Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Порівняння Spring MVC, Spring Webflux, Quarkus, Micronaut

Текстова частина до курсової роботи

за спеціальністю „Інженерія програмного забезпечення” 121

Керівник курсової роботи
асистент _Андрощук М__
(прізвище та ініціали)
_____
(підпис)
“_____” _____ 2023 р.

Виконав студент _____
___Загривий О.С._____
(прізвище та ініціали)
“_____” _____ 2023 р.

Київ 2023

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав. кафедри інформатики,
доцент, кандидат наук
_____ Гороховський С.С.
(підпис)
“_____” _____ 2023 р.

І

НДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту __Загривому Олегу Сергійовичу факультету інформатики 3-го курсу____

ТЕМА Порівняння Spring MVC, Spring Webflux, Quarkus, Micronaut

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Вступ
1. Аналіз предметної області
2. Порівняння можливостей
3. Тестування та вимірювання характеристик
Висновки
Список літератури
Додатки (за необхідністю)

Дата видачі: “____ ” ____ 2022 р.

Керівник _____ (підпис)

Завдання отримав _____ (підпис)

Тема: Порівняння Spring MVC, Spring Webflux, Quarkus, Micronaut.

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	01.10.2022	
2.	Огляд та аналіз матеріалів за темою роботи, дослідження фреймворків.	05.11.2022	
3.	Проектування проєктів для кожного фреймворку.	15.12.2022	
4.	Будування проєктів.	25.01.2023	
5.	Визначення формату тестів.	20.02.2023	
6.	Тестування проєктів.	15.03.2023	
7.	Написання текстової частини роботи.	26.04.2023	
8.	Коригування роботи.	01.05.2023	
9.	Створення презентації.	11.05.2023	
10.	Захист курсової роботи.	24.05.2023	

Загривий О. С. _____ _____

Андрощук М. В. _____ _____

“_____” _____ _____ р.

ЗМІСТ

АНОТАЦІЯ

Роботу присвячено порівнянню та дослідженню фреймворків для веб-розробки, зокрема Spring MVC, Webflux, Quarkus та Micronaut, описані їх переваги та недоліки. Написані проєкти для тестування ефективності та продуктивності цих фреймворків та проведено аналіз результатів.

ВСТУП

У сучасному світі розробки програмного забезпечення веб-додатки стають все більш поширеними, і вибір правильного інструментарію може мати велике значення для успіху проекту.

Метою цієї курсової роботи є дослідження фреймворків та аналіз їх переваг і недоліків, а також визначення кращих з них для різних цілей у програмуванні.

У даній роботі будуть розглянуті різні аспекти розробки веб-додатків, включаючи маршрутизацію, обробку запитів, безпеку, автоматичне перезавантаження та перекомпілювання, а також підтримку тестування та документацію кожної з інструментаріїв. Метою цього дослідження є надання об'єктивної інформації, яка допоможе розробникам обрати найкращий набір засобів для своїх потреб.

Дослідження включатиме проведення тестів та вимірювання різних параметрів продуктивності, таких як час компіляції та виконання тестів, час запуску додатків, час першої відповіді, продуктивність запитів та використання пам'яті. Результати тестування будуть порівняні для кожного інструментарію, що дозволить здійснити об'єктивну оцінку їх характеристик та зрозуміти, який з них найкраще відповідає вимогам розробки веб-додатків.

Окрім того, будуть розглянуті переваги та недоліки кожного фреймворку, що допоможе з'ясувати, які особливості мають вплив на розробку та підтримку проекту. Крім того, розглянуті будуть можливості реактивного програмування, підтримка тестування та наявність документації, що є важливими аспектами при виборі інструментарії для розробки веб-додатків.

Ця курсова робота буде корисною для розробників, які планують створювати веб-додатки та розглядають різні варіанти інструментарію для розробки. Результати тестування та перелік переваг і недоліків допоможуть їм зробити обґрунтований вибір та використовувати найбільш підходящий набір засобів для своїх проектів.

РОЗДІЛ 1: АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Веб-розробка та серверна частина

Веб-розробка є суттєвою складовою сучасного інформаційного світу і має ключове значення у створенні та розгортанні веб-додатків. Веб-додаток - це програмне забезпечення, розроблене для роботи у веб-середовищі, і доступне користувачам через веб-браузер. При розробці веб-додатків важливими концепціями є фронтенд та бекенд.

Фронтенд (англ. front-end), також відомий як клієтська частина, відповідає за візуальну складову додатку та те, як користувачі можуть взаємодіяти з ним.

Бекенд (англ. back-end), або серверна частина, відповідає за обробку запитів, взаємодію з базами даних, бізнес-логіку та інші аспекти, необхідні для коректного функціонування веб-додатків. Він забезпечує обробку даних та взаємодію з клієнтською частиною.

Важливість бекенду полягає у забезпеченні зручності, ефективності та масштабованості веб-додатків. Бекенд повинен надавати надійну та швидку обробку запитів користувачів, ефективно взаємодіяти з базами даних для зберігання та отримання інформації, а також забезпечувати бізнес-логіку додатку. Крім того, бекенд має бути масштабованим, щоб забезпечити ріст та підтримку великого числа користувачів.

Фреймворк (англ. framework, іноді також називають каркасом, структурою) - це набір інструментів, бібліотек та структур, які надають розробникам програмного забезпечення готові рішення для побудови додатків або систем. Він визначає загальну структуру та організацію проекту, надає набір правил і шаблонів, за допомогою яких можна розробляти програмне забезпечення з меншими зусиллями та витратами часу.

Основна мета фреймворку полягає в спрощенні розробки програмного забезпечення шляхом надання готових рішень для типових завдань та проблем, з якими стикаються розробники.

Каркаси бекенду які вивчаються у цьому дослідженні (Spring MVC, Spring Webflux, Quarkus, Micronaut) забезпечують готові рішення для розробників, спрощуючи роботу зі стандартними завданнями, такими як маршрутизація, обробка HTTP-запитів, взаємодія з базами даних та забезпечення безпеки. Вони надають абстракції та інтерфейси, що дозволяють розробникам швидко реалізовувати функціональність своїх додатків, використовуючи готові модулі та компоненти.

Ці фреймворки спеціалізуються на розробці бекенду веб-додатків. Вони надають розробникам необхідні інструменти та структуру для реалізації серверної логіки, обробки запитів, взаємодії з базами даних та інших компонентів системи. Такі структури дозволяють прискорити розробку, забезпечити кращу організацію коду, а також надають можливості для розширення та налаштування веб-додатків.

1.2. Фреймворк Spring та його компоненти

1.2.1 Spring

Spring є одним з найпопулярніших фреймворків для розробки додатків на мові програмування Java [1,2]. Він пропонує високорівневі інструменти та компоненти, які допомагають розробникам будувати ефективні та масштабовані програми з меншими зусиллями. Основні принципи фреймворка Spring включають:

- Інверсія керування (Inversion of Control, IoC): Spring використовує контейнер інверсії керування для створення та керування об'єктами. Це дозволяє розробникам зосередитися на реалізації бізнес-логіки, незалежно від деталей управління об'єктами.
- Управління життєвим циклом об'єктів: Spring дозволяє визначати життєвий цикл об'єктів, включаючи їх створення, ініціалізацію та

знищення. Це забезпечує більшу контрольованість та підтримує зв'язування залежностей між об'єктами.

- Інверсія керування за допомогою аспектів (AOP): Spring надає можливість для застосування принципу інверсії керування за допомогою аспектів. AOP дозволяє відокремлювати перехресну функціональність, таку як запис подій (logging), транзакції та безпеку, від основної бізнес-логіки програми. Це полегшує управління аспектами, забезпечує модульність та підвищує повторне використання коду.
- Інструменти для тестування: Spring надає підтримку для тестування, що допомагає розробникам створювати автоматизовані тести для перевірки функціональності своїх додатків. Це включає можливості для модульного тестування, інтеграційного тестування та тестування залежностей.
- Розширюваність: Spring пропонує розширюваність через використання плагінів та розширень. Розробники можуть використовувати вже існуючі модулі та розширення Spring або створювати свої власні, щоб додати специфічну функціональність або властивості до своїх додатків.

Spring також надає широкий набір модулів та розширень, таких як Spring Security для забезпечення безпеки, Spring Data для роботи з базами даних, Spring Cloud для розробки мікросервісів та багато інших. Ця екосистема дозволяє розробникам використовувати потужні інструменти та бібліотеки для реалізації різних аспектів своїх додатків.

1.2.2. Spring Boot

Spring Boot - це розширення фреймворка Spring, яке дозволяє розробникам створювати самостійні, готові до використання та легко налаштовувані додатки на основі Spring з мінімальними зусиллями[3]. Він надає простий спосіб

розробки веб-додатків, зменшуючи необхідність у ручному налаштуванні та конфігурації. Основні особливості Spring Boot:

- Автоматична конфігурація: Spring Boot має вбудований механізм автоматичної конфігурації, який дозволяє автоматично налаштовувати додаток, виходячи з його залежностей та налаштувань. За замовчуванням, Spring Boot надає розумні значення конфігураційних параметрів, що дозволяє швидко створювати додатки без необхідності в ручній конфігурації.
- Вбудований контейнер: Spring Boot має вбудований сервер додатків, що дозволяє запускати додаток без необхідності встановлення та конфігурації окремого сервера. Це полегшує розгортання та тестування додатків.
- Управління залежностями: Spring Boot надає велику кількість готових залежностей та бібліотек, які можна використовувати для розробки додатків. Він має вбудований менеджер залежностей Maven або Gradle, що дозволяє легко додавати та керувати залежностями в проекті.
- Сервіси моніторингу та адміністрування: Spring Boot надає вбудовані сервіси, які дозволяють отримувати різноманітну інформацію про додаток та його стан. Це включає здатність перевіряти стан додатка, моніторити використання ресурсів, збирати метрики та багато іншого. Це полегшує процес моніторингу та управління додатками.
- Вбудована безпека: Spring Boot надає вбудовану безпеку, що допомагає захистити додаток від загроз та забезпечити автентифікацію та авторизацію. Він має підтримку різних механізмів безпеки, таких як використання ролей, налаштування прав доступу та інтеграція зі сторонніми системами аутентифікації.
- Легка розширюваність: Spring Boot дозволяє легко розширювати його функціональність шляхом додавання власних модулів та розширень. Він надає гнучкий механізм конфігурації та розширення, що дозволяє розробникам адаптувати фреймворк під свої потреби.

Spring Boot дозволяє розробникам швидко створювати функціональні веб-додатки без необхідності ускладненої конфігурації та налаштувань. Він забезпечує готове середовище для розробки, що дозволяє розробникам фокусуватися на бізнес-логіці своїх додатків. Це робить Spring Boot одним з найпопулярніших виборів для розробки веб-додатків на платформі Java.

1.2.3. Spring MVC

Spring MVC є одним з найпопулярніших фреймворків для веб-розробки на платформі Java [4]. Він надає потужні засоби для розробки веб-додатків з використанням архітектурного підходу Model-View-Controller (MVC).

Модель-Вид-Контролер (Model-View-Controller, MVC) є популярним шаблоном проектування, який використовується для розділення бізнес-логіки додатка, представлення даних та управління користувацьким інтерфейсом. Основні компоненти MVC включають:

- Модель (Model): Відповідає за бізнес-логіку та обробку даних. Вона зазвичай представляє собою об'єкти доменної моделі або сервіси, які взаємодіють з базою даних або іншими джерелами даних.
- Вид (View): Відповідає за представлення даних користувачу. Це може бути HTML-шаблон, який відображає дані з моделі, або інші типи представлення, такі як PDF-файли або XML-документи.
- Контролер (Controller): Відповідає за обробку запитів користувача та керування потоком виконання. Контролер приймає запити від користувача, взаємодіє з моделлю для отримання потрібних даних та вибирає відповідне представлення для відображення результату користувачеві.

Spring MVC пропонує ефективну обробку запитів, маршрутизацію, керування сесіями, підтримку технологій відображення (наприклад, JSP (JavaServe Pages), Thymeleaf) та інші інструменти для створення веб-додатків з високою продуктивністю та добре організованою структурою коду.

1.2.4. Spring Webflux

Spring Webflux є фреймворком реактивної веб-розробки, який входить до складу Spring [5]. Цей фреймворк надає можливості для розробки реактивних веб-додатків на основі асинхронного та неблокуючого програмування. Він заснований на принципах реактивного програмування, що дозволяє ефективно обробляти велику кількість запитів і забезпечувати швидку відповідь на них.

Асинхронне програмування - це підхід до розробки програмного забезпечення, в якому виконання операцій відбувається незалежно від основного потоку виконання програми. Основна ідея полягає у тому, що програма не чекає завершення операції, але вона все ще очікує результат. Це дозволяє досягти більшої продуктивності та ефективного використання ресурсів в програмах, особливо при роботі з багатопотоковими та паралельними операціями.

Неблокуюче програмування - це підхід, в якому операції не блокують виконання програми, коли вони очікують результату. Результат неблокуючої операції може бути ігнорований, а може бути використаний або потрібний пізніше. Використовуючи неблокуючий підхід, програма може ефективно виконувати багато операцій одночасно, не затримуючи виконання через очікування завершення певних операцій. Це особливо важливо в сучасних системах, де потрібно обробляти багато одночасних запитів і забезпечувати мінімальну часову затримку відповіді.

1.3. Quarkus

Quarkus є новим поколінням фреймворків Java, яке спеціалізується на створенні швидких та ефективних веб-додатків [6,7]. Він призначений для розробки мікросервісів та хмарних додатків, з фокусом на оптимізацію продуктивності та зниження витрат пам'яті.

Одна з ключових особливостей Quarkus - це його швидкий запуск. Quarkus використовує компіляцію-заздалегідь (Ahead-of-Time, AoT) для компіляції додатка перед його запуском. Це допомагає знизити час запуску додатка та зменшити споживання пам'яті, оскільки код компілюється у машинний код заздалегідь.

Завдяки використанню технологій, таких як гаряче перезавантаження (hot reload), Quarkus дозволяє розробникам швидко вносити зміни в код без необхідності перезапуску додатку. Це сприяє збільшенню продуктивності розробників та спрощенню процесу впровадження змін.

Quarkus заснований на філософії "Optimize for the Cloud" (Оптимізація для хмари), що означає, що фреймворк пристосований до хмарного середовища і надає розробникам можливість ефективно працювати з хмарними ресурсами, такими як контейнери і мікросервіси. Це дозволяє легко розгортати додатки на хмарних платформах, таких як Amazon Web Services (AWS), Microsoft Azure або Google Cloud Platform (GCP).

Quarkus також славиться своєю ефективністю використання ресурсів. Він використовує оптимізовані механізми обробки HTTP-запитів та мінімізує споживання пам'яті. Це особливо важливо при розгортанні мікросервісів та контейнеризованих додатків, де ефективне використання ресурсів може значно покращити масштабованість та продуктивність системи.

Quarkus пропонує злагоджену платформу, спрямовану на оптимізацію задоволення розробників. Вона забезпечує єдину конфігурацію та генерацію нативних виконуваних файлів без зайвих зусиль. Завдяки нульовій конфігурації та миттєвому перезавантаженню, Quarkus надає зручність розробки на

максимально високому рівні для 80% загальних випадків використання. Водночас, він залишається гнучким для тих 20% випадків, які вимагають більшої настроюваності та специфічного підходу.

Ці характеристики спрямовані на полегшення розробки та забезпечення високої продуктивності розробників. Quarkus надає розробникам простоту та ефективність в розробці, одночасно забезпечуючи потужні можливості для налаштування та розширення, що задовольняють різноманітні потреби проектів.

Загалом, Quarkus є потужним фреймворком, який дозволяє створювати швидкі, ефективні та масштабовані веб-додатки на основі Java. Його низький час запуску, ефективне використання ресурсів та розширюваність роблять його привабливим вибором для розробки сучасних мікросервісних та хмарних додатків.

1.4. Micronaut

Micronaut є новим фреймворком Java, який розроблено з метою створення швидких, ефективних та легких веб-додатків[8,9]. Він спрямований на розробку мікросервісних та хмарних додатків, з фокусом на оптимізацію продуктивності та ефективного використання ресурсів.

Однією з ключових особливостей Micronaut є його низьке споживання пам'яті та швидкий час запуску. Це досягається завдяки оптимізованій роботі фреймворку та використанню ефективних механізмів, також через AoT, як в Quarkus. Micronaut дозволяє швидко розгортати додатки та забезпечує ефективне використання ресурсів.

Іншою особливістю Micronaut є його використання статичного аналізу та інформації про структуру додатку для мінімізації використання рефлексії.

Рефлексія – це механізм в Java, який дозволяє програмам аналізувати та змінювати свою структуру під час виконання.

Мінімізація використання рефлексії у Micronaut має декілька переваг. По-перше, це забезпечує покращену продуктивність додатку, оскільки відсутність

зайвих рефлексивних операцій дозволяє покращити швидкодію та ефективність роботи програми. По-друге, це знижує ризик помилок, пов’язаних з некоректним використанням рефлексії, оскільки Micronaut перевіряє структуру додатку ще до його запуску.

Фреймворк Micronaut забезпечує сумісність з відомими анотаціями, що використовуються в інших фреймворках або стандартах, таких як Java EE або Spring. Це дозволяє розробникам швидко адаптуватися до Micronaut і використовувати свої наявні знання та навички без необхідності вивчати нові підходи або синтаксис.

Також Micronaut надає інструменти для автоматичної генерації документації API, яка описує сервіси, їх маршрути, параметри та типи даних. Це дозволяє забезпечити зрозумілість та доступність API для інших розробників, які хочуть використовувати ці сервіси. За допомогою OpenAPI та Swagger, можна створювати документацію, яка описує API у стандартизованому форматі, що спрощує процес інтеграції та спілкування з іншими командами розробників.

Загалом, Micronaut є інноваційним фреймворком, який дозволяє розробникам створювати швидкі, ефективні та легкі веб-додатки. Його низьке споживання пам'яті, швидкий час запуску та розширені можливості роблять його привабливим вибором для розробки мікросервісів та хмарних додатків.

РОЗДІЛ 2: ПОРІВНЯННЯ МОЖЛИВОСТЕЙ

2.1. Маршрутизація

Маршрутизація у Spring MVC, Quarkus та Micronaut заснована на використанні анотацій для визначення шляхів URL та обробників запитів. Це дозволяє розробникам легко налаштовувати маршрутизацію у своїх додатках.

У випадку Spring Webflux, маршрутизація використовує функціональний підхід. Замість використання анотацій, розробники визначають маршрути за допомогою функцій, які приймають на вхід HTTP-запит та повертають відповідь. Це дає більшу гнучкість і контроль над маршрутизацією, особливо у складних сценаріях.

Таким чином, основна відмінність полягає в підході до визначення маршрутів. Spring MVC, Quarkus та Micronaut використовують анотації, в той час як Spring Webflux використовує функціональний підхід. Кожен з цих підходів має свої переваги та підходить для різних сценаріїв розробки.

2.2. Обробка запитів

Фреймворки Spring MVC, Quarkus та Micronaut використовують схожий підхід до обробки HTTP-запитів, вони дозволяють визначати маршрути та обробники запитів за допомогою анотацій або декларативного підходу.

У випадку Spring Webflux використовується асинхронний та реактивний підхід до обробки HTTP-запитів. Він побудований на основі реактивних бібліотек, що дозволяє ефективно обробляти багато запитів одночасно та працювати з неблокуючими операціями вводу/виводу.

Отже, хоча всі чотири фреймворки можуть обробляти HTTP-запити, Spring Webflux має свою особливу функціональність, пов'язану з асинхронним та реактивним програмуванням.

2.3. Безпека

Кожен з фреймворків - Quarkus, Micronaut, Spring Webflux та Spring MVC -надає можливості безпеки та захисту. Кожен фреймворк має свій підхід до безпеки і може надавати вбудовані засоби для забезпечення безпеки веб-додатків.

Quarkus та Micronaut надають вбудовану підтримку безпеки, включаючи механізми автентифікації, авторизації, захисту від атак та керування доступом до ресурсів. Вони мають вбудовані анотації та конфігураційні можливості для налаштування правил безпеки.

У свою чергу, Spring Webflux та Spring MVC також надають можливості безпеки через бібліотеку Spring Security. Spring Security є розширеною бібліотекою безпеки, яка працює на рівні Spring-екосистеми і надає широкий набір можливостей для автентифікації, авторизації та іншого. Але варто зауважити, що Spring Security хоча і є основною бібліотекою для безпеки в Spring, вона не вбудована ні у Webflux, ні у WebMVC за замовчуванням, хоча це і не заважає легко інтегрувати та налаштувати її.

2.4. Автоматичне перезавантаження та перекомпяція

Дослідження функціональності автоматичного перезавантаження коду та гарячої перекомпіляції є важливим аспектом для швидкості розробки та зручності. Перевагою Quarkus та Micronaut є те, що вони пропонують підтримку автоматичного перезавантаження коду та гарячого перекомпілювання (hot reloading), що дозволяє розробникам швидше і зручніше вносити зміни до своїх додатків та спостерігати результати без необхідності повного перезапуску сервісу.

Quarkus використовує технологію, відому як "Dev Mode", яка дозволяє автоматично перезавантажувати змінений код без перезапуску додатку. При внесенні змін до коду Quarkus автоматично виявляє ці зміни та застосовує їх, що

дозволяє розробникам бачити зміни миттєво. Це значно прискорює процес розробки та тестування, оскільки розробникам не потрібно чекати на повний перезапуск додатку для перегляду змін.

У Micronaut також є подібна функціональність гарячої перекомпіляції, яка називається "Restart", і вона дозволяє автоматично перезавантажувати змінений код без перезапуску додатку. Коли розробник вносить зміни до коду, Micronaut виявляє ці зміни та виконує hot-reloading, що дозволяє бачити зміни миттєво без затримок.

Обидва фреймворки, Quarkus та Micronaut, надають зручні інструменти для автоматичного перезавантаження та гарячого перекомпілювання, що робить їх відмінними виборами для швидкої розробки та тестування додатків. Ці функції забезпечують зручність та продуктивність розробників, дозволяючи їм швидко ітерувати над кодом та переглядати результати без зайвих затримок.

2.5. Порівняння наявної документації та активної спільноти

Spring Framework існує вже протягом багатьох років і набув великого популярності та прихильності серед розробників. Через це у Spring є широка спільнота, активні форуми, блоги, статті, підручники та інші ресурси, які надають багато інформації та підтримку для розробників. Багато програм і проєктів розроблені на основі Spring, що свідчить про його широке використання та популярність у реальних сценаріях.

Хоча Quarkus і Micronaut також мають активні форуми, GitHub-репозиторії, документацію та інші ресурси, що надають підтримку розробникам, їх спільнота може бути меншою порівняно з Spring. Вони все ще є новішими проектами, які набувають популярності у розробницькій спільноті. Це означає, що менша кількість програм що використовують ці фреймворки, і менша кількість компаній вже мають досвід роботи з ними порівняно з Spring.

Цей факт може вплинути на інтеграцію нових фреймворків та їх використання, оскільки багато розробників вже мають досвід роботи з Spring.

Вони можуть відчувати схильність до використання вже знайомого і перевіреного досвідом фреймворку, а не вводити нові технології.

Отже, через те, що Quarkus і Micronaut новіші, у них менша кількість документації та спільноти, менша кількість компаній що вже мали досвід роботи з ними та менша популярність. Однак, з часом ці фреймворки можуть зрости в популярності і набути більшої кількості ресурсів та підтримки, що робить їх біль привабливими для розробників, які бажають використовувати нові технології та експериментувати з ними.

2.6. Переваги та недоліки кожного фреймворку

У цьому підрозділі розглянуто переваги та недоліки кожного з розглянутих фреймворків: Spring MVC, Spring Webflux, Quarkus і Micronaut. Звернено увагу на те, які переваги вони пропонують у розробці веб-додатків, а також на проблеми, які можуть виникати при їх використанні і способи їх вирішення.

2.6.1. Spring MVC:

Переваги:

- Широке поширення та велика спільнота розробників, що забезпечує багато ресурсів та підтримку.
- Висока стабільність та тривалість на ринку.
- Простота використання та зрозуміла документація.
- Широкий набір інтегрованих інструментів та бібліотек.

Недоліки:

- Більш традиційний підхід, що може бути обмежуючим для більш сучасних архітектур та сценаріїв.

- Синхронний стиль програмування, що може призводити до блокувань та неефективного використання ресурсів.

2.6.2. Spring Webflux:

Переваги:

- Реактивний підхід, що дозволяє ефективно обробляти багато запитів одночасно та зменшує блокування.
- Підтримка нестандартних протоколів та розширень, таких як WebSocket.
- Інтеграція з реактивними бібліотеками та фреймворками.

Недоліки:

- Вимагає особливих знань та досвіду для ефективного використання реактивного підходу.
- Обмежена підтримка деяких інтегрованих інструментів та бібліотек, які не підтримують реактивну модель.

2.6.3. Quarkus:

Переваги:

- Швидкість запуску та відгуку, що підходить для мікросервісної архітектури.
- Низьке споживання пам'яті та ресурсів.
- Підтримка гарячого перезапуску та інші інструменти для продуктивної розробки.

Недоліки:

- Обмежена підтримка деяких інтегрованих бібліотек та інструментів, які потребують специфічного налаштування.

- Можливі проблеми з сумісністю та підтримкою старших версій фреймворків.

2.6.4. Micronaut:

Переваги:

- Швидкість та ефективність, особливо при роботі з обмеженими ресурсами.
- Низьке споживання пам'яті та мінімальні затримки під час відгуку.
- Вбудована підтримка обробки залежностей та інструменти для швидкої розробки.

Недоліки:

- Менша популярність та менша спільнота розробників, що може призвести до обмежених ресурсів підтримки та документації.
- Обмежена підтримка деяких інтегрованих бібліотек та інструментів, які вимагають додаткового налаштування.

РОЗДІЛ 3: ТЕСТУВАННЯ ТА ВИМІРЮВАННЯ ХАРАКТЕРИСТИК

3.1. Опис застосунків та способів тестування

3.1.1. Структура проєктів

Кожен з фреймворків має свій власний проєкт з однаковою функціональністю. Структура кожного з проєктів складається з трьох основних компонентів: сервісу, фабрики сервісу та контролера.

Сервіс відповідає за логіку обробки запитів, фабрика сервісу створює екземпляр сервісу з налаштованими параметрами, а контролер визначає шлях та обробляє HTTP-запити до кінцевої точки API (endpoint, ендпоїнт).

3.1.2. Маршрут

В кожному з проєктів присутня однакова структура з одним маршрутом. Ось опис цього ендпоїнту, використовуючи наданий код:

URL-шлях: /hello/{target}
HTTP-метод: GET
Параметри запиту:

- target (шляховий параметр) - це ціль, до якої буде застосоване привітання.

Логіка ендпоїнту:

- Ендпоїнт отримує параметр target, який визначає ціль для привітання.
- За допомогою сервісу, який приймає початкове привітання в конструкторі, генерується відповідь, яка складається з привітання та цілі.
- Згенерована відповідь повертається як результат ендпоїнту.

Цей маршрут призначений для прийому запитів у форматі /hello/{target}, де {target} може бути будь-яким рядком. При запиті на цей ендпоїнт, він поверне відповідь, що складається з привітання та заданої цілі.

Однак, важливо відзначити, що ці структура та логіка ендпоїнту не є принциповими для порівняння фреймворків. Їх основна мета полягала в створенні рівних умов для порівняння продуктивності, використання пам'яті, часу запуску та інших вимірюваних характеристик фреймворків.

3.1.3. Формат тестування

З метою порівняння фреймворків, кожен проєкт має однакові тести. Тести перевіряли правильність повернення привітання для певного цільового значення.

Тестування було проведено за допомогою відповідних фреймворків та методів тестування. Були використані наступні способи тестування:

- Micronaut: У тесті MicronautExampleTest використовується аннотація @MicronautTest, що дозволяє інтегрувати тестове середовище Micronaut. Тест виконує HTTP-запит до веб-додатку за допомогою DefaultHttpClient, передає шлях запиту та очікувану відповідь. Після цього порівнюються отримані результати з очікуваними.
- Quarkus: Для тестування Quarkus використовується фреймворк Quarkus Test, який надає підтримку для інтеграційних тестів. Деталі виконання тесту аналогічні до Micronaut тесту.
- Spring MVC і Spring WebFlux: Для тестування використовується фреймворк Spring Test, який надає підтримку для інтеграційних тестів. Логіка тестів аналогічна до інших.

3.2. Тестування та оцінка результатів

3.2.1. Час компіляції та виконання тестів

У цьому підрозділі порівнюються час компіляції та виконання тестів для кожного фреймворку. Результати тестування та пояснення таких результатів наведені нижче:

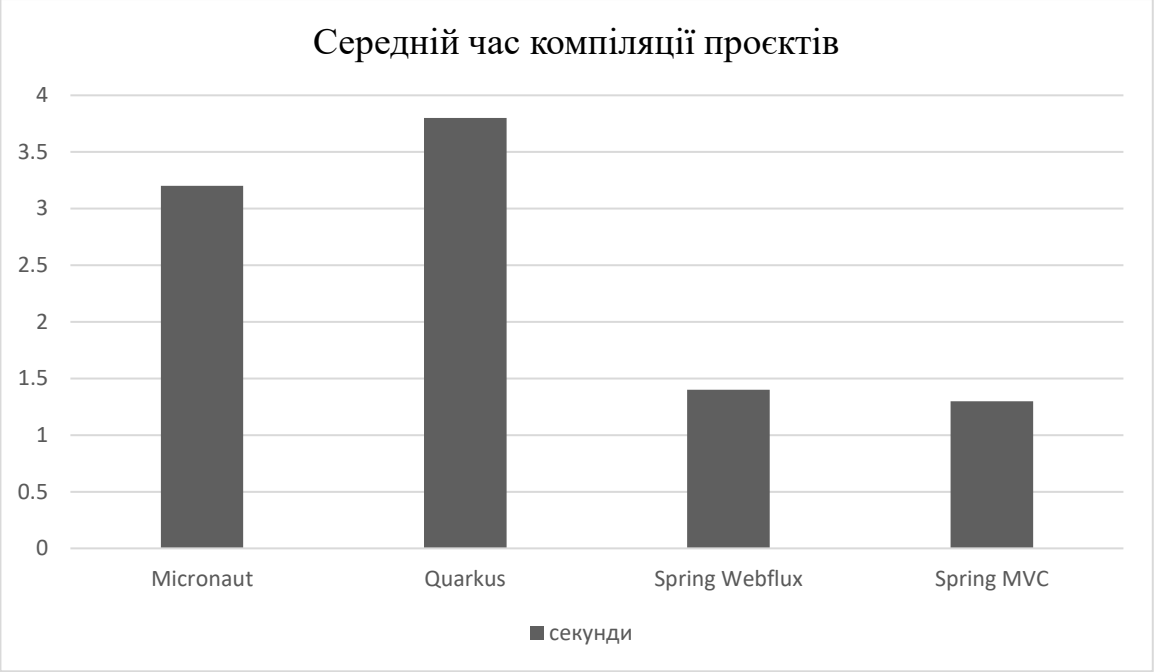

Рис. 1 Середній час компіляції проєктів

Micronaut та Quarkus використовують оптимізацію на етапі компіляції, виконуючи аналіз коду під час компіляції, що дозволяє виконувати певні оптимізації коду. Цей підхід може збільшити кількість часу витраченого на етапі компіляції, але компенсується підвищеною продуктивністю під час роботи програми.

Насупроти цьому, Spring Webflux та Spring MVC не проводять такі оптимізації на етапі компіляції, вони покладаються на інші механізми, такі як рефлексія. Це призводить до швидшого часу компіляції, але може негативно вплинути на продуктивність під час виконання додатка.

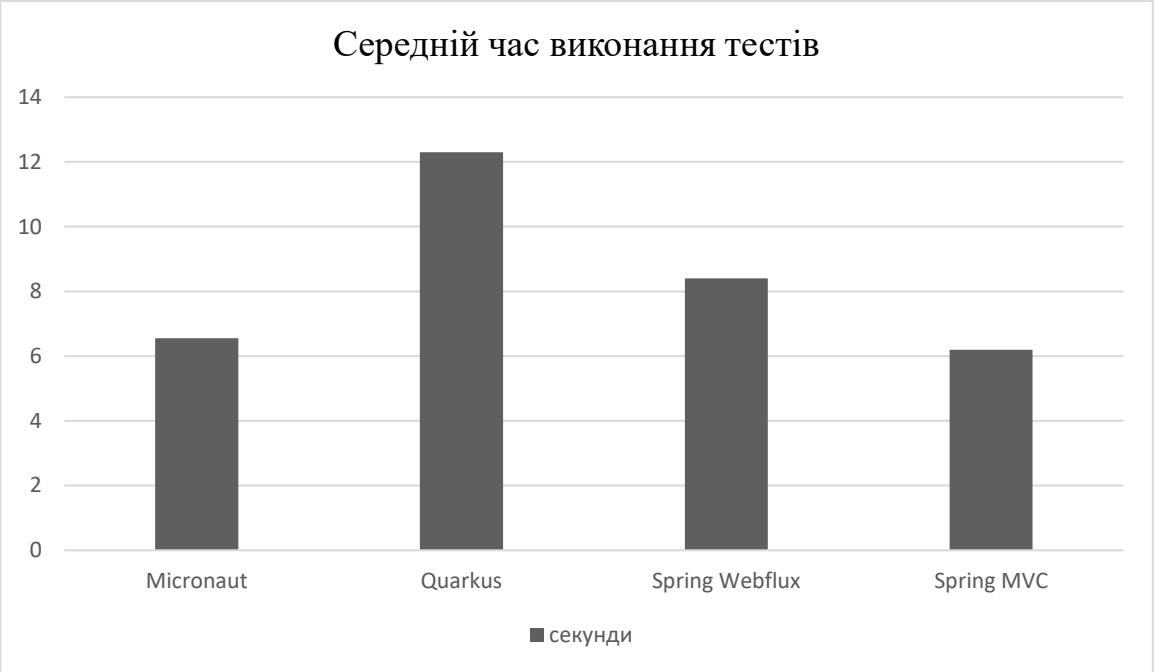

Рис. 2 Середній час виконання тестів

Один із можливих факторів, який може впливати на тривалість виконання тестів у Quarkus, є сам процес Ahead-of-Time (AoT) компіляції, який використовується в Quarkus для створення нативних виконуваних файлів.

Цей підхід може забезпечити кращу продуктивність та швидкодію додатків Quarkus у виробничому середовищі, але в той же час може впливати на тривалість виконання тестів. Залежно від розміру та складності проєкту, виконання AoT компіляції може займати певний час.

Важливо зазначити, що швидкодія та продуктивність під час реальної роботи додатків Quarkus можуть бути значно покращені завдяки AoT компіляції. Тому, хоча виконання тестів може займати більше часу, саме виробниче середовище може виявитися більш ефективним та швидкодійним.

3.2.2. Час запуску

У цьому підрозділі будуть порівнюватися час запуску додатків для кожного фреймворка у двох режимах: режимі розробки (dev mode) та режимі релізу (release mode). Нижче наведені результати та причини таких результатів.

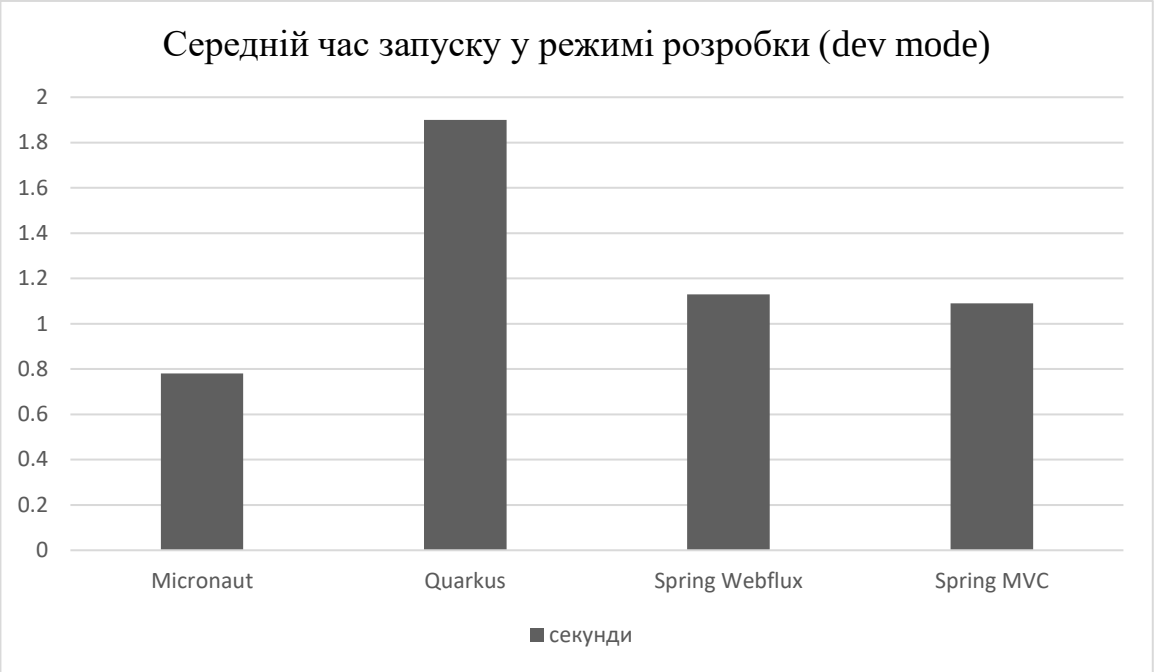

Рис. 3 Середній час запуску у режимі розробки

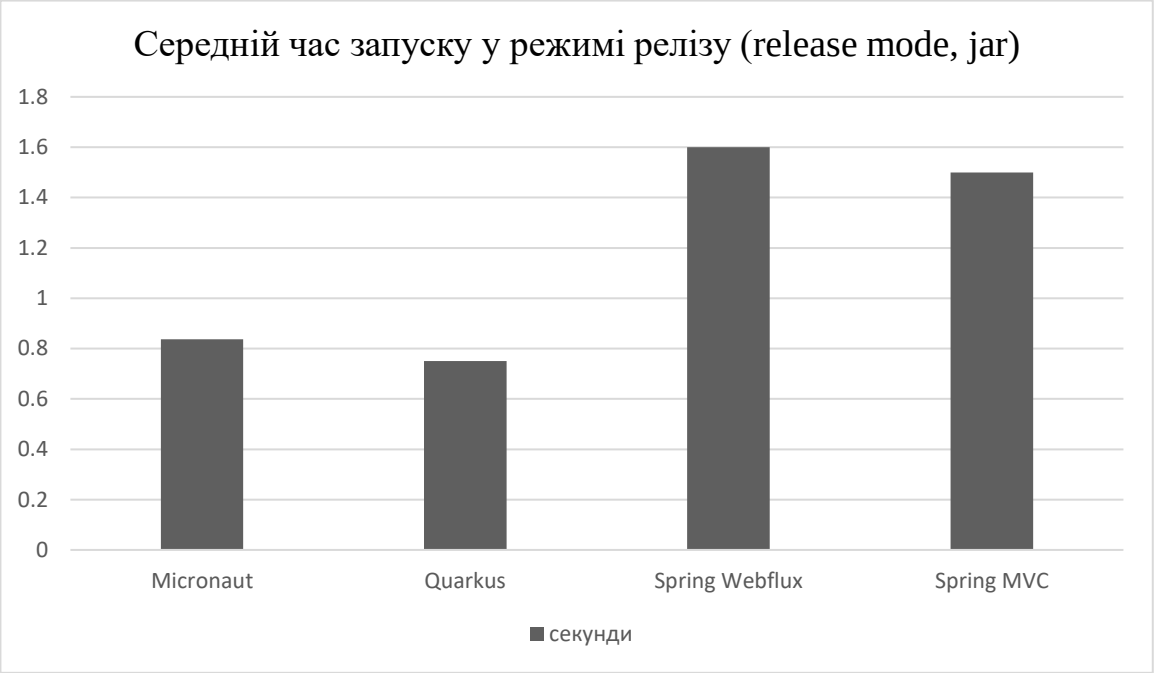

Рис. 4 Середній час запуску у режимі релізу

У режимі розробки (dev mode) час запуску Quarkus може бути довший через використання JVM. Однак, у режимі релізу (release mode) Quarkus показує швидший час запуску завдяки використанню AoT та GraalVM. Ця особливість

Quarkus може бути перевагою для додатків, які потребують високої продуктивності та швидкості запуску у режимі релізу.

Порівнюючи час запуску додатків, можна помітити, що Micronaut та Quarkus мають найменший час запуску, Spring Webflux та Spring MVC є повільнішими.

3.2.3. Час до першої відповіді

Час до отримання першої відповіді є важливою метрикою продуктивності, яка визначає, скільки часу займає процес від запуску додатка до отримання першої відповіді від сервера. Нижче наведені порівняльні дані щодо часу першої відповіді між фреймворками:

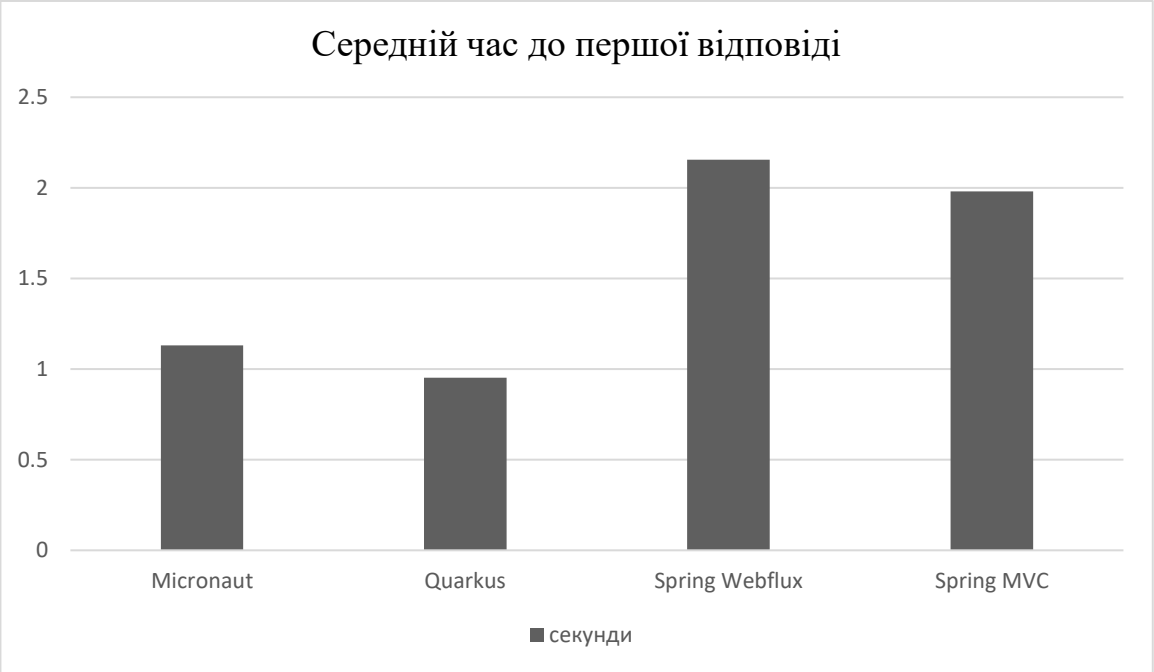

Рис. 5 Середній час до першої відповіді

На основі цих результатів можна зробити такі спостереження:

Quarkus і Micronaut показують найшвидший час першої відповіді, вдвічі менший за Spring MVC та Spring Webflux.

Також ці результати свідчать про те, що Quarkus має найшвидший час першої відповіді після запуску додатку. Це може бути важливим фактором для додатків, які потребують миттєвої реакції на запити зразу після запуску.

3.2.4. Продуктивність запитів

Продуктивність запитів була виміряна за допомогою утиліти ab (Apache Bench) на одному маршруті для кожного з фреймворків.

Spring Webflux не був включений у вимірювання продуктивності запитів через те, що він не підтримує "keep-alive" (постійне з'єднання) за замовчуванням. У вимірюваннях продуктивності запитів, де використовується утиліта ab (Apache Bench), "keep-alive" є важливим фактором, оскільки він дозволяє повторне використання з'єднання між запитами, що покращує продуктивність і зменшує накладні витрати на встановлення нового з'єднання для кожного запиту. Проте, варто зазначити, що Spring Webflux може бути використаний для реалізації реактивних додатків і мати свої переваги в інших аспектах, таких як обробка багатьох одночасних запитів.

Результати показують кількість запитів, яку кожен фреймворк здатен обробити за одну секунду. Ось отримані результати:

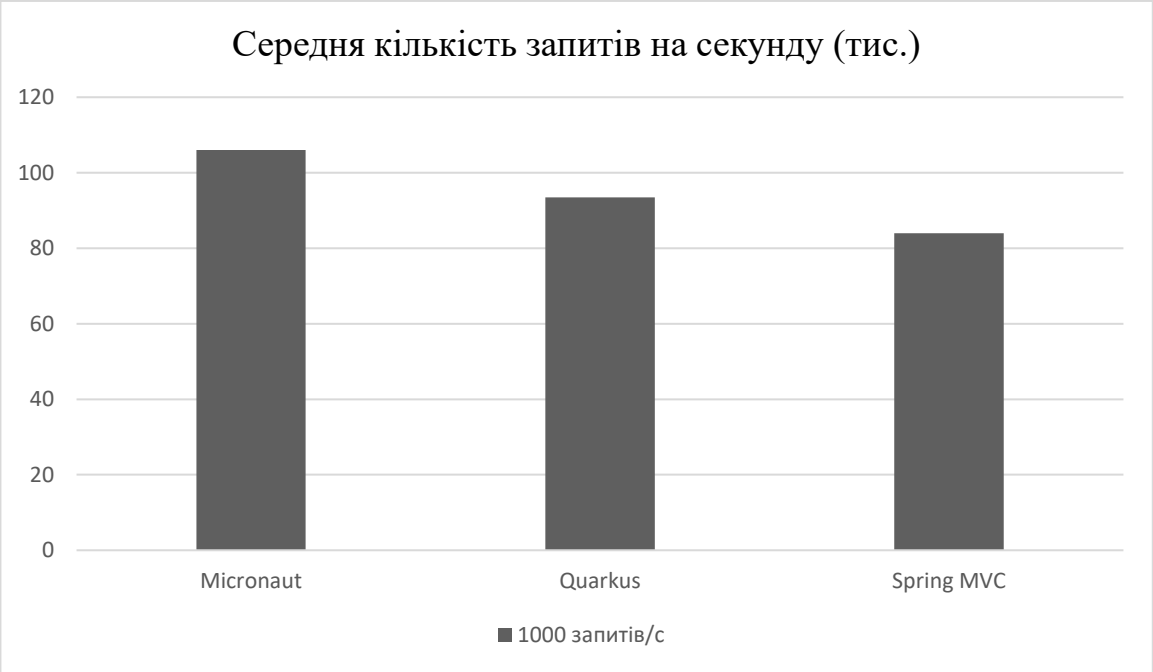

Рис. 6 Середня кількість запитів на секунду

Ці результати свідчать про різну продуктивність обробки запитів у кожному з фреймворків. Micronaut показав найкращі показники, здатність обробити найбільшу кількість запитів на секунду, залишаючи позаду інші фреймворки. Spring WebMVC та Quarkus також продемонстрували гарні показники, але трохи менші порівняно з Micronaut.

Це порівняння продуктивності дозволяє визначити, який фреймворк може ефективно обробляти більший обсяг запитів і забезпечити швидку відповідь на них. При розробці додатків, де кількість запитів на секунду грає важливу роль, вибір фреймворка з кращою продуктивністю може мати велике значення для досягнення оптимальної швидкодії.

3.2.5. Використання пам’яті

У цьому підрозділі проведено вимірювання споживання пам'яті після запуску фреймворків та відправки 1 мільйона запитів (середні значення багатьох тестів). Нижче наведені розміри пам'яті для кожного фреймворка:

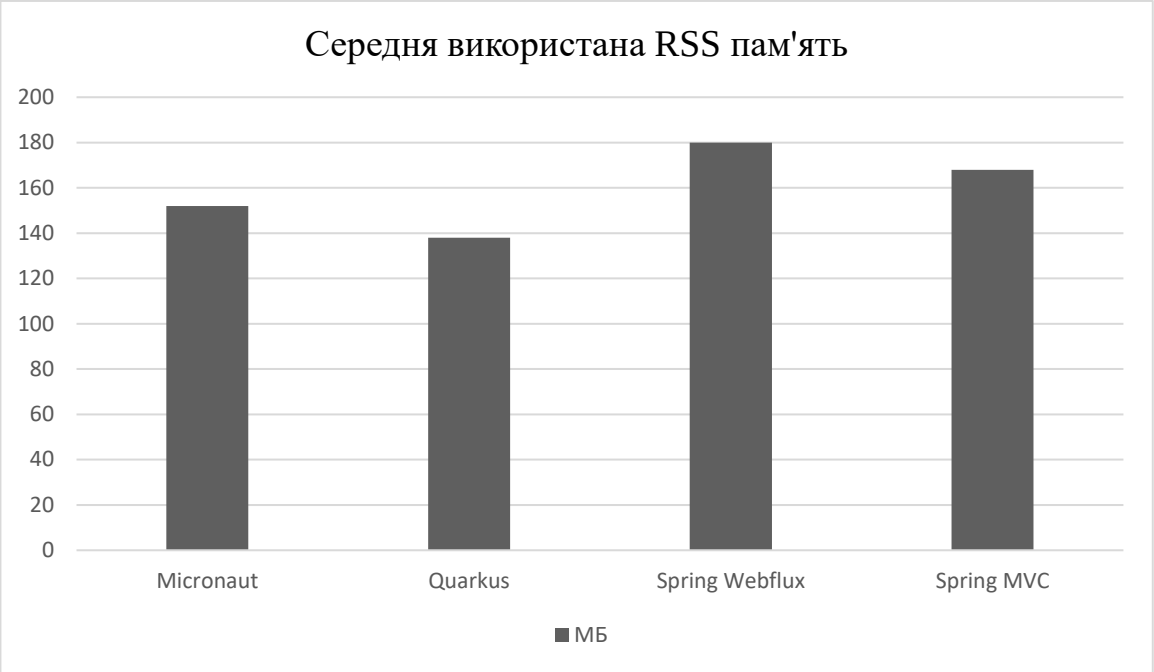

Рис. 7 Середня використана пам’ять

Зазначені результати вимірювання вказують на різницю у використанні пам'яті між різними фреймворками. Micronaut та Quarkus продемонстрували менше споживання пам'яті, витрачаючи в середньому від 138 до 152 MB. У той же час, Spring WebFlux та Spring WebMVC показали більше споживання пам'яті, з розмірами від 168 до 180 MB.

Micronaut та Quarkus відомі своєю ефективністю та низьким споживанням ресурсів, що може пояснити їх нижче споживання пам'яті. З іншого боку, Spring WebFlux та Spring WebMVC, хоча можуть бути потужними та гнучкими фреймворками, можуть вимагати більше ресурсів для своєї роботи, що впливає на використання пам'яті.

3.2.6. Вплив обмеження розміру пам’яті на продуктивність

Вимірювання були проведені з використанням обмеження розміру пам'яті (тегу -Xmx18m при запуску у режимі релізу) для кожного фреймворка. Оцінено вплив цього обмеження на продуктивність кожного фреймворка:

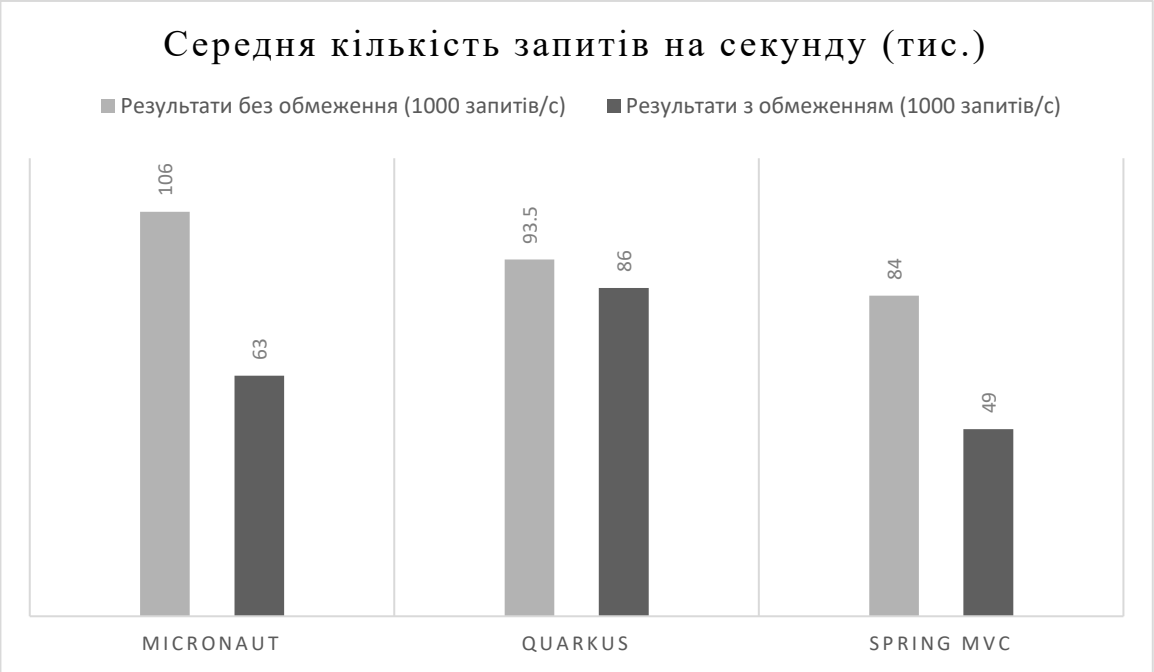

Рис. 8 Вплив обмеження розміру пам’яті на продуктивність

Результати вказують на те, що Quarkus показав найбільшу стабільність продуктивності при обмеженому розмірі пам'яті. Він втратив лише близько 9% продуктивності порівняно зі швидкістю без обмеження пам'яті. Micronaut також демонстрував добру стійкість і зазнав втрати приблизно 25% продуктивності. Натомість, Spring WebMVC показав найбільші втрати продуктивності, знизившись аж на 42% у швидкодії.

ВИСНОВКИ

В результаті виконання цієї роботи було визначено що у кожного з досліджуваних фреймворків є свої переваги і недоліки, які варто враховувати при виборі інструментарію для розробки веб-додатків.

Spring MVC – широко використовуваний і стабільний фреймворк з великою спільнотою розробників. Він простий у використанні, але він може бути обмежений для сучасних архітектур і сценаріїв.

Spring Webflux пропонує реактивний підхід, що дозволяє ефективно обробляти багато запитів одночасно. Цей фреймворк має підтримку нестандартних протоколів і розширень, але вимагає додаткових знань та досвіду для ефективного використання.

Quarkus – це швидкий та ефективний фреймворк, особливо для мікросервісної архітектури. Він має низьке споживання пам’яті та широкі можливості для продуктивної розробки, однак, може мати обмежену підтримку деяких бібліотек і вимагати специфічного налаштування.

Micronaut є альтернативою до Quarkus, також відомий своєю швидкістю та ефективністю. Однак, Micronaut також має меншу популярність та спільноту, що може обмежити доступ до ресурсів і допомоги.

З порівняння фреймворків було визначено, що Micronaut і Quarkus перевершують Spring Webflux і Spring MVC у швидкості виконання, продуктивності, споживанні пам’яті і часі запуску, і програвали лише у часі компіляції та тестів, і це було лише через те, що Webflux та WebMVC не проводять аналіз коду та оптимізації на етапі компіляції, тому і займають менше часу.

Отже, кожен фреймворк має свою нішу і може бути підходящим для конкретних вимог і сценаріїв розробки. Вибір оптимального фреймворку повинен базуватися на потребах проєкту, зручності для розробників та їхньому знайомстві з інструментарієм.

ВИКОРИСТАНІ ДЖЕРЕЛА

1. Official Spring Framework site [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/.
2. Spring Framework Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.spring.io/spring-framework/reference/.
3. Official Spring Boot documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/.
4. Official Spring WebMVC documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html.
5. Official Spring Webflux documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.spring.io/spring-framework/docs/5.0.0.M5/spring-framework-reference/html/web-reactive.html.
6. Official Quarkus site [Електронний ресурс] – Режим доступу до ресурсу: https://quarkus.io/.
7. Official Quarkus documentation [Електронний ресурс] – Режим доступу до ресурсу: https://quarkus.io/guides/.
8. Official Micronaut site [Електронний ресурс] – Режим доступу до ресурсу: https://micronaut.io/.
9. Official Micronaut documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.micronaut.io/.