцентиль, особливо 95-й та 99-й, допомагають зрозуміти якість обслуговування для всіх користувачів, включаючи "найгірші" випадки.	Важливі для систем, де важливо гарантувати високу продуктивність для великої частини користувачів, наприклад, в електронній комерції чи фінансових послугах, де підтримання продуктивності під час пікових навантажень критично важливо.
Мінімальне та максимальне значення (Min/Max)	Ці метрики показують крайні межі часів відповіді, виявляючи найкращий і найгірший випадки.	Використовуються для ідентифікації потенційних проблем у продуктивності, які можуть вимагати негайного втручання, особливо при тестуванні нових оновлень або змін у системі. Максимальне значення може вказувати на надмірне навантаження чи неефективність, а мінімальне демонструє найкращий час відповіді.

Кінець таблиці 3.1

Стандартне відхилення (Standard Deviation)	Показує розкид часів відповіді від середнього значення, допомагаючи зрозуміти, наскільки значення згруповані або розкидані.	Корисне для систем, які піддаються частим змінам або оновленням. Велике стандартне відхилення може вказувати на непостійність в системі, що вимагає додаткового аналізу і потенційних оптимізацій.
Інтерквартильний розмах (IQR)	Визначає діапазон між 25-м (першим квартилем) і 75-м (третім квартилем) перцентилями, дозволяючи виявити основний діапазон даних та ідентифікувати викиди.	Цей показник корисний для аналізу даних з великою кількістю викидів і для систем, де важлива стабільність. У критично важливих застосунках IQR допомагає зосередитися на "типовому" діапазоні даних, виключаючи аномалії.
Розподіл (крива Гауса)	Дозволяє візуально оцінити характер сітки запитів та знайти патерни	Різні види розподілу можуть вказувати на якісь сценарії які викликають різні піки на розподілі, або в залежності від сценарію використання – крива буде зміщуватись, наприклад це може бути використання кешу або звертання до спіцефічних сервісів із певними обмеженнями

Кожна із цих метрик має свої переваги і може бути особливо корисною в певних умовах тестування [11]. Використовуючи їх у комбінації, можна отримати

глибокий і всебічний аналіз продуктивності системи, що допоможе не тількизнайти слабкі місця, а й оцінити загальну стабільність та надійність системи.

3.4 Порівняння із BenchmarkDotNet

Загалом у висновку варто зазначити що не один бенчмарк чи бібліотека яка так чи інакше дотична до вимірювання швидкості роботи запитів – не має функціоналу для збору та відображення колекцій сумованих запитів та часу виконання цих запитів для відображення на лінії кривої розподілу, також варто відмітити що багато бібліотек також не мають єдиного стандартизованого підходу до відображення статистики.

BenchmarkDotNet є високоефективним і популярним інструментом для мікро-бенчмаркінгу коду в .NET, зокрема для вимірювання продуктивності окремих методів чи операцій у межах .NET додатків. Він чудово підходить для тестування продуктивності на рівні коду та виявлення різних характеристик як латентності, пропускної спроможності, розподілу пам'яті тощо. Проте, він не завжди підходить для всіх сценаріїв, особливо коли мова йде про бази даних або інтегровані системні тести. Ось кілька причин, чому наша система "QPressure" може бути потрібна поряд із таким інструментом як BenchmarkDotNet:

- спеціалізовані сценарії для баз даних: наша система може бути налаштована для специфічних тестів продуктивності баз даних, які включають складні запити sql, транзакції або інтеракції, що не можуть бути легко виміряні benchmarkdotnet. такі тести можуть включати великі об'єми даних, різні рівні ізоляції транзакцій або специфічні конфігурації баз даних;
- тестування під реальним навантаженням: "querypressure" може бути призначена для моделювання реального користувацького навантаження на базу даних, що дозволяє аналізувати як система веде себе під час інтенсивного використання. це включає одночасне виконання багатьох запитів, що може відтворювати типове або пікове навантаження виробничої системи;

- визначення проблем проектування бази даних: іноді проблеми з продуктивністю пов'язані не з кодом, а зі структурою бази даних, такими як індекси, плани запитів або конфігурація сервера. "querypressure" може допомогти ідентифікувати такі вузькі місця, які можуть не бути виявлені підчас стандартного бенчмаркінгу коду;
- інтегровані системні тести: наш інструмент може інтегруватися з іншими системами та сервісами, що дозволяє виконувати комплексні тести, які вимірюють не тільки локальну продуктивність коду, а й загальну поведінку інтегрованої системи;
- кастомізація та гнучкість: "querypressure" може надавати більшу гнучкість у налаштуванні параметрів тестів, виборі баз даних або в конфігурації середовища, що є критично важливим для тестування специфічних сценаріїв або виробничих умов.

Хоча BenchmarkDotNet є відмінним інструментом для мікро-бенчмаркінгу в рамках .NET, наша система "QPressure" може забезпечити додаткові інструменти та можливості для тестування та оптимізації баз даних і інтегрованих систем, які важливі для загальної продуктивності та стабільності бізнес-додатків.

4 ПРОВЕДЕННЯ ЕКСПЕРЕМЕНТАЛЬНОЇ РЕАЛІЗАЦІЇ

4.1 Опис реалізації

Ми проведемо експериментальну реалізацію завданням якої буде демонстрація можливостей реалізації деяких метрик для тестування бази даних а з нею і серверної частини за допомогою розробки модулю статистичних розрахунків на платформі .Net. Ми реалізуємо деякі ключові метрики для демонстрації, такі як:

- середнє;
- медіана;
- квартилі;
- стандартне відхилення;
- модуль збору колекцій для розподілу.

Також згодом буде можливість розширити функціонал до бібліотеки для інтеграції у ваб-застосунки для впровадження метрик як окремого модулю для тестування.

Наша система "QPressure" розроблена як інструмент для проведення навантажувального тестування та аналізу продуктивності баз даних і веб API. Цей інструмент дозволяє інженерам та розробникам здійснювати детальний моніторинг і оптимізацію відповідей систем на різні типи запитів під час різних умов використання [12]. Ось основні характеристики та можливості системи "QPressure":

- багатопотокове тестування:система підтримує запуск множини запитів одночасно, що дозволяє імітувати реальні умови використання і визначити, як система справляється з високим навантаженням;
- підтримка різних типів баз даних:"qpressure" може тестувати різні типибаз даних, включаючи sql і nosql, забезпечуючи гнучкість у виборі тестових сценаріїв та інтеграції з різними технологіями;
- розширені метрики продуктивності:система збирає детальні метрики, такі як час відповіді, пропускна спроможність та розподіл часу відповідей, що допомагає аналізувати продуктивність на глибокому рівні;

- конфігурація через файли:тести можна налаштовувати і керувати через зовнішні конфігураційні файли, що робить систему легкою для налаштування та розширення;
- аналіз результатів:"qpressure" забезпечує засоби для аналізу результатів, що дозволяє користувачам швидко ідентифікувати потенційні проблеми та оптимізувати конфігурацію своїх баз даних або API.

Розробники можуть використовувати "QPressure" для тестування та оптимізації SQL запитів і схем баз даних, щоб мінімізувати час відповіді та підвищити пропускну здатність.

IT-інженери можуть використовувати систему для стрес-тестування інфраструктури перед розгортанням, щоб переконатися в її стабільності та надійності під високим навантаженням.

Автоматизація Тестування: "QPressure" можна інтегрувати в автоматизовані пайплайни CI/CD для регулярного виконання тестів продуктивності.

"QPressure" розроблена з можливістю легкої інтеграції з іншими системами і платформами, забезпечуючи API для взаємодії або як модуль, який може бути інтегрований в більш широкі системи для керування тестуванням та

моніторингом.

4.2 Аналіз файлової системи

Для більшого розуміння структури – проаналізуємо файлову структуру системи (див. рис. 4.1).

ApplicationLoader.cs відповідає за ініціалізацію та завантаження основних компонентів системи, управління залежностями та конфігурацією.

AppModule.cs – модуль, який забезпечує зв'язок різних компонентів системи. Він може містити конфігураційну логіку, необхідну для інтеграції різних сервісів і модулів.

ArgumentDescriptor.cs, ArgumentsExtensions.cs, ArgumentsSection.cs – класи, пов'язані з обробкою аргументів командного рядка або конфігураційних параметрів. Використовуються для параметризації запуску тестів і налаштувань.

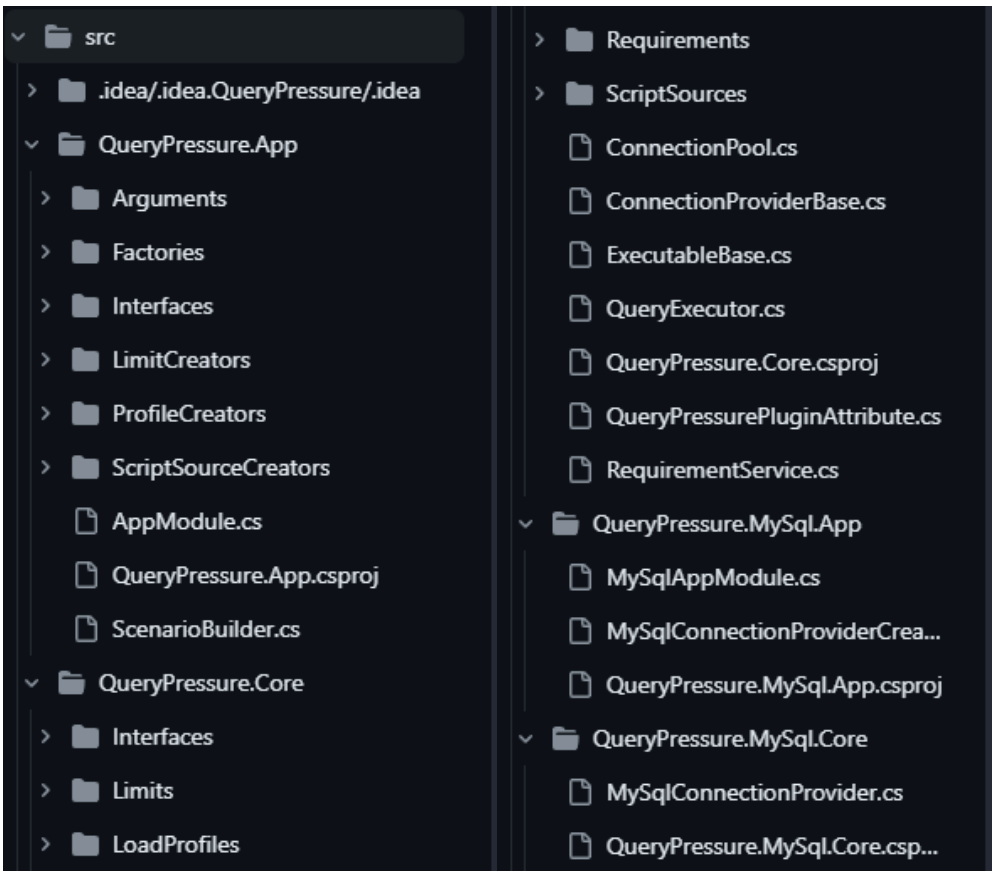

Рисунок 4.1 – Файлова система

ConsoleApplicationLoader.cs – специфічний клас для завантаження консольних додатків, можливо з консольним інтерфейсом для взаємодії з користувачем. ConsoleMetricFormatter.cs, ConsoleMetricFormatterProvider.cs, ConsoleMetricsVisualizer.cs – класи для форматування і відображення метрик у консолі. Вони відіграють ключову роль у представленні результатів тестів у зрозумілому форматі.

EmbeddedResourceDiscovery.cs використовується для виявлення і завантаження вбудованих ресурсів, таких як конфігураційні файли або скрипти, які вбудовані в асемблеї проекту.

ExecutionResultStore.cs – клас для збереження результатів виконання тестів. Використовує LinkedList для зберігання результатів, дозволяючи ефективно зберігати та доступати до історії тестів.

MetricsModule.cs, StatisticalMetricsProvider.cs – компоненти, які забезпечують збір та аналіз статистичних метрик виконання. Вони важливі для оцінювання

продуктивності та ефективності виконуваних операцій.

Docker та конфігураційні файли: docker-compose.yml і спеціалізовані версії для різних баз даних (MySQL, PostgreSQL тощо) використовуються для налаштування середовищ виконання в Docker, забезпечуючи легке розгортання та ізоляцію сервісів.

Скрипти баз даних script.mysql.sql, script.postgres.sql і подібні файли містять SQL скрипти для ініціалізації баз даних, налаштування схем, індексів тощо.

Для більшого розуміння структури наведемо рисунок Б.1 у Додатку Б де зображено структуру класів.

Рисунок включає основні компоненти нашої системи, показуючи, як модулі Connectors інтегровані з модулем Metrics та як вони взаємодіють із основними компонентами системи, такими як Loader, ScenarioBuilder, та ExecutionResultStore.

4.3 Цікаві складові елементи

Далі розглянемо більш детально деякі елементи системи для більшого розуміння принципу роботи. Але спочатку опишімо пайп-лайн роботи тесту, від вводу конфігураційного файлу – до отримання результатів метрик. Далі розглянемо складові та їх функціонал:

- введення даних та налаштувань тесту: файли конфігурації: appsettings.json, sample.mysql.yml та інші специфічні файли для різних субд. класи: configurationmanager (гіпотетична назва) для завантаження та обробкиконфігурацій;
- завантаження конфігурації: клас: loader або startup для ініціалізації та встановлення параметрів тестування;
- ініціалізація тестового середовища: файли: docker-compose.yml для налаштування середовищ.класи: databasesetup (гіпотетична назва) для створення та конфігурації баз даних;
- створення тестових сценаріїв:файли: querytemplates.sql (гіпотетичнаназва) для sql скриптів. класи: testscenariobuilder для генерації тестових сценаріїв;

– запуск тестів:класи: testrunner, apitestscenario для виконання і оркестрації тестів. функціонал: використання httpclient для api запитів та sqlcommand для sql запитів;
– збір результатів:клас: executionresultstore для зберігання результатів виконання;
– класи: metriccalculator, statisticalmetricsprovider для обрахунку та аналізу метрик.

Далі на рисунку 4.2 наведемо візуальну інтерпретацію процесу обробки тесту у системі.

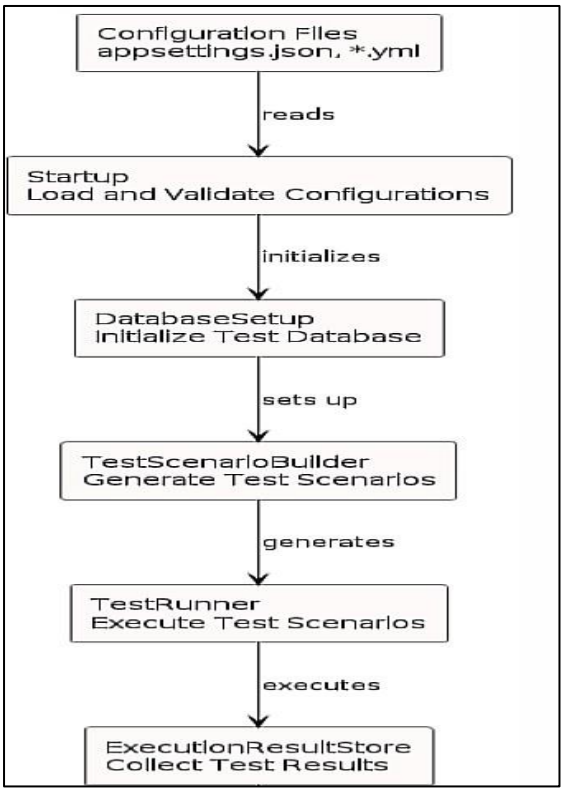

Рисунок 4.2 – Пайп-лайн тесту

Також варто розглянути основні діючі елементи по типу MetricsCalculator та ExecutionResultStore.

Цей клас відповідає за обчислення метрик виконання. Він реалізує інтерфейс IMetricsCalculator, що робить його основним компонентом для вимірювання та аналізу виконання запитів:

```
      public class MetricsCalculator : IMetricsCalculator
      {
        private readonly  IEnumerable<IMetricProvider>
   _providers; public
MetricsCalculator(IEnumerable<IMetricProvider> providers)
        {
          _providers = providers;
        }
      public async Task<IEnumerable<IMetric>>
 CalculateAsync(IExecutionResultStore store,
 CancellationTokencancellationToken)
        {
          var result =
     ImmutableArray.CreateBuilder<IMetric>();foreach
     (var provider in _providers)
          {
      var metrics = await
 provider.CalculateAsync(store,cancellationToken);
            result.AddRange(metrics);
          }
          return result;}}
```

Основний метод CalculateAsync(), який асинхронно викликає кожен провайдер метрик для обчислення метрик на основі збережених результатів запитів. Результати з усіх провайдерів збираються та повертаються як один набір метрик.

Цей клас реалізує IExecutionResultStore та зберігає всі результати виконання запитів у LinkedList<ExecutionResult>. Він відіграє роль центрального сховища для всіх результатів тестів:

```
      public class ExecutionResultStore : IExecutionResultStore
      {
        private readonly LinkedList<ExecutionResult> _results = new();
      public Task OnQueryExecutedAsync(ExecutionResult
 result,CancellationToken cancellationToken)
        {
          _results.AddLast(r
     esult); return
     Task.CompletedTask;
        }
        public IEnumerator<ExecutionResult> GetEnumerator()
        {
          return _results.GetEnumerator();
        }
        IEnumerator IEnumerable.GetEnumerator()
        {
          return GetEnumerator();}}
```

OnQueryExecutedAsync() додає результат запиту в зв'язний список.GetEnumerator(): Повертає ітератор для перебору результатів виконання.

Кожен з цих класів є важливою частиною припайну метричного обчислення в системі "QPressure", де MetricsCalculator відповідає за обчислення метрик з використанням даних, зібраних ExecutionResultStore, і кожен результат виконання запитів передається через інтерфейс IExecutionHook для можливих додаткових обробок.

Далі варто також розглянути методи реалізації метрик, що є най важливішою частиною проекту. Отже, нижче наведено приклад вирахування перцентилів:

```
      public static class StatisticsCalculator
      {
          private static double GetPercentile(double[] sortedData, double
percentile)
          {
              int N = sortedData.Length;
              double n = (N - 1) * percentile / 100.0 + 1;
              if (n == 1d)
              {
                  return sortedData[0];
              }
              else if (n == N)
              {
                  return sortedData[N - 1];
              }
              else
              {
                  int k = (int)n;
                  double d = n - k;
                  return sortedData[k - 1] + d * (sortedData[k] -
sortedData[k - 1]);
              }
          }
          public static double[] CalculatePercentiles(List<double> values,
params int[] percentiles)
          {
              var sortedValues = values.OrderBy(x => x).ToArray();
              return percentiles.Select(p => GetPercentile(sortedValues,
p)).ToArray();}}
```

Цей публічний метод використовує GetPercentile для обчислення перцентилів зі списку значень. Це зручно для обробки реальних даних, де значення не завжди можуть бути заздалегідь впорядкованими.

List<double> values – це список значень, для яких потрібно обчислити

перцентилі.

params int[] percentiles – масив перцентилів, які потрібно обчислити (наприклад, 25, 50, 75).

Спочатку відбувається сортування списку значень.

Потім для кожного вказаного у масиві перцентиля викликається GetPercentile, результати обчислень збираються в масив і повертаються.

Також варто відзначити метод збору статистики для відображення відхилення, код наведено у ДОДАТОК В.

Клас HistogramConsoleMetricFormatter є прикладом спеціалізованого

форматера для представлення гістограм метрик у консолі. Він реалізує інтерфейс IConsoleMetricFormatter, який забезпечує можливість форматувати метрики для виводу в консольному додатку "QPressure". Наведемо детальний розбір компонентів та логіки цього класу.

Параметри:

- ConsoleOptions consoleOptions – залежність, яка містить налаштування консолі, такі як символи-роздільники;
- IResourceManager resourceManager використовується для доступу до ресурсів, які можуть залежати від локалізації.

Логіка:

- ініціалізація _consoleoptions та витягування локалізованих ресурсів у _locale за допомогою resourcemanager, що дозволяє адаптувати відображення до встановленої культури;
- створення заголовка з використанням назви метрики, заміна символів для відповідності налаштуванням консолі;
- використання методу tostring з класу histogram, передаючи делегат для форматування значень часу з використанням класу timeinterval.

Методи:

- priority (властивість): повертає пріоритет форматера. у цьому випадку,це 1, що може вказувати на порядок використання серед інших форматерів;
- canformat(string metricname, object metricvalue): перевіряє, чи може цей

форматер обробити метрику, яка передається. він обробляє метрики типу histogram;

– format(string metricname, object metricvalue): формує рядок для виведення гістограми в консоль. використовує налаштування та локалізацію для адаптації представлення.

Приблизний вигляд розподілу із піками на ліміти часу запиту – наведено на рисунку 4.3.

```
[17.000 ; 17.500) | @
[17.500 ; 18.000) | @@@@@@
[18.000 ; 18.500) | @@@@@@@@@
[18.500 ; 19.000) | @@@@@@@@@@@@@@@@@@@@@@@
[19.000 ; 19.500) | @@@@@@@@@@@@@@@@@@@@@@@@@@@
[19.500 ; 20.000) | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
[20.000 ; 20.500) | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
[20.500 ; 21.000) | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
[21.000 ; 21.500) | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
[21.500 ; 22.000) | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
[22.000 ; 22.500) | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
[22.500 ; 23.000) | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
[23.000 ; 23.500) | @@@@@@@@@@@@@
[23.500 ; 24.000) | @@@@@@@@@@@@@
[24.000 ; 24.500) | @@@
```

Рисунок 4.3 – Схематична крива розподілу

Цей клас є хорошим прикладом того, як можна адаптувати відображення даних в консолі, використовуючи налаштування та локалізацію. Він забезпечує зрозуміле візуальне представлення складних метрик, що є критично важливим для аналізу продуктивності та оптимізації процесів.

4.4 Вхідні дані до реалізації

Профілі навантаження:

– обмеження: для забезпечення ефективного тестування на навантаження необхідно встановити чіткі обмеження, які дозволять контролювати процес і забезпечувати повторюваність результатів;

– час: визначте максимальний час виконання тесту, після якого тестування буде автоматично зупинено. це може бути кількість годин або днів, в залежності від тестового сценарію;

- кількість запитів: обмеження за кількістю запитів, які можуть бути відправлені до бази даних під час одного тесту. це дозволяє зрозуміти обробку базиданих при великих обсягах інформації;
- до першої помилки: тестування продовжується до моменту виникнення першої помилки, що дозволяє ідентифікувати потенційні вузькі місця або вразливості системи.

При проектуванні тестів важливо визначити, як буде генеруватися навантаження.

Характеристики навантаження:

- послідовне: запити відправляються один за одним без затримки. це симулює реальне навантаження від окремих користувачів, які виконують дії по черзі;
- з затримкою: запити відправляються з певною затримкою між кожним,що дозволяє тестувати систему під менш інтенсивним навантаженням;
- цільова пропускна спроможність: встановлення конкретної кількості запитів за секунду, яка повинна оброблятися системою.

Зростаюче навантаження:

- дельта/час: поступове збільшення кількості запитів на одиницю часудля визначення меж продуктивності системи;
- з затримкою: визначення максимальної кількості одночасних запитів, які можуть бути оброблені системою, з можливістю введення затримок між серіямизапитів для імітації реального користувацького доступу.

Важливо забезпечити можливість порівняння результатів різних тестів для оцінки впливу змін в системі на її продуктивність.

Звітування:

- ефективне звітування є ключовим елементом навантажувального тестування, оскільки воно дозволяє аналізувати результати і робити висновки про продуктивність та стабільність системи;
- розробка системи звітування повинна включати автоматизовані інструменти для збору даних, їх агрегації та візуалізації;

– звіти мають включати деталізацію по кожному з проведених тестів, включаючи час виконання, кількість і тип помилок, час відгуку та інші важливі метрики.

Далі варто розглянути питання доступу до СУБД та взаємодії із базою.

У нашій системі взаємодія з базами даних та використання плагінів для різних СУБД відіграють ключову роль у функціонуванні та гнучкості. Система налаштована таким чином, щоб легко інтегруватися з різними типами баз даних, використовуючи спеціалізовані плагіни.

Система використовує поняття "Connection Providers" для управління з'єднаннями з базами даних. Ці провайдери відповідають за створення та підтримку з'єднань з конкретними типами баз даних. Наприклад, можуть бути різні провайдери для SQL Server, MySQL, PostgreSQL та MongoDB.

Кожен провайдер включає логіку для ініціації з'єднань, виконання запитів та обробки результатів.

Система дозволяє виконання запитів через спеціалізовані класи, які можуть включати параметризовані SQL запити або скрипти. Завдяки цьому, вона може виконувати запити, що є критично важливими для навантажувального тестування.

Результати запитів збираються та аналізуються для вимірювання продуктивності баз даних.

Також нами було прийнято рішення використовувати плагіни до різних СУБД. Архітектура побудована таким чином, що вона підтримує плагіни для різних СУБД. Це означає, що для кожної СУБД може бути розроблений спеціалізований плагін, який імплементує інтерфейси та класи, необхідні для взаємодії з цією конкретною системою. Плагіни та вибір конфігурації із ними зображено на рисунку 4.4.

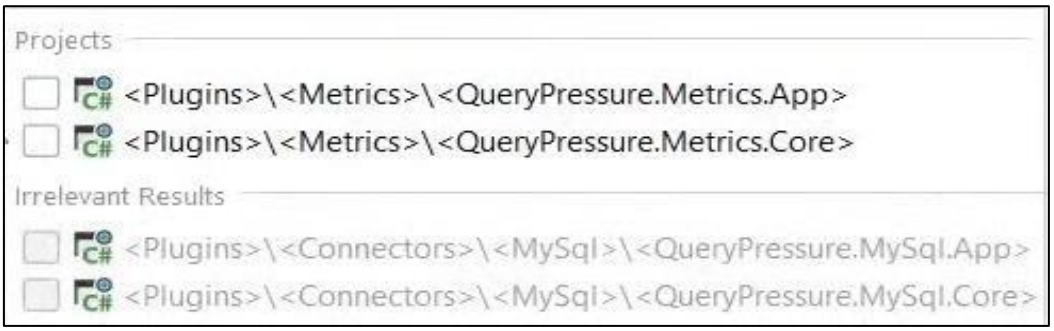

Рисунок 4.4 – Плагіни БД

Плагіни можуть включати специфічні оптимізації або методи взаємодії, які найкраще підходять для певних СУБД.

5 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Для аналізу результатів варто навести конфігураційний файл із прикладом тестового сценарію, конфігурація наведена на рисунку 5.1.

```
profile:
  type: targetThroughput
  arguments:
    rps: 20
limit:
  type: queryCount
  arguments:
    limit: 400
connection:
  type: mysql
  arguments:
    connectionString: Host=localhost;Database=query_pressure_db;User Id=root;SSL Mode=None

```

Рисунок 5.1 – Тестова конфігурація

У файлі ми зазначили виведені раніше конфігурайійні обмеження, та встановили ліміт на 400 запитів та строку підключення до бд а також профілзапиту та кількість запитів на секунду.

На рисунку 5.2 наведено процес вибору плагіну конфігурації.

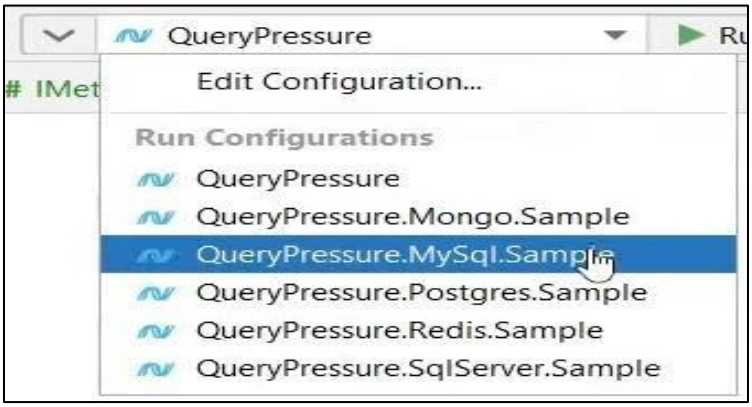

Рисунок 5.2 – Вибір плагіну конфігурації

Після завершення роботи ми отримали результати реалізованих метрик середнього, квартилів та медіани та стандартного відхилення (див. рис. 5.3).

Наш метод та реалізація вирізняється на тлі інших рішень та методів, комплексних чи одинарних для навантажувального тестування, завдяки своєму інноваційному підходу до аналізу і візуалізації результатів. Особливо важливим є впровадження кривої розподілу у форматі гістограм, що дає змогу бачити не просто основні метрики, а детальний розподіл часів відповідей на запити. Це значно

розширює можливості аналізу даних, дозволяючи краще розуміти поведінку системи під навантаженням.

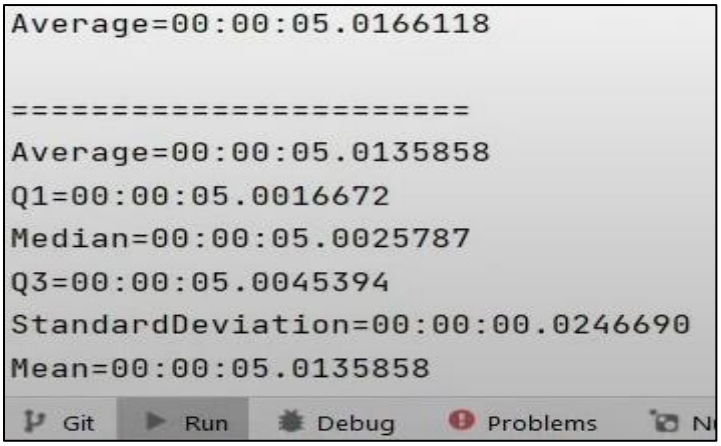

Рисунок 5.3 – Результати реалізованих метрик

Включення гістограми розподілу в "QPressure" дозволяє відслідковувати не тільки середні значення часів відповідей, але й бачити, як розподіляються відповіді в межах тестового сценарію. Це особливо корисно для виявлення неочікуваних затримок або варіабельності в часах відповідей, що може вказувати на потенційні проблеми з продуктивністю або стабільністю.

Система також розширила стандартний набір метрик, зазвичай використовуваних у навантажувальному тестуванні, включаючи не тільки середні значення, але і перцентилі, мінімальні та максимальні відповіді, а також статистичні агрегації, як-от стандартне відхилення. Це надає користувачам комплексний інструментарій для детального аналізу продуктивності їхніх систем.

Система плагінів у "QPressure" дозволяє легко інтегрувати підтримку нових баз даних, що робить її сумісною з широким спектром СУБД, від традиційних SQL-серверів до новітніх NoSQL рішень. Кожен плагін може бути налаштованийдля оптимальної взаємодії з конкретною системою, забезпечуючи точність і ефективність тестування.

На даний момент, "QPressure" є однією з небагатьох, якщо не єдиною, системою на платформі .NET, що включає такий рівень деталізації в аналізі продуктивності та надає таку широку підтримку різноманітних СУБД через модульну систему.

ВИСНОВКИ

У даній роботі нами був проведений аналіз, який охоплював область тестування веб-застосунків, а саме – навантажувальне тестування. Було проаналізовано важливість проведення дослідження цієї області та сформульовано основні вимоги до дослідження яке буде мати на меті вдосконалення методів навантажувального тестування а також їх порівняння та вибір необхідних.

Нами було проведено аналіз методів знаходження середнього значення, проведено аналіз медіанного часу швидкості запитів а також досліджено та проаналізовано метод перцентилів.

Нами було визначено що вищезазначені методи дають дуже узагальнену оцінку ситуації та оминають виключні випадки та аномалії. Тому надалі нами було проаналізовано метод нормального розподілу, метод квартилів та міжквартального інтервалу, ці методи не прийнято використовувати у розрізі навантажувального тестування, але ми визначили що їх можна адаптувати під наші задачі.

Загалом нами було впроваджено метод розподілу для більшого спектру можливостей для аналізу результатів навантажувального тестування, цей підхід також можна використати для замірів швидкості запитів у веб-додатках.

За результатами нами було реалізовано програмну систему, фактично модуль тестування із можливістю налаштування параметрів навантаження, вибору бази даних. Система має змогу відображати комплекс метрик та гістограм які допоможуть та прокриють весь об’єм інформації яка потрібна розробника для аналізу запитів та продуктивності серверної частини.

Отримані результати можуть знадобитись розробникам та системним адміністраторам яким необхідно аналізувати та знаходити аномалії у значеннях навантажувального тестування. Також варто зазначити що на даний момент не має фактичних аналогів на платформі .Net для цих цілей а отже ця система приверне увагу розробників та набуде багатокористувацького застосування та імплементації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Аналіз предметної області, URL: https://www.blazemeter.com/blog/performance-testing-vs-load-testing-vs-stress-testing (дата звернення: 01.04.2024).

2. Load and stress testing in .NET 6, URL: https://medium.com/@matias.paulo84/load-and-stress-testing-in-net-6-725ba346cf78 (дата звернення: 01.04.2024).

3. Аналіз навантажувального тестування, URL: https://www.blazemeter.com/blog/performance-testing-vs-load-testing-vs-stress-testing (дата звернення: 01.04.2024).

4. Performance Testing For .NET Applications, URL: https://www.radview.com/solution/net/ (дата звернення: 09.04.2024).

5. Аналіз середнього показника, URL: https://www.blazemeter.com/blog/performance-testing-vs-load-testing-vs-stress-testing (дата звернення: 09.04.2024).

6. Аналіз методу медіанного значення, URL: https://ca.indeed.com/career-advice/career-development/formula-of-median (дата звернення: 09.04.2024).

7. Аналіз перцентилів, URL: https://www.statisticshowto.com/probability-and- statistics/percentiles-rank-range/ (дата звернення: 09.04.2024).

8. Аналіз нормального розподілу, URL: https://statisticsbyjim.com/basics/normal-distribution/ (дата звернення: 20.04.2024).

9. Deep dive API: a statistical approach of API performance, URL: https://maximechampoux.medium.com/deep-dive-api-a-statistical-approach-of-api-performance-9004e851924b / (дата звернення: 20.04.2024).

10. Optimization Factors in Modeling and Testing Hardware and Semiconductor Defects by Dynamic Discrete Event Simulation \ Arabian, J. H., Видавництво: ХНУРЕ, 2009р., URL: https://openarchive.nure.ua/entities/publication/be918fba-8755- 4d34-be6d-fc1464089d77 (дата звернення: 20.04.2024).

11. Falatiuk H., Shirokopetleva M., Dudar Z. Investigation of Architecture and Technology Stack for e-Archive System. 2019 IEEE International Scientific-Practical

Conference Problems of Infocommunications, Science and Technology (PIC S&T), Kyiv, Ukraine, 8-11October 2019, URL: https://doi.org/10.1109/picst47496.2019.9061407 (дата звернення: 05.05.2024).

12. DATA EXCHANGE MODEL IN THE INTERNET OF THINGS CONCEPT / I. Afanasieva et al. Telecommunications and Radio Engineering. 2019. Vol. 78,no. 10.P. 869–878. URL: https://doi.org/10.1615/telecomradeng.v78.i10.30 (date of access: 11.05.2024).