International Journal of Science and Research (IJSR)
ISSN: 2319-7064
ResearchGate Impact Factor (2018): 0.28 | SJIF (2019): 7.583

Optimizing Web Applications Performance with Java: Best Practices

Pavan Kumar Joshi

Fiserv, USA
Email: *pa1n12[at]gmail.com*

Abstract: ***In the era of pervasive computing, the demand for high - quality web applications has surged, prompting developers to adopt robust frameworks and programming languages. This paper explores the significance of Java in web application development, emphasizing its reliability, portability, and object - oriented nature. By leveraging Java's features, developers can create scalable applications that meet evolving customer and business requirements. We discuss the architectural principles of web applications, as an example focusing on the performance implications of adopting a 3 - Tier architecture using Java technologies such as Servlets, JSP, and AJAX. Furthermore, we analyze factors affecting web application performance, including user experience, server capabilities, network conditions, and database efficiency. The insights provided in this paper aim to guide developers in optimizing web applications, ensuring enhanced user satisfaction and operational effectiveness.***

Keywords: Web Application, Optimization, Best Practices, Performance, 3 - Tier Architecture.

1. Introduction

Web application development has recently focused heavily on a variety of quality - related challenges [1] in response to the skyrocketing demand for web apps in the age of ubiquitous and pervasive computing. The intense level of competition in the online application industry has made developers of web applications more price - conscious. The ability of a web application to withstand the ever - evolving demands of both customers and businesses is a key asset [2].

The development of high - quality web applications is an arduous and demanding process. To produce high - quality web - applications, however, one must have the correct development process, methodologies, tools, and people behind them. To achieve high - quality web - applications with a simple, efficient, and resilient development process, the development platform is crucial since it affects related development processes, methodologies, tools, and people [2].

For application programming interfaces (APIs) and online applications, Java has long been a top choice. It is a stable and trustworthy language that gives programmers access to a wealth of resources for creating high - performance, scalable apps. People and companies alike use Java for various purposes, including chatting, providing services, sharing content, and much more besides. This includes conversing, providing services, exchanging materials, and much more. [3] [4]. The Java platform is an open - source, standards - based, operating system - and hardware - independent platform for creating and running distributed corporate applications. Because Java programs do not favour any one vendor over another, the company is free from vendor lock - in [2].

A rapid growth of web applications in today's digital landscape necessitates a focus on performance, scalability, and user experience. As businesses increasingly rely on online platforms to meet customer demands, the challenges associated with developing high - quality web applications have become more pronounced. This paper is motivated by the need to explore effective strategies for building robust web applications using Java, a language known for its reliability and versatility. By addressing the architectural principles, performance factors, and best practices in Java web development, we aim to equip developers with the insights needed to enhance application efficiency and user satisfaction. The following paper contributes as:

- Provides a comprehensive overview of Java's frameworks (Servlets, JSP, AJAX) that facilitate the development of high - performance web applications.
- Discusses the advantages of adopting a 3 - Tier architecture, highlighting its role in separating concerns and improving scalability and maintainability.
- Identifies and analyzes critical factors affecting web application performance, including user experience, network conditions, server efficiency, and code quality.
- Offers practical recommendations for optimizing web applications through effective database management, efficient coding practices, and hardware considerations.
- Illustrates the application of theoretical principles with real - world examples, demonstrating how Java - based solutions can enhance web application performance in high - traffic scenarios.

1.1 Structure of the paper

The rest of the paper are organized as: Section II - introduction overview of java programming language, Then Section III - performance testing web applications, in section IV - web applications performance with java: an example, Section V - analysis of factors affecting web application system performance, Section VI discussed as literature review, Section VII provide the best practices for optimizing web application performance with Java, at last provide the Conclusion and future work.

2. Overview Of Java Programming Language

Java is a widely used object - oriented, high - level programming language that may be used to create safe, cross - platform applications. It has become one of the most popular

languages for online and enterprise application development since James Gosling of Sun Microsystems introduced it in 1995. With Java's support for the "write once, run anywhere" (WORA) concept, programs written in Java may run on any device running the Java Virtual Machine (JVM), independent of the device's operating system or hardware [5]. The following seven requirements must be met in order for object - oriented programming to be considered pure: No.1: Bequests (2) Data Encapsulation 3. Polymorphism 4. Decoupling 5. Every type that is predefined is an object.6. Messages are sent to objects to accomplish all actions.7 Objects are the only type that may be user - defined. Figure 1 displays the JAVA architecture.

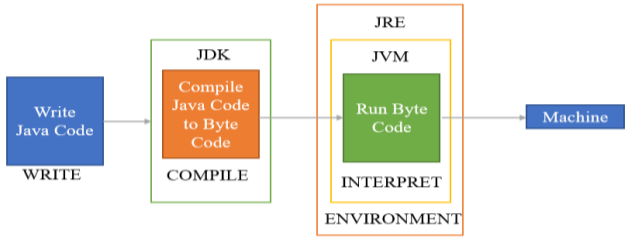

Figure 1: Java Architecture [6]

Compilation and Interpretation are the two main steps of the Java programming language.

- The compiler receives the Java source code.
- It is transformed into byte codes by the Java Compiler.
- The JVM takes the bytes codes and turns them into machine code.
- It is the machine itself (the operating system) that runs the code.

Java is not a true object - oriented programming language as it does not support the #5 requirement. However, it is an object - oriented programming language. In order to construct a robust online or mobile application, several technologies and frameworks make use of object - oriented ideas.

- **Java:** Java is a platform or language that is safe, well - structured, object - oriented, and works at a high level. Given its platform independence, it can be executed in any software or hardware environment.
- **Servlet:** Web applications may be developed and deployed on servers using the Servlet technology, which is also based on Java. The employment of this technology is not without its many pros and cons.
- **JSP:** This is similar to the Servlet technology used to build web applications, but it adds features like expression language, JSTL, etc.

2.1 Key Principles of Java

Java has become the most essential programming language in the era of internet because of its web application environment and its built - in networking environment [7]. There are five main principles of Java language listed below as:

- **Java is Architecture - neutral and portable:** Java's universal byte - code simplifies porting. Programs execute slower than native executables due to the overhead of byte - code interpretation into machine instructions. Java is platform - independent. The Java Virtual Machine converts Java byte - code into the platform's native machine language.
- **Java is simple and object - oriented:** Java uses C syntax and C++ technologies to produce bug - free code and enhance object - oriented principles. Java eliminates the most perplexing element of memory allocation and de - allocation, making it bug - free. Programmers spend less time on memory allocation and garbage collection since the JRE does it. The whole thing is object - oriented.
- **Java is Robust:** Java has efficient memory allocation and auto garbage collection. It provides superior exception handling and type checking compared to other programming languages. Java's compiler and interpreter check for basic syntactic and semantic problems and run - time code, making the architecture resilient.
- **Java is dynamic, interpreted and threaded;** Java is dynamic because of byte - code (class files). It is dynamic because it stores all the types needed at runtime to check and resolve object access rights. Any platform can execute Java source code. It loads class files during runtime, therefore anything at runtime is dynamic. Java programs must be executed using the interpreter since byte - code runs anywhere. The programs are compiled into byte - code Java Virtual Machine code. Multithreading allows multiple tasks to run simultaneously. Multithreading makes Java powerful. Lightweight multithreaded may do several tasks simultaneously.
- **Java is secure:** Java's security paradigm prohibits fraudulent users by not allowing read or write access to files or runtime process creation. It uses “sandbox” to protect users against malicious programs downloaded from a network. Java programs can only run in the sandbox. The JVM checks executable files for malware and other threats as they are built by the Java compiler from byte - code in class files.

3. Performance Testing Web Applications

Web apps are often recognised as the foundational elements of standard SOA applications. Web application components are crucial to the operation of such an application system. People who use the Internet often engage with websites, and a lot of those websites are dynamic. These websites don't only provide static web pages; they create content according to user demands. Figure 2 indicates that these dynamic websites are more accurately referred to be Web Applications Architecture because of the flexibility and interaction they provide [8].

- Client tier (the browser) - displays data that has been requested.
- Presentation layer, also known as Middle Tier or Application Server (the Web server): manages the business logic and provides client (s) with data.
- The database server, which is part of the data storage layer, is responsible for storing the system's data in a relational database.

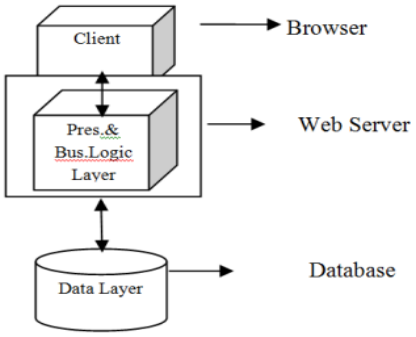

Figure 2: Web Application Architecture [9]

A web application performance testing tool's main function is to simulate user load [4]. The number of concurrent users accessing the program under test is a measure of load. Their status as "virtual users" stems from the fact that the testing instrument just mimics their actions. One real person using the app is symbolized by each virtual user [3].

4. Web Applications Performance With Java: An Example

In this example, the web application is designed based on a 3 - Tier architecture, replacing the existing 2 - Tier Client/Server (C/S) model. The application leverages Java open - source libraries such as Struts, ibatis, and Ajax, enabling ease of development and maintenance. The main goal is to combine the user - friendly interface of C/S models with the flexibility and scalability of a web - based system. This new architecture enhances development efficiency, reduces maintenance efforts, and improves the overall user experience [10].

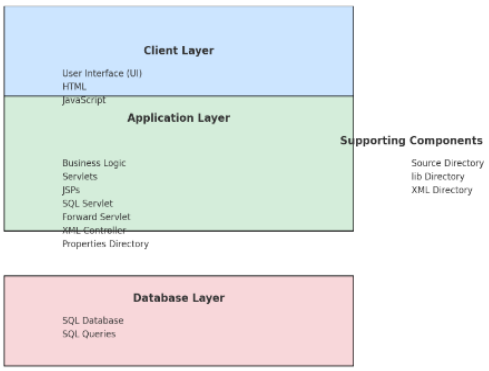

Figure 3: Web Applications Performance with Java

Here's figure 3 representation of the architecture for optimizing web application performance with Java. It illustrates the 3 - Tier architecture, highlighting the Client Layer, Application Layer, and Database Layer, along with supporting components like directories and libraries. Each layer is organized to show the specific roles and components involved in the system, providing a clear view of how the application is structured. The each steps of web application development with Java discussed below:

4.1 Directory Structure

The directory structure organizes the web application into multiple components, each serving a specific role:

- Properties Directory: Contains environment configuration files for the web application.
- Source Directory: Holds Java business logic files that are critical to application functionality.
- Web Page Directory: Stores static content like HTML, JavaScript, and images, ensuring a clean separation between presentation and business logic.
- WEB - INF Directory: This is where servlets, JSP files, and related classes reside. It is secured from direct URL access, protecting critical logic from external threats.
- Classes Directory: Contains servlet classes, JavaBeans, and other related files, facilitating modular and maintainable code.
- lib Directory: Stores necessary external libraries, in compressed formats like. jar or. zip.
- XML Directory: Holds SQL Data Manipulation Language (DML) statements used within the web application, separating them from Java or JSP code for easier management.

4.2 Web Application Whole Process

The web application employs an Ajax - driven process, enabling asynchronous requests for efficient data communication between client and server. The steps are as follows:

- User Request via JavaScript: A user initiates a request through a JavaScript function, which then communicates with the web server via the Ajax engine.
- Business Logic Processing: Upon receiving the HTTP request, the web application server processes the business logic and communicates with the database server if necessary.
- XML Data Response: The server returns the processed result in XML format to the client, ensuring fast data retrieval and reducing bandwidth usage.
- Callback Function Processing: The XML data is processed by the callback function in the browser, rendering the required information for the user.

4.3 Client Process Flow

The client process revolves around presenting dynamic data and responding to user requests:

- Initial GUI Request: The client requests the graphical user interface (GUI) using an extended JSTL Custom Tag Library.
- JavaScript Function Execution: All user requests are managed by JavaScript functions, which validate and forward them to the server via the Ajax engine.
- XML Response Parsing: The XML data returned from the server is parsed and displayed on the client screen using defined callback functions.

4.4 Server Process Flow

On the server side:

- Loading the Application: The web application is loaded into memory, where configuration settings and business logic are prepared using properties files.
- Processing Requests: When a client request arrives, the server processes it, communicates with the database if necessary, and sends the result back in XML format.

- SQL Handling via XML Controller: SQL queries are handled via an XML controller that retrieves and executes SQL statements from XML files, optimizing database interactions.

4.5 Compile Process Flow

The web application leverages Apache ANT, a Java - based build tool, to manage the compile process:

- Building with ANT: The build process compiles Java classes, packages them into. jar files, and moves them to the appropriate directories within the application.
- Automated Deployment: Once compiled, the libraries and class files are automatically placed in the WEB - INF directory, making the deployment process smooth and efficient.

4.6 SQL Servlet Process

The proposed SQL servlet handles SQL - related requests by reading SQL DML statements from the XML directory. Key steps include:

- SQL Parsing and Execution: SQL statements are parsed based on identifiers (SQL ID) and processed in the database.
- Multi - SQL Handling: The servlet can manage multiple SQL queries simultaneously, enhancing efficiency and reducing the need for multiple servlets.
- XML Response: The result is returned in XML format, eliminating the need for additional servlets for database interactions.

4.7 Forward Servlet Process

The Forward servlet is responsible for handling business logic in JSP files, offering several benefits:

- Efficient Maintenance: Since the business logic resides in JSPs, changes to the code do not require server restarts, reducing the risk of service interruptions.
- XML Output: After processing the JSP logic, the servlet forwards the result in XML format, streamlining the integration of presentation and business logic.

The described Java - based web application architecture is designed to enhance performance, scalability, and maintainability. By utilizing a 3 - Tier model, separating concerns, and leveraging industry - standard tools like Apache ANT, the system efficiently manages client requests and server processing while maintaining a clean, modular structure.

5. Analysis of Factors Affecting Web Application System Performance

Taobao and Jingdong are two examples of massive e - commerce Web applications that immediately spring to mind when one thinks of high concurrency software. The server experiences a surge in traffic during the yearly "double eleven, " leading to issues including poor page response times, crashes, and commodities that cannot be exchanged. As a result, the platform's functioning and user experience are negatively impacted [11].

Thus, it is important to consider potential operational issues while designing the system, include suitable technologies to address them, and make it easy to expand and maintain. As a result, studying what makes the Web application system tick is essential. It is important to consider the following details while dissecting the elements that influence the Web application system's performance [12]:

1) User experience layer

Many issues, like slow system throughput, high rates of front - end script errors, high rates of asynchronous requests, static resource 404 errors, and so on, might impact the user experience when they use a web application. The request will time out if the requested resource has any of the aforementioned issues, leading to a sluggish response and an unpleasant experience for the user. Consequently, we need to check whether the system has the aforementioned issues.

2) The network layers

A web application's response time is directly proportional to the bandwidth available on the server's network. In theory, static resources with a large exit bandwidth should load more quickly, and the ping command is a good way to check the communication speed of a network.

3) Server layer

The effectiveness of the system's service is both restricted by the number of requests that the server can handle and by the current condition of the server's load. For that reason, it's important to examine the server's settings and look for any issues.

4) Code layer

The code's quality also affects the system's speed. The user experience layer may display a sluggish response or possibly crash if the system contains several enquiries, such as multiple nested loop queries. The efficiency of the Java programming language must, therefore, be considered throughout optimisation and development.

5) Data storage layer

The database concurrency issue is particularly challenging to resolve in systems that support high concurrency web applications. When analyzing the system database's performance, it's important to zero in on SQL, common database operations, and the concurrent "dirty read" issue because of the constraints imposed by locking mechanisms, concurrency, and SQL statements.

6) Physical hardware layer

Processor and memory - dependent system business processing mostly reflects this layer.

6. Literature Review

Developers don't have to put in as much effort after incorporating the framework's numerous pre - built features, which are provided in the form of jars.

J. Correa et al. (2014), Google App Engine and Google Web Toolkit are frameworks for building web applications. These technologies have a specific way of working and certain limitations that impact the design and functionality of these

apps. On the other hand, they offer great benefits like a more intuitive interface, faster performance, more scalability, and the ability to use specific services that make applications work with different systems [13].

Rajeev BV et al. (2015), shows how developers may use several strategies to optimise and assess the performance of mobile web applications at various levels, including HTML, CSS, and Java script, as well as during deployment. This article starts with a use case application, sets performance goals and baselines, and then uses a number of methods to determine the impact of each method on the application's performance [14].

Javier Verdú et al. (2016), releases the first study to examine the scalability of performance in web applications that use numerous workers in parallel. Here, we focus in on two case examples that illustrate distinct approaches to worker execution. The Web Workers API, which is part of the more modern HTML5 standard, enables the execution of JavaScript programs on many threads, or workers. Nevertheless, the inner workings of the browser's JavaScript virtual engine conceal any direct correlation between the number of workers and threads operating in the browser and the utilisation of the processor's logical cores. Consequently, programmers are clueless about the ideal amount of workers for parallel JavaScript scripts and how performance really grows across various settings. We have tested three of the most popular web browsers and found that performance scales across a variety of parallel processing microarchitectures. On top of that, we investigate how web app performance is affected by running apps simultaneously. These findings have important implications for future methods that aim to automatically determine the optimal workforce size in order to maintain system responsiveness and user experience in the face of unpredictable changes in system workload, while also optimising resource utilisation [15].

Peng Li et al. (2017), The goal of their research is to find a programming language and middleware abstraction that provides a different way to build client/server web applications. This approach would help with issues like component coordination between the client and server, as well as the flexibility to map components to different physical locations. In order to circumvent issues with the two paradigms mentioned in this Introduction, they are developing a JavaScript interpreter and middleware that facilitates modular coordination. In this method, developers would mediate the client - server workflow by writing a distinct coordination specification. They have changed a popular open - source web app, Java Pet Store, using my framework, and then tested it to see whether it has little performance overhead, all in an effort to establish that my framework is effective [16].

Yohei Ueda et al. (2017), performed a comparison between two dynamically compiled languages—JavaScript and Java—and the widely used statically compiled language Go. For each of the three languages, we tested three different implementations of the Acme Air benchmark. Following some minor tweaks to the server settings, our experiments showed that the Go code outperformed the JavaScript and Java implementations in terms of throughput, with the former achieving a 3.8x increase and the latter a 2.4x boost [17].

Palacios et al. (2018) research presents the analysis of the Prime Faces and Rich Faces libraries, in their average page response time dimensions, and average Ajax response time, to determine which one offers better performance. The analysis was carried out through a Web page N layers, applied in the management of academic tutoring University, a test environment was set up on an Apache Tomcat Web server in a Linux environment, with each of the libraries, also used the JSF, Prime Faces and Rich Faces technologies. Performance tests were based on the Neoload tool, simulating 350 requests per second observing significant differences between the two component libraries [18].

K Munonye et al. (2018) analyses and contrasts the efficiency of REST APIs built using Java and Microsoft. Net. We used Jersey to build RESTful APIs in Java and then deployed them on top of the Apache Tomcat server. The same parameters were used to construct RESTful APIs with MS. Net as well. The Microsoft IIS Server was used to deliver Net Framework 6.0. The results were compared and assessed across several test cases. In terms of processing PUT operations, the findings demonstrated that the. net API outperformed the Java - based API. Compared to the other API, the Net API had an 11.6% faster processing time. On the other hand, the API that was based on Java improved the speed of the GET operations by simultaneously processing 80.36 percent more data [19].

Ansari et al. (2019), Programmers used key Java ideas like Applet, Multithreading, and Polymorphism to create standalone or mobile applications. However, web applications also need extra technologies such as Servlet and JSP. Spending more time and money developing a web app utilising servlet, JSP, and core java methods increases the likelihood of new issues and requires rewriting a large amount of code in the event that the program needs future modifications. Many large software organisations are working on new frameworks and tools to alleviate these issues, with the hope of reducing the workload and associated costs of programmers. Any application may be produced more quickly and at a lower cost by using one of the common Java frameworks that are currently available. There are many different frameworks available, each with its own set of modules that may be used according to the needs of a project. Some examples are Spring, Hibernate, Spring Boot, Struts, and EJB. These frameworks cannot be used without first doing the necessary research and analysis [20].

This table 1 outlines the key comparisons between the research papers, highlighting their technological focus, performance metrics, and major findings.

Table 1: Summarizing the key aspects of the related work

Author (s)	Focus Area	Technologies /Frameworks	Performance Metrics/Analysis	Key Findings

J. Correa et al. [13]	Web application development with Google Web Toolkit (GWT) and Google App Engine	GWT, Google App Engine	Design and functionality considerations under framework restrictions, usability, UI, scalability	Great benefits in UI, performance, and scalability but constrained by framework limitations
Rajeev BV et al. [14]	Optimizing mobile web application performance at various layers	HTML, CSS, JavaScript	Use case application with performance baselining and optimization techniques applied across layers	Demonstrates the importance of multi - layered optimization, improving performance step by step
Javier Verdú et al. [15]	Performance scalability of parallel web applications with multiple workers	HTML5, Web Workers API	Analysis of scaling JavaScript performance with multiple threads/workers, assessing co - running application impact	Provides insights into the optimal number of workers for best performance across browsers and hardware configurations
Peng Li et al. [16].	Client/server Web application coordination via middleware and JavaScript interpreter	Middleware, JavaScript interpreter	Modular coordination in client/server workflows, evaluating performance overhead with the Java Pet Store	Coordination framework provides low - performance overhead, offering flexibility in client - server application development
Yohei Ueda et al. [17]	Comparison of statically compiled and dynamically compiled languages for server - side web applications	Go, JavaScript, Java	Acme Air benchmark to evaluate throughput after tuning server configuration	Go significantly outperforms JavaScript and Java in throughput (3.8x vs. JS, 2.4x vs. Java)
Palacios et al. [18]	Performance comparison of Prime Faces and Rich Faces libraries for web application development	Prime Faces, Rich Faces, JSF, Apache Tomcat	Average page and Ajax response times analyzed using Neoload in simulated load tests (350 requests/second)	RichFaces shows better performance in average page response and Ajax times
K Munonye et al. [19]	Performance comparison of REST APIs implemented in Java and Microsoft. Net	Java (Jersey),. Net Framework, Apache Tomcat, IIS	Performance comparison of RESTful APIs (PUT, GET operations)	. Net API performs better for PUT operations, while Java API is superior for GET, with 80.36% better processing rate
Ansari et al. [20]	Comparison of Java frameworks for web application development	Java (Servlet, JSP, Spring, Hibernate, Struts)	Comparative analysis of Java frameworks on developer effort, cost, and ease of modification	Frameworks like Spring and Hibernate reduce development time and cost, simplifying updates and modifications compared to traditional Servlet/JSP approaches

7. Best Practices for Optimizing Web Application Performance with Java

Here's a concise summary of best practices for optimizing web application performance with Java:

- Code Optimization: Focus on using efficient algorithms and data structures. Minimize object creation in loops to reduce garbage collection overhead and implement lazy loading to defer data loading until necessary.
- Database Optimization: Utilize connection pooling to manage database connections efficiently. Create indexes on frequently queried columns and optimize SQL queries by avoiding SELECT * and implementing pagination.
- Caching Strategies: Implement in - memory caching using libraries like Ehcache or Redis for frequently accessed data. Use HTTP caching strategies, such as ETag and Cache - Control headers, to reduce server load for static resources.
- Asynchronous Processing: Utilize Completable Future for concurrent tasks and consider using message queues (e. g., RabbitMQ, Kafka) to decouple processing and handle tasks asynchronously.
- Web Server and Application Server Tuning: Optimize thread pool configurations based on expected load and set appropriate connection timeouts to prevent hanging requests.

Front - End Optimization:

- Minify JavaScript, CSS, and HTML files, and use Gzip or Brotli compression to reduce payload sizes. Leverage a Content Delivery Network (CDN) for serving static resources.
- Monitoring and Profiling: Use profiling tools like VisualVM and performance monitoring solutions like Prometheus with Grafana to identify bottlenecks and monitor application performance.
- JVM Tuning: Choose the right garbage collector (e. g., G1GC, ZGC) and tune GC parameters to minimize pause times. Set appropriate initial and maximum heap sizes based on memory requirements.
- Spring Framework Optimization: Limit excessive auto wiring and use Spring Profiles to load only necessary beans for specific environments, enhancing application performance.
- Security Considerations: Limit data exposure by sending only necessary data over the network and ensure data validation and sanitization to prevent vulnerabilities.

8. Conclusion

In conclusion, Java continues to serve as a powerful tool for developing web applications, offering scalability, performance, and security. This paper has highlighted the benefits of adopting a 3 - Tier architecture and best practices for optimizing web application performance. By implementing these practices, developers can significantly enhance the user experience, operational efficiency, and scalability of their Java based applications. Future research could explore further optimization strategies, such as advanced caching techniques and emerging technologies like machine learning for performance prediction.

References

[1] L. Olsina, G. Lafuente, and G. Rossi, "Specifying Quality Characteristics and Attributes for Websites, " 2001. doi: 10.1007/3 - 540 - 45144 - 7_26.
[2] H. B. Prajapati and V. K. Dabhi, "High quality web - application development on java BE platform, " in *2009 IEEE International Advance Computing Conference, IACC 2009*, 2009. doi: 10.1109/IADCC.2009.4809267.
[3] F. R. Gilbert and D. B. Dahl, "jsr223: A Java platform integration for R with programming languages Groovy, JavaScript, JRuby, Jython, and Kotlin, " *R J.*, 2019, doi: 10.32614/RJ - 2018 - 066.
[4] V. K. Yarlagadda and R. Pydipalli, "Secure Programming with SAS: Mitigating Risks and Protecting Data Integrity, " *Eng. Int.*, vol.6, no.2, pp.211–222, Dec.2018, doi: 10.18034/ei. v6i2.709.
[5] V. V Kumar, M. Tripathi, M. K. Pandey, and M. K. Tiwari, "Physical programming and conjoint analysis - based redundancy allocation in multistate systems: A Taguchi embedded algorithm selection and control (TAS& C) approach, " *Proc. Inst. Mech. Eng. Part O J. Risk Reliab.*, vol.223, no.3, pp.215–232, Sep.2009, doi: 10.1243/1748006XJRR210.
[6] W. Del Ra, "Java application architecture, " *ACM SIGSOFT Softw. Eng. Notes*, 2013, doi: 10.1145/2413038.2413053.
[7] S. K. R. A. Sai Charan Reddy Vennapusa, Takudzwa Fadziso, Dipakkumar Kanubhai Sachani, Vamsi Krishna Yarlagadda, "Cryptocurrency - Based Loyalty Programs for Enhanced Customer Engagement, " *Technol. Manag. Rev.*, vol.3, no.1, pp.46–62, 2018.
[8] S. Elbaum, S. Karre, and G. Rothermel, "Improving web application testing with user session data, " in *Proceedings - International Conference on Software Engineering*, 2003. doi: 10.1109/icse.2003.1201187.
[9] M. S. Sharmila and E. Ramadevi, "Analysis of Performance Testing on Web Applications, " *Int. J. Adv. Res. Comput. Commun. Eng.*, 2014.
[10] O. Kwon and H. Bang, "Design Approaches of Web Application with Efficient Performance in JAVA, " 2011.
[11] A. S. Harji, P. A. Buhr, and T. Brecht, "Comparing high - performance multi - core web - server architectures, " in *ACM International Conference Proceeding Series*, 2012. doi: 10.1145/2367589.2367591.
[12] X. Tan, "A database optimization model for java web architecture, " *Int. J. Simul. Syst. Sci. Technol.*, 2016, doi: 10.5013/IJSSST. a.17.10.04.
[13] J. D. Y. Correa and J. A. B. Ricaurte, "Web application deveplopment technologies using google web toolkit and google app engine - java, " *IEEE Lat. Am. Trans.*, 2014, doi: 10.1109/TLA.2014.6749559.
[14] Rajeev BV and K. Bakula, "A developer's insights into performance optimizations for mobile web apps, " in *2015 IEEE International Advance Computing Conference (IACC)*, IEEE, Jun.2015, pp.671–675. doi: 10.1109/IADCC.2015.7154791.
[15] J. Verdu and A. Pajuelo, "Performance Scalability Analysis of JavaScript Applications with Web Workers, " *IEEE Comput. Archit. Lett.*, 2016, doi: 10.1109/LCA.2015.2494585.
[16] P. Li, "Modular and flexible coordination for web - based applications, " in *2nd International Conference on Computer and Communication Systems, ICCCS 2017*, 2017. doi: 10.1109/CCOMS.2017.8075176.
[17] Y. Ueda and M. Ohara, "Performance competitiveness of a statically compiled language for server - side Web applications, " in *ISPASS 2017 - IEEE International Symposium on Performance Analysis of Systems and Software*, 2017. doi: 10.1109/ISPASS.2017.7975266.
[18] D. Palacios, J. Guamán, and S. Contento, "Analysis of the performance of Java Server faces component libraries in Web application development, " vol.1, pp.54–59, 2018, doi: 10.37135/unach. ns.001.02.06.
[19] K. Munonye and P. Martinek, "Performance Analysis of the Microsoft. Net - and Java - Based Implementation of REST Web Services, " in *SISY 2018 - IEEE 16th International Symposium on Intelligent Systems and Informatics, Proceedings*, 2018. doi: 10.1109/SISY.2018.8524705.
[20] A. Ansari, "Analysis and Performance Issue of Java and Its Framework and Impacts on Web Application, " 2019. doi: 10.13140/RG.2.2.11281.48489.