PORÓWNANIE WYDAJNOŚCI TECHNOLOGII IMPLEMENTACJI REST API Z WYKORZYSTANIEM ASP.NET CORE I SPRING BOOT

european student magazine
ISSN 2956-834X
Numer 2 / 2025

Paweł **Piotrowski**[1], Aleksandra **Sikora**[2*]

[1] Politechnika Świętokrzyska, Wydział Elektrotechniki, Automatyki i Informatyki, al. Tysiąclecia Państwa Polskiego 7, 25-314 Kielce, Polska, s091310@student.tu.kielce.pl
[2] Politechnika Świętokrzyska, Wydział Elektrotechniki, Automatyki i Informatyki, al. Tysiąclecia Państwa Polskiego 7, 25-314 Kielce, Polska, asikora@tu.kielce.pl
[*] Corresponding author

Streszczenie – W artykule przeprowadzono porównanie wydajności dwóch popularnych technologii do budowy usług sieciowych zgodnych z architekturą REST API: ASP.NET Core oraz Spring Boot. Ocenie poddano aplikację sklepu internetowego realizującą operacje CRUD, uwierzytelnianie i autoryzację, w której zastosowano powszechnie wykorzystywane rozwiązania ORM: Entity Framework Core dla ASP.NET Core oraz Spring Data JPA dla Spring Boot. Testy wydajności przeprowadzono z użyciem Apache JMeter, analizując średni czas odpowiedzi oraz przepustowość w różnych scenariuszach obciążeniowych. Wyniki wykazały wyraźną przewagę ASP.NET Core, szczególnie w operacjach modyfikujących dane, gdzie uzyskano nawet 36% większą przepustowość oraz krótsze czasy odpowiedzi. W przypadku operacji GET ALL lepsze rezultaty osiągnął Spring Boot. Zaprezentowana analiza wskazuje na wyższą efektywność platformy ASP.NET Core w środowiskach wymagających intensywnego przetwarzania wielu równoczesnych żądań.

Słowa kluczowe – ASP.NET Core, CRUD, REST API, Spring Boot, wydajność

WSTĘP

Wraz z rozwojem aplikacji webowych i mobilnych rośnie znaczenie wydajnej komunikacji między warstwą kliencką a serwerową, w której kluczową rolę odgrywa architektura REST. REST API stanowi powszechnie stosowany mechanizm wymiany danych, szczególnie w systemach backendowych realizujących operacje CRUD. W środowiskach produkcyjnych, takich jak systemy medyczne czy platformy zakupowe, istotna staje się nie tylko poprawność działania, ale także responsywność i skalowalność rozwiązania.

W odpowiedzi na te potrzeby powstało wiele technologii umożliwiających tworzenie usług sieciowych, spośród których szczególnie popularne są ASP.NET Core (firmy Microsoft oparty o język C#) oraz Spring Boot (Java). Oba rozwiązania oferują bogate wsparcie w zakresie integracji z bazami danych, bezpieczeństwa i monitorowania.

Celem niniejszego artykułu jest porównanie wydajności dwóch technologii w kontekście implementacji aplikacji backendowej realizującej operacje CRUD. Warto podkreślić, że obie wersje oparto na otwarto-źródłowych bibliotekach. Analizie poddano takie parametry jak średni czas odpowiedzi, przepustowość oraz skuteczność przy równoczesnym przetwarzaniu wielu żądań, aby ocenić ich przydatność w scenariuszach wymagających wysokiej wydajności.

I. PRZEGLĄD LITERATURY

Artykuł *"A Performance Comparison of RESTful Applications Implemented in Spring Boot Java and MS.NET Core"* autorstwa Hardeep Kaur Dhalla analizuje wydajność aplikacji REST napisanych w Java (Spring Boot) i C# (.NET Core) [1]. Obie implementacje realizowały te same procesy biznesowe, umożliwiając rzetelne porównanie ich działania. Testy przeprowadzono za pomocą Apache JMeter 5.2.1. W przeglądzie literatury odnotowano również badania wskazujące na wyższą wydajność aplikacji Java EE Struts w porównaniu z VB.NET.

Głównymi metrykami oceny wydajności były: średni czas odpowiedzi oraz procentowy udział błędów przy obsłudze żądań. Równocześnie monitorowano zużycie zasobów systemowych, w tym procesora (CPU) i pamięci operacyjnej (RAM). Scenariusze testowe zakładały stopniowe zwiększanie liczby jednoczesnych użytkowników od 1000 do 6400. Wyniki badań wykazały, że aplikacja zaimplementowana z wykorzystaniem platformy MS.NET Core cechowała się krótszymi czasami odpowiedzi i mniejszym zużyciem zasobów systemowych w porównaniu do aplikacji opartej na Java Spring Boot.

Zagadnienie porównania wydajności i kompletności ekosystemów ASP.NET Core oraz Spring Boot zostało szczegółowo przeanalizowane w pracy *„Comparison of ASP.NET Core and Spring Boot ecosystems"*, autorstwa Petra Kopyla, Teofila Rozaliuka oraz Jakuba Smołki,

opublikowanej na łamach czasopisma *Journal of Computer Sciences Institute*. Badanie oparto na dwóch autorskich aplikacjach implementujących identyczną funkcjonalność, których wspólnym elementem był silnik bazodanowy PostgreSQL [2]. Aplikacje zostały opracowane przy użyciu typowych dla danego środowiska bibliotek i narzędzi: do realizacji operacji bazodanowych wykorzystano biblioteki ORM Spring Data dla Spring Boot oraz Entity Framework Core dla ASP.NET Core.

W projekcie uwzględniono także technologie wspierające mechanizmy uwierzytelniania i autoryzacji. Odpowiednio Spring Security oraz ASP.NET Core Identity, a także zestaw dodatkowych bibliotek upraszczających proces implementacji. Porównanie koncentrowało się na trzech głównych aspektach: intuicyjności i łatwości implementacji, możliwościach narzędzi do obsługi uwierzytelniania, a także surowej wydajności operacji wykonywanych na bazie danych.

Pomiar czasów odpowiedzi operacji bazodanowych prowadzono bezpośrednio w kodzie aplikacji, z wyłączeniem jakiejkolwiek logiki biznesowej, brano pod uwagę wyłącznie czas wykonania żądania do bazy danych. Wyniki testów wykazały, że Spring Data przewyższa Entity Framework Core pod względem szybkości realizacji zapytań. Jednocześnie zauważono, że Spring Security. W przeciwieństwie do ASP.NET Core Identity cechuje się mniejszym poziomem integracji z całym ekosystemem frameworka, oferując ograniczony zestaw gotowych rozwiązań w zakresie integracji z warstwą danych.

W artykule *„Performance analysis of REST API technologies using Spring and Express.js examples”* autorstwa Macieja Wichy oraz Beaty Pańczyk, opublikowanym w *Journal of Computer Sciences Institute* (tom 29, s. 352–359), przeprowadzono porównanie dwóch popularnych technologii wykorzystywanych do budowy aplikacji webowych zgodnych z architekturą REST — frameworka Spring (Java) oraz Express.js (Node.js) [3]. Badanie miało charakter empiryczny i zostało zrealizowane poprzez implementację testowej aplikacji w obu technologiach. Wydajność mierzono przy pomocy narzędzia Apache JMeter, koncentrując się na czasie przetwarzania żądań HTTP dla podstawowych operacji CRUD (GET, POST, PUT, DELETE) przy zmiennej liczbie użytkowników (od 10 do 100) i stałej częstotliwości zapytań.

Wyniki eksperymentu wskazują na wyraźną przewagę Express.js w zakresie obsługi żądań – w niektórych scenariuszach czas odpowiedzi był nawet o 249% krótszy niż w przypadku Spring. Wyjątkiem był scenariusz dodawania rekordów dla 70 użytkowników, gdzie Express osiągnął najgorszy wynik. Autorzy zwracają również uwagę na zmienność czasu odpowiedzi w zależności od obciążenia oraz niedostateczne wykorzystanie zasobów dla niskiej liczby równoczesnych użytkowników. Zwrócono też uwagę na rosnącą popularność Express.js wśród programistów. Podkreślono, że choć Express cechuje się lepszą wydajnością czasową, dobór technologii powinien uwzględniać również inne kryteria, takie jak dokumentacja, architektura kodu czy wykorzystanie zasobów systemowych, które planuje się uwzględnić w dalszych badaniach.

Porównanie technologii Java EE 7 oraz ASP.NET Core w kontekście tworzenia usług REST API zostało przeprowadzone przez Kristiansa Kronisa i Marinę Uhanovą w publikacji *„Performance Comparison of Java EE and ASP.NET Core Technologies for Web API Development”*, opublikowanej w czasopiśmie *Applied Computer Systems* [4]. Autorzy dużą wagę przywiązali do zachowania spójnych warunków eksperymentalnych, dlatego testowane aplikacje uruchomiono na tym samym serwerze VPS. Środowisko to pozostawało niezmienione przez cały czas trwania pomiarów, nie przeprowadzano aktualizacji systemu ani modyfikacji konfiguracji oprogramowania.

Zarówno aplikacja w Java EE, jak i ta oparta na ASP.NET Core korzystały z bazy danych MySQL, a dane przesyłane były w formacie JSON. Ocenie poddano szereg operacji, w tym przetwarzanie krótkich tekstów, generowanie danych losowych, działania arytmetyczne na 1000 losowo wybranych liczbach oraz manipulację rozbudowanymi strukturami JSON.

Analiza wyników pokazała, że aplikacja w ASP.NET Core uzyskiwała krótsze czasy przetwarzania żądań, jednak serwer Kestrel wykorzystywany przez tę technologię wykazywał dłuższy czas wysyłania odpowiedzi niż serwer Tomcat używany w przypadku rozwiązania opartego na Java EE.

II. BADANE TECHNOLOGIE

ASP.NET Core - to nowoczesny framework firmy Microsoft, umożliwiający tworzenie aplikacji webowych, w tym usług REST API [5]. Jego architektura opiera się na modularnym podejściu i elastycznej konfiguracji. Centralnym elementem frameworka jest potok pośredników middleware, odpowiadający za obsługę żądań HTTP oraz ich transformację w odpowiedzi. Kluczową zaletą jest też możliwość selektywnego dobierania komponentów przez NuGet, co sprzyja optymalizacji aplikacji [7].

Najważniejsze funkcjonalności:

- Wydajny serwer Kestrel jako domyślne środowisko uruchomieniowe,
- Elastyczny potok middleware z możliwością konfiguracji żądań (uwierzytelnianie, logowanie, kompresja),
- Wzorzec Model-View-Controller i kontrolery Web API oznaczone [ApiController],
- Wbudowany mechanizm Dependency Injection (DI) do zarządzania zależnościami,
- Routing atrybutowy [HttpGet], [HttpPost], zgodny z zasadami REST,
- Automatyczna serializacja danych z wykorzystaniem JSON (domyślnie) lub
- Newtonsoft.Json,
- Integracja z OpenAPI/Swagger (np. Swashbuckle.AspNetCore) do dokumentowania API,

- ASP.NET Core Identity – system uwierzytelniania, autoryzacji i zarządzania użytkownikami,
- Obsługa mechanizmów zabezpieczeń: JWT, OAuth2, polityki i role autoryzacyjne,
- Ochrona przed CSRF, XSS oraz wsparcie dla CORS i bezpiecznej komunikacji HTTPS.

Framework ten został zaprojektowany z myślą o wysokiej wydajności i bezpieczeństwie.

Spring Boot - to framework oparty na ekosystemie Spring, zaprojektowany z myślą o uproszczeniu tworzenia aplikacji webowych i usług REST w języku Java. Wyróżnia się podejściem konwencji ponad konfigurację, integracją z wieloma bibliotekami Spring [6]. Jego modułowa struktura umożliwia podzielenie projektu na warstwy, przy czym interakcja między komponentami oparta jest na mechanizmie wstrzykiwania zależności [8].

Kluczowe funkcjonalności Spring Boot:

- Wbudowana obsługa REST API z wykorzystaniem @RestController, @GetMapping, @PostMapping,
- Wstrzykiwanie zależności poprzez adnotacje: @Autowired, @Service, @Component,
- Obsługa warstwowej architektury aplikacji: kontrolery, serwisy i repozytoria,
- Spring Data JPA umożliwia automatyczne generowanie zapytań na podstawie repozytoriów,
- Wbudowany serwer Tomcat do lokalnego uruchamiania aplikacji,
- Integracja z OpenAPI/Swagger za pomocą SpringDoc do generowania dokumentacji API,
- Spring Security jako mechanizm uwierzytelniania i autoryzacji z obsługą JWT,
- Możliwość deklaratywnego zabezpieczania zasobów przy użyciu adnotacji takich jak @PreAuthorize czy @Secured,
- Elastyczna konfiguracja reguł zabezpieczeń w klasie SecurityFilterChain.

III. CEL I METODYKA BADANIA

Celem badania było porównanie wydajności aplikacji zaimplementowanych w dwóch wyżej opisanych technologiach. Przeprowadzono serię testów obejmujących podstawowe operacje zgodne z protokołem HTTP (POST, GET, PUT, DELETE), odpowiadające operacjom CRUD. Dodatkowo wykonano test złożony, symulujący jednoczesne występowanie różnych typów żądań. Na podstawie zebranych danych przeprowadzono analizę porównawczą, uwzględniającą kluczowe wskaźniki wydajnościowe.

SCENARIUSZE TESTOWE

Przy użyciu narzędzia JMeter przygotowano następujące scenariusze testowe:

- **PostProducersAndCategories** - Testowanie tworzenia nowych producentów i~kategorii (metoda POST). Weryfikacja poprawności dodawania zasobów i~generowania ich ID.
- **PostProductsAndOrders** - Testowanie tworzenia produktów i zamówień (metoda POST). Sprawdzenie integralności danych (np. powiązań produkt–kategoria) oraz mechanizmów zamówień.
- **PutProducersAndCategories** - Aktualizacja istniejących producentów i kategorii (metoda PUT). Weryfikacja modyfikacji pól (np. nazwa, opis) i o
- **PutProductsAndOrders** - Modyfikacja produktów i zamówień (metoda PUT/PATCH). Testowanie zmian stanu zamówień oraz aktualizacji danych produktów.
- **GetProducersAndCategories** - Pobieranie pojedynczych producentów i kategorii (metoda GET). Sprawdzenie wydajności odczytu pojedynczych zasobów oraz poprawności zwracanych danych.
- **GetProductsAndOrders** - Pobieranie szczegółów produktów i zamówień (metoda GET). Weryfikacja powiązań (np. produkt–zamówienie) i czasu odpowiedzi.
- **GetAllProducersAndCategories** - Pobieranie wszystkich producentów i kategorii (metoda GET). Testowanie skalowania API przy dużych zestawach danych.
- **GetAllProductsAndOrders** - Pobieranie pełnych list produktów i zamówień (metoda GET). Analiza wydajności przy masowym odczycie i paginacji.
- **DeleteProducersAndCategories** - Usuwanie producentów i kategorii (metoda DELETE). Sprawdzenie mechanizmów usuwania oraz obsługi zależności (np. czy usunięcie kategorii wpływa na produkty).
- **DeleteProductsAndOrders** - Usuwanie producentów i kategorii (metoda DELETE). Sprawdzenie mechanizmów usuwania oraz obsługi zależności (np. czy usunięcie kategorii wpływa na produkty).
- **DemandingRestApi** - Symulacja kompleksowego obciążenia systemu poprzez mieszane operacje (CRUD) na wszystkich zasobach jednocześnie. Sprawdzenie działania przy wysokiej konkurencji dostępu do danych podczas równoległego tworzenia, modyfikacji i usuwania zasobów.

Testy 1--10 skupiają się na pojedynczych operacjach, podczas gdy **DemandingRestApiSpring** imituje rzeczywiste obciążenie np. ruch wielu użytkowników podczas korzystania z usługi.

Co warte podkreślenia polecenia oznaczone jako **GET ALL** służą do pobrania wszystkich zasobów danego typu np. wszystkich, produktów z serwera w ramach żądania HTTP.

Tabela 1. Platformy testowe

Parametr	Wartość
Komputer nr 1	
Platforma	Lenovo Legion Y530-15ICH
Procesor (CPU)	Intel(R) Core(TM) i7-8750H – 6 rdzeni / 12 wątków
Pamięć operacyjna (RAM)	2 x 16 GB SODIMM DDR4 – 2666 MHz
Dysk	WD Black SN770 1TB
System operacyjny	Windows 11 Home 64-bitowy; build 26100.3775

IV. Materiał badawczy

Materiał badawczy stanowiła aplikacja sklepu internetowego zrealizowana w architekturze REST API, zaimplementowana w dwóch wybranych technologiach: ASP.NET Core w wersji 9.0 oraz Spring Boot w wersji 3.4.5. W obu programach wykorzystano mechanizmy ORM: Entity Framework Core w przypadku ASP.NET Core oraz Spring Data JPA dla Spring Boot.

Celem obu aplikacji było realizowanie prostych operacji CRUD (ang. Create, Read, Update, Delete), uwierzytelnianie oraz autoryzacja użytkowników.

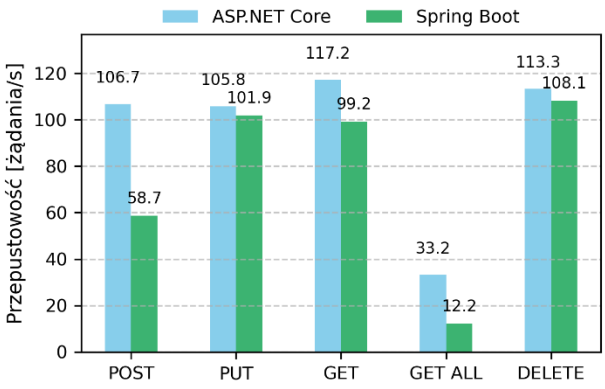
W celu zapewnienia rzetelnych warunków pomiarowych obie aplikacje działały na tym samym silniku bazodanowym PostgreSQL 17.3 [9].

Elementy modelu danych, na których były przeprowadzone testy to: Prodeucent, Kategoria, Produkt, Zamówienie.

Co warte podkreślenia oba rozwiązania wykorzystują otwarto-źródłowe biblioteki. Dzięki temu możliwe jest ich powszechne stosowanie i analizowanie.

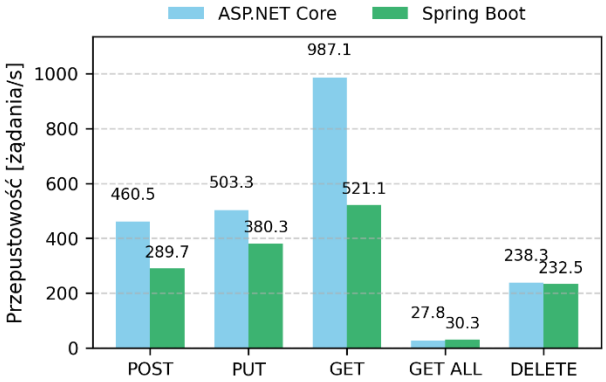
V. Analiza porównawcza

Poniżej zaprezentowano wyniki badania.



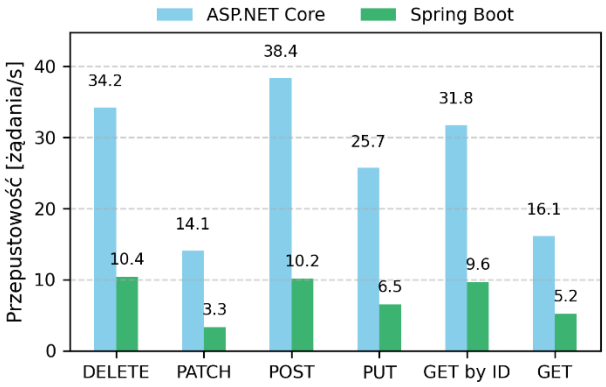
Rys. 1. Porównanie przepustowości różnego typu żądań dla testu podstawowego i danych o mniejszej złożoności.

Przedstawione wyniki wskazują na zbliżoną przepustowość analizowanych technologii w przypadku testów o niższym stopniu złożoności. Jednakże występują większe różnice w poleceniu GET ALL i POST na korzyść ASP.NET Core. W przypadku dodawania danych technologia Microsoftu jest niemal dwukrotnie wydajniejsza. Co może mieć kluczowe znaczenie podczas projektowania systemu.

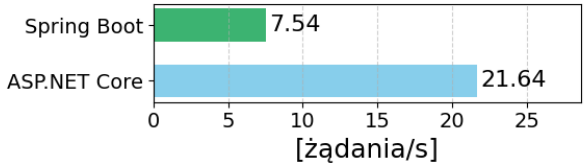


Rys. 2. Porównanie przepustowości różnego typu żądań dla testu podstawowego i danych o większej złożoności.

W scenariuszach obejmujących operacje na większych zbiorach danych oraz przy bardziej skomplikowanych relacjach, technologia ASP.NET Core wykazuje istotnie wyższą przepustowość. Jest to szczególnie zauważalne w przypadku żądań typu GET, POST oraz PUT, które wiążą się z przetwarzaniem większych zestawów danych. Zauważalne różnice występują dla polecenia GET, które w przypadku ASP.NET Core jest niemal dwukrotnie bardziej wydajne w porównaniu do rozwiązania opartego o język Java. Polecenia związane ze zmianą lub zapisem danych POST i PUT są wyraźnie szybsze na ASP.NET Core, którego przewaga w przypadku polecenia POST wynosi około 37%. Zaznacza się tutaj wyraźna tendencja, gdzie operacje związane z dodawaniem lub edycją danych wyraźnie lepiej wypadają dla technologii działającej na ASP.NET Core.

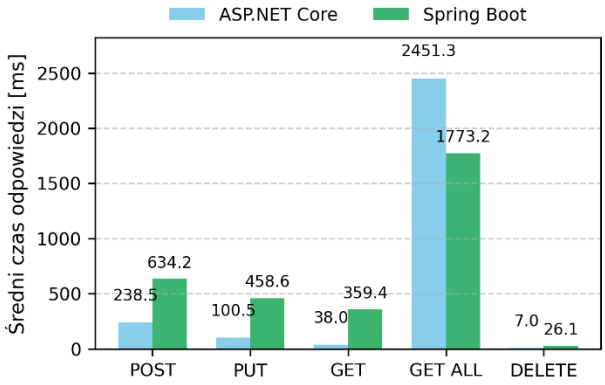


Rys. 3. Porównanie przepustowości różnego typu żądań dla testu złożonego (wymagającego).



Rys. 4. Porównanie wartości średnich dla przepustowości żądań testu złożonego (wymagającego).

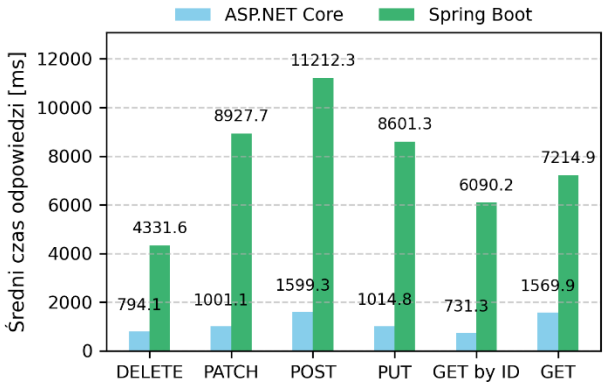
W przypadku tego testu zaobserwowano wyraźne różnice w wydajności obu rozwiązań. Test, który symuluje realistyczne obciążenie poprzez jednoczesne wykonywanie różnych typów żądań, wykazał, że rozwiązanie oparte na języku C# osiąga średnio kilkukrotnie lepsze wyniki. Średnio ASP.NET Core okazał się być od 3 do nawet 4 razy wydajniejszy – zwłaszcza przy operacjach na danych, takich jak POST, PATCH czy PUT. Wyniki te potwierdzają wnioski płynące z wcześniejszych, bardziej podstawowych testów.



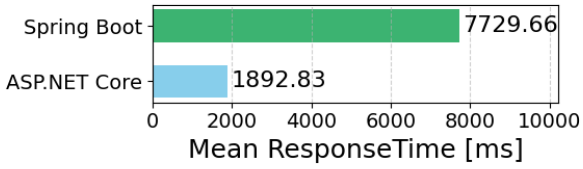
Rys. 5. Porównanie średniego czasu odpowiedzi różnego typu żądań dla testu podstawowego i danych o większej złożoności.

Na podstawie zaprezentowanych wyników można stwierdzić, że **ASP.NET Core charakteryzuje się wyraźnie lepszą wydajnością** w większości testowanych scenariuszy. Szczególnie przy operacjach na bardziej złożonych danych osiąga niższy średni czas odpowiedzi, co świadczy o bardziej efektywnym wykorzystaniu zasobów w porównaniu do aplikacji opartej na Spring Boot.

Zarówno dla PC1 jak i PC2 Wyjątkiem był jedynie test GET ALL, w którym lepszy wynik uzyskało rozwiązanie oparte na Spring Boot. Najprawdopodobniej ASP.NET Core nie był w stanie przetworzyć efektywnie tylu żądań z dużą ilością danych.



Rys. 6. Porównanie średniego czasu odpowiedzi różnego typu żądań dla testu złożonego (wymagającego).



Rys. 7. Porównanie średniej ze średnich czasów odpowiedzi dla różnych typów żądań w teście złożonym (wymagającym).

Biorąc pod uwagę przedstawione powyżej dane średniego czasu odpowiedzi aplikacja Rest API napisana w technologii firmy Microsoft jest w większości przypadków nawet kilkadziesiąt razy szybsza od tej bazującej na Java Spring Boot. Oznacza to, że klient musi dość długo czekać na odpowiedź serwera aplikacji napisanej w języku Java.

VI. WNIOSKI

Zaprezentowana analiza wskazuje na wyraźnie wyższą wydajność technologii ASP.NET Core w porównaniu do Spring Boot, co potwierdzają wyniki przeprowadzonych testów. W testach podstawowych, o mniejszym stopniu złożoności, ASP.NET Core osiągał nawet kilkudziesięciokrotnie krótsze czasy odpowiedzi, co potwierdzają średnie opóźnienia realizacji żądań. Również pod względem przepustowości ta technologia umożliwiała obsługę od kilku do kilkudziesięciu procent większej liczby żądań na jednostkę czasu w porównaniu do Spring Boot. Szczególną uwagę zwraca operacja GET ALL, która ze względu na zwracanie dużej ilości danych wykazała wyraźnie wyższe czasy odpowiedzi oraz niższą przepustowość w obu rozwiązaniach, choć Spring Boot pozostawał niemal trzykrotnie wolniejszy pod względem liczby przetworzonych żądań na sekundę.

Analiza bardziej złożonych operacji, takich jak tworzenie produktów i zamówień (POST, PUT), potwierdziła lepszą wydajność ASP.NET Core, czasy odpowiedzi były zdecydowanie krótsze, a przepustowość systemu wyraźnie wyższa nawet do 36% większa. W tych scenariuszach Spring Boot wykazywał nawet do dziesięciokrotnie większe opóźnienia, szczególnie

w przypadku zapytań GET, które jednocześnie charakteryzowały się przepustowością na poziomie około połowy wartości notowanej w ASP.NET Core. Wyjątkowo operacja GET ALL przyniosła lepsze wyniki w Spring Boot zarówno pod względem opóźnień, jak i przepustowości, co prawdopodobnie wynika z ograniczeń ASP.NET Core w obsłudze dużej liczby równoległych żądań zwracających rozbudowane dane.

W testach obciążeniowych **ASP.NET Core** konsekwentnie utrzymywał przewagę, osiągając średni czas odpowiedzi nawet **osiem razy krótszy** niż **Spring Boot**. W rozbudowanym scenariuszu z wykorzystaniem metody **POST**, technologia ta uzyskała odpowiedzi średnio o **85% szybsze**. Przewaga była również widoczna w przypadku pozostałych żądań modyfikujących dane, takich jak **PUT** i **PATCH**, gdzie ASP.NET Core pozwalał na obsługę nawet **czterokrotnie większej liczby równoczesnych żądań**. Tego typu różnice mają istotne znaczenie w kontekście wydajności i skalowalności nowoczesnych aplikacji webowych.

PERFORMANCE COMPARISON OF REST API IMPLEMENTATION TECHNOLOGIES USING ASP.NET CORE AND SPRING BOOT

The article presents a performance comparison of two popular technologies for building web services based on the REST API architecture: ASP.NET Core and Spring Boot. The evaluation focused on an online store application implementing CRUD operations, authentication, and authorization, utilizing commonly used ORM solutions: Entity Framework Core for ASP.NET Core and Spring Data JPA for Spring Boot. Performance tests were conducted using Apache JMeter, analyzing average response time and throughput under various load scenarios. The results demonstrated a clear advantage of ASP.NET Core, particularly in data-modifying operations, where it achieved up to 36% higher throughput and shorter response times. In the case of GET ALL operations, Spring Boot delivered better results. The presented analysis indicates higher efficiency of the ASP.NET Core platform in environments requiring intensive processing of multiple concurrent requests.

Key words: ASP.NET Core, CRUD, REST API, Spring Boot, performance

BIBLIOGRAFIA

[1] Dhalla H.K. (2021) “A Performance Comparison of RESTful Applications Implemented in Spring Boot Java and MS.NET Core”. J. Phys.: Conf. Ser. 1933 012041, doi: 10.1088/1742-6596/1933/1/012041
[2] Rozaliuk, T., Kopyl, P., & Smołka, J. (2022). Comparison of ASP.NET Core and Spring Boot ecosystems. Journal of Computer Sciences Institute, 22, 40–45. https://doi.org/10.35784/jcsi.2794
[3] Wicha, M., & Pańczyk, B. (2023). Performance analysis of REST API technologies using Spring and Express.js examples. Journal of Computer Sciences Institute, 29, 352–359. https://doi.org/10.35784/jcsi.3796
[4] Kronis, K., & Uhanova, M. (2018). Performance Comparison of Java EE and ASP.NET Core Technologies for Web API Development. Applied Computer Systems, 23(1), 37–44. https://doi.org/10.2478/acss-2018-0005
[5] Microsoft. Oficjalna strona ASP.NET Core, https://dotnet.microsoft.com/en-us/apps/aspnet, [Dostęp: 20 czerwca 2025]
[6] Spring.io. Oficjalna strona Spring Boot, https://spring.io/projects/spring-boot, [Dostęp: 20 czerwca 2025]
[7] de Sanctis V. Building Web APIs with ASP.NET Core. Packt Publishing, 2023.
[8] Walls C. Spring in Action - Sixth Edition. Manning Publications, 2022.
[9] Serwer bazodanowy PostgreSQL, https://www.postgresql.org/, [Dostęp: 22 czerwca 2025]
[10] Operacje bazodanowe typu CRUD, https://www.codecademy.com/articles/what-is-crud, [Dostęp: 20 czerwca 2025]