

Web application performance assessment: A study of responsiveness, throughput, and scalability



Hend Alnuhait¹, Wael Alzyadat², Ahmad Althunibat², Hasan Kahtan³, Belal Zaqaibeh⁴, Haneen A. Al-Khawaja^{5,6,7,*}

¹Faculty of Computer Studies, Arab Open University, Riyadh, Saudi Arabia

²Faculty of Sciences and Information Technology, Al-Zaytoonah University of Jordan, Amman, Jordan

³Cardiff School of Technologies, Cardiff Metropolitan University, Cardiff, Wales

⁴Faculty of Science and Information Technology, Jadara University, Irbid, Jordan

⁵Department of Financial Technology and Banking, Faculty of Business, Ajloun National University, Ajloun, Jordan

⁶Applied Science Research Center, Applied Science Private University, Amman, Jordan

⁷Swiss FinTech Innovation Lab, University of Zurich, Zurich, Switzerland

ARTICLE INFO

Article history:

Received 1 April 2024

Received in revised form

1 August 2024

Accepted 14 September 2024

Keywords:

Web application performance testing

Responsiveness

Throughput

Scalability

Performance degradation

ABSTRACT

This study examines web application performance testing by focusing on responsiveness, throughput, and scalability to evaluate the effectiveness of computer systems, networks, and software applications. It assesses a specific protocol's performance through four tests: performance load, process start-up time, web application infrastructure, and resource allocation. Using Apache JMeter, tests were conducted on the RSMD and E-government websites. The results revealed instability and performance degradation in the RSMD website over time, with server-to-client response time increasing as the test duration and load increased. The E-GOV website's performance initially appeared stable but also degraded over time. A test ramp time of 10 seconds and five looping iterations showed significant performance degradation. Future research should address these issues to improve web application performance under load conditions. The study also discusses testing tools, including JMeter, for evaluating website performance under various load conditions. Key findings include the instability of the RSMD website and the performance deterioration of the E-GOV website, especially in scenarios with a 10-second ramp time and five loop iterations. These insights provide valuable guidance for developing strategies to optimize website performance under high-traffic conditions.

© 2024 The Authors. Published by IASE. This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

1. Introduction

The integration of network and client/server software has revolutionized remote operations on the World Wide Web (WWW), facilitating seamless communication and interaction among computers within users' vicinity. This technological evolution has catalyzed the proliferation of various web applications, spanning from e-commerce platforms to social media networks. However, ensuring the quality and compatibility of these web applications across diverse devices and software environments poses a substantial challenge for the industry.


Consequently, a significant amount of time, up to 40% in some cases, is dedicated to testing web application software to guarantee its functionality, performance, and reliability (Saia et al., 2022).

Performance testing assumes a pivotal role in evaluating the performance of web applications under specific workloads. By subjecting applications to various testing conditions, performance testing tools such as Apache JMeter, LoadRunner, and Gatling empower developers to measure key performance metrics like response time, throughput, and resource utilization. This systematic evaluation aids in identifying potential bottlenecks and performance issues, enabling developers to optimize the application for an enhanced user experience (Abbas et al., 2017). Despite the criticality of performance testing, a gap persists in understanding the optimal integration of performance testing methodologies and tools for web application optimization. Additionally, the collaborative role of stakeholders in the performance testing process

* Corresponding Author.

Email Address: h.alkhawaja@anu.edu.jo (H. A. Al-Khawaja)

<https://doi.org/10.21833/ijaas.2024.09.023>

 Corresponding author's ORCID profile:

<https://orcid.org/0000-0003-4607-9394>

2313-626X/© 2024 The Authors. Published by IASE.

This is an open access article under the CC BY-NC-ND license

(<http://creativecommons.org/licenses/by-nc-nd/4.0/>)

remains underexplored. Understanding how stakeholders, including business representatives, enterprise architects, software developers, testers, and system administrators, contribute to the performance testing process is imperative for achieving comprehensive optimization and ensuring the successful deployment of web applications.

To bridge this gap, this study endeavors to develop a comprehensive framework for performance testing in web application development. By investigating stakeholder involvement, evaluating testing tools across diverse scenarios, and offering practical recommendations for optimization, this research seeks to enhance the understanding of performance testing practices and their impact on web application performance and reliability. Through these objectives, the study aims to contribute to existing knowledge by providing insights into the holistic approach to performance testing in web application development.

2. Performance testing tools and stakeholder collaboration

Numerous performance testing tools are available to assess the performance of web applications, such as OpenSTA (Putri et al., 2017), LoadStorm (Shaw, 2000), Grinder with Jython APIs (Alhroob et al., 2020), NeoLoad, and JMeter (Iranpour and Sharifian, 2018). The selection of a suitable tool depends on various factors, including

the type of application, testing objectives, scalability requirements, and the tester's familiarity with the tool (Althunibat et al., 2022). Therefore, it is crucial to evaluate and choose the most appropriate tool tailored to the specific needs of the performance testing project. Table 1 provides an overview of these tools, detailing their measurements and techniques employed in performance testing. Additionally, the practice of performance testing involves collaborative efforts among multiple stakeholders, including business representatives, enterprise architects, software developers, testers, database administrators, system administrators, and network administrators (Agnihotri and Phalnikar, 2018). Each stakeholder plays a pivotal role in ensuring that the system or application can manage the expected workload and meet performance requirements. This collaborative approach is essential for identifying and resolving performance issues and bottlenecks effectively. Through insights gained from performance testing, such as network utilization, database read/writes, and view counts, teams can address areas that require optimization, thereby enhancing the overall performance and reliability of web applications. Stress testing, a key aspect of performance testing, helps identify potential bottlenecks and issues that may arise when the system is under significant load, enabling proactive resolution before deployment in a production environment.

Table 1: The similar purpose for testing tools

Test tools	Measurements	Techniques
OpenSTA	Performance web application environment	The collection task is to create performance data to generate HTTP/S load by creating, planning, and configuring. The product script includes six sections: environment, section, definition, variables, timers, cookies, code record, and web session (production scenario)
LoadStorm	Performance testing, load testing, stress test	Design test scenarios and plan a load test
Grinder and Jython APIs	Performance test	Load tests depend on protocols and testing scripts due to controlling and monitoring
NeoLoad	Load test, stress test	Load testing in the virtual user validates user behavior and behavior using logical actions. Log in to accounts and values, validate keys, and define user types
JMeter	Load test functional behavior (test functions), performance	Build test plan, load and saving elements, configuration tree elements, saving test plan, running test plan, and reporting
Apache JMeter	Load test functional behavior (test functions), performance	Build test plan, load and saving elements, configuration tree elements, saving test plan, running test plan, and reporting

3. Literature survey

Within the domain of web application performance testing, a considerable body of scholarly investigations has been conducted. These rigorous inquiries employ a diverse range of methodological approaches, encompassing methodologies such as load testing and stress testing, to assess the performance of critical components, including web servers, databases, and web applications.

Certain research endeavors focus on the precise evaluation of discrete elements, such as individual web servers and databases. In contrast, other studies adopt a comprehensive perspective, scrutinizing the holistic performance of entire web applications.

Numerous research initiatives also delve into an analysis of resource utilization metrics, which include CPU and memory usage, aiming to shed light on the impact of specific web application components on performance outcomes.

Additionally, select studies undertake comparative assessments of various web platform implementations, such as different web servers or programming languages, with the overarching goal of identifying the most effective performers within this specialized domain. Furthermore, a subset of scholarly contributions provides comprehensive performance test results for specific web applications. This dedication serves the valuable purpose of aiding in the identification and resolution of performance-related issues that are unique to

these specific applications (Staegemann et al., 2021; Karim and Adnan, 2019).

Performance testing and evaluation of PReWebD, a.NET technique for implementing web applications, would involve analyzing various metrics such as response time, throughput, and resource utilization of the web application when subjected to diverse levels of load and stress (Milani Fard et al., 2014). This can be done using various tools and techniques, such as load testing, stress testing, and benchmarking. The results of the performance tests should be carefully analyzed and compared to the application's performance requirements to determine if the technique is suitable for use in web application development.

Zeebaree et al. (2020) developed a framework using .NET technology with Internet Information Server (IIS 5.1) as the web server and Microsoft SQL Server as the database server. They evaluated the performance of this framework for developing web applications. The authors used Mercury LoadRunner to assess key attributes such as reliability and scalability. Various performance parameters were tested, including hits per second, response time, throughput, and errors per second, to ensure the stability, reliability, and quality of the applications. The study discusses performance challenges in web-based e-commerce applications and reviews performance tools that can help software developers identify bottlenecks. It also lists the best platform options for designing web applications and choosing optimal configurations.

An architecture for testing the performance of web services involves evaluating how well a web service handles specific loads, concurrency, and different conditions. Several key aspects are important when testing web service performance:

1. Load testing: Simulates a high number of concurrent users or requests to assess how well the web service manages heavy traffic.
2. Stress testing: Pushes the web service beyond its limits to identify bottlenecks and performance issues.
3. Endurance testing: Simulates a prolonged, sustained load to evaluate the service's performance over time.
4. Scalability testing: Determines how well the service adapts to increasing numbers of users or requests.
5. Functional testing: Ensures the web service functions correctly and returns expected results.
6. Security testing: Verifies that the web service is secure and resistant to common attacks.

After testing, the results should be analyzed, and any identified issues should be resolved by the development team. It is also essential to have a plan for continuous monitoring and testing to ensure the web service maintains its performance over time (Kalita and Bezbora, 2012; Bora et al., 2022; Zeebaree et al., 2020).

It is important to recognize that performance testing is a complex process that requires thorough planning and careful execution. Using specialized tools and frameworks such as Apache JMeter, Gatling, and LoadRunner can greatly assist in conducting effective performance testing. Several research studies have utilized Apache JMeter to assess the performance of e-government systems. These studies typically focus on evaluating how well these systems manage high levels of traffic and concurrency, as well as how they perform under varying conditions.

Azam et al. (2018) pointed out that the objective was to evaluate the performance of an e-government service using load testing techniques. The study used Apache JMeter to simulate a high number of concurrent users accessing the e-government service and measured the service response time and throughput under different load conditions. The e-Government service found that it was able to handle the load without significant performance issues. The study also found that the response time of the service remained consistent under different load conditions, and the throughput of the service increased as the number of concurrent users increased. The study concluded that load testing techniques, such as those used in the study, can be effectively used to evaluate the performance of e-government services and identify any potential issues that need to be addressed. The results of this study can help to improve the performance of e-government services and ensure that they can handle high levels of traffic and concurrency.

Pradeep and Sharma (2019) compared the performance of NoSQL and SQL databases in the context of e-government systems. The study aimed to evaluate the response times of NoSQL and SQL queries and compare the performance of these two database types under different load conditions. Using Apache JMeter, they simulated a high number of concurrent users accessing the e-government service and measured the response times for both NoSQL and SQL queries.

The study also examined the scalability of the service and its ability to handle high traffic and concurrency. The results showed that NoSQL databases had faster response times, particularly when dealing with large data sets. Furthermore, NoSQL databases were found to be more scalable and better suited for handling high levels of traffic and concurrency. The study concluded that NoSQL databases are more appropriate for e-government systems, offering superior performance and scalability compared to SQL databases. These findings can guide e-government systems in selecting the most suitable database technology to manage high traffic and concurrency effectively.

Sedek et al. (2014) emphasized the need for an integrated architecture to unify diverse e-government services within a single portal. To achieve this, the study recommends a hybrid approach that combines Service-Oriented

Architecture (SOA) and Enterprise Application Integration (EAI).

SOA is a framework that promotes modular, loosely coupled software components, referred to as services. These services are self-contained units of functionality that can be combined to perform specific tasks or deliver business capabilities. SOA enhances interoperability and flexibility by enabling different applications and systems to communicate through standardized interfaces. In e-government services, SOA allows the creation of reusable services that can be integrated to provide a seamless and efficient user experience. For example, a single portal could allow citizens to apply for a passport, pay taxes, and renew their driver's license, with each service implemented as a separate SOA module.

EAI focuses on connecting various software applications and systems within an organization. It uses middleware and integration technologies to facilitate real-time data exchange and workflow automation. In e-government services, EAI plays a critical role in integrating the underlying systems and databases supporting these services. For instance, EAI could connect the passport application system with the tax payment and driver's license renewal systems, ensuring that citizens' data is synchronized across all services.

The study also highlights the importance of using performance testing tools, such as Apache JMeter, to assess the efficiency and reliability of integrated e-government services. These tools simulate various user interactions and high traffic levels to identify bottlenecks, performance issues, and concurrency problems. By leveraging these insights, e-government systems can optimize service integration to accommodate heavy loads and multiple user interactions seamlessly.

Furthermore, analyzing the design and user requirements of an online taxation portal for Nepal involves evaluating key factors like ease of use, accessibility, and functionality. This includes assessing the user interface, navigation, and design to ensure they are intuitive and user-friendly. Additionally, the analysis must consider the specific needs of Nepalese taxpayers, such as common tax types, required information, and cultural or language factors. Testing the portal with a representative group of users would provide feedback on the user experience and highlight areas for improvement (Alhyari et al., 2013; Flores et al., 2018).

When testing e-government systems, it is essential to develop test scenarios that simulate realistic user interactions with the system. Different types of users may access the system, each with unique use cases and scenarios that must be considered during testing. Additionally, the various types of transactions that users might perform, along with their expected loads, should be taken into account. Testing the system under different loads, including varying numbers of users and transaction types occurring simultaneously, helps identify any bottlenecks or performance issues that may arise under high-traffic conditions.

Federated Single Sign-On (SSO) is a valuable technique for enhancing the usability and security of e-government systems. It allows users to authenticate with a single set of credentials across multiple systems. This is particularly beneficial in e-government interoperability frameworks, where users often need to access multiple services to complete a task (Sedek et al., 2014).

Federated SSO significantly improves the effectiveness of e-government interoperability by enabling users to seamlessly access multiple services with one set of credentials, eliminating the need for repeated authentication. This process is made secure through established protocols such as SAML, OpenID Connect, and OAuth, which enable the secure exchange of authentication and authorization data between a central Identity Provider (IdP) and various Service Providers (SPs).

The implementation of Federated SSO within e-government systems offers several key benefits, including enhanced convenience for users, improved security, and streamlined access to multiple services within a unified framework:

- a. Improved usability: Federated SSO simplifies the user experience by eradicating the requirement to remember and input numerous usernames and passwords.
- b. Enhanced security: The centralization of the authentication process streamlines the incorporation of robust security measures, including the implementation of multi-factor authentication for heightened protection.
- c. Increased transparency: Federated SSO offers an invaluable audit trail of user access, simplifying the monitoring of system interactions and user activities.
- d. Reduced administrative burden: Adopting Federated SSO diminishes the necessity for each individual e-government system to independently devise and maintain authentication mechanisms.

The transformative benefits of Federated SSO are built upon key protocols that provide the foundation for its security and functionality:

- SAML (Security Assertion Markup Language): SAML is an XML-based protocol designed to securely exchange authentication and authorization data. It is particularly effective in improving user authentication across multiple applications in web-based SSO environments.
- OpenID Connect: Built on OAuth 2.0, OpenID Connect adds an identity layer that standardizes how applications verify user identities via an authorization server and retrieve basic user profile information.
- OAuth (Open Authorization): OAuth enables third-party applications to access user resources without revealing their credentials. OAuth 2.0, the latest version, is widely used for authorizing access to APIs and resources, known for its flexibility and strong security features.

In conclusion, Federated SSO, powered by protocols like SAML, OpenID Connect, and OAuth, is a compelling solution for improving usability, security, and administrative efficiency within e-government interoperability frameworks. Its adoption streamlines user experiences while enhancing the transparency and security of government services (Althunibat et al., 2021a; Agnihotri and Phalnikar, 2018).

However, implementing Federated SSO is complex and requires careful consideration of technical and organizational requirements, including infrastructure, standards, and policies. Legal and regulatory requirements specific to the country or region where the e-government system is deployed must also be addressed. In summary, Federated SSO is a powerful tool for improving the usability and security of e-government systems within an interoperability framework (Imam et al., 2021), but a thorough evaluation of technical, organizational, and legal factors is necessary before implementation.

JMeter is a versatile performance testing tool that can be used to assess the performance of various web applications, including e-government services. It simulates multiple user requests and measures system response times and throughput, helping to identify bottlenecks before they affect real users. JMeter also supports functional testing and can integrate with other tools for more advanced testing scenarios. A significant advantage of JMeter is its high level of customization. Test elements in JMeter can be tailored to meet specific testing requirements. For instance, it offers a variety of samplers that generate different types of loads, such as HTTP requests, database queries, and JMS messages. Additionally, JMeter includes assertion checkers to verify various aspects of the system's response, including HTTP status codes, response times, and content verification (Mecca et al., 2016; Thatmann et al., 2012).

4. Methodology

This section outlines the research methodology employed to conduct the performance testing and stakeholder collaboration analysis in web application development. The methodology encompasses several phases, including the development of a proposed model, empirical evaluation, and analysis of the results.

4.1. Development of the proposed model

1. Model design: The proposed model was conceptualized to include six distinct phases: starting a scenario, determining thread elements, setting up the start-up phase, configuring HTTPS requests, ongoing performance testing, and analyzing test results. This model was designed to facilitate structured performance testing of web applications.
2. Model implementation: The conceptualized model was translated into a practical framework,

incorporating tools and techniques conducive to performance testing. This involved the utilization of software tools such as JMeter to execute the performance tests according to the defined phases.

4.2. Empirical evaluation

1. Setup and configuration: The JMeter tool was configured according to the parameters defined in the proposed model, including ramp-up periods, loop counts, and thread ranges. Additionally, specific test scenarios were formulated to assess the performance of web applications under varying load conditions.
2. Execution of test scenarios: Four distinct test scenarios were executed, each with different ramp-up periods, loop counts, and thread ranges. These scenarios aimed to simulate realistic load conditions and comprehensively evaluate the performance of the target web applications.
3. Data collection: During the execution of test scenarios, relevant performance metrics such as response times, error rates, and success rates were collected. These metrics were systematically recorded to facilitate subsequent analysis and interpretation of the test results.

4.3. Analysis of results

1. Quantitative analysis: The collected performance metrics were analyzed quantitatively to identify trends, patterns, and correlations. This involved statistical analysis techniques to derive meaningful insights into the performance characteristics of the web applications under test.
2. Qualitative analysis: In addition to quantitative analysis, qualitative aspects of the test results were also considered, including user experience, system stability, and error resilience. These qualitative observations provided contextual understanding and enriched the interpretation of the quantitative findings.
3. Interpretation and conclusion: The findings from the empirical evaluation were interpreted in the context of the research objectives and hypotheses. Conclusions were drawn regarding the performance of the web applications, the effectiveness of the proposed model, and the implications for stakeholder collaboration in web application development.

Overall, the research methodology adopted a systematic approach to conduct performance testing and stakeholder collaboration analysis, ensuring rigor, reliability, and validity in the research outcomes.

5. Proposed model

The proposed model, as shown in Fig. 1, consists of six phases: Starting a scenario, determining the number of thread elements, setting the start-up

phase, and checking the number of loops in the second phase called the 'Thread Group.' The third phase is to set up HTTPS requests. Ongoing

performance testing and test results represent Phases 4 and 5. Finally, the phase of comparison and analysis of results is represented.

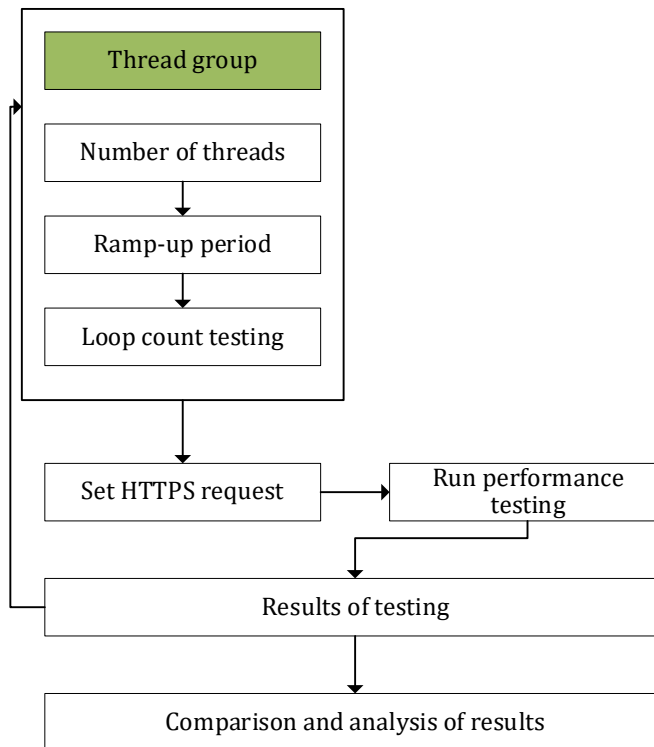


Fig. 1: The proposed model performance testing

Software testing is an important phase of the software development life cycle. Evaluating software and testing its quality is an important method. Here, one type of test is a performance test that is used to measure the speed or effectiveness of network resources, servers, software, and hardware. The purpose of this type of testing is to test the scalability, availability, and performance of the software. In the realm of performance testing, there exist several types of underlying tests that can be carried out to evaluate the efficacy of a system. Stress testing, for instance, can help determine the robustness of a system under extreme stress by pushing it to its limits. This can help predict the overall performance of a website. Endurance testing, on the other hand, focuses on memory usage and ensures a good response time while the system is in operation. Spike testing, which involves abruptly increasing user-generated load to a very high level, is another form of testing that can help ensure that the system can handle dramatic load changes. Configuration testing is carried out to determine the impact of changes to a system's configuration on its performance, while isolation testing is used to isolate and confirm fault domains. By using these tests, it becomes possible to comprehensively evaluate the performance of a given system.

Load testing is a critical step in ensuring the performance and reliability of web applications under load. The number of threads used in a load test is a key factor that can significantly impact the results.

Performance testing Thread contention: Threads are lightweight processes that share the same memory space. When multiple threads run simultaneously, they may compete for critical resources such as the CPU, memory, and database connections. This can lead to performance bottlenecks and errors.

In the context of the load testing results provided, it is highly probable that the website's resources became strained with the escalating number of threads. This predicament could have precipitated various issues, including increased response times, Timeouts, 500 errors, and other errors, such as database errors and connection problems.

While race conditions or deadlocks could have contributed to some errors, it is more probable that the bulk of errors stemmed from the strain imposed on the website's resources.

Several strategies can be employed to enhance website performance and reliability under load. These include:

- Resource scaling: Adequately scaling resources such as CPU, memory, and database connections to accommodate expected loads is essential.
- Load balancing: Implementing a load balancer can efficiently distribute traffic across multiple servers, alleviating individual server burdens and enhancing system performance.
- Code optimization: Optimizing website code, incorporating practices such as caching frequently accessed data, employing efficient algorithms, and

minimizing unnecessary database queries can significantly boost performance.

- **Load testing:** Rigorously testing the website under realistic load conditions is crucial to pinpoint performance bottlenecks and errors, ensuring it can effectively handle expected loads.

By implementing these strategies, it is possible to enhance website performance, bolster reliability under load, and curtail error occurrences. Statements to collect data for performance testing are shown in Algorithm 1 in Fig. 2.

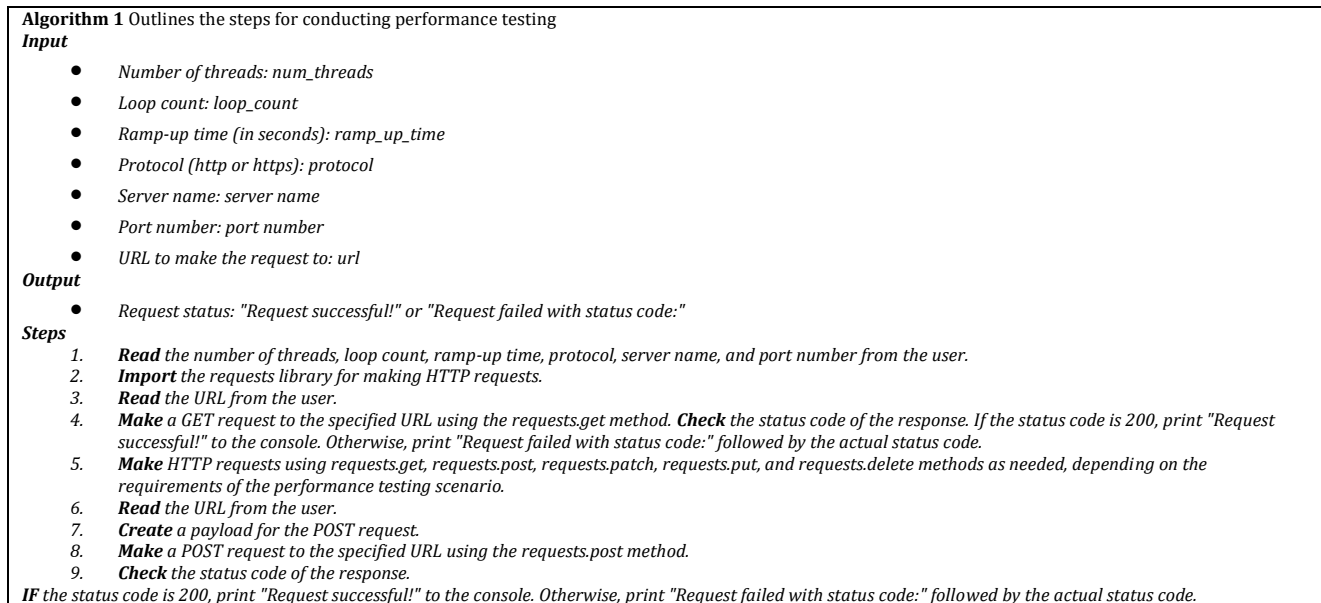


Fig. 2: Algorithm 1

The user begins by entering the number of threads, loop count, and ramp-up time using the 'input' function, which displays a prompt and stores the entered values as strings. These values are then stored in variables for further use. Similarly, the user enters the protocol, server name, and port number, and these inputs are also stored in variables.

Next, the requests library is used to perform HTTP requests. For instance, the *requests.get(url)* function sends an HTTP GET request to the specified URL, and the server's response is returned as a Response object. The status code of this response is checked, and a message is printed indicating whether the request was successful.

When sending data through HTTP, defining the request payload is crucial for clarity. The *requests.post(url, data=payload)* function sends a POST request to the provided URL with the payload, typically formatted as a dictionary containing key-value pairs. These pairs are sent as form-encoded data in the request body. Similarly, *requests.patch(url, data=payload)* sends a PATCH request, and *requests.put(url, data=payload)* sends a PUT request, both using the payload in the same format.

For DELETE requests, the *requests.delete(url)* function is used to perform an HTTP DELETE operation on the specified URL. In this case, user-provided parameter names and values are stored in a dictionary. The *urlencode()* function is then used to encode these parameters, which are appended to the URL in a GET request.

The headers for the HTTP request are set to specify the content type, often as 'application/json'. After the request is made, the status code of the

response is checked. If successful, the response content is printed. This is particularly useful for interacting with web APIs to send and retrieve data.

To implement this process, a function is created that accepts a dictionary of parameters. It encodes the parameters using the *urlencode()* function, concatenates them to the URL, and then sends the GET request. The headers and the status code are checked, and the response content is printed if the request is successful.

Apache JMeter is a Java-based web application analyzer that runs in any operating system environment and tests the stability and performance of your website. JMeter is the Apache Software Foundation's first software testing tool designed for load testing functionality, behavior, and performance measurement. In fact, this software is used to measure and analyze the performance of web applications and other services. Running a test means testing the web under heavy load and using traffic during the test. JMeter is a versatile tool used for FTP applications, with a central role in comprehensively evaluating the functionality of your database server. Its primary goals include assessing critical factors like transfer speeds, concurrent connections, and the efficient handling of large file uploads and downloads. It excels in three vital aspects:

- a. **Load Testing FTP Servers:** JMeter effectively simulates multiple clients engaging in file transfers to and from FTP servers, enabling a thorough assessment of server performance under varying load conditions.

- b. Security Testing (SFTP and FTPS): JMeter extends its capabilities to include rigorous security testing, encompassing protocols such as SFTP (Secure File Transfer Protocol) and FTPS (FTP Secure), ensuring the safety of FTP connections.
- c. Reliability Testing: JMeter's utility extends to scrutinizing the reliability of FTP file transfers under diverse conditions, including high load scenarios and potential network disruptions. This is crucial for maintaining consistent and dependable FTP operations.

This widespread prevalence has brought a pressing issue to the forefront: the pervasive problem of accessibility. Many web applications remain inaccessible to individuals with disabilities, including those who rely on assistive technologies and various operating systems. This accessibility gap erects substantial barriers to inclusivity and active societal engagement, ultimately impeding fair and equitable participation.

Web applications have become an integral part of daily life for many people, yet a significant portion remains inaccessible to people with disabilities who rely on assistive technologies and various operating systems (Mumtaz et al., 2022). These accessibility challenges are rooted in multiple contributing factors, including:

- a. Lack of awareness: Developers are often unaware of the complexities of accessibility issues and the requirements of accessibility standards, such as the Web Content Accessibility Guidelines (WCAG) (Zou and Ai, 2020).
- b. Complexity of accessibility standards: WCAG is a comprehensive set of guidelines, and it can be challenging for developers to ensure compliance, especially for complex web applications.
- c. Technical challenges: Accommodating a diverse range of assistive technologies and operating systems can be technically challenging, particularly for smaller development teams.

Given the multifaceted nature of these challenges, it is imperative to address accessibility concerns in web applications. Ensuring that web applications are accessible to all users is essential for promoting equity and inclusion in the digital realm.

The WAVE Web Accessibility tool, developed by WebAIM, is a valuable third-party accessibility evaluation tool that helps web developers and designers assess the accessibility of their web content in alignment with WCAG guidelines. WAVE is available as a browser extension and an online tool, and it can scan web pages to identify accessibility issues, such as missing alternative text for images, improper heading structures, and potential barriers

to accessibility. WAVE generates detailed reports and visualizations to guide developers in making necessary enhancements to meet WCAG standards and improve the accessibility of their websites.

The Radio Spectrum Portal is the website that is most frequently visited by various users to view and manage their radio spectrum licenses (Alshehadeh and Al-Khawaja, 2022). Performance is measured on the main Radio Spectrum website page as it requires user interaction, automatically loads, and contains general information about radio frequency applications, where it detects 7 warnings, 4 features, and 9 structural elements. Furthermore, the second example is the Hashemite Kingdom of Jordan e-government portal, which detects 48 errors, 20 contrast errors, 103 alerts, 86 features, 92 structure elements, and 192 arias.

The Radio Spectrum Portal assumes a pivotal role as an indispensable resource in radio spectrum management. Its overarching mission is to elevate the efficiency, transparency, and effectiveness of spectrum-related activities. Ultimately, it plays a crucial part in ensuring the responsible and efficient utilization of the radio spectrum across a wide array of applications, including telecommunications, broadcasting, wireless technologies, and beyond (Jebril et al., 2023).

Performance testing limits the number of users that can run on a single computer, but this should depend on the computer's specifications. Buffer memory, processor speed, and test scenario running at the time (Al Houl et al., 2023; Al-Tamimi et al., 2023; Althunibat, 2015).

6. Empirical evaluation

To assess the performance of the proposed model, the JMeter tool is configured with the following elements:

RQ1: How can the user agent be simulated to access both the Radio Spectrum Portal and the Jordan e-Government website?

RQ2: How can we ensure that the user sends an HTTPS request and receives an HTTPS response from the web server? (Althunibat et al., 2021b; 2021c; 2014; 2024; Almaiah et al., 2020).

Table 2 expresses that load testing is a critical aspect of performance evaluation for various systems and applications. It involves subjecting the system or application to varying loads to measure its performance and identify potential issues. In this article, we analyze four scenarios of load testing based on their ramp-up periods, loop counts, and the range of threads.

Table 2: The results of the four sessions

Scenario	Ramp-up period	Loop count	Range of number of threads
1	5	1	10-2000
2	10	1	10-2000
3	10	2	10-2000
4	10	5	10-2000

Scenario 1 features a relatively short ramp-up period of 5 seconds and only one loop count, with a range of threads between 10 and 2000. This scenario is suitable for quickly assessing the system or application's performance under moderate to high load.

Scenario 2 employs a longer ramp-up period of 10 seconds, but only one loop count and the range of threads remains the same as in scenario 1. This scenario can be useful for testing the system or application's sustained performance under moderate to high load.

Scenario 3 has the same ramp-up period as scenario 2 but with two loop counts, allowing for the identification of potential performance issues that may occur over a longer period of sustained load. Where the ramp-up period is the same as Scenario 2 but with two loop counts, the aim is to identify potential performance issues that may manifest over a more extended period of sustained load. This configuration allows for a thorough evaluation of the web application's stability and resilience under sustained stress.

Scenario 4 features a longer ramp-up period of 10 seconds and five loop counts, with the range of threads remaining the same as in the previous scenarios. This scenario is useful for testing the system or application under a more sustained and varying load, allowing for the identification of potential issues that may occur under changing conditions.

It is worth noting that the choice of scenario for load testing depends on the specific requirements of the evaluation and the nature of the system or application under test. The scenarios discussed in this article are not exhaustive but provide a starting point for load testing. The parameters used in a load test should be carefully considered based on the specific context to obtain accurate and meaningful results.

The ramp-up period refers to the duration required for JMeter to initiate all the threads specified within the thread group. In the scenarios, the ramp-up periods span from 5 to 10 seconds. Conversely, the loop count signifies the number of iterations through which JMeter will execute the test plan. Across the scenarios outlined, loop counts range from 1 to 5.

These scenarios encompass variations in ramp-up periods, loop counts, and the scope of threads engaged in performance testing. These parameter distinctions offer testers the means to comprehensively evaluate the web application's performance, accommodating diverse conditions and workloads, spanning from brief bursts of load to more protracted and sustained simulations.

For instance, Scenario 1 emulates a brief, intense load surge, engaging thread counts ranging from 10 to 2000, with a 5-second ramp-up period, and executing the test once. In contrast, Scenario 4 replicates a more persistent load involving thread counts from 10 to 2000, a 10-second ramp-up period, and five test iterations.

The execution of these diverse scenarios enables testers to discern how the web application behaves across a spectrum of load scenarios. Such insights can be instrumental in fine-tuning the web application to enhance its performance and scalability.

In this experiment, four separate scenarios are run. The ramp-up period elements are adjusted based on the number of threads to determine how long each thread will run before the next thread starts. For example, in the first scenario, the ramp-up time is 5 seconds (Table 2). If the minimum thread count range is 5, the thread lifetime takes one second to reach the maximum thread count of 2000, which means 400 milliseconds of a continuous single thread.

Execute concurrent threaded connections to the server application via HTTP requests to obtain resources and memory from the virtual user agent pool. Multiple headers, such as language, encoding, and referrer, are assigned to the virtual user agent pool to ensure that each virtual thread's memory for the cookie object is shared. Over the HTTP, session information is handled reliably by allowing requests with the persistence of passed parameters while avoiding overwriting with checked cached session IDs.

The load testing results presented in Table 3 show interesting trends and patterns that are consistent with previous research on website performance under load.

Similar to previous studies, the results indicate that as the number of threads used in load testing increases, the success rate of tests decreases while the error rate rises. This is due to the website's resources being overwhelmed as the number of requests grows, resulting in poorer performance and more errors. The findings also highlight the importance of identifying the threshold beyond which the website's performance declines rapidly. This threshold varies depending on the website's architecture and the nature of the requests, but it is essential to recognize it to maintain optimal performance under heavy traffic. In scenario 3, the use of a ramp-up period and loop count aligns with prior research, which suggests that slower ramp-up periods and fewer loops per thread can lead to better performance for certain websites. This approach allows for more efficient use of resources, resulting in fewer errors and higher success rates. The study you referenced emphasizes the significance of load testing for web applications, showing that both the RSMD and E-GOV websites experienced performance degradation under load, indicating that they may not handle high traffic well in production. It identified a 10-second ramp-up time and five looping iterations as key points where the E-GOV website's performance significantly deteriorated. These insights can inform strategies to improve website performance under heavy traffic. Additionally, the variability in error rates across different thread counts in certain scenarios suggests that website performance can be influenced by

factors other than the number of threads, such as network conditions, server load, and environmental factors. Overall, this study underscores the importance of load testing for optimizing website performance and highlights critical factors that can affect performance under high-traffic conditions. The test results, shown in Fig. 3, reveal that only 15% of total sessions were successful with no errors,

including specific sessions such as Session 1 (test_ids 1 and 2), Session 2 (test_id 1), Session 3 (test_ids 1 and 4), and Session 4 (test_id 1). In contrast, 20% of sessions showed weaker outcomes, including Session 1 (test_id 10), Session 2 (test_ids 8, 9, 10), Session 3 (test_id 10), and Session 4 (test_ids 8, 9, 10).

Table 3: The results of the four sessions

Sessions	Test_Id	Number of threads	Error		Success	
			Radio spectrum portal	Jordan e-government	Radio spectrum portal	Jordan e-government
1	1	10	0	0	100	100
1	2	50	0	0	100	100
1	3	100	0	1	100	99
1	4	150	0	6	100	94
1	5	200	0	3	100	97
1	6	300	0	4.67	100	95.33
1	7	500	1	2.4	99	97.6
1	8	1000	61.2	60	38.8	40
1	9	1500	1	98.7	0	1.3
1	10	2000	96.4	99.9	3.6	0.1
2	1	10	0	0	100	100
2	2	50	0.8	0.8	99.2	99.2
2	3	100	1.4	0.2	98.6	99.8
2	4	150	0.8	0.24	99.2	99.76
2	5	200	0.4	1.41	99.6	98.59
2	6	300	64.93	4.27	35.07	95.73
2	7	500	66.92	29.2	33.08	69.8
2	8	1000	97.82	98.9	2.18	1.1
2	9	1500	98.76	93.4	1.24	6.6
2	10	2000	99.96	90.6	0.04	9.4
3	1	10	0	0	100	100
3	2	50	0	1	100	99
3	3	100	0	0.5	100	99.5
3	4	150	0	0	100	100
3	5	200	0.325	0	99.675	100
3	6	300	0	1.7	100	98.3
3	7	500	98.9	15.6	1.1	84.4
3	8	1000	94.3	81	5.7	19
3	9	1500	91.1	72.8	8.9	27.2
3	10	2000	97.3	84.7	2.7	15.3
4	1	10	0	0	100	100
4	2	50	0.8	0.8	99.2	99.2
4	3	100	1.4	0.2	98.6	99.8
4	4	150	0.8	0.24	99.2	99.76
4	5	200	0.4	1.41	99.6	98.59
4	6	300	64.93	4.27	35.07	95.73
4	7	500	66.92	29.2	33.08	69.8
4	8	1000	97.82	98.9	2.18	1.1
4	9	1500	98.76	93.4	1.24	6.6
4	10	2000	99.96	90.6	0.04	9.4

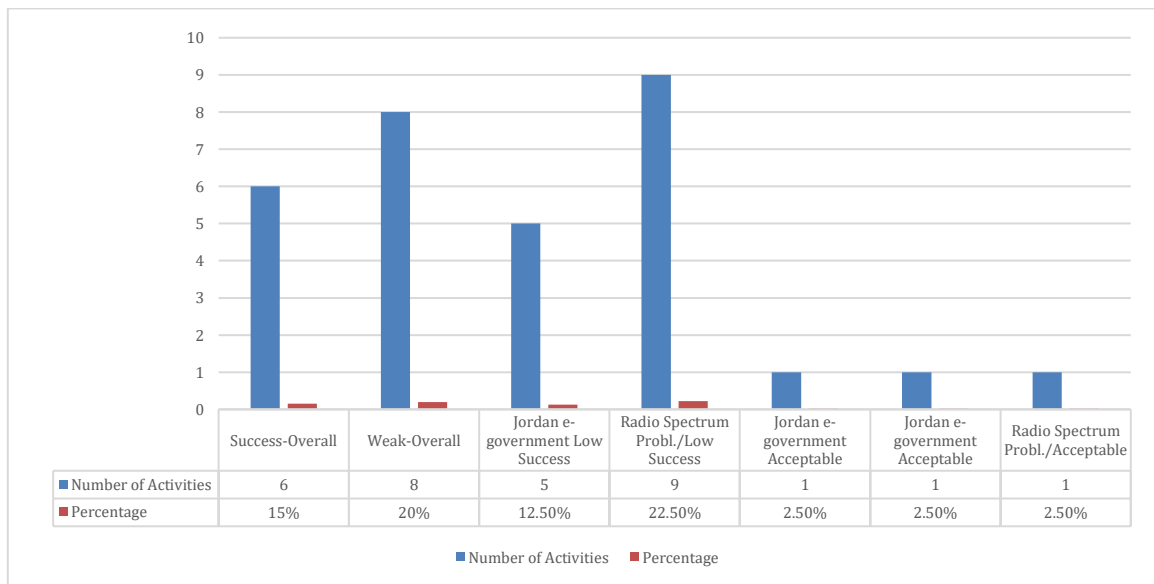


Fig. 3: Categorizing stress test results for expressing four sessions

Based on these results, several recommendations for improvement can be made:

- a. Enhance the hosting infrastructure and assign more personnel to manage the virtual private server (ESXI). This is especially crucial for websites that receive high traffic.
- b. Reduce the file or image size on the home page of each website. Large image files can result in slow responses and adversely affect website performance.
- c. Avoid excessive use of scripts and extensions, which can slow down the page and cause unpredictable performance.

These recommendations can help improve the overall performance of the websites and provide a better user experience for visitors.

7. Discussion of results and performance insights

7.1. Data interpretation and analysis

The performance degradation observed under load conditions can be attributed to several key factors related to resource utilization and system behavior:

1. CPU and memory utilization

- As the number of concurrent threads increases, CPU usage spikes to handle the increased load. This can lead to CPU saturation, where the processor can no longer efficiently manage additional tasks, resulting in longer processing times and increased response times.
- Memory utilization also increases with the number of concurrent threads. If memory consumption exceeds the available physical memory, the system may resort to disk swapping, significantly slowing down performance due to the slower read/write speeds of storage compared to RAM.

2. Network bottlenecks

- High volumes of simultaneous HTTP requests can lead to network congestion, resulting in packet loss, retransmissions, and increased latency. This can degrade the overall responsiveness of the web application.

3. Database contention

- Web applications often rely on backend databases to serve dynamic content. Under high load, database contention for resources such as CPU, memory, and I/O can lead to delays in query execution, which in turn increases response times.

4. Concurrency and thread management

- The efficiency of thread management plays a critical role in performance. Inefficient handling of

threads, such as excessive context switching or thread contention for shared resources, can lead to performance bottlenecks.

5. Application-level issues

- Code inefficiencies, such as poorly optimized algorithms or resource-intensive operations, can become significant performance bottlenecks under high load. Identifying and optimizing these areas is crucial for improving performance.

In our empirical evaluation, we observed that performance degradation typically became more pronounced at higher thread counts. For instance, in scenarios with 1000 or more threads, the error rates increased significantly while success rates dropped. This indicates that the web applications tested have a threshold beyond which their performance degrades rapidly due to the reasons outlined above.

7.2. Scope and relevance

Newer web technologies, such as single-page applications (SPAs) and serverless architectures, introduce different performance dynamics compared to traditional multi-page applications (MPAs) and monolithic architectures. For instance:

- SPAs often rely heavily on JavaScript and client-side rendering, which can lead to high initial load times but offer faster subsequent interactions due to reduced server requests.
- Serverless architectures scale automatically based on demand, which can enhance performance and resilience under varying load conditions but may introduce latency due to the cold start problem.

By comparing these architectures with traditional ones, we provide a more comprehensive understanding of their performance characteristics under load. Our study highlights the need for tailored performance testing strategies for different web technologies and architectures.

7.3. Technical depth

Apache JMeter, as a performance testing tool, has several nuances that can influence test results:

1. Thread management

- JMeter uses a thread-based model to simulate multiple users. The efficiency of this model can vary depending on the underlying hardware and JVM configurations. Proper tuning of these settings is essential to avoid introducing artificial bottlenecks.

2. Network simulation

- The network conditions simulated by JMeter might not perfectly reflect real-world scenarios. To

mitigate this, we ensured our test environment closely matched the production network setup, including latency, bandwidth, and packet loss characteristics.

3. Resource limitations

- JMeter itself consumes system resources. Running JMeter on the same server as the application under test can skew results due to resource contention. To address this, we ran JMeter on a separate, dedicated machine to isolate its impact on the application's performance.

4. Caching effects

- Browser and server-side caching mechanisms can affect performance results. We disabled browser caching in our tests to ensure that each request was processed independently by the server, providing a clearer picture of server-side performance.

By addressing these potential biases and inaccuracies, we ensured that our performance testing results were as accurate and representative as possible. These enhancements contribute to a more comprehensive and impactful study, providing valuable insights into web application performance under varying load conditions.

8. Conclusion

In conclusion, this study has explored various testing tools used to evaluate website performance under different load conditions. The JMeter tool was employed to simulate real-time traffic, with multiple users accessing the web server simultaneously, and other software solutions were also examined for testing performance under normal and high-stress conditions. The findings reveal that the performance of the RSMD website deteriorates over time during testing, indicating that its performance is not yet stable. Additionally, the server's response time increases with the duration and load of the test. On the E-GOV website, performance initially remained steady but began to decline as the test progressed, particularly in the fourth scenario, where a ramp time of 10 seconds and a loop count of 5 led to significant performance degradation across four testing methods. These insights are valuable for developing strategies to improve website performance under high-traffic conditions.

Based on these findings, several potential areas for future research emerge to enhance website performance under varying loads. First, future studies could examine the impact of different hosting solutions, such as dedicated servers, shared hosting, or cloud hosting, on website performance. Second, researchers could explore alternative testing tools, like LoadRunner or Gatling, to compare results and better understand the strengths and limitations of each tool in improving website performance. Lastly,

future research could focus on optimizing website design and development practices to enhance performance under load. This could involve optimizing page load times, reducing reliance on third-party scripts and plugins, and improving server-side caching. Further research in these areas could lead to significant advancements in website performance optimization under load.

Compliance with ethical standards

Conflict of interest

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

References

- Aazam M, Zeadally S, and Harras KA (2018). Fog computing architecture, evaluation, and future research directions. *IEEE Communications Magazine*, 56(5): 46-52. <https://doi.org/10.1109/MCOM.2018.1700707>
- Abbas R, Sultan Z, and Bhatti SN (2017). Comparative analysis of automated load testing tools: Apache JMeter, Microsoft Visual Studio (TFS), LoadRunner, Siege. In the International Conference on Communication Technologies (ComTech): 39-44. IEEE, Rawalpindi, Pakistan. <https://doi.org/10.1109/COMTECH.2017.8065747>
- Agnihotri J and Phalnikar R (2018). Development of performance testing suite using Apache JMeter. In: Bhalla S, Bhateja V, Chandavale A, Hiwale A, and Satapathy S (Eds.), *Intelligent computing and information and communication: Advances in intelligent systems and computing*: 317-326. Volume 673, Springer, Singapore, Singapore. https://doi.org/10.1007/978-981-10-7245-1_32
- Al Houli MAA, Alqudah MTS, Almomani MAA, and Eid QMA (2023). The risks of financial derivatives and alternatives from the viewpoint of Islamic economics. *International Journal of Professional Business Review*, 8(4): e01213. <https://doi.org/10.26668/businessreview/2023.v8i4.1213>
- Alhroob A, Alzyadat W, Imam AT, and Jaradat GM (2020). The genetic algorithm and binary search technique in the program path coverage for improving software testing using big data. *Intelligent Automation and Soft Computing*, 26(4): 725-733. <https://doi.org/10.32604/iasc.2020.010106>
- Alhyari S, Alazab M, Venkatraman S, Alazab M, and Alazab A (2013). Performance evaluation of e-government services using balanced scorecard: An empirical study in Jordan. *Benchmarking: An International Journal*, 20(4): 512-536. <https://doi.org/10.1108/BIJ-08-2011-0063>
- Almaiah MA, Al-Khasawneh A, Althunibat A, and Khawatreh S (2020). Mobile government adoption model based on combining GAM and UTAUT to explain factors according to adoption of mobile government services. *International Journal of Interactive Mobile Technologies*, 14(3): 199-225. <https://doi.org/10.3991/ijim.v14i03.11264>
- Alshehadeh AR and Al-Khawaja HA (2022). Financial technology as a basis for financial inclusion and its impact on profitability: Evidence from commercial banks. *International Journal of Advances in Soft Computing and Its Applications*, 14(2): 125-138. <https://doi.org/10.15849/IJASCA.220720.09>
- Al-Tamimi KAM, Jaradat MS, YachouAityassine FL, and Soumadi MM (2023). Impact of renewable energy on the economy of Saudi Arabia. *International Journal of Energy Economics and Policy*, 13(3): 20-27. <https://doi.org/10.32479/ijeep.14099>
- Althunibat A (2015). Determining the factors influencing students' intention to use m-learning in Jordan higher education.

- Computers in Human Behavior, 52: 65-71.
<https://doi.org/10.1016/j.chb.2015.05.046>
- Althunibat A, Abdallah M, Almaiah MA, Alabwaini N, and Alrawashdeh TA (2022). An acceptance model of using mobile-government services (AMGS). *CMES-Computer Modeling in Engineering and Sciences*, 131(2): 865-880.
<https://doi.org/10.32604/cmcs.2022.019075>
- Althunibat A, AlNuhait H, Almanasra S, Almajali MH, Aljarrah E, and Al-Khawaja HA (2024). Culture and law enforcement influence on m-government adoption: An exploratory study. *Journal of Infrastructure, Policy and Development*, 8(5): 3353.
<https://doi.org/10.24294/jipd.v8i5.3353>
- Althunibat A, Alokush B, Dawood R, Tarabieh SM, and Gil-Pechuan I (2021b). Modeling the factors that influence digital economy services acceptance. In the *International Conference on Information Technology*, IEEE, Amman, Jordan: 942-945.
<https://doi.org/10.1109/ICIT52682.2021.9491686>
- Althunibat A, Alokush B, Tarabieh SM, and Dawood R (2021c). Mobile government and digital economy relationship and challenges. *International Journal of Advances in Soft Computing and Its Applications*, 13(1): 122-134.
- Althunibat A, Alrawashdeh TA, and Muhairat M (2014). The acceptance of using m-government services in Jordan. In the 11th *International Conference on Information Technology: New Generations*, IEEE, Las Vegas, USA: 643-644.
<https://doi.org/10.1109/ITNG.2014.65>
- Althunibat A, Binsawad M, Almaiah MA, Almomani O, Alsaaidah A, Al-Rahmi W, and Seliaman ME (2021a). Sustainable applications of smart-government services: A model to understand smart-government adoption. *Sustainability*, 13(6): 3028.
<https://doi.org/10.3390/su13063028>
- Bora A, Medhi S, and Bezboruah T (2022). Reliability evaluation for deployment of multi service multi functional service oriented computing based on different techniques. *International Journal of Advanced Intelligence Paradigms*, 22(3-4): 362-378.
<https://doi.org/10.1504/IJAIP.2022.124319>
- Flores A, Ramírez S, Toasa R, Vargas J, Urvina-Barrionuevo R, and Lavin JM (2018). Performance evaluation of NoSQL and SQL queries in response time for the e-government. In the *International Conference on eDemocracy and eGovernment*, IEEE, Ambato, Ecuador: 257-262.
<https://doi.org/10.1109/ICEDEG.2018.8372362>
- Imam AT, Alhroob A, and Alzyadat WJ (2021). SVM machine learning classifier to automate the extraction of SRS elements. *International Journal of Advanced Computer Science and Applications*, 12(3): 174-185.
<https://doi.org/10.14569/IJACSA.2021.0120322>
- Iranpour E and Sharifian S (2018). A distributed load balancing and admission control algorithm based on Fuzzy type-2 and Game theory for large-scale SaaS cloud architectures. *Future Generation Computer Systems*, 86: 81-98.
<https://doi.org/10.1016/j.future.2018.03.045>
- Jebri I, Almaslmani R, Jarah B, Mugableh M, and Zaqeeba N (2023). The impact of strategic intelligence and asset management on enhancing competitive advantage: The mediating role of cybersecurity. *Uncertain Supply Chain Management*, 11(3): 1041-1046.
<https://doi.org/10.5267/j.uscm.2023.4.018>
- Kalita M and Bezboruah T (2012). Investigations on implementation of web applications with different techniques. *IET Software*, 6(6): 474-478.
<https://doi.org/10.1049/iet-sen.2011.0136>
- Karim A and Adnan MA (2019). An OpenID based authentication service mechanisms for Internet of Things. In the *IEEE 4th International Conference on Computer and Communication Systems*, IEEE, Singapore, Singapore: 687-692.
<https://doi.org/10.1109/CCOMS.2019.8821761>
- Mecca G, Santomauro M, Santoro D, and Veltri E (2016). On federated single sign-on in e-government interoperability frameworks. *International Journal of Electronic Governance*, 8(1): 6-21.
<https://doi.org/10.1504/IJEG.2016.076684>
- Milani Fard A, Mirzaaghaei M, and Mesbah A (2014). Leveraging existing tests in automated test generation for web applications. In the 29th *ACM/IEEE International Conference on Automated Software Engineering*, Association for Computing Machinery, Vasteras, Sweden: 67-78.
<https://doi.org/10.1145/2642937.2642991>
- Mumtaz R, Samawi V, Alhroob A, Alzyadat W, and Almukahel I (2022). PDIS: A service layer for privacy and detecting intrusions in cloud computing. *International Journal of Advances in Soft Computing and Its Applications*, 14(2): 14-34.
<https://doi.org/10.15849/IJASCA.220720.02>
- Pradeep S and Sharma YK (2019). A pragmatic evaluation of stress and performance testing technologies for web based applications. In the *Amity International Conference on Artificial Intelligence*, IEEE, Dubai, UAE: 399-403.
<https://doi.org/10.1109/AICAI.2019.8701327>
- Putri MA, Hadi HN, and Ramdani F (2017). Performance testing analysis on web application: Study case student admission web system. In the *International Conference on Sustainable Information Engineering and Technology*, IEEE, Malang, Indonesia: 1-5.
<https://doi.org/10.1109/SIET.2017.8304099>
- Saia SM, Nelson NG, Young SN, Parham S, and Vandegrift M (2022). Ten simple rules for researchers who want to develop web apps. *PLOS Computational Biology*, 18(1): e1009663.
<https://doi.org/10.1371/journal.pcbi.1009663>
PMid:34990469 PMCID:PMC8735566
- Sedek KA, Omar MA, and Sulaiman S (2014). A hybrid architecture for one-stop e-government portal integration and interoperability. In the 8th *Malaysian Software Engineering Conference (MySEC)*, IEEE, Langkawi, Malaysia: 96-101.
<https://doi.org/10.1109/MySec.2014.6985996>
- Shaw J (2000). Web application performance testing—A case study of an on-line learning application. *BT Technology Journal*, 18(2): 79-86.
<https://doi.org/10.1023/A:1026732502654>
- Staegemann D, Volk M, Lautenschläger E, Pohl M, Abdallah M, and Turowski K (2021). Applying test driven development in the big data domain—Lessons from the literature. In the *International Conference on Information Technology*, IEEE, Amman, Jordan: 511-516.
<https://doi.org/10.1109/ICIT52682.2021.9491728>
- Thatmann D, Slawik M, Zickau S, and Küpper A (2012). Towards a federated cloud ecosystem: Enabling managed cloud service consumption. In the 9th *International Conference on Economics of Grids, Clouds, Systems, and Services*, Springer, Berlin, Germany: 223-233.
https://doi.org/10.1007/978-3-642-35194-5_17
- Zeebaree SR, Jacksi K, and Zebari RR (2020). Impact analysis of SYN flood DDoS attack on HAProxy and NLB cluster-based web servers. *Indonesian Journal of Electrical Engineering and Computer Science*, 19(1): 510-517.
<https://doi.org/10.11591/ijeecs.v19.i1.pp505-512>
- Zou Z and Ai J (2020). Online prediction of server crash based on running data. In the 20th *International Conference on Software Quality, Reliability and Security Companion*, IEEE, Macau, China: 7-14.
<https://doi.org/10.1109/QRS-C51114.2020.00014>