DOI https://doi.org/10.30525/978-9934-26-459-7-10

TESTING AUTOMATISATION IN MODERN WEB-APPLICATIONS

Umarjon Khamidullaev*
ISMA University of Applied Sciences, Latvia
**Corresponding author's e-mail: khamidullayev011@gmail.com*

Abstract
There are many different programming languages, frameworks and architectural styles available today for developing web services. It is often difficult for a developer to decide what the best implementation solution would be and how it would affect system performance.
Key words: *automation testing, quality control, test automation tools.*

1. Introduction

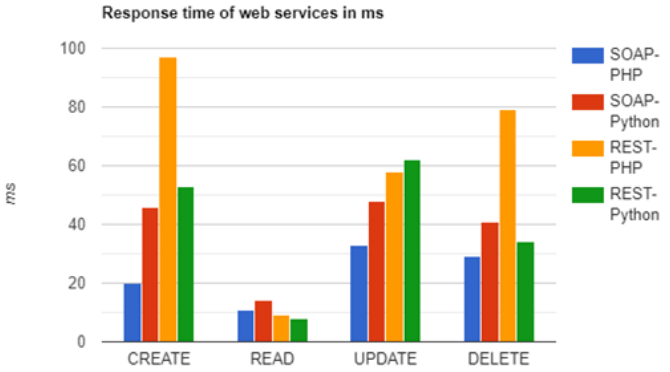
Web services have acted as the catalyst for digital transformation over the past two decades. What began as basic protocols designed to foster inter-system communication have matured into sophisticated architectures that ensure seamless data exchange across diverse platforms. The transition from the monolithic constructs of the early 2000s to today's agile microservices demonstrates the adaptability of web services, echoing the ever-changing tech environment. By offering standardized interfaces, web services have not just promoted system compatibility they have been the bedrock for innovation, enabling businesses to integrate various software components with ease [1].

However, with the diversification of web service technologies, a significant challenge has emerged: the conundrum of selection. Developers and enterprises find themselves navigating a complex array of programming languages, developmental frameworks, and architectural patterns. While this vast array of choices showcases the rich evolution of web technologies, it often introduces uncertainty. The technological choices made have lasting implications, affecting a system's efficiency, scalability, and maintainability. Thus, the need for an evidence-based assessment of these tools and architectures is paramount [2].

Overview

The experimental section of this research stands as its backbone, offering tangible insights into the actual workings and efficacy of web services

implemented using different paradigms and languages. This section encapsulates the meticulous process of setting up, developing, testing, and evaluating web services. While the choice of languages, PHP and Python, and architectures, REST and SOAP, might vary, a few constants persist, ensuring uniformity in evaluation and analysis.



Methodology: Using Apache JMeter, a series of CRUD (Create, Read, Update, Delete) operations were sent to each of our web services: SOAP-PHP, SOAP-Python, REST-PHP, and REST-Python. The objective was to monitor and compare the response time (in milliseconds) for each type of request across the services.

Decision

My analysis reveals the strengths and potential weak spots of each web service, laying a foundation for further refinement and optimization. While certain services like SOAP-PHP shine in many aspects, others like REST-Python raise flags in specific scenarios. Ultimately, the choice and optimization of a web service will hinge on the specific use-case requirements, anticipated load, and the criticality of performance. This data-driven insight empowers developers and businesses to make informed decisions, ensuring web services that are both robust and efficient.

Conclusion

As I conclude this research, it's essential to synthesize the findings, observations, and analyses into cohesive insights that have been derived from this experimental work. The primary aim of this thesis was to understand the nuances of different web services, contrasting and comparing their performance across various parameters, including response times and resource utilization.

References

1. Pautasso, C., Zimmermann, O., & Leymann, F. (2008). Restful web services vs. “big” web services: making the right architectural decision. Proceedings of the 17th international conference on World Wide Web. ACM.

2. Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures. Doctoral dissertation, University of California, Irvine.

3. Lerman, K., & White, T. (2008). Designing web services and service-oriented architectures for dynamic and adaptable systems. *Software: Practice and Experience*, 38(13), 1341–1363.

4. Emmerich, W. (2000). Software engineering and middleware: a roadmap. *Proceedings of the Conference on The Future of Software Engineering*, 117–129.

5. Winer, D. (2001). XML-RPC Specification. UserLand Software