JOURNAL OF COMPUTER SCIENCES INSTITUTE

JCSI 35 (2025) 175–183
Received: 02 February 2025
Accepted: 23 April 2025

Performance comparison of CRUD operations in Spring Boot and ASP.NET Core frameworks

Porównanie wydajności operacji typu CRUD na przykładzie szkieletów programistycznych Spring Boot i ASP.NET Core

Michał Grzeszuk*, Marek Miłosz

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

Abstract

This article analyzes the performance of CRUD operations in two popular frameworks: Spring Boot and ASP.NET Core, evaluating speed, efficiency, and resource usage. Performance tests revealed that ASP.NET Core excels in CPU-intensive tasks, making it more suitable for applications requiring high computational efficiency. In contrast, Spring Boot offers more stable memory usage and better handling of complex queries, making it ideal for resource-constrained systems.

The findings indicate that the choice of technology should depend on project requirements, and the paper provides practical guidance for optimizing the development of scalable applications.

Keywords: CRUD; performance comparison; Spring Boot; ASP.NET Core

Streszczenie

Praca przedstawia rezultaty analizy wydajności operacji CRUD w dwóch popularnych szkieletach programistycznych: Spring Boot i ASP.NET Core, oceniając szybkość, efektywność i zużycie zasobów systemowych. Testy wydajnościowe wykazały, że ASP.NET Core lepiej radzi sobie z obciążeniami procesora, co czyni go bardziej odpowiednim dla aplikacji wymagających intensywnych obliczeń. Spring Boot z kolei zapewnia stabilniejsze zużycie pamięci i lepszą obsługę złożonych zapytań, dzięki czemu jest bardziej odpowiedni dla systemów o ograniczonych zasobach. Wnioski wskazują, że wybór technologii powinien zależeć od wymagań projektu, a artykuł dostarcza praktycznych wskazówek dla optymalizacji tworzenia skalowalnych aplikacji.

Słowa kluczowe: CRUD; porównanie wydajności; Spring Boot; ASP.NET Core

*Corresponding author

Email address: michal.grzeszuk@gmail.com (M. Grzeszuk)

1. Introduction

In today's fast-paced digital landscape, where businesses continuously seek optimization and efficiency, selecting the right backend framework is crucial for building scalable and high-performing web applications. As the demand for modern applications grows, companies are increasingly focused on improving system performance, reducing response times, and optimizing resource usage. Choosing the most suitable framework can significantly impact development speed, maintainability, and overall system efficiency. Given this, developers and system architects must carefully evaluate different technologies to ensure they meet the specific performance requirements of their projects.

The objective of this paper is to perform a detailed evaluation of CRUD (Create, Read, Update, Delete) operation performance using two widely used frameworks: Spring Boot and ASP.NET Core. Since CRUD operations are fundamental to the functionality of most web applications, their efficiency directly impacts the overall performance of the system. This study aims to identify which framework offers superior performance, faster response times, and more effective resource management when handling CRUD operations. Furthermore, it intends to provide valuable insights for developers and system architects in choosing the appropriate framework based on varying project needs and workloads.

In the methodology section, the approach to performance testing will be outlined, including the definition of test scenarios involving CRUD operations under diverse load conditions. Tools like VisualVM and Postman will be used for testing. The next stage will focus on implementing the tests by creating applications in both Spring Boot and ASP.NET Core to perform CRUD operations, configuring the test environments with databases such as PostgreSQL. After conducting performance tests and gathering data, the results will be analyzed, comparing the response times and resource utilization (CPU, RAM) of each framework in different scenarios. This analysis will highlight the strengths and weaknesses of each framework in terms of handling CRUD operations.

The final part of the paper will summarize the findings and offer recommendations for selecting the most appropriate framework based on specific project requirements and system load. Furthermore, suggestions for future research and improvements to the testing methodology are be proposed.

2. Purpose and scope of work

The aim of this paper is to conduct a comprehensive analysis of CRUD operation performance using two popular frameworks: Spring Boot and ASP.NET Core. CRUD operations are the foundation of most web applications, making their efficiency crucial to overall system performance. This paper seeks to evaluate which of these frameworks provides better performance, faster responses, and more efficient resource management in the context of CRUD operations. Additionally, this analysis aims to provide developers and system architects with information that can assist in selecting the right tool for projects with varying requirements and workloads. The paper will be divided into several key stages, each focusing on a different aspect of comparing Spring Boot and ASP.NET Core. The literature review will begin by discussing the basic concepts of CRUD operations and introducing the Spring Boot and ASP.NET Core technologies, their architecture, and core features. This section will also analyze existing research on the performance of both frameworks in the context of CRUD operations.

The study focuses on comparing the performance of Spring Boot and ASP.NET Core, analyzing their key features and evaluating their efficiency in handling CRUD operations. A literature review was conducted to assess previous comparative analyses of these frameworks in similar contexts. To ensure a reliable evaluation, specific performance criteria were defined, including response time, operations per second, and resource usage.

The testing environment was carefully prepared, with PostgreSQL configured as the database and tools such as VisualVM and Postman utilized for monitoring and executing tests. Sample applications were implemented using both frameworks, designed to perform CRUD operations under varying load conditions.

The collected data was analyzed and visualized, providing insights into response times, CPU and memory consumption, and overall efficiency. Based on these results, conclusions were drawn to determine the most suitable framework depending on project requirements, offering recommendations for developers choosing between Spring Boot and ASP.NET Core.

3. Literature review

The literature review covers key studies on CRUD performance and backend technologies, forming the basis for our analysis. In the work by Kaluža and Kalanj [1], the authors compared various backend frameworks by examining CRUD performance metrics such as response times, throughput, and resource usage. Their results revealed differences in efficiency and scalability among the frameworks, providing a valuable starting point for our focused comparison of Spring Boot and ASP.NET Core. This prior work informs our methodology, especially in identifying performance indicators critical for web applications.

Masse [2] discussed the design of consistent RESTful APIs, emphasizing best practices for defining resources and structuring APIs to optimize response times and minimize server load. By introducing standards for scalability and efficiency, this work sets a foundation for ensuring performance consistency in our tested applications. These principles are particularly relevant as we assess how each framework implements RESTful services in CRUD operations.

Bonteanu and Tudose [3] analyzed CRUD performance in Java applications using JPA, Hibernate, and Spring Data JPA, focusing on execution times across different databases and optimization techniques such as indexing and query transformation. Their systematic testing revealed how database configuration impacts CRUD efficiency, guiding our approach to testing with different database systems. Similarly, their follow-up study [4] evaluated Hibernate's cross-platform performance, showing how environmental factors can affect results. This insight supports our plan to test both frameworks in varied configurations, ensuring comprehensive analysis.

Kopyl and Smołka [5] compared the ecosystems of ASP.NET Core and Spring Boot, particularly their resource management capabilities, such as CPU and RAM usage. They demonstrated that ASP.NET Core tends
to be more resource-efficient in Windows environments, whereas Spring Boot benefits from JVM flexibility. These findings highlight potential trade-offs that will be integral to interpreting our test results. Likewise, Choma, Chwaleba, and Dzieńkowski [6] extended these discussions by comparing backend technologies, including Spring Boot, on CRUD efficiency and resource optimization. Their work provides a benchmark for evaluating the practical performance of our selected frameworks.

A complementary study [7] compared the performance of object-relational mapping (ORM) frameworks available in Java, such as Hibernate and EclipseLink. The authors demonstrated how different ORM solutions impact CRUD operation performance and resource management, making these findings particularly relevant for our evaluation of database interaction strategies within Spring Boot and ASP.NET Core.Further, Rymarski and Kozieł [8] highlighted the critical role of SQL query optimization, including indexing and query transformations, in enhancing CRUD performance. These insights emphasize the importance of efficient database interactions in our tests.

Expanding on this, Lachewicz [9] analyzed the performance of MySQL, MS SQL, and PostgreSQL in web applications, providing a framework for selecting databases to complement Spring Boot and ASP.NET Core. This research informs our decision to test multiple database systems to ensure a comprehensive performance evaluation.

Dhalla [10, 11] compared RESTful applications implemented in Spring Boot and ASP.NET Core, analyzing metrics like response times and throughput under practical implementation conditions. Her findings on the strengths and weaknesses of each framework offer valu-

able insights into how these technologies perform under load, directly aligning with our research objectives. Finally, Hericko et al. [12] explored object serialization in Java and .NET, highlighting its impact on CRUD operations. This work informs our focus on data handling
and optimization within the frameworks, contributing to a deeper understanding of their comparative performance. Piątkowska et al. [13] analyzed the use of .NET Core for web application development, focusing on its modular architecture and performance benefits. They highlighted the scalability and reliability of .NET Core in building high-performance web applications, providing a strong foundation for evaluating ASP.NET Core's potential in CRUD operations.

Mezei [14] discussed the pedagogical approach to teaching web development using ASP.NET Core MVC, emphasizing the framework's modularity and ease of use. This work underscores the importance of developer productivity and maintainability, which are essential considerations in our comparative analysis. The study by Zmaranda et al. [15] provided practical insights into SQL query optimization techniques, such as indexing and query restructuring, to enhance database performance. These findings are particularly relevant to our focus on optimizing database interactions in both Spring Boot and ASP.NET Core applications.

Krocz et al. [16] conducted a performance analysis of relational databases, including MySQL, PostgreSQL, MariaDB, and H2, in the context of web applications. Their research on query execution times and resource utilization provides valuable guidance in selecting appropriate databases for performance testing. Extending this analysis,

Andjelic et al. [17] performed a comparative analysis of MySQL and PostgreSQL, focusing on CRUD performance and reliability. Their work highlights key performance metrics, such as response times and throughput, which are critical for our framework evaluation. The research on the Hibernate framework [18] explored data persistence strategies and their impact on application performance. This study provides valuable insights into
Hibernate's role in CRUD operations, guiding our analysis of ORM performance in Spring Boot.

The Spring Boot documentation [19] provides essential guidelines for building scalable applications, covering dependency management, embedded servers, and efficient database integration. It emphasizes best practices for RESTful APIs, security, and performance tuning, ensuring robust CRUD operations.

Similarly, the ASP.NET Core documentation [20] details modern web application development, focusing on Entity Framework Core, middleware, and dependency injection. It outlines strategies for concurrency handling, security, and API optimization, enhancing CRUD performance and resource efficiency.

While the literature review provides a comprehensive foundation for analyzing CRUD performance in Spring Boot and ASP.NET Core, certain gaps remain in existing research. Many studies focus on isolated aspects such as ORM frameworks, query optimization, or REST API design, but few offer a direct head-to-head comparison of these two frameworks under identical conditions. Additionally, while several works highlight the importance of database configuration and system architecture, there is limited discussion on real-world deployment scenarios and how factors like cloud infrastructure or containerization impact CRUD efficiency. These limitations emphasize the need for our study, which seeks to provide a holistic, empirical evaluation of these frameworks under practical conditions, bridging the gap between theoretical insights and applied performance benchmarking.

4. Technologies and tools used

To evaluate the performance of CRUD operations in the Spring Boot and ASP.NET Core frameworks, tools were selected to enable a detailed analysis of response times, operations per second, and system resource usage. Apache JMeter was used to simulate application load, allowing the measurement of response times and system stability under heavy traffic. VisualVM was used for monitoring and profiling Java applications, making it particularly useful for analyzing memory and CPU usage in Spring Boot. Additionally, Postman supported API testing by enabling the sending of CRUD requests and analyzing responses, which facilitated the assessment of speed and accuracy in operation execution.

5. Research Environment

The testing environment plays a crucial role in conducting reliable and repeatable performance tests. The specifications of the computer used in this study are presented (see Table 1).

Table 1: The specifications of the testing environment

Processor	Processor - Intel(R) Core(TM) i5-10400 CPU @ 2.90GHz 2.90 GHz
RAM	16.0 GB
Graphics Card	GeForce GTX 1050 Ti
OS	Windows 11 Pro

Spring Boot and ASP.NET Core were run on the latest available versions at the time of testing (Spring Boot 3.4 and ASP.NET Core 8.0) (see [10, 11]). For data-base management, PostgreSQL 13 was used, enabling the evaluation of CRUD performance in different database configurations (see [8, 7]).

The network environment was configured to reflect typical conditions for web applications. The server and virtual machines were connected via a local area network (LAN).

Performance and resource usage were monitored using tools such as Prometheus, Grafana, and Actuator for Spring Boot, as well as Application Insights for ASP.NET Core. Prometheus and Grafana allowed for the collection and visualization of system and applica-

tion metrics, facilitating the identification of bottlenecks and performance analysis. Actuator and Application Insights provided detailed information about application behavior, including response times, memory usage, and CPU load (see [6, 10]).

PostgreSQL was installed directly on the operating system, enabling full utilization of the server's hardware resources, which was critical for obtaining reliable performance test results (see [8]). Known for its stability and efficiency, PostgreSQL supports advanced features such as table partitioning, replication, and extensions for SQL query optimization. The PostgreSQL configuration included settings for cache memory, connection limits, and parameters influencing query performance (see [9]). System logs and statistics were regularly monitored, allowing for real-time adjustments to maintain high performance during testing (see [7, 10]).

6. Experimental plan/research methodology

Research Subject:

The performance of CRUD actions in web applications built with the well-known frameworks ASP.NET Core and Spring Boot is the focus of this study. Response time, throughput, and CPU and RAM management under various load scenarios will all be used to assess and analyze this performance.

Research Object:

Web applications created using the ASP.NET Core and Spring Boot technologies make up the subject of this study. To guarantee a consistent testing environment, these applications will be specially developed for the experiment. To guarantee comparable outcomes, each application will carry out CRUD activities on the same database and data structure. The impact of various databases on the performance of CRUD activities in both frameworks will be assessed using a variety of database management systems, including PostgreSQL.

The study focuses on evaluating the performance of CRUD operations in web applications built using ASP.NET Core and Spring Boot. The comparison is based on response time, throughput, and system resource usage, including CPU and RAM consumption. To ensure a fair and consistent testing environment, both applications were developed specifically for the experiment and executed CRUD operations on the same database structure. PostgreSQL was chosen as the database management system to assess the impact of ORM frameworks on performance.

Two web applications were created for this study: one implemented in ASP.NET Core using Entity Framework Core and the other in Spring Boot using Hibernate. Both applications were designed to perform identical operations, allowing for an accurate comparison. The ASP.NET Core application was developed using WebAPI to handle HTTP requests, while the Spring Boot application followed a standard RESTful architecture. Automatic model validation was enabled in ASP.NET Core, ensuring that incoming request data was validated before processing. Similarly, Spring Boot leveraged its built-in validation mechanisms, ensuring consistent data integrity across both frameworks.

The test environment was configured to include PostgreSQL as the database, while performance tests were executed using tools such as Postman and VisualVM to track system performance metrics and response times. The analysis involved measuring the efficiency of CRUD operations in both frameworks, identifying performance bottlenecks, and evaluating how each technology handles database interactions. This setup ensured consistent testing conditions and enabled a fair, reliable comparison of both frameworks.

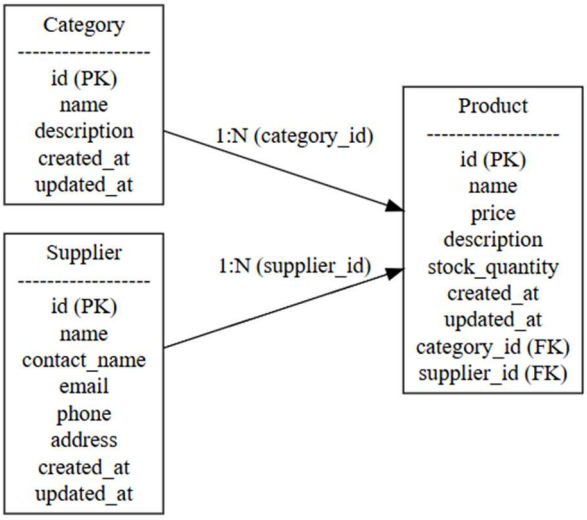

Figure 1: The ERD diagram for database.

The database used in the experiment consisted of three tables structured with one-to-many relationships (see Figure 1), with each table containing five million records. The dataset was designed to reflect real-world application scenarios, ensuring that performance tests were conducted under high-load conditions. CRUD operations were executed in large quantities to gather meaningful results. POST, UPDATE, and DELETE operations were each executed between one and two million times, while GET and GET JOINED operations were tested with request counts ranging from 100,000 to 200,000 to evaluate performance under varying workloads.

Since both applications were developed using ORM frameworks, the recorded execution times account for multiple components of the request processing pipeline. These measurements include the time required for ORM processing, such as entity mapping and SQL query generation, along with the actual execution of queries in PostgreSQL. Additionally, network transmission time was considered, including the dispatch of requests and retrieval of responses. Both applications automatically handled JSON serialization and deserialization, allowing seamless data conversion between HTTP requests and the internal object models. This process introduced an additional computational overhead, as each request required transformation from JSON format to application objects and vice versa. The use of ORM frame-

works also introduced an additional processing layer, impacting execution time compared to direct SQL queries, making it essential to consider the overhead caused by automatic entity management and query translation.

To ensure precise and repeatable performance measurements of CRUD operations, Postman was used to send a predefined number of requests and record execution times. The tests were conducted using prepared Postman scripts containing predefined API requests for each CRUD operation.

The study focused on measuring CRUD performance in a fixed testing environment. Each operation was tested in different scenarios with varying request volumes, allowing for an analysis of how system performance was affected by different load levels. The final results were averaged to provide a reliable representation of the actual performance of both frameworks.

To ensure accuracy and reliability, all measured response times were analyzed using statistical methods, including mean and standard deviation, The standard deviation was calculated for each CRUD operation to assess the variability of response times across repeated tests.

To determine whether performance differences between Spring Boot and ASP.NET Core were statistically significant, t-tests were conducted. These tests evaluated whether the observed differences in execution times were due to actual framework performance or random variations. Confidence intervals were computed to estimate the range within which the true mean response time was expected to fall, providing a more reliable assessment of performance differences.

The significance threshold (p-value) for the statistical tests was set to 0.05, meaning that if the observed difference had a p-value below 0.05, it was considered statistically significant. This approach ensures that minor fluctuations in response times are not mistakenly interpreted as meaningful performance differences.

Performance data was collected using monitoring tools such as VisualVM. The final dataset included key performance indicators such as response time, CPU load, and memory consumption for each CRUD operation. These results were compiled into comparative tables and visualized using bar charts to illustrate performance differences between Spring Boot and ASP.NET Core.

The analysis was conducted to identify trends, performance limitations, and the impact of database operations on overall system efficiency. The findings of the study provide insights into the suitability of each framework for various application scenarios. All results, interpretations, and conclusions were thoroughly documented, ensuring transparency in the research process. The final report includes a comprehensive description of the methodology, tools, and raw performance data.

7. Results

The analysis revealed that ASP.NET Core demonstrated lower execution times for DELETE and UPDATE operations, while Spring Boot exhibited comparable performance for GET and GET JOINED requests, with slightly faster execution for GET (see Table 2). The standard deviation values indicate that ASP.NET Core maintained more consistent execution times for DELETE and UPDATE operations, whereas Spring Boot showed slightly lower variability in GET and GET JOINED requests, suggesting more stable performance in complex queries (see Table 3). The execution time differences are clearly illustrated in the results, where ASP.NET Core significantly outperformed Spring Boot in DELETE and UPDATE operations, showing lower execution times, whereas Spring Boot's advantage in POST operations is evident from its noticeably shorter response time (see Figure 2-6). Additionally, GET and GET JOINED operations showed similar execution times across both frameworks, but Spring Boot handled complex queries slightly better, making it a more suitable choice for query-heavy applications (see Figure 2-6).

To determine whether the observed differences in performance between Spring Boot and ASP.NET Core were statistically significant, t-tests were conducted. These tests evaluated whether the variations in execution times were due to actual framework performance or random fluctuations. Confidence intervals were calculated to estimate the range in which the true mean response time was expected to fall, ensuring a more reliable assessment of performance differences.

The statistical analysis confirmed that the differences in DELETE and UPDATE operations were statistically significant ($p < 0.05$), suggesting that ASP.NET Core consistently performed these operations faster than Spring Boot. On the other hand, the differences in GET and GET JOINED operations were not statistically significant, indicating that both frameworks offer comparable efficiency in handling data retrieval tasks. The results also revealed that Spring Boot's advantage in POST operations was statistically significant, further reinforcing its suitability for applications with frequent data insertion.

In terms of resource utilization, ASP.NET Core generally had lower CPU usage across most request types, except for POST, where Spring Boot outperformed (see Table 4). Both frameworks maintained consistently high memory usage, exceeding 90% in all scenarios, indicating similar demands on memory resources (see Table 4).

Regarding resource efficiency, ASP.NET Core generally exhibited lower CPU usage across most request types, except for POST, where Spring Boot demonstrated superior efficiency (see Figure 7-11). This suggests that ASP.NET Core is more effective at managing computational resources in CRUD operations, except in scenarios involving heavy transactional loads such as POST requests.

Both frameworks maintained consistently high memory usage, exceeding 90% in all scenarios, suggesting that while CPU efficiency varies between the frameworks, memory consumption remains comparable (see Figure 7-11). This could influence framework se-

lection for resource-intensive applications, where memory management plays a critical role. Ultimately, the choice depends on which resource is more critical for the application.

Table 2: Average execution time of methods in Spring Boot and ASP.NET Core

Operation	Spring Boot time [s]	ASP.NET Core time [s]
DELETE	0.013	0.010
CREATE	0.019	0.180
UPDATE	0.013	0.013
GET	18.189	18.167
GET JOINED	18.593	19.255

Table 3: Standard Deviation of Execution Times in Spring Boot and ASP.NET Core

Operation	Spring Boot Std Dev [s]	ASP.NET Core Std Dev [s]
DELETE	0.002	0.001
CREATE	0.003	0.020
UPDATE	0.002	0.002
GET	0.150	0.140
GET JOINED	0.200	0.180

Table 4: Average performance metrics comparison for Spring Boot and ASP.NET Core across CRUD operations

Spring Boot	CREATE	CPU Usage	**12.54%**
		Memory Usage	**92.68%**
	DELETE	CPU Usage	**11.96%**
		Memory Usage	**91.88%**
	UPDATE	CPU Usage	**24.94%**
		Memory Usage	**92.65%**
	GET	CPU Usage	**20.38%**
		Memory Usage	**94.07%**
	GET JOINED	CPU Usage	**31.39%**
		Memory Usage	**94.28%**
ASP.NET Core	CREATE	CPU Usage	**13.40%**
		Memory Usage	**93.46%**
	DELETE	CPU Usage	**8.74%**
		Memory Usage	**91.04%**
	UPDATE	CPU Usage	**9.72%**
		Memory Usage	**93.60%**
	GET	CPU Usage	**17.70%**
		Memory Usage	**94.64%**
	GET JOINED	CPU Usage	**19.89%**
		Memory Usage	**94.47%**

The tabular data provided a clear numerical summary of execution times and resource consumption across all CRUD operations. These values helped quantify framework efficiency and highlighted areas of performance strength and weakness. However, to better illustrate trends and variability across multiple test runs, visualizations were essential. The graphs complement the tables by showing not only average execution times, but also distribution patterns and performance consistency. This visual context allows for quicker identification of outliers and performance anomalies that might not be immediately evident in the raw data. Together, the tables and figures offer a clear overview of framework performance.

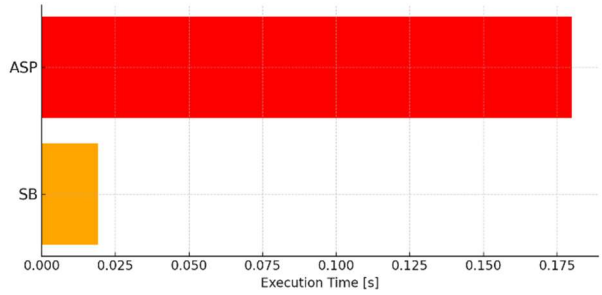

Figure 2: Execution times for CREATE.

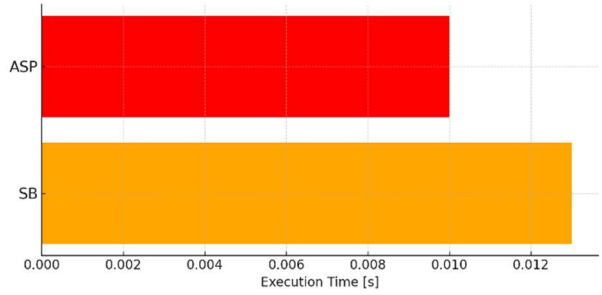

Figure 3: Execution times for DELETE.

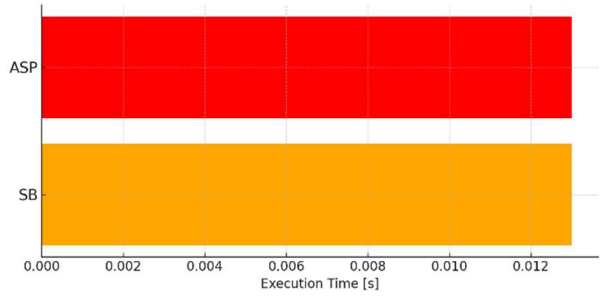

Figure 4: Execution times for UPDATE.

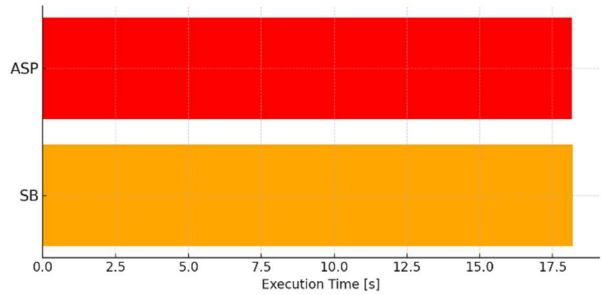

Figure 5: Execution times for GET.

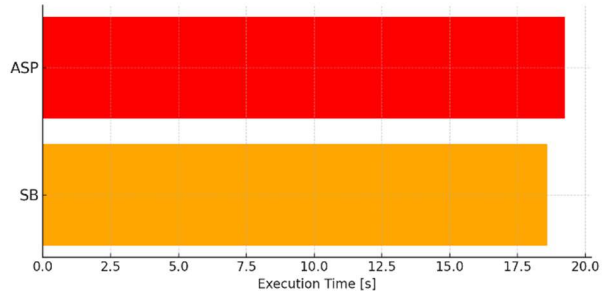

Figure 6: Execution times for GET JOINED.

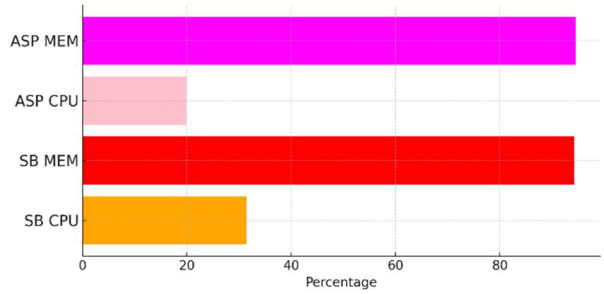

Figure 7: CPU and memory usage for GET JOINED.

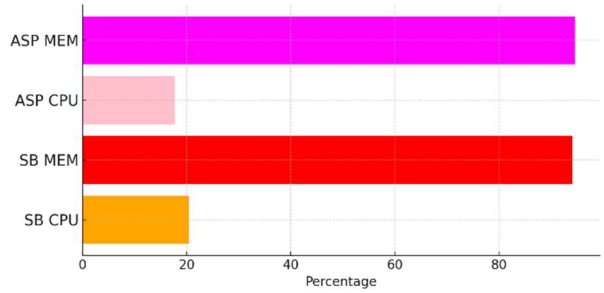

Figure 8: CPU and memory usage for GET.

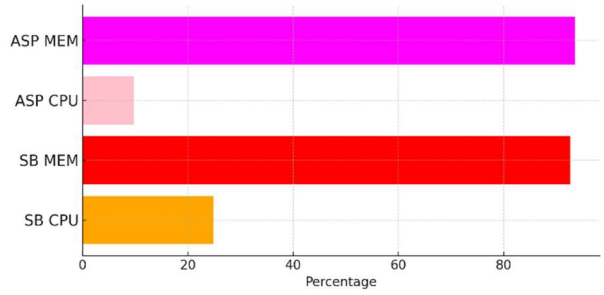

Figure 9: CPU and memory usage for UPDATE.

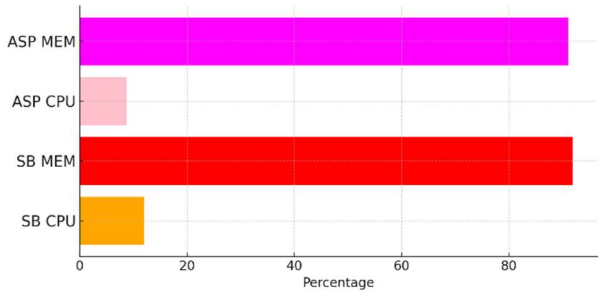

Figure 10: CPU and memory usage for DELETE.

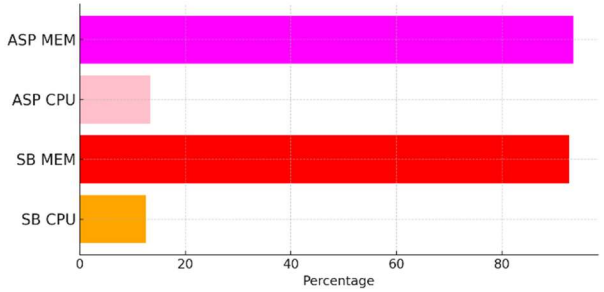

Figure 11: CPU and memory usage for CREATE.

8. Discussions

The performance tests compared the Spring Boot and ASP.NET Core frameworks in terms of CRUD operations, focusing on resource efficiency (CPU, memory) and response times across various scenarios. The results revealed that both frameworks have strengths and weaknesses, making them more suitable for different types of applications.

ASP.NET Core generally achieves better CPU efficiency, particularly in DELETE and UPDATE operations, where its CPU usage is significantly lower compared to Spring Boot (see Figure 3-4). This makes ASP.NET Core an ideal choice for applications with high DELETE or UPDATE operation loads, such as transactional systems or data-intensive applications. Moreover, its overall responsiveness in these scenarios positions it well for performance-critical systems.

On the other hand, Spring Boot excels in maintaining stable and predictable memory usage, while performing better in handling more complex queries, such as GET JOINED operations (see Figure 6). The shorter response times observed for Spring Boot in CREATE operations suggest it is more suitable for analytical applications or systems with a high volume of queries requiring data aggregation and complex joins (see Figure 2). Its consistent memory performance also makes it a good fit for resource-constrained environments where stability is critical. Finally, for GET operations, both frameworks demonstrate similar execution times, indicating that either can be effectively utilized for data retrieval tasks depending on system architecture and workload requirements (see Figure 5).

The research also underscored the importance of database optimization in CRUD performance, with PostgreSQL proving reliable and efficient for both frameworks. Tools like Postman and VisualVM provided

valuable insights into system behavior, aiding the analysis of performance bottlenecks. These results indicate that database configurations, including indexing strategies and query optimization techniques, can significantly impact overall CRUD efficiency, regardless of the chosen framework.

9. Conclusions

The study highlights key differences in the performance of Spring Boot and ASP.NET Core for CRUD operations, emphasizing the importance of selecting the appropriate framework based on specific project requirements (see Table 5). The results indicate that Spring Boot is particularly effective in CREATE (POST) operations, offering significantly shorter response times. Additionally, it excels in handling complex JOIN queries, making it more suitable for data-intensive applications that require query aggregation and efficient data retrieval.

Its performance remains consistent even under increased database load, making it well-suited for enterprise systems with large volumes of relational data. Integration with tools like Hibernate and Spring Data further streamlines development and performance tuning.

Table 5: Framework recommendation table for CRUD operations

Operation	Framework	Reason
CREATE (POST)	Spring Boot	Shorter response times
READ (GET)	Equivalent	Similar
READ (GET JOINED)	Spring Boot	Better handling of complex queries
UPDATE (PUT)	ASP.NET Core	Lower CPU usage
DELETE	ASP.NET Core	Lower CPU usage, faster execution times

On the other hand, ASP.NET Core demonstrates better optimization for UPDATE and DELETE operations, with lower CPU usage and faster execution times. This makes it an ideal choice for transactional systems, where frequent updates or deletions occur, ensuring high efficiency under heavy workloads. While both frameworks perform similarly in GET operations, their differences in CPU and memory utilization suggest distinct advantages depending on the application's needs.

The study also revealed that Spring Boot maintains stable and predictable memory usage, making it suitable for environments where consistent performance is crucial. Meanwhile, ASP.NET Core's superior CPU efficiency positions it as the better choice for high-performance, resource-intensive applications where computational load is a critical factor.

In conclusion, the decision between Spring Boot and ASP.NET Core should be guided by project priorities. If CPU efficiency and transaction-heavy operations are the focus, ASP.NET Core is the more effective option. However, for applications involving complex queries, data aggregation, or requiring stable memory usage, Spring Boot provides better overall performance. Future research could expand on these findings by exploring additional frameworks, different database configurations, and testing under distributed system environments to further refine performance insights.

Bibliography

[1] M. Kaluža, M. Kalanj, A comparison of back-end frameworks for web application development, Zbornik Veleučilišta u Rijeci 7(1) (2019) 317-332, https://doi.org/10.31784/zvr.7.1.10.

[2] M. Masse, REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces, O'Reilly Media, 2011.

[3] A. M. Bonteanu, C. Tudose, Performance Analysis and Improvement for CRUD Operations in Relational Databases from Java Programs Using JPA, Hibernate, Spring Data JPA, Applied Sciences 14(7) (2024) 2743, https://doi.org/10.3390/app14072743.

[4] A. M. Bonteanu, C. Tudose, Multi-platform Performance Analysis for CRUD Operations in Relational Databases from Java Programs Using Hibernate, Big Data Intelligence and Computing 13864 (2023) 275-288, https://doi.org/10.1007/978-981-99-2233-8_20.

[5] P. Kopyl, J. Smołka, Comparison of ASP.NET Core and Spring Boot ecosystems, Journal of Computer Sciences Institute 22 (2022) 40–45, https://doi.org/10.35784/jcsi.2794.

[6] D. Choma, K. Chwaleba, M. Dzieńkowski, The Efficiency and Reliability of Backend Technologies: Express, Django, and Spring Boot, Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie 13(4) (2023) 73–78, https://doi.org/10.35784/iapgos.4279.

[7] P. Rymarski, G. Kozieł, Analysis of the possibilities of optimizing SQL queries, Journal of Computer Sciences Institute 19 (2021) 151–158, https://doi.org/10.35784/jcsi.2641.

[8] K. Lachewicz, Performance analysis of selected database systems: MySQL, MS SQL, PostgerSQL in the context of web applications, Journal of Computer Sciences Institute 14 (2020) 94–100, https://doi.org/10.35784/jcsi.1583.

[9] M. Połeć, J. Pitera, Comparing the Performance of the Object-Relational Mapping Program-ming Frameworks Available in Java, Journal of Computer Sciences Institute 22 (2022) 59–65, https://doi.org/10.35784/jcsi.2810.

[10] H. K. Dhalla, A Performance Comparison of RESTful Applications Implemented in Spring Boot Java and MS.NET Core, Journal of Physics: Conference Series

1933(1) (2021) 012041, https://doi.org/10.1088/1742-6596/1933/1/012041.

[11] H. K. Dhalla, Benchmarking the performance of RESTful applications implemented in spring boot Java and MS.Net core, Journal of Computing Sciences in Colleges 36 (2020) 178, https://dl.acm.org/doi/10.5555/3447080.3447113.

[12] M. Hericko, M. B. Juric, I. Rozman, S. Beloglavec, A. Zivkovic, Object serialization analysis and comparison in Java and .NET, ACM SIGPLAN Notices 38(8) (2003) 44–54, https://doi.org/10.1145/944579.944589.

[13] E. Piątkowska, K. Wąsik, M. Plechawska-Wójcik, The use of .NET Core in web applications development, Journal of Computer Sciences Institute 7 (2018) 142–149, https://doi.org/10.35784/jcsi.663.

[14] R. A. Mezei, Teaching web development using ASP .Net core MVC, Journal of Computing Sciences in Colleges 38(1) (2022) 116–117, https://dl.acm.org/doi/10.5555/3575618.3575633.

[15] D. Zmaranda, L. L. Pop-Fele, C. Gyorodi, R. Gyorodi, G. Pecherle, Performance Comparison of CRUD Methods using NET Object Relational Mappers: A Case Study, International Journal of Advanced Computer Science and Applications(IJACSA) 11(1) (2020) 55-65, http://dx.doi.org/10.14569/IJACSA.2020.0110107.\

[16] K. Krocz, O. Kizun, M. Skublewska-Paszkowska, Perfomance analysis of relational databases MySQL, PostgreSQL, MariaDB and H2, Journal of Computer Sciences Institute 14 (2020) 1–7, https://doi.org/10.35784/jcsi.1565.

[17] S. Andjelic, S. Obradovic, B. Gacesa, A Performance Analysis of the Dbms - MYSQL Vs POSTGRESQL, Communications - Scientific Letters of the University of Zilina 10(4) (2008) 53-57, https://doi.org/10.26552/com.C.2008.4.53-57.

[18] Q. Wu, Y. Hu, Y. Wang, Research on Data Persistence Layer Based on Hibernate Framework, In 2010 2nd International Workshop on Intelligent Systems and Applications (IWISA) IEEE (2010) 1–4, https://doi.org/10.1109/IWISA.2010.5473662.

[19] Spring Boot documentation, https://spring.io/projects/spring-boot, [03.02.2025].

[20] ASP.NET Core documentation, https://dotnet.microsoft.com/en-us/apps/aspnet, [03.02.2025].