JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 29 (2023) 352–359
Received: 27 June 2023
Accepted: 5 July 2023

Performance analysis of REST API technologies using Spring and Express.js examples

Analiza wydajności technologii tworzenia REST API na przykładzie Spring i Express.js

Maciej Wicha*, Beata Pańczyk

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

Abstract

The purpose of this article is a comparative analysis of two technologies for building applications in REST architecture. A Java-based development framework - Spring, and a framework designed for JavaScript language and Node environment - Express.js were analyzed. The test application was designed and implemented in both studied technologies. Using the Apache JMeter tool, HTTP request processing times were measured by operating on simple text data. The experiment was based on 5 scenarios repeated for a different number of users in the range of 10 to 100, with a constant number of executed requests to the server. The analysis conducted showed that the application implemented in Express.js handles HTTP requests up to 249% more efficiently than its counterpart in Spring.

Keywords: Spring; Express.js; REST API; performance benchmarking

Streszczenie

Tematem niniejszego artykułu jest analiza porównawcza dwóch technologii do budowania aplikacji w architekturze REST. Badania dotyczą opartego na języku Java szkieletu programistycznego - Spring oraz szkieletu przeznaczonego dla języka JavaScript i środowiska Node – Express.js. Aplikację testową zaimplementowano w obu badanych technologiach. Przy wykorzystaniu narzędzia Apache JMeter dokonano pomiaru czasów przetwarzania żądań HTTP operując na prostych danych tekstowych. Eksperyment opierał się na 5 scenariuszach powtórzonych dla różnej liczby użytkowni (od 10 do 100), przy stałej częstotliwości wykonywanych zapytań do serwera. Przeprowadzone analizy pozwoliły określić, że aplikacja zaimplementowana w Express.js obsługuje żądania HTTP nawet o 249% sprawniej niż jej odpowiednik w Spring.

Słowa kluczowe: Spring; Express.js; REST API; analiza wydajności

*Corresponding author

Email address: **maciejwicha8@gmail.com** (M. Wicha)

1. Wstęp

Pojawienie się smartfonów łączących funkcje telefonu komórkowego i komputera zrewolucjonizowały wiele aspektów naszego życia. Standardem stało się projektowanie trzech wariantów aplikacji – dostępnej dla urządzeń z systemem Android, iOS oraz w klasycznej postaci WEB.

W przypadku aplikacji działających w trybie online, pojawia się problem pobierania danych, tak aby na wszystkich trzech platformach dostępne były te same treści. Naturalnym zabiegiem zdaje się być zaimplementowanie jednej centralnej aplikacji udostępniającej dane do wszystkich aplikacji klienckich.

Nieodłącznym zjawiskiem w wytwarzaniu różnego rodzaju aplikacji jest integracja z istniejącymi już systemami lub uwzględnienie jej w przyszłości. Twórcy portali społecznościowych udostępniają API swoich aplikacji, aby inni mogli z niego korzystać poszerzając standardowe funkcjonalności macierzystej aplikacji. Odpowiednie usługi pozwalające np. udostępniać posty za pomocą odpowiednich punktów końcowych udostępniają autorzy zarówno Facebooka [1] jak i Twittera [2].

Aplikacje serwerowe w technologii REST rozwiązują wszystkie powyższe kwestie. Jedna aplikacja serwerowa jest w stanie obsłużyć wiele aplikacji klienckich, bez względu na przeznaczenie, środowisko czy funkcjonalności.

Wśród wielu dostępnych technologii, istnieje jeszcze więcej szkieletów programistycznych ułatwiających pracę nad tego typu aplikacjami. Ich wybór nie należy do najprostszych i uzależniony jest od wielu czynników, takich jak stopień skomplikowania projektu czy jego przeznaczenie. Oprócz szeregu funkcji, rozbudowanej społeczności dzielącej się rozwiązaniami i wspólnie pomagającym wyeliminować pojawiające się błędy, każda technologia posiada również swoje wady.

Celem niniejszej pracy jest analiza porównawcza dwóch popularnych technologii tworzenia aplikacji internetowych w architekturze REST – Spring i Express, pod kątem popularności oraz czasów przetwarzania żądań HTTP odpowiadającym podstawowym operacjom CRUD.

W artykule postawiono hipotezę badawczą, że technologia Express zapewnia wydajniejsze rozwiązanie do tworzenia REST API niż Spring.

2. Przegląd literatury

Marcin Grudniak wraz z Mariuszem Dzieńkowskim w artykule *Porównanie wydajności aplikacji internetowych REST API opartych na szkieletach programistycznych JavaScript* opublikowanym na łamach JCSI poddali analizie porównawczej dwa szkielety programistyczne Express.js i Hapi [3]. Autorzy przygotowali aplikację testową, a następnie opracowali 4 scenariusze testowe, w ramach których poddano badaniom 4 podstawowe typy operacji HTTP (POST, PUT, GET, DELETE). Każdy ze scenariuszy operował na innym typie danych, w eksperymencie uwzględniono łańcuchy znaków, tablicę 100 elementów zawierających 100 znakowe losowe ciągi znaków, obiekt o 100 atrybutach oraz 50 elementowej tablicy zawierającej 50 atrybutów każdy po 50 losowych znaków. Aplikacja utrwalała dane w nierelacyjnej bazie danych MongoDB. Czasy przetwarzania żądań zostały zebrane za pomocą modułu wewnątrz-serwerowego oraz za pomocą aplikacji zewnętrznej rejestrującej czas odpowiedzi. Na podstawie przeprowadzonych badań autorzy doszli do wniosku, że aplikacja zbudowana w oparciu o szkielet Express.js osiągnęła lepsze wyniki niż jej odpowiednik w Hapi. Jedynie dla operacji na tablicy obiektów, czasy obu technologii były do siebie zbliżone.

Celem artykułu *A performance comparision of RESTful Applications Implemented in Spring Boot Java and MS.NET Core* autorstwa Hardeep Kaur Dhalla jest porównanie wydajności aplikacji typu REST przy użyciu dwóch różnych szkieletów programistycznych opierających się na języku Java i C# [4]. Obie aplikacje były implementacją tych samych procesów biznesowych. Do zebrania wyników autor wykorzystał narzędzie Apache JMeter w wersji 5.2.1. W trakcie przeprowadzonego przeglądu literatury autor natknął się na badania wskazujące lepszą wydajność aplikacji opartej o Java EE Struts niż VB.NET. Jako kryterium oceny wydajności aplikacji autor obrał średni czas odpowiedzi oraz procentowy błąd przy obsłudze żądania. Dodatkowo monitorowano również zużycia zasobów takich jak CPU czy pamięć RAM. Scenariusze testowe zakładały wzrost liczby aktywnych użytkowników od 1000 do 6400. Uzyskane wyniki pozwoliły stwierdzić, że aplikacja zaimplementowana w MS.NET Core zapewnia krótsze czasy odpowiedzi, oraz zużywa mniej zasobów takich jak CPU czy RAM w stosunku do aplikacji utworzonej przy użyciu Java Spring Boot.

W artykule *Performance Comparision of Java EE and ASP.NET Core Technologies for Web API Development* opublikowanej w Applied Computer Systems przez Kristians Kronis i Marina Uhanova poruszono analizę porównawczą szkieletów ASP.NET Core i Java EE w wersji 7 [5]. Autorzy zwrócili uwagę na istotność niezmienności środowiska testowego przez cały okres przeprowadzania eksperymentu, dlatego aplikacje zostały wdrożone na serwer VPS (ang. Virtual Private Server), który przez cały okres prowadzenia pomiarów utrzymywał ten sam stan (aktualizacje, wersja oprogramowania, zainstalowane programy). Jako miejsce przechowywania danych przez aplikacje testowe wybrano bazę danych MySQL, a dane przesyłane za pomocą API miały format JSON. Scenariusze badawcze dotyczyły m.in. wysyłania i odbierania krótkich ciągów tekstowych, odbierania danych o zadanym rozmiarze i generowaniu liczb pseudolosowych, operacji arytmetycznych na 1000 losowo wybranych liczbach oraz operacji na rozbudowanych plikach JSON. Analiza uzyskanych wyników pozwoliła określić, że o ile ASP.NET Core uzyskał krótsze czasy przetwarzania żądań to sam serwer Kestrel potrzebował więcej czasu na wysłanie odpowiedzi niż Tomcat.

Performance Comparision and Evaluation of Web Development Technologies in PHP, Python and Node.js autorstwa Kai Lei, Yinng Ma oraz Zhi Tan w kompleksowy sposób porównuje trzy różne technologie tworzenia aplikacji internetowych [6]. Jako narzędzie pomiarowe wykorzystali oni dwa programy ApacheBench oraz LoadRunner dla testów obciążeniowych. Testy przeprowadzono dla 10, 100, 200, 500 i 1 000 użytkowników. Materiałem badawczym były trzy proste aplikacje: klasyczny program wyświetlający „Hello World!", program wyznaczający *n*-ty wyraz ciągu Fibbonacciego (autorzy wybrali 10, 20 i 30 wyraz) oraz program dokonujący operacji *select* na bazie danych MySQL. Uzyskane wyniki wskazały, że aplikacja zaimplementowana za pomocą PHP i Python obsługuje znacznie mniej żądań niż Node.js w określonym czasie. Dodatkowo autorzy pokusili się o wskazanie przeznaczenia każdej z technologii, dla aplikacji napisanej w języku Python zaproponowano duże rozbudowane aplikacje biznesowe, podczas gdy dla języka PHP małe i średnie przedsiębiorstwa. Technologię Node.js polecają natomiast dla stron wykorzystujących intensywnie operacje wejścia-wyjścia.

3. Badane technologie

Analizie porównawczej poddano dwa popularne szkielety programistyczne Java Spring (z konfiguracją Spring Boot) oraz Express.js.

3.1. Szkielet programistyczny Spring

Spring framework to od lat jeden z najpopularniejszych szkieletów programistycznych do tworzenia aplikacji w języku Java [7].

Jednym z popularnych modułów Spring jest Spring Boot, który w prosty sposób pozwala skonfigurować i wdrożyć aplikację. Spring Initializr pozwala wygenerować początkowy szablon aplikacji Spring wraz z dodatkowymi zależnościami.

Wśród wielu cech i zalet szkieletu Spring wymienić można między innymi:

- Wstrzykiwanie zależności (ang. Dependency Injection – DI) – jest to mechanizm wyręczający programistę w ręcznym tworzeniu obiektów. Szkielet automatycznie wstrzykuje odpowiednie zależności.
- Model MVC (ang. Model-View-Controler) – obsługa wzorca projektowego pozwalającego na budowę aplikacji internetowych, zapewniającego gotowe komponenty do obsługi żądań HTTP, obsługę for-

mularzy, zarządzanie sesją użytkownika i generowanie widoków.
- Modularność – aplikacje Spring można zintegrować z wieloma narzędziami i bibliotekami, pozwalającymi na szybsze i sprawniejsze programowanie, np. Spring Security (konfiguracja zabezpieczeń aplikacji, autoryzacji), Spring Data (zarządzanie danymi w aplikacji), Thymeleaf (silnik szablonów HTML) [8].

3.2. Szkielet programistyczny Express.js

Express.js to minimalistyczny szkielet programistyczny języka JavaScript przeznaczony na platformę Node.js zyskujący popularność w ostatnich latach [9]. W swoim założeniu jest prostym i elastycznym narzędziem do tworzenia aplikacji internetowych, zapewniając prostszy sposób obsługi żądań HTTP niż sam Node.js. Podobnie jak Spring, udostępnia narzędzie do generowania podstawowego szablonu aplikacji. Razem z Angular.js, Node.js oraz MongoDB należy do tzw. MEAN Software Stack, czyli zestawu technologii i narzędzi napisanych w języku JavaScript pozwalających na kompleksowe pisanie aplikacji internetowych [10].

Wśród zalet wykorzystania szkieletu Express.js wymienić można:
- Trasowanie (ang. routing) – intuicyjny system routingu, pozwalający programistom prosty sposób definiować ścieżki URL i ich obsługę poprzez specjalne funkcje (tzw. middleweare).
- Funkcje pośredniczące (ang. middleware) – funkcje wywoływane sekwencyjnie podczas przetwarzania żądań HTTP przesyłanych do serwera aplikacji. Odpowiadają za obsługę żądań, wysyłania odpowiednich wiadomości zwrotnych w postaci danych lub zakończenia przetwarzania żądań.
- Model MVC (ang. Model-View-Controler, model-widok-kontroler) – podobnie jak Spring, zapewnia obsługę wzorca projektowego MVC ułatwiającego rozwój aplikacji i zarządzanie kodem aplikacji.
- Prostota i elastyczność – w przeciwieństwie do Spring, aplikacja zaimplementowana z wykorzystaniem szkieletu Express nie posiada narzuconej struktury projektu. Pomimo, że w dokumentacji znajduje się zalecana struktura projektu to cały kod aplikacji może znajdować się w jednym pliku – twórcy nie narzucają w tej kwestii żadnych ograniczeń, a większość standardów wyznacza społeczność [10].

3.3. Popularność technologii w latach 2019-2023

Zarówno Express.js, jak i Spring zdobyły przez lata grono swoich zwolenników i przeciwników. Wyniki corocznej ankiety „Developer Survey" przeprowadzanej przez Stack Overflow widoczne w Tabeli 1. wskazują, że popularność obu technologii jest do siebie zbliżona. W latach 2019-2023 Express.js jest nieznacznie bardziej popularny wśród profesjonalnych programistów pracujących na pełnym etacie niż jego odpowiednik Spring.

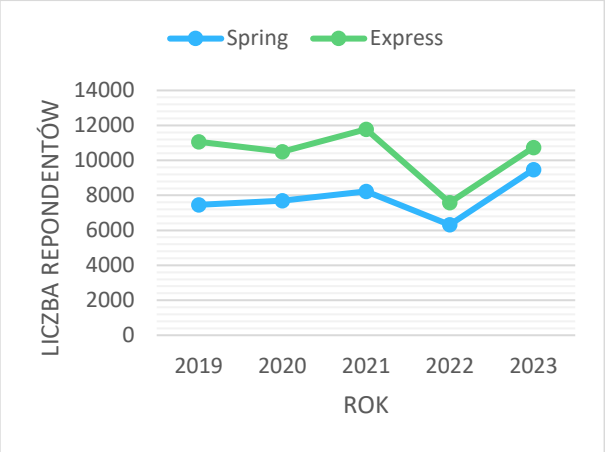
W przypadku Spring zauważyć można, że w latach 2019-2021 liczba respondentów rosła (Rysunek 1), gdy popularność Express zachowywała się niemonotonicznie – w roku 2020 spadła z 11 070 głosów na 10 504, by w kolejnym roku zanotować wzrost o 12%.

Tabela 1: Popularność Spring i Express - wyniki ankiety przeprowadzonej przez Stack Overflow w latach 2019-2023 [7-9, 11-13]

ROK	Spring	Express
2019	7 463	11 070
2020	7 695	10 504
2021	8 226	11 780
2022	6 315	7 585
2023	9 474	10 740

W roku 2022 obie technologie straciły na popularności osiągając najniższe wyniki w badanym okresie (Spring - 6 315 głosów, Express – 7 585). Dla Express był to spadek o 36% w stosunku do roku poprzedniego.

Pomimo tego, obie technologie nadal znajdują się w czołówce popularności szkieletów programistycznych przeznaczonych dla aplikacji internetowych.



Rysunek 1: Popularność badanych szkieletów programistycznych w latach 2019-2023 na podstawie ankiety przeprowadzonej przez Stack Overflow [7-9, 11-13].

4. Metoda badań

Zebranie materiału badawczego polegało na przeprowadzeniu eksperymentu, w ramach którego testowano odpowiedniki operacji CRUD dla protokołu HTTP (POST, GET, PUT, DELETE) dla zaimplementowanych aplikacji. Następnie wykorzystując statystykę opisową przeanalizowano zebrane wyniki.

4.1. Scenariusze badawcze

Przy użyciu narzędzia Apache JMeter zdefiniowano 5 scenariuszy testowych, które pozwoliły zbadać czasy odpowiedzi aplikacji i przetwarzania żądań HTTP.

Ogólnymi założeniami dla wszystkich testów były:
- stała liczba zapytań wysyłana do serwera REST (1000 zapytań na minutę);
- każdy scenariusz powtarzano 10 krotnie, dla różnej liczby użytkowników (od 10 do 100);
- każdy z użytkowników próbuje jednocześnie odczytać swoje dane;
- użytkownik jest zautoryzowany i ma wygenerowany token JWT;

- użytkownik próbuje odczytać tylko dane, do których ma dostęp.

Poszczególne scenariusze zostały zdefiniowane następująco:

1. *Scenariusz I – metoda POST* – polegający na tworzeniu przez użytkownika 100 rekordów, przekazując pełen obiekt zadania. Przy każdym żądaniu zmianie ulegała data ukończenia zadania ustawiana na aktualną datę.
2. *Scenariusz II – metoda GET dla pojedynczego elementu* – użytkownik pobierał pojedyncze zadanie z bazy danych, operację powtarzał dla każdego ze 100 zadań.
3. *Scenariusz III – metoda GET dla 100 elementów* – użytkownik pobierał wszystkie dodane zadania 100 krotnie.
4. *Scenariusz IV – metoda PUT* – użytkownik aktualizował wartość kolumny „isCompleted" z wartości false na true. Operację wykonywał dla każdego ze 100 zadań. W ciele zapytania przekazywano tylko aktualizowaną wartość.
5. *Scenariusz V – metoda DELETE* – użytkownik usuwał każde ze 100 zadań.

4.2. Platforma testowa

W Tabeli 2. przedstawiono konfigurację środowiska testowego, na którym dokonywane były pomiary, oraz na których uruchomione były wszystkie 3 usługi – aplikacja kliencka (scenariusz w narzędziu Apache JMeter), aplikacja serwerowa (aplikacja „to-do" w technologii Express lub Spring) oraz baza danych (MariaDB za pośrednictwem narzędzia XAMPP).

Tabela 2: Środowisko testowe

NPM	9.5.1
JDK	17
Procesor	Intel Core i7-9750H (2.60 GHz, 6 rdzeni, 12 wątków)
RAM	2x16GB (DDR4, 2667 MHz)
Dysk #1	SSD 1TB 2.5" SATA
Dysk #2	SSD 512 GB M.2 PCIe
System operacyjny	Windows 10 Pro

4.3. Narzędzie Apache JMeter

Apache JMeter jest rozbudowanym narzędziem typu open-source wyprodukowanym przez Apache Software Foundation, pozwalającym na wykonywanie testów wydajności różnych usług, w tym aplikacji internetowych. Do jego zalet zaliczyć można:

- symulowanie dostępu do aplikacji przez wielu użytkowników jednocześnie;
- pomiar dynamicznych i statycznych parametrów (np. czas odpowiedzi serwera, jego obciążenie lub przepustowość sieci, itp.);
- testowanie wielu protokołów i usług (m.in. HTTP, HTTPS, REST, SOAP, SMTP, itp.);
- samodzielne budowanie scenariuszy testowych przez użytkownika;
- tworzenie scenariuszy za pomocą interfejsu graficznego lub poprzez specjalny plik;
- dostęp do wielu wtyczek (oficjalnych i tworzonych przez społeczność) rozszerzających podstawowe funkcjonalności narzędzia.

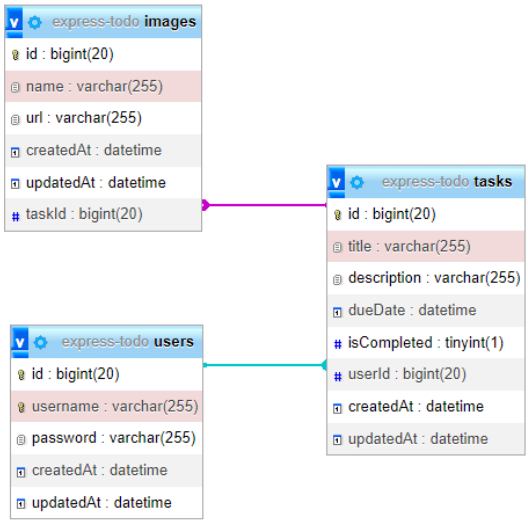
5. Materiał badawczy

Materiałem badawczym była prosta aplikacja w architekturze REST zaimplementowana w dwóch wybranych technologiach w następujących wersjach – Spring 3.0.4 oraz Express 4.16.1.

5.1. Aplikacja testowa

Zaimplementowano aplikację „to-do" o strukturze bazy danych widocznej na Rysunku 2. Dla każdej z testowanej aplikacji baza danych tworzona była przez badany szkielet programistyczny przy użyciu dodatkowych modułów (np. dla Express.js – Sequalizer, dla Spring – JPA).

Aplikacja w swoim założeniu pozwala na rejestrację i logowanie użytkownika, oraz dodawanie i zarządzanie zadaniami. Odpowiednie punkty końcowe wymagają autoryzacji. Wszystkie wymagania funkcjonalne i niefunkcjonalne zawarte zostały w kolejnych podrozdziałach.



Rysunek 2: Struktura bazy danych wygenerowana przez moduł Sequelizer (Express.js).

5.2. Wymagania funkcjonalne

Określenie wymagań funkcjonalnych było istotnym elementem planowania eksperymentu w celu zachowania spójności pomiędzy obiema wersjami aplikacji testowej.

1. Użytkownik
   - 1.1. Logowanie
     - 1.1.1. Weryfikacja poprawności danych
     - 1.1.2. Wygenerowanie JWT
   - 1.2. Rejestracja
     - 1.2.1. Weryfikacja poprawności danych
     - 1.2.2. Weryfikacja unikalności loginu
     - 1.2.3. Rejestracja użytkownika

2. Zadanie
   2.1. Dodanie zadania
      2.1.1. Uzupełnienie tytułu
      2.1.2. Uzupełnienie opisu
      2.1.3. Uzupełnienie daty wykonania zadania
      2.1.4. Uzupełnienie statusu zadania
      2.1.5. Utrwalenie zadania w bazie danych
   2.2. Edycja zadania
      2.2.1. Weryfikacja identyfikatora zadania
      2.2.2. Zmiana tytułu
      2.2.3. Zmiana opisu
      2.2.4. Zmiana daty wykonania zadania
      2.2.5. Zmiana statusu zadania
      2.2.6. Utrwalenie zmian w bazie danych
   2.3. Pobranie zadania
      2.3.1. Wskazanie zadania do wyświetlenie
      2.3.2. Weryfikacja identyfikatora zadania
      2.3.3. Zwrócenie danych w postaci JSON
   2.4. Pobranie wszystkich zadań
      2.4.1. Zwrócenie danych w postaci JSON
   2.5. Usunięcie zadania
      2.5.1. Wskazanie zadania do usunięcia
      2.5.2. Weryfikacja identyfikatora zadania
      2.5.3. Zwrócenie komunikatu o statusie

5.3. Wymagania niefunkcjonalne

Oprócz funkcjonalności zdefiniowano również ograniczenia aplikacji określające architekturę i format przesyłanych danych.

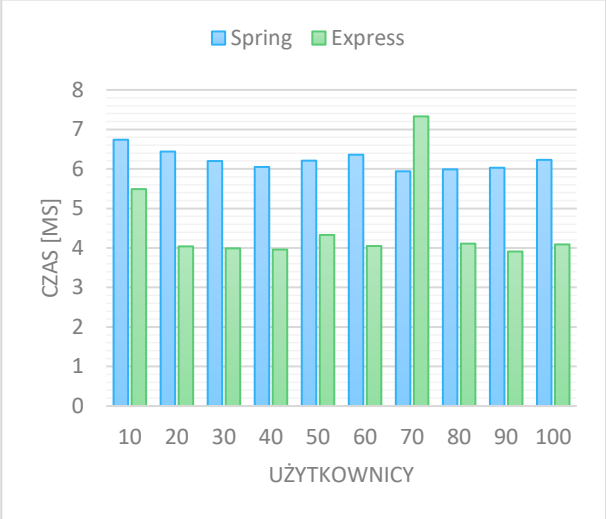
Dostępność systemu:

a) serwis dostępny jest jako aplikacja REST,
b) każdej funkcjonalności odpowiadają odpowiednie punkty końcowe (ang. endpoints),
c) komunikaty zwracane przez aplikację mają postać zgodną z formatem JSON.

6. Wyniki badań

6.1. Scenariusz I – metoda POST

Na Rysunku 3 przedstawiono wyniki, będące średnią arytmetyczną uzyskanych wyników dla poszczególnych iteracji testu odpowiednio od 10 do 100 użytkowników.



Rysunek 3: Średnie czasy przetwarzania żądania POST.

W przeważającej większości Express.js uzyskuje niższe czasy przetwarzania żądania POST odpowiedzialnego za utworzenie nowego rekordu w bazie danych. Jedynie dla 70 użytkowników potrzebuje 23% więcej czasu niż Spring.

Tabela 3 przedstawia szczegółowe informacje na temat wybranych parametrów dla 50 użytkowników. Pomimo, że maksymalny czas, który Express.js potrzebuje na przetworzenie żądania wynosi 43,00 ms, a Spring jedynie 25,00 to zarówno średnia jak i mediana jest niższa dla Express i wynosi odpowiednio 4,33 ms i 4,00 ms.

Tabela 3: Wartości parametrów dla metody POST – 50 użytkowników

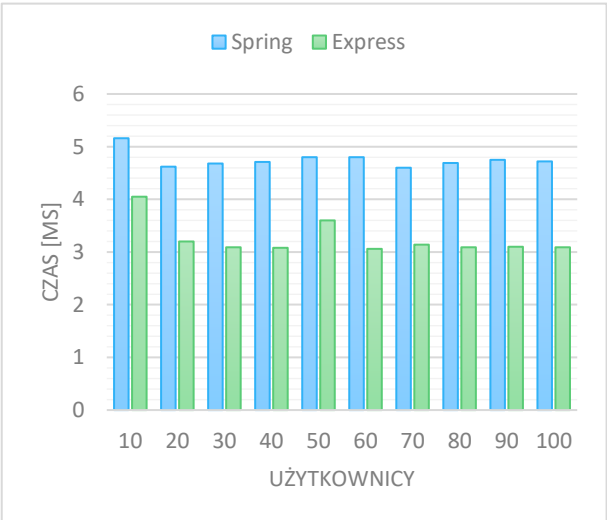
Parametr	Spring	Express
Średnia (ms)	6,21	4,33
Wartość maksymalna (ms)	25,00	43,00
Wartość minimalna (ms)	4,00	3,00
Odchylenie standardowe (ms)	1,04	1,55
Mediana (ms)	6,00	4,00

6.2. Scenariusz II – metoda GET dla pojedynczego obiektu

Średnie czasy przetwarzania żądania z metodą GET dla pojedynczego obiektu wskazanego poprzez parametr adresu URL przekazywanego do aplikacji testowej widoczne są na Rysunku 4.

Podobnie jak w przypadku tworzenia nowego rekordu, również i przy odczycie danych Spring radzi sobie gorzej od Express. Obie technologie potrzebują jednak najwięcej czasu w przypadku obsługi 10 użytkowników (Spring – 5,16 ms, Express – 4,05 ms).

Najkrótszy średni czas obsługi żądania dla Express występował dla 60 użytkowników (3,06 ms) oraz 70 użytkowników dla Spring (4,6 ms).



Rysunek 4: Średnie czasy przetwarzania żądania GET dla pojedynczego elementu.

W Tabeli 4 porównano statystyki dla 50 użytkowników. W tym przypadku dla obu technologii mediana jest sobie równa i wynosi 5,00 ms. Podobnie jak w przypadku poprzedniego scenariusza najdłuższy zarejestrowany czas przetwarzania żądania należy do Express (19 ms). Średni czas odpowiedzi wynosi 4,80 ms dla Spring oraz 3,60 dla Express.

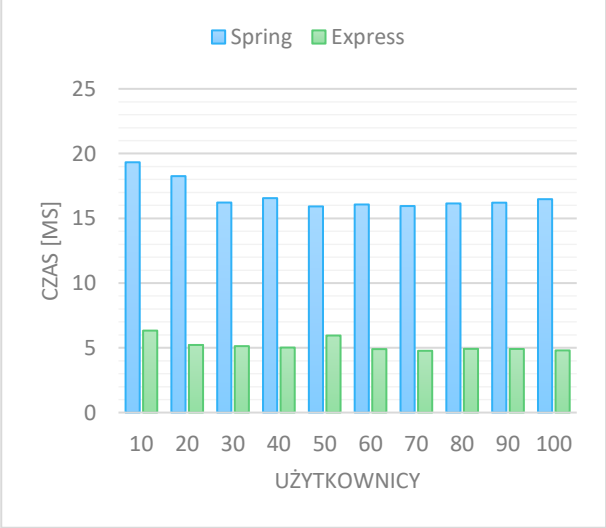
Tabela 4: Wartości parametrów dla metody GET dla pojedynczego elementu – 50 użytkowników

Parametr	Spring	Express
Średnia (ms)	4,80	3,60
Wartość maksymalna (ms)	9,00	19,00
Wartość minimalna (ms)	3,00	2,00
Odchylenie standardowe (ms)	0,74	1,23
Mediana (ms)	5,00	5,00

6.3. Scenariusz III – metoda GET dla 100 obiektów

Znaczącą przewagę w czasie odpowiedzi można zauważyć podczas analizy wyników trzeciego scenariusza, pobierającego 100 obiektów dodanych przez zalogowanego użytkownika (Rysunek 5).

Spring potrzebuje ponad 15 ms na odesłanie danych w czasie gdy Express około 5 ms.



Rysunek 5: Średnie czasy przetwarzania żądania GET dla 100 elementów.

Podobnie jak w dwóch poprzednich scenariuszach, tak i w tym przypadku najdłuższe czasy osiągane są dla 10 użytkowników (Spring – 19,33 ms, Express – 6,33 ms). Analizując czasy dla szkieletu JavaScript mają one tendencję malejącą do 50 użytkowników gdzie widoczny jest wzrost o 18% w porównaniu do 40 użytkowników. Następnie dla 60. i 70. użytkowników czas ponownie maleje. W przypadku Spring, czasy maleją do 30 użytkowników, następnie dla 40 użytkowników czas wzrasta o 2% i znowu maleje. Dla kolejnych użytkowników widać niewielką tendencję wzrostową.

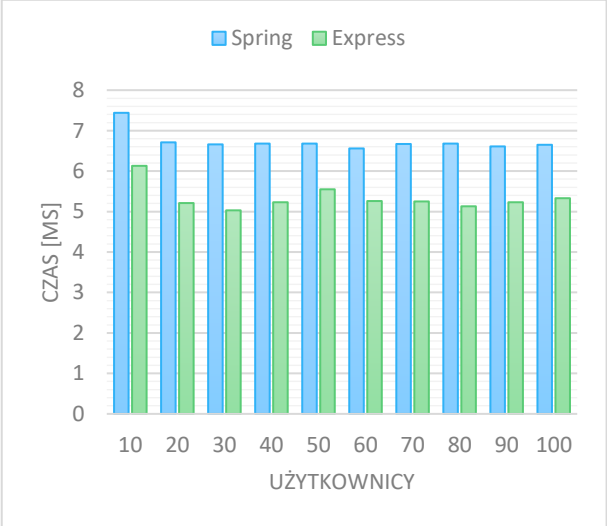
Tabela 5: Wartości parametrów dla metody GET dla 100 elementów – 50 użytkowników

Parametr	Spring	Express
Średnia (ms)	15,92	5,95
Wartość maksymalna (ms)	29,00	25,00
Wartość minimalna (ms)	11,00	4,00
Odchylenie standardowe (ms)	1,69	2,04
Mediana (ms)	16,00	3,00

Zarówno średnia, wartość maksymalna, minimalna oraz mediana największe są dla aplikacji zaimplementowanej w Spring. Potrzebuje ona co najmniej 11 ms na przetworzenie pojedynczego żądania, podczas gdy Express tylko 4 ms (Tabela 5).

6.4. Scenariusz IV – metoda PUT

Średnie czasy uzyskane w trakcie czwartego scenariusza zostały zwizualizowane na Rysunku 6. Najszybciej zostały przetworzone żądania w przypadku 30 użytkowników dla szkieletu Express (5,03 ms) oraz 60 użytkowników dla aplikacji w Spring (6,56 ms). Najdłuższe czasy uzyskane zostały dla 10 użytkowników (Spring – 7,44 ms, Express – 6,13 ms).



Rysunek 6: Średnie czasy przetwarzania żądania PUT.

W zakresie od 20 do 100 użytkowników, czasy uzyskane w aplikacji utworzonej przy użyciu Spring oscylują w okolicy 6,65 ms, osiągając wyniki delikatnie poniżej lub powyżej tej wartości (w granicach około 1%).

W przypadku Express widoczny jest wzrost czasu potrzebnego na przetworzenie żądania w zakresie od 20 do 50 użytkowników. W kolejnych iteracjach następuje skrócenie czasu z 5,55 ms (dla 50 użytkowników) do 5,13 ms (dla 80 użytkowników). Dla dwóch ostatnich powtórzeń scenariusza czas wzrasta odpowiednio do 5,23 ms i 5,33 ms.

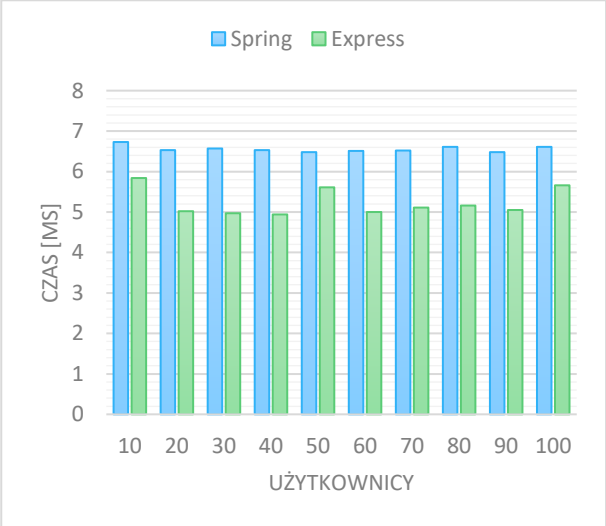
Najdłuższy jednostkowy czas potrzebny na przetworzenie żądania wynosił 22,00 ms, najszybciej zaś odpowiedzi udzielił Express i potrzebował na to 2,0 ms. Mediana w obu przypadkach jest zbliżona do średniej i wynosi – 7,00 ms i 6,68 ms dla Spring oraz 5,00 ms i 5,55 ms dla drugiego szkieletu (Tabela 6).

Tabela 6: Parametry dla metody PUT – 50 użytkowników

Parametr	Spring	Express
Średnia (ms)	6,68	5,55
Wartość maksymalna (ms)	22,00	19,00
Wartość minimalna (ms)	4,00	2,00
Odchylenie standardowe (ms)	1,04	1,50
Mediana (ms)	7,00	5,00

6.5. Scenariusz V – metoda DELETE

Rysunek 7 przedstawia wizualizację czasów przetwarzania żądania HTTP dla usuwania rekordów z bazy danych. Podobnie jak w poprzednich scenariuszach iteracja dla 10 aktywnych użytkowników potrzebuje średnio najwięcej czasu na odpowiedź (Spring – 6,73 ms, Express – 5,84 ms).



Rysunek 7: Średnie czasy przetwarzania żądania DELETE.

Czasy dla Spring oscylują w okolicach 6,50 ms i osiągają wartość ± 1%, podczas gdy zmiany wartości dla Express są wyraźnie widoczne w 3 punktach – dla 10, 50 i 100 użytkowników. Wynoszą one odpowiednio 5,84 ms, 5,61 ms oraz 5,66 ms. Dla pozostałych przebiegów testów wartości te są bliskie 5 ms (w zakresie od 4,97 ms dla 30 użytkowników, do 5,16 ms dla 80 użytkowników.

Szczegółowe statystyki dla 50 użytkowników wykonujących operacje usunięcia widoczne są w Tabeli 7. Najdłuższy czas potrzebny na usunięcie pojedynczego rekordu wynosi 17,00 ms dla Spring i 15,00 ms dla Express. Najkrótszy czas jest taki sam dla obu technologii i wynosi 4,00 ms.

Tabela 7: Wartości parametrów dla metody DELETE – 50 użytkowników

Parametr	Spring	Express
Średnia (ms)	6,48	5,61
Wartość maksymalna (ms)	17,00	15,00
Wartość minimalna (ms)	4,00	4,00
Odchylenie standardowe (ms)	0,83	1,37
Mediana (ms)	6,00	5,00

7. Wnioski

W niniejszym artykule położono nacisk na pomiary wydajności obsługi żądań HTTP – GET, POST, PUT, DELETE dla aplikacji wykonanych w technologii Spring i Express.

Biorąc pod uwagę całość przeprowadzonych badań w 5 scenariuszach testowych aplikacja zaimplementowana z wykorzystaniem mikro-szkieletu programistycznego Express.js radziła sobie lepiej z przetwarzaniem żądań uzyskując krótsze czasy niż odpowiednik aplikacji w szkielecie dla Javy. Przewaga Express nad Spring widoczna jest szczególnie w scenariuszu trzecim. W najlepszym przypadku czas oczekiwania jest dla Spring dłuższy o 167%, a w najgorszym przypadku 249% dla tej samej liczby użytkowników w aplikacji opartej o Express. Jedynym miejscem w którym Express.js przegrywa ze Spring jest pierwszy scenariusz dodawania nowych rekordów dla 70 użytkowników – średni czas jest tutaj najwyższy ze wszystkich.

Ponadto podczas analizy uzyskanych czasów dla szkieletu opartego o język JavaScript i środowisko Node.js można zauważyć tendencję spadkową w zakresie 10 – 40 użytkowników. W scenariuszach 2 – 5 czas odpowiedzi chwilowo rośnie dla 50 użytkowników. W porównaniu do wyników uzyskanych przez drugą aplikację czasy również maleją w stosunku do pierwszej iteracji testów, jednak utrzymują się na zbliżonym poziomie bez względu na liczbę aktywnych użytkowników.

Z podstawowych operacji CRUD najdłużej zajęły operacje aktualizacji i usuwania wskazanych rekordów.

W pracy porównano również popularność obu technologii na przestrzeni kilku lat, wskazując że Express.js jest aktualnie popularniejszym szkieletem niż Spring wśród zawodowych programistów.

We wszystkich przeprowadzonych scenariuszach czasy uzyskane dla 10 użytkowników były największe. W tych przypadkach mechanizmy zarządzające wielowątkowością w Express i Spring nie przydzieliły większej liczby zasobów.

Na podstawie uzyskanych wyników i przeprowadzonych wniosków można stwierdzić, że postawiona we wstępie hipoteza badawcza była słuszna wskazując, że

technologia Express zapewnia wydajniejsze rozwiązanie do budowy aplikacji w architekturze REST.

Zakres przeprowadzonych badań skupiał się wyłącznie na wydajności czasowej aplikacji. Nie jest to jednak jedyne kryterium doboru technologii przy planowaniu projektu. W dalszej pracy należałoby uwzględnić parametry takie jak dostępna dokumentacja, sposób implementacji kluczowych fragmentów kodu, struktura projektu, obciążenie procesora i pamięci RAM, wydajność z innymi bazami danych, czy zachowanie szkieletów dla zmiennej liczby jednoczesnych żądań HTTP dochodzących do serwera. Te i wiele innych kryteriów stanowić będą obszar dalszych badań autorów.

Literatura

[1] Dokumentacja API - Meta for Developers, https://developers.facebook.com/docs/pages/publishing/, [18.06.2023].

[2] Dokumentacja API - Twitter for Developers, https://developer.twitter.com/en/docs/twitter-api/tweets/manage-tweets/api-reference/post-tweets, [18.06.2023].

[3] M. Grudniak, M. Dzieńkowski, REST API performance comparison of web applications based on JavaScript programming frameworks, Journal of Computer Sciences Institute, 19 (2021) 121-125, https://doi.org/10.35784/jcsi.2620.

[4] K. K. Dhall, A performance comparison of restful applications implemented in Spring Boot Java and MS.NET Core, Journal of Physics: Conference Series, 1933 (2021) 12-41, https://doi.org/10.1088/1742-6596/1933/1/012041.

[5] K. Kronis, M. Uhanova, Performance comparison of Java EE and ASP.NET Core Technologies for web API development, Applied Computer Systems 23 (2018) 37-44, https://doi.org/10.2478/acss-2018-0005.

[6] K. Lei, Y. Ma, Z. Tan, Performance comparison and evaluation of Web Development Technologies in PHP, python, and node.js, 2014 IEEE 17th International Conference on Computational Science and Engineering, (2014) 661-668, https://doi.org/10.1109/cse.2014.142.

[7] Stack Overflow Developer Survey 2023, https://survey.stackoverflow.co/2023#most-popular-technologies-webframe-prof, [26.06.2023].

[8] Spring - prostota i uniwersalność najpopularniejszego frameworku Java, https://boringowl.io/tag/spring, [26.06.2023].

[9] Stack Overflow Developer Survey 2022, https://survey.stackoverflow.co/2022#most-popular-technologies-webframe-prof, [26.06.2023].

[10] Express.js - MVC Framework Node.js, https://boringowl.io/tag/express-js, [19.06.2023].

[11] Stack Overflow Developer Survey 2018, https://insights.stackoverflow.com/survey/2018#most-popular-technologies, [26.06.2023].

[12] Stack Overflow Developer Survey 2020, https://insights.stackoverflow.com/survey/2020#technology-web-frameworks-professional-developers2, [26.06.2023].

[13] Stack Overflow Developer Survey 2021, https://insights.stackoverflow.com/survey/2021#most-popular-technologies-webframe-prof, [26.06.2023].