JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 23 (2022) 66-70
Received: 19 October 2021
Accepted: 5 November 2021

Comparative analysis of selected programming frameworks of Java-based web applications

Analiza porównawcza wybranych szkieletów programistycznych aplikacji webowych opartych na języku Java

Radosław Książek*, Beata Pańczyk

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
This article presents the results of a comparative analysis of Spring (with Spring Boot configuration), Micronaut and Quarkus programming frameworks. The recently observed increase in popularity of these solutions made it necessary to determine their application profile. In order to determine the characteristics of the researched technologies, a series of performance and optimization tests of applications built on the basis of the above-mentioned programming frameworks were carried out. The results of the analyzes showed that thanks to the high degree of optimization, the Micronaut and Quarkus frameworks are perfectly adapted to work in cloud environments, while Spring (Boot) framework, despite its lower efficiency, is an irreplacible solution in complex projects.

Keywords: comparative study; Spring Boot; Micronaut; Quarkus

Streszczenie
Tematem artykułu jest analiza porównawcza szkieletów programistycznych Spring (z konfiguracją Spring Boot), Micronaut oraz Quarkus. Obserwowany w ostatnim czasie wzrost popularności tych rozwiązań uwarunkował konieczność wyznaczenia ich profilu zastosowań. W celu wyznaczenia charakterystyk badanych technologii, przeprowadzono szereg testów wydajnościowych oraz optymalizacyjnych aplikacji zbudowanych w oparciu o wymienione szkielety programistyczne. Rezultaty analiz wykazały, iż dzięki wysokiemu stopniu optymalizacji, szkielety Micronaut oraz Quarkus są doskonale przystosowane do pracy w środowiskach chmurowych, natomiast rozbudowany szkielet Spring (Boot), pomimo mniejszej wydajności, jest pozycją niezastąpioną w bardziej złożonych projektach.

Słowa kluczowe: analiza porównawcza; Spring Boot; Micronaut; Quarkus

*Corresponding author
Email address: radoslaw.ksiazek@pollub.edu.pl (R. Książek)

©Published under Creative Common License (CC BY-SA v4.0)

1. Wstęp

Rozwój rozbudowanych aplikacji internetowych spełniających szereg złożonych reguł biznesowych jest zagadnieniem pochłaniającym dużą ilość zasobów czasowych oraz ludzkich. Pożądany w dzisiejszych czasach wysoki stopień optymalizacji procesu planowania i implementacji takich serwisów istotnie redukuje koszty związane z wytwarzaniem, bądź utrzymywaniem oprogramowania. Dokonane przez autora pracy poszukiwania dostępnych płaszczyzn optymalizacji wspomnianych procesów wykazały, iż znaczący wpływ na łączny czas projektowania i opracowywania serwisów ma szkielet programistyczny wybrany do realizacji danego projektu.

2. Cel i zakres badań

Celem badań jest porównanie następujących szkieletów programistycznych:

- Spring MVC (z konfiguracją Spring Boot),
- Micronaut,
- Quarkus

w kontekście implementacji przykładowej aplikacji internetowej typu CRUD w technologii REST, w języku Java (wersja 15).

Zakres badań obejmuje dokonanie analizy porównywanych technologii z uwzględnieniem różnic strukturalnych i funkcjonalnych. Każdy szkielet programistyczny poddany został testom wydajnościowo-obciążeniowym, których rezultaty poddano analizie w sekcjach końcowych artykułu.

3. Przedmioty badań

Jako obiekty badań wybrano trzy z najpopularniejszych szkieletów programistycznych stanowiących podstawę rozwoju aplikacji napisanych w języku Java [1].

Pierwszą analizowaną pozycją jest Spring MVC (wydany w 2002r.) z bibliotekami konfiguracyjnymi Spring Boot (wydanymi w roku 2014.). Głównym celem tego szkieletu było uproszczenie procesu rozwoju aplikacji internetowych poprzez natywne wspieranie paradygmatu odwróconego sterowania (ang. IoC, Inversion of Control) oraz stosowania mechanizmów wstrzykiwania zależności [2]. Szkielet ten był przez długi czas najpopularniejszym następnikiem domeny JavaEE (ang. Java Enterprise Edition) i technologii EJB (ang. Enterprise JavaBeans).

W czasach wzrostu popularności architektury mikroserwisowej, zaszła potrzeba optymalizacji istniejących rozwiązań. Organizacja Micronaut Foundation, dostrze-

gając kluczowe wady w istniejących szkieletach programistycznych, stworzyła projekt Micronaut. Główną innowacją, którą wprowadzał ten szkielet było natywne wsparcie kompilacji projektu do obrazu kontenera platformy GraalVM [3]. Mechanizm ten znacznie zmniejsza zapotrzebowanie aplikacji na zasoby pamięciowe oraz przyczynił się do istotnej redukcji czasu uruchamiania serwisów. W celu zwiększenia stopnia optymalizacji szkieletu wdrożono koncepcję rozszerzonego przetwarzania kodu na etapie kompilacji z jednoczesnym odstąpieniem od stosowania mechanizmów refleksji [4].

Ostatnią z uwzględnionych w badaniach pozycji jest Quarkus. Szkielet ten został stworzony przez organizację RedHat w 2019r. jako odpowiedź na dynamicznie rosnące zainteresowanie technologiami chmurowymi. Głównym założeniem twórców szkieletu Quarkus była minimalizacja zasobów zużywanych przez opracowywane z jego wykorzystaniem aplikacje [5] (m.in. poprzez kompilację do natywnych obrazów kontenerów technologii GraalVM [6]) oraz natywne wsparcie dla stosu Kubernetess [7].

4. Analiza literatury

Z uwagi na długi okres aktywności projektu Spring na rynku deweloperskim aplikacji internetowych dostępne są liczne publikacje odnoszące się do zagadnień wydajności i funkcjonalności tego szkieletu programistycznego. W przypadku szkieletów Quarkus oraz Micronaut, dostępna baza literatury jest niewielka.

Dostępne porównania możliwości implementacji architektury REST w aplikacjach opartych m.in. o szkielet Spring Boot [8] wskazują na rozbudowaną pulę modułów wspierających obsługę zapytań w omawianej konwencji wymiany informacji.

W pracy Sayagh M. et al. [9] autorzy wskazują na wysoki stopień wsparcia konfiguracji wielu aspektów projektu (m.in. ustawienia warstwy repozytoriów oraz kontekstu aplikacji) przez szkielet konfiguracyjny Spring Boot.

Kwestie optymalizacji środowisk uruchomieniowych aplikacji poruszane są stosunkowo rzadko. Jedna z pozycji nawiązujących do tego zagadnienia ("Enhancing Performance of Cloud-based Software Applications with GraalVM and Quarkus") [10] odnosi się do aspektu wsparcia implementacji rozwiązań chmurowych w technologii GraalVM oraz powiązanym z nią szkieletem Quarkus. Rezultaty badań tej pracy podkreślają zwiększoną wydajność oprogramowania zbudowanego w oparciu o szkielety programistyczne wspierające technologię tworzenia obrazów natywnych aplikacji.

Przeprowadzone studia literaturowe wskazują na brak kompleksowych analiz szkieletów Micronaut oraz Quarkus. Niniejsza praca ma na celu uzupełnienie tego obszaru badań o szczegółowe zestawienia porównywanych technologii z uwzględnieniem analiz funkcjonalności poszczególnych szkieletów programistycznych.

5. Metoda badań

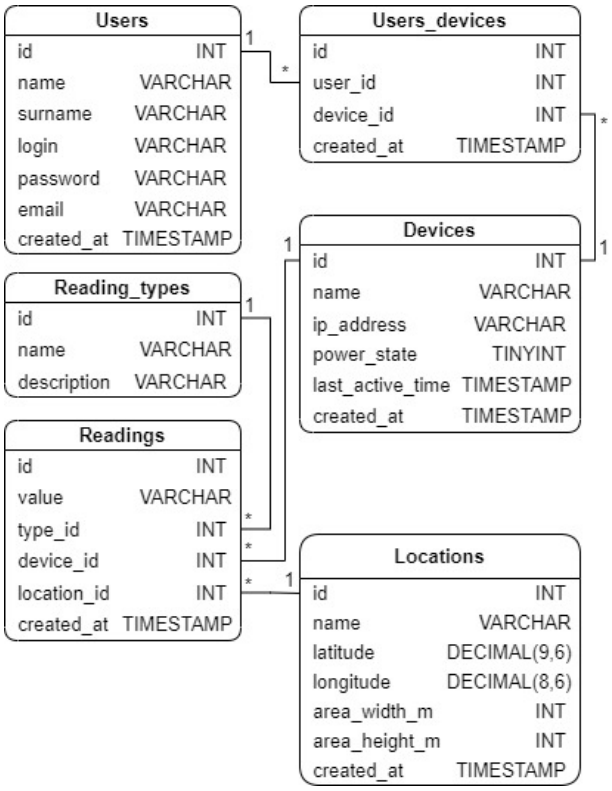
Eksperyment polegał na zaprojektowaniu oraz implementacji referencyjnej aplikacji internetowej typu REST API, umożliwiającej zarządzanie systemem pomiarów opartym o rozproszoną sieć czujników i powiązanych z nimi urządzeń.

W celu wyznaczenia wartości kryteriów przyrównania szkieletów programistycznych, dla każdej z porównywanych technologii przeprowadzone zostały testy wydajnościowe oraz ogólna analiza struktury kodu opracowanych aplikacji i użytych w nich funkcjonalności. Eksperyment został wykonany według scenariuszy testowych przedstawionych w Tabeli 1.

W celu zapewnienia odpowiedniej jakości rezultatów uzyskanych w toku dokonanej analizy, porównywane projekty aplikacji zostały zaimplementowane według opracowanej specyfikacji aplikacji referencyjnej. Zdefiniowana w niej prototypowa struktura projektu, encje oraz zachodzące między nimi relacje (Rys. 1) pozwoliły na zachowanie spójności porównywanych aplikacji w kontekście ich wymagań funkcjonalnych.

Tabela 1: Lista scenariuszy testowych realizowanych w ramach przeprowadzanej analizy szkieletów programistycznych

Lp.	Nazwa	Jednostka miary rezultatów	Opis
1.	Rozmiar pliku wykonywalnego	liczba megabajtów [MB]	Badanie rozmiaru pliku wykonywalnego będącego artefaktem końcowym procesu budowy projektu
2.	Czas budowania projektu	czas [s]	Badanie czasu kompilowania i pakowania kodu źródłowego projektu
3.	Czas uruchamiania	czas [s]	Badanie czasu uruchomienia aplikacji (do momentu obsługi pierwszych żądań)
4.	Zużywana pamięć operacyjna	liczba megabajtów [MB]	Badanie zużytej przez proces aplikacji, systemowej pamięci RAM
5.	Wydajność operacji odczytu z bazy danych	czas [s] wykonania 1 000 odczytów	Test polegający na odczycie 1 000 wpisów z encji ”readings”
6.	Wydajność operacji zapisu do bazy danych	czas [s] wykonania 1 000 zapisów	Test polegający na zapisie 1 000 wpisów do encji ”readings”
7.	Czas pierwszej odpowiedzi (treść statyczna/dynamiczna)	czas [s]	Badanie czasu wymaganego do przetworzenia pierwszej odpowiedzi aplikacji na żądanie od momentu jej uruchomienia (dla odpowiedzi o treści statycznej oraz dynamicznej)
8.	Liczba zapytań / sekundę	liczba	Badanie liczby zapytań możliwych do obsłużenia przez aplikację w ciągu jednej sekundy



Rysunek 1: Schemat przedstawiający zaprojektowaną strukturę bazy danych.

6. Eksperyment

Aplikacje opracowane w oparciu o analizowane szkielety programistyczne poddane zostały testom wydajnościowym oraz optymalizacyjnym zgodnie z wyznaczonymi scenariuszami testowymi. W Tabeli 2. przedstawiono konfigurację środowiska testowego, na którym dokonywane były pomiary.

Tabela 2: Specyfikacja środowiska testowego

Wersja JDK	15
Procesor	AMD Ryzen 4600H (3.0GHz, 6 rdzeni/12 wątków)
Pamięć operacyjna	16 GB (DDR4, 3200MHz)
Dysk	SSD 250GB (PCIe NVMe, PCIe 3.0 x4)
System operacyjny	Windows 10

7. Rezultaty

Rezultaty badania rozmiaru plików wykonywalnych aplikacji (Tabela 3.) wskazały, iż największy rozmiar pliku osiągnęła aplikacja oparta o szkielet Spring (Boot). Uzasadnieniem tego wyniku jest duża liczba dołączonych do projektu aplikacji bibliotek. Rozmiar plików wykonywalnych projektów opartych o szkielety Micronaut oraz Quarkus był zbliżony i wynosił niecałe 30MB.

Tabela 3: Rozmiary plików wykonywalnych aplikacji opracowanych w poszczególnych szkieletach programistycznych

Szkielet programistyczny	Spring (Boot)	Micronaut	Quarkus
Rozmiar pliku wykonywalnego [MB]	38,03	29,91	29,59

Eksperyment weryfikujący średni czas budowania projektów wykazał, iż proces ten trwał najkrócej w przypadku aplikacji opartej o szkielet Quarkus (11,43s). Najdłuższym czasem budowania projektu (14,53s) cechował się projekt wykorzystujący szkielet Spring (Boot). Otrzymane rezultaty przedstawiono w Tabeli 4.

Tabela 4: Średnie czasy budowania projektów aplikacji opracowanych w poszczególnych szkieletach programistycznych

Szkielet	Spring (Boot)	Micronaut	Quarkus
Czas budowania projektu [s]	14,53	12,02	11,43

Testy wykonane w oparciu o scenariusz rozpatrujący czasy uruchamiania aplikacji (Tabela 5.) wykazały, iż najszybciej uruchamiała się aplikacja oparta o szkielet Quarkus (3,24s). Jest to wynik o 1,71s mniejszy od rezultatu najwyższego, uzyskanego dla aplikacji wykorzystującej szkielet Spring (Boot) – 4,95s.

Tabela 5: Czasy uruchamiania aplikacji opracowanych w poszczególnych szkieletach programistycznych

Szkielet	Spring (Boot)	Micronaut	Quarkus
Czas uruchamiania aplikacji [s]	4,95	3,77	3,24

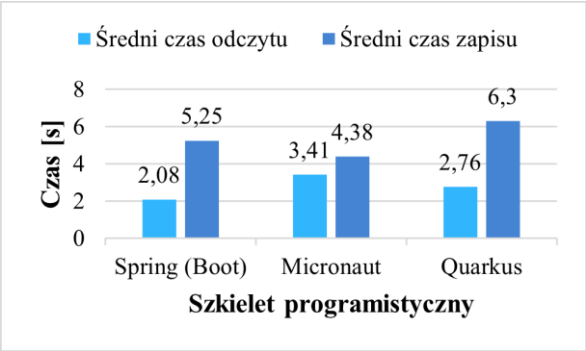
Badanie użycia pamięci operacyjnej przez aplikacje wykazało, iż największe zapotrzebowanie na pamięć RAM (ang. Random Access Memory) wykazała aplikacja oparta o szkielet Spring (Boot) (36,6MB). Najmniejszym użyciem pamięci charakteryzowała się aplikacja wykorzystująca technologię Micronaut (23,5MB). Wyniki badania przedstawione zostały w Tabeli 6.

Tabela 6: Zużywana pamięć operacyjna przez aplikacje opracowane w poszczególnych szkieletach programistycznych

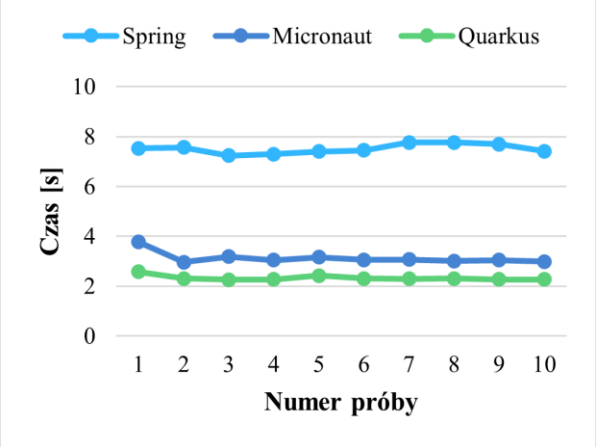
Szkielet	Spring (Boot)	Micronaut	Quarkus
Użyta pamięć operacyjna [MB]	36,6	23,5	24,8

Średni czas odczytu danych z bazy był najmniejszy dla aplikacji opartej o szkielet Spring (Boot) i wynosił 2.08s, zaś najmniejszy średni czas zapisu danych został odnotowany dla aplikacji zbudowanej na podstawie szkieletu Micronaut - 4,38s (Rys 2.).

Badania czasu pierwszej odpowiedzi przedstawiono na Rys 3. Aplikacja oparta o szkielet Quarkus wykazała najkrótszy czas pierwszej odpowiedzi (średnio około 2s), podczas gdy najdłuższe czasy zwłoki były generowane przez aplikację wykorzystującą technologię Spring (Boot) (niecałe 8s).



Rysunek 2: Rezultaty badania czasu trwania wymiany informacji z bazą danych.



Rysunek 3: Rezultaty badania czasu pierwszej odpowiedzi aplikacji.

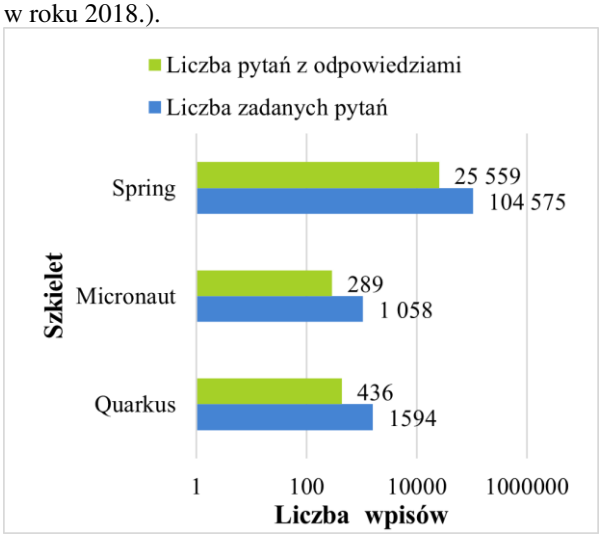
Badanie liczby zapytań obsłużonych przez aplikacje w ciągu sekundy (Tabela 7.) wykazało, iż najwięcej zapytań zostało obsłużonych przez serwis zbudowany w oparciu o szkielet Micronaut (8 962 zapytania). Aplikacja opracowana na podstawie szkieletu Spring (Boot) obsłużyła najmniej zapytań (6 558) spośród analizowanych pozycji.

Tabela 7: Wyniki badania liczby zapytań obsługiwanych przez aplikacje w ciągu jednej sekundy

Szkielet programistyczny	Spring (Boot)	Micronaut	Quarkus
Liczba zapytań na sekundę	6 558	8 962	8 560

W ramach weryfikacji dodatkowego kryterium porównawczego analizowanych szkieletów programistycznych przeprowadzono analizę stopnia wsparcia porównywanych technologii przez społeczność deweloperów. W tym celu przeprowadzono badanie liczby pytań oraz odpowiedzi związanych z konkretnymi tagami (w tym przypadku nazwami szkieletów programistycznych) w serwisie „stackoverflow.com". Rezultaty analiz przedstawiono na Rys 4.

Statystyki odnotowane dla szkieletu Spring (Boot) znacznie przewyższały wyniki pozostałych pozycji w zestawieniu. Jest to spowodowane istotnie dłuższym okresem egzystencji tego szkieletu na rynku deweloperskim aplikacji internetowych. Faktem wartym zaznaczenia w tym kontekście jest większe zainteresowanie zagadnieniami dotyczącymi szkieletu Quarkus (wydanym w roku 2019.) niż Micronaut (pierwsze wydanie w roku 2018.).



Rysunek 4: Statystyki odpowiedzi na pytania odnoszące się do zagadnień związanych z poszczególnymi szkieletami programistycznymi na stronie stackoverflow.com.

8. Wnioski

Przeprowadzone badania wykazały, iż przewodnim zastosowaniem technologii Spring MVC z konfiguracją Spring Boot jest implementacja aplikacji o wysokim poziomie złożoności. Wysoka dostępność kompatybilnych, zewnętrznych bibliotek zwiększa możliwości integracji dodatkowych funkcjonalności w aplikacji [11], a wysokie wsparcie społeczności ułatwia proces tworzenia oprogramowania mniej doświadczonym deweloperom. Rezultaty analiz wydajnościowych i optymalizacyjnych wskazały, iż szkielet Spring (Boot) w wielu scenariuszach był najmniej wydajnym rozwiązaniem spośród innych porównywanych technologii.

Szkielet Micronaut jest wysoce zoptymalizowanym rozwiązaniem, doskonale wpisującym się w infrastruktury systemów informatycznych opartych o architekturę mikroserwisową. Potwierdzają to przeprowadzone testy wydajnościowe, w których technologia ta zajmowała czołowe pozycje. Jedną z głównych cech specyficznych tego szkieletu programistycznego na tle porównywanych pozycji jest unikanie wykorzystywania refleksji i rozszerzone przetwarzanie kodu podczas kompilacji projektu [12]. Mechanizm ten znacznie optymalizuje zużycie zasobów przez działającą aplikację.

Jednym z głównych profili zastosowań szkieletu Quarkus jest tworzenie aplikacji wdrażanych w ekosystem technologii chmurowych. Dokonane analizy wydajnościowo-optymalizacyjne, w większości scenariuszy testowych, wykazały najwyższy stopień optymalizacji zasobów zużywanych przez działającą aplikację w porównaniu do pozostałych, badanych szkieletów programistycznych. Cechy te sprawiają, iż technologia Quarkus może w istotnej mierze przyczyniać się do redukcji kosztów związanych z rozwojem oraz utrzymaniem aplikacji internetowych działających w chmurze.

Literatura

[1] Analiza ankiety dotyczącej dziesięciu najpopularniejszych szkieletów programistycznych języka Java w roku 2020. https://jaxenter.com/java-trends-top-10-frameworks-2020-168867.html , [19.10.2021].

[2] M. Moises, Learn Microservices with Spring Boot: A Practical Approach to Restful Services Using RABBITMQ, Eureka, Ribbon and Cucumber, APRESS, 2018.

[3] T. W. Pusztai, T. Christos, D. Schahram, Engineering Heterogeneous Internet of Things Applications: From Models to Code. 2019 IEEE 5th International Conference on Collaboration and Internet Computing (CIC) (2019) 222-231, https://doi.org/10.1109/cic48465.2019.00036.

[4] N. Singh, Z. Dawood, Building Microservices with Micronaut: A Quick-Start Guide to Building High-Performance Reactive Microservices for Java Developers, Birmingham: Packt Publishing, 2021.

[5] F. Marchioni. Hands-on Cloud-Native Applications with Java and QUARKUS: Build High Performance Java Microservices on Kubernetes. PACKT Publishing Limited, 2019.

[6] S. A. Bueno, J. Porter, Quarkus Cookbook: Kubernetes-Optimized Java Solutions. Beijing: O'Reilly, 2020.

[7] T. Koleoso, Beginning Quarkus Framework Build Cloud-Native Enterprise Java Applications and Microservices, Berkeley, CA: Apress, 2020.

[8] R. Kwiatkowski, P. Kopniak, Comparison of Capabilities to Implement REST Services in Java Language Using the Popular Web Application Frameworks, Journal of Computer Sciences Institute 6 (2018) 92-96, https://doi.org/10.35784/jcsi.648.

[9] M. Sayagh, Z. Dong, A. Andrzejak, B. Adams, Does the Choice of Configuration Framework Matter for Developers? Empirical Study on 11 Java Configuration Frameworks, In 2017 IEEE 17th International Working Conference on Source Code Analysis and Manipulation (SCAM) (2017) 41-50.

[10] M. Šipek, D. Muharemagić, B. Mihaljević, A. Radovan, Enhancing Performance of Cloud-based Software Applications with GraalVM and Quarkus, In 2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO) (2020) 1746-1751.

[11] Lista oficjalnych projektów-bibliotek Spring. https://spring.io/projects , [17.10.2021].

[12] A. B. Kumar, Vijay, Supercharge Your Applications with Graalvm: Hands-on Examples to Optimize and Extend Your Code Using GRAALVM's High Performance and Polyglot Capabilities, Birmingham: Packt Publishing, 2021.