COMPARATIVE ANALYSIS OF FULLSTACK DEVELOPMENT TECHNOLOGIES: FRONTEND, BACKEND AND DATABASE

by

QOZEEM ODENIRAN

(Under the Direction of Hayden Wimmer)

ABSTRACT

Accessing websites with various devices has brought changes in the field of application development. The choice of cross-platform, reusable frameworks is very crucial in this era. This thesis embarks in the evaluation of front-end, back-end, and database technologies to address the status quo. Study-a explores front-end development, focusing on angular.js and react.js. Using these frameworks, comparative web applications were created and evaluated locally. Important insights were obtained through benchmark tests, lighthouse metrics, and architectural evaluations. React.js proves to be a performance leader in spite of the possible influence of a virtual machine, opening the door for additional research. Study b delves into backend scripting by contrasting node.js with php. The efficiency of sorting algorithms—binary, bubble, quick, and heap—is the main subject of the research. The performance measurement tool is apache jmeter, and the most important indicator is latency. Study c sheds light on database systems by comparing and contrasting the performance of nosql and sql, with a particular emphasis on mongodb for nosql. In a time of enormous data volumes, reliable technologies are necessary for data management. The five basic database activities that apache jmeter examines are insert, select, update, delete, and aggregate. The performance indicator is the amount of time that has passed. The results showed that the elapsed time for insert operations was significantly faster in nosql than in sql. The p-value for each operation result was less than 0.05, indicating that the performance difference is not significant. The results also showed that the elapsed time of update, delete, select, and aggregate operations are less in nosql than in sql. This suggests that the performance difference between sql and nosql is not significant. These research studies are combined in this thesis to provide a comprehensive understanding of database management, backend programming, and development frameworks. The results provide developers and organisations with the information they need to make wise decisions in this constantly changing environment and satisfy the expectations of a dynamic and diverse technology landscape.

INDEX WORDS: Framework, JavaScript, frontend, React.js, Angular.js, Node.js, PHP, Backend, technology, Algorithms, Performance, Apache JMeter, T-test, SQL, NoSQL, Database management systems, Performance comparison, Data operations, Decision-making.

COMPARATIVE ANALYSIS OF THE FULLSTACK DEVELOPMENT TECHNOLOGIES: FRONTEND, BACKEND AND DATABASE

by

QOZEEM ODENIRAN

B.S., University of Ilorin, Nigeria, 2018

A Thesis Submitted to the Graduate Faculty of the Georgia Southern University in the Partial Fulfilment of the Requirement for the Degree

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY

STATESBORO, GEORGIA

COMPARATIVE ANALYSIS OF THE FULLSTACK DEVELOPMENT TECHNOLOGIES: FRONTEND, BACKEND AND DATABASE

by

QOZEEM ODENIRAN

Major Professor:	Hayden Wimmer
Committee:	Meenalosini Vimal Cruz
	Kim Jongyeop Kim

Electronic Version Approved:

December 2023

DEDICATION

I dedicate this to Almighty God, and a very few good people.

ACKNOWLEDGMENTS

I would like to sincerely thank Dr. Hayden Wimmer, my thesis advisor, for his excellent advice, steadfast support, and knowledgeable insights during the study process. I would like to express my sincere gratitude to every member of my thesis committee for their insightful criticism and contributions to this work.

In addition, I want to thank my fellow students for their support and companionship throughout this academic adventure. Your assistance has greatly influenced how this thesis has turned out.

I appreciate all your help and inspiration.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Development frameworks and stacks have evolved and improved over time. Deciding on which framework to apply to a particular application is a choice that deserves careful consideration. In a multi-tier framework, one must consider front-end, back-end, and database technologies. Angular and React are modern front—end frameworks with Angular developed by Google and React developed by Facebook to address specific issues for the Facebook application. PHP, or PHP Hypertext Processor, has been around for decades and dominated the development landscape due to its power, performance, and scalability and is the most common framework on the web today; however, JavaScript has emerged as a more modern alternative and challenger to PHP's legacy. Similar, MySQL is the most common back-end database technology due to its performance and open-source support; however, MongoDB has emerged as a challenger and seeks to improve performance over MySQL. In this work, we seek to compare these frameworks from a front-end, back-end, and database level. We first focus on Angular vs React, followed by PHP vs JavaScript, and finally we compare MySQL to MongoDB.

The first study focuses on Angular versus React. The front-end engineering of web-pages continuously experience growth as the number of devices used in accessing it increases. Users tend to spend more time on a webpage if it has a simple and appealing interface. In the past decades, building a responsive webpage is tedious as the developer must make use of HTML, CSS, and pure (vanilla JavaScript). Nowadays, all these have been combined into an easy-to-use template known as a framework. However, many frameworks have been developed by developers from diverse backgrounds, and the choice of which framework to use must be made rightly. Hence, we embark on a research study to unveil features, power, and benchmark characteristics of some selected JavaScript frontend frameworks (React.js and Angular.js) to ease the decision-making process for developers.

Following front-end development, we move to back-end web technology. Within the field of back-end web scripting, architecture, scalability, and performance are frequently discussed. In our research study, PHP and Node.js are used to illustrate this argument. The popularity of Node.js, a contemporary and effective back-end technology, has grown, while PHP continues to keep its historical position. Performance issues are very important in the quick-paced digital world of today. Our goal is to evaluate the performance of PHP and Node.js using popular algorithms to shed light on this matter. The temporal complexities of binary, bubble, quick sorts, and heap algorithms allowed for effective performance differentiation in their selection. Through rigorous data gathering using Apache JMeter, latency comparisons, and extensive testing with varying array sizes, this study provides valuable insights for software engineers and developers. It facilitates the process of deciding which backend scripting is best.

Finally, we examine the performance between SQL and NoSQL databases. Within the field of data management, our research study centers on the crucial decision between NoSQL and SQL database systems, tackling the needs of a constantly changing, diverse data environment. By carefully comparing the performance of these two different database systems across essential activities, including as Insert, Select, Update, Delete, and Aggregate (Average), it acts as a guide for decision-makers, database administrators, and developers. The study highlights the superiority of SQL databases in reporting and aggregating tasks, facilitated by Apache JMeter and the T-Test statistical approach, while revealing the strengths of NoSQL databases, exemplified by MongoDB, in data manipulation tasks.

CHAPTER 2

LITERATURE REVIEW

2.1 STUDY A – Literature Review

Presently, devices of various sizes are used to access webpages. This resulted in an increased demand for a cross-platform, reusable, and easy-to-maintain front-end development code to enhance the development process and efficiency of the overall front-end system. Framework technology offers well-defined code structures embedded with enough features to accomplish a speedy front-end development solution [1]. Xu [1] focused on a thorough evaluation of popular JavaScript frameworks in order to make available insightful options while developers are in the decision-making stage of which frontend JavaScript framework to use for a particular project. In Xu [1]'s research, the same website was created with selected frameworks to evaluate the advantages and disadvantages, and the kind of projects they can be used for. Also, performance metrics for each website were also recorded to compare the efficiency of these frameworks in terms of DOM operations. The experiment in Xu [1] research unveiled the important characteristic features of the selected JavaScript frontend frameworks making the selection process easier for developers when they are about to embark on a project.

JavaScript is one of the most popular scripting languages used in the development of standard, interactive and easy-to-maintain websites [2]. Hence, many frontend frameworks leverage the outstanding features of JavaScript, which in turn make the design and maintenance of small and large websites a breeze. Among others, React.js, Angular.js, and Vue.js are the most popular JavaScript frontend frameworks. study aimed to carry out a comprehensive evaluation of the most used JavaScript frontend frameworks in order to establish which framework is advantageous in terms of DOM performance benchmark, as well as determining the strength and weaknesses of each framework in the realm of web development. Assessing Document Object Method (DOM) performance metrics was what Levlin [2] based the evaluation of his experiment on. In the study, he created four similar applications in React.js, Angular.js, Vue.js, and Svelte. These applications all have the same functionalities and HTML, CSS files, and test files are located in an 'src' folder, to avoid unbiased testing and performance. In his result, Levlin [2] inferred that the React.js framework had the overall best performance having been observed to be the most used framework as well as satisfying standard scores in DOM performance metrics, popularity, documentation languages, and so on.

The inability of Hypertext Markup Language (HTML) to satisfy the need for application integration, model flexibility and cross-platform birthed the three-layers architecture which supports the separation of the logical layers in an application. These layers include the presentation layer, application layer, and a dedicated database layer [3]. Many frameworks were created following this style and have been experiencing exponential growth since inception as the latest technologies were been introduced to complement its evolvement. The objective of Verma [3] research was to examine the performance of different frameworks by executing similar chat applications built in selected web frameworks and native frameworks under the same server. Background network communication makes obtaining targeted data from native applications more difficult than web applications. Hence, while other background-running network-related activities were isolated from the network requests on the chat server, Verma [3] obtained datasets for the selected frameworks via network methodology. The allocation of performance scores to selected frameworks was based on criteria such as development, debugging, modifying, and testing of the chat applications. From Verma [3] it was inferred that the association of React.js and Rails framework satisfied the overall standard performance, hence emerging as the most efficient frameworks. While the second-rated performance goes to the combination of Laravel and Vue.js frameworks.

As the request to build more complex web applications such as Multiple Pages Applications (MPA) and Single Page Applications (SPA) rises, the decision-making process as to which framework to use could pose a challenge to the developer [4]. However, making the wrong choice of framework might result in some setbacks in the development process. Hence, Vukelić [4] worked on research that examines the available frameworks and proffers some decision-making advice on which framework is suitable for which web application project. In this research, Vukelić [4] hoped to give a detailed explanation of MPA and SPA web applications emphasizing on advantages and disadvantages, and characteristic features expected of a framework to make it suitable for specific web application projects. Vukelić [4] adopted the style of initially gathering some probing questions on MPA and SPA development, then based the analysis of available frameworks on these questions. Doing this ensured narrowed-down research as to which features of the available framework to look out for. At the end of the experimental analysis, Vukelić [4] presented a table composed of obtained performance metrics showing a comparison among React.js, Angular.js, and Vue.js frameworks, and how they can enhance the development of MPA and SPA.

JavaScript is indeed one of the most famous and used scripting languages among developers of today. Its complementing features embedded in a lot of frameworks and libraries are another supporting factor for its popularity among developers, especially frontend-heavy programmers [5]. In order to properly utilize different JavaScript frameworks, and to beat the challenges in choosing the right framework for a specific project, it is important to investigate the available frameworks to get more insight into specific features which might or not meet the requirements of a project. Yorulmaz [5]'s goal of this research was to thoroughly asses some selected JavaScript frameworks (Angular and React), then be able to give solid suggestions as to which framework best suits a project. For this research, Yorulmaz [5] chose to develop two To-do-list applications, one in Angular.js, and the other in React.js. He conducted a comparative analysis of these applications based on some chosen factors which appeared to be significant to developers, as well as performed an extensive evaluation. At the end of the experimental processes, Yorulmaz [5] gave detailed results on the selected frameworks based on the comparative analysis conducted. The results comprise of suitability and updates of the frameworks where Angular.js and React.js were almost rated the same, variables and functionalities where React.js is a bit complex to comprehend. Other evaluation factors considered are documentation, number of libraries, performance, security, localization, and so on.

In order to achieve absolute productivity in a shorter time, developers leverage the power of a framework [6]. Apart from the fact that the use of frameworks facilitates a shorter time to market in application development via design and code reuse, there are some overhead issues a developer should be aware of as regards development. Furthermore, another issue associated with making choice-of-framework is that, as long as JavaScript frameworks are open-source, a lot of JavaScript developers continually propose new features which force the framework users to remain updated with the new development in the selected framework. In a thesis titled '***Supporting web development decisions by comparing three major JavaScript frameworks: Angular, React and Vue***' by Wohlgethan [6], a constructive overview of three JavaScript frameworks which are currently trending in the market was present. This aims to provide software developers with indications in the decision-making process of which framework to use for front-end development. Wohlgethan [6] summarized the criteria for analysis of the selected framework into three aspects which are ***stability, learning curve,*** and ***JavaScript integration***. In the first aspect, sub-criteria like versioning, release policy, maintainability, and licensing was discussed. The second criterion considered available documentation, knowledge requirements, and career opportunities. While the third consists of stacks, development languages, and syntax. A comparative analysis among the three frameworks was performed and compared baes on selected features. In the results, Wohlgethan [6] gave conclusion that there is no constructive conclusion that states that one framework is better than another. However, the choice of a framework sits on the shoulder of the nature of the project being considered, the developer's knowledge. Other points such as community support, and file separation technology which React.js and Angular.js score a high score due to big tech companies' support.

Mohammadi [7] classified the traditional website's model as being multipage which is characterized by bad responsiveness, meaning that it takes a considerable amount of time for the browser to load/refresh when users perform operations like a page change or data retrieval from the server. As more and more operations are performed, more time is required to finish a process. This calls for a system where the client (front end) and the server (back end) can be separated such that logic and processing are handled by the server. Hence, a single-page application (SPA) was proposed as a feasible solution. The objectives behind (Mohammadi)'s research includes the following: Introduction of JavaScript's framework (React.js and Angular.js) along with its structure and functionalities, as well as analyzing what key features are available in them, unveiling some key advantages between selected framework and making the right choice in the project, to create a SPA in

Angular in order to be able to assess its complexity then make an inference as to which kind of project it is most suited for. In [7]'s research, in order to carry out a comparative analysis of the selected frameworks of study, a single-page application (***property service website)*** was developed and various benchmark features such as communication trends between clients and servers, and the number of libraries available were evaluated. [7] concluded, regarding his experiment that Angular turned out to be the most used for SPA due to its data binding features and considerably large support from Google. Also, the Angular.js framework is capable of working successfully with the SPA model.

With the rate at which JavaScript frameworks are evolving, KEPLER [8] suggested that users (i.e., developers) must meticulously choose which one to use in a project, otherwise, they would be investing their time and money wrongly. For the fact that if developers are not familiar with the fundamental programming language of a framework, the need for changing to another framework is the next thing to do, and this might happen at any point in the project life cycle, it becomes important to analyze and review some merits and demerits of a bunch of the available frameworks of today as well as being informed of its overall requirements [8].

Defining the research's problem statement, KEPLER [8] aimed to offered to carry out a comparative analysis by experimenting with and evaluating the three most popular JavaScript frontend frameworks (React.js, Angular.js, and Vue.js) in order to establish some solid facts about the frameworks that would be useful to developers and companies in the decision-making process stage. In order to perform the experiment, a website that displays data from an Android Device Security Rating was developed with one of the frameworks. KEPLER [8] placed some important requirements such as presenting data in a table, filtering of table data, URL representation of filters, and full responsiveness on the website in order to be able to grab some metrics while performing various operations. Also, KEPLER [8] made sure to implement the operations with huge data to measure efficiency. The collection of relevant criteria was done via the study of literature and their comparison methods on web applications and was used to evaluate selected frameworks. In the results of his experiments, KEPLER [8] inferred that Angular is characterized by a smooth learning process and clear structures, hence selected for the applications like the one developed in the research.

Due to an increased demand for better functionalities and usability, web applications needed to adopt new techniques, hence abandoning the idea of browser plugins and embracing the SPA (single-page application) model [9]. The embraced model leveraged the powerful feature of JavaScript to asynchronously fetch data and partially or completely update web pages' contents without refreshing the page. Mousavi [9] embark on research to evaluate the two most popular JavaScript frameworks namely: React.js and Angular.js. In the midst of competitive markets, rapid requirements, and accelerated delivery, Mousavi [9] main objective was to evaluate and compare the selected frameworks' maintainability, and complexity, to establish firm facts about the frameworks while helping developers gain more insights. In the research, Mousavi [9] evaluated prototype applications in both React and Angular and performed an extensive comparative analysis of components of the applications and frameworks involved from basic to advanced levels. An open-source tool, ***Plato*** was used to obtaining performance metrics. Then, presented results and statistics in tabular formats to give quantitative reports. Also, qualitative reports were obtained. The result from the Mousavi [9] experiment showed that both frameworks significantly differ in terms of the number of files and lines of code. Also, Angular.js uses more function calls for retrieving server data, and splitting data into related chunks, while React.js leverage state mutation to separate functions into two files which cancel the need for factory and abstract class for exercise. This results in React.js using fewer lines of code. However, the object-oriented style of Angular.js makes a better splitting with respect to higher cohesion [9].

The correlation between the framework used in the development of SPA (Single Page Application) and the application itself is positive [10]. The workflow of an application is dependent on the framework it was developed in. Also, the use of frameworks speeds up the development process and cancels the occurrence of possible errors. However, the uniqueness of each framework cannot be underestimated. Hence, a decision on which framework to use for a project must be strategic [10]. Saks [10]'s interest was to make the decision-making process a breeze for developers while trying to figure out what framework is best to use for what project. Hence, Saks [10] conducted a comparative analysis of the three most popular JavaScript frameworks: React.js, Angular.js, and

Vue.js. For the purpose of obtaining performance metrics, Saks [10] developed three similar small-sized web applications in React.js, Angular.js, and Vue.js. In these applications, a key performance metric, ***loading speed*** was evaluated while some defined operations were performed. After an extensive evaluation exercise Saks [10] inferred that React.js is the most popular and perhaps the best framework to consider learning to land a well-paying job. Also, Vue surfaces to be the fastest framework and easiest to learn. While Angular seems to be the hardest to learn and is characterized by the slowest performance of the three [10].

As the era of web development is fast advancing, the need to develop interactive web applications becomes significant [11]. JavaScript frontend frameworks help facilitate the development of single-page web applications, hence referred to as one of its most important parts. Kumar and Singh [11]'s research aims to offer an in-depth understanding of the two most popular JavaScript frameworks, React.js and Angular.js. First, Kumar and Singh [11] comprehensively introduced the client side of the web application, and technologies used such as HTML and JavaScript. Then, the frameworks of concern, React.js and Angular.js were also discussed, as well as the Virtual DOM which serves as a tree having nodes listing HTML elements, attributes, and contents in terms of properties and objects [11]. Lastly, the two JavaScript frameworks were compared in terms of some benchmark features and performances to unveil the power of each of the frameworks. Obtained results showed that both frameworks differ in features including the size of the project, supporting the community, debugging techniques, Document Object Model (DOM), as well as mobility (Kumar and Singh [11]

The frameworks created in JavaScript are commonly used in both client and server-side development of web applications [12].However, one of the most dangerous security issues in web applications is cross-site scripting (XSS), and this is pertinent to JavaScript applications. Hence Peguero, et al. [12] did extensive research to contribute to the understanding of how a JavaScript's framework embedded security features can impact the application's security developed in such framework. Peguero, et al. [12] demonstrated four places in an application where an XSS reduction can be done, in relation to the framework being utilized. Peguero, et al. [12] conducted an experimental analysis of the three most-used template engines namely: Jade/Pug, EJS, and Angular used by JavaScript applications. Peguero, et al. [12] determined the number of projects vulnerable to XSS, as well as the number of vulnerabilities in each of the projects, depending on the framework used, using an automatic and manual examination of each group of applications. At the end of the empirical study, Peguero, et al. [12] inferred that application security is affected by the location of the mitigation; framework-based mitigations make apps more secure.

Lots of frameworks have been developed and adopted by IT companies as well as researchers [13]. This is a result of new requirements for the features of web and mobile-based applications. However, Sultan [13]understands the fact that these frameworks differ from each other in one way or the other. The difference in characteristics and features of the existing and newly developed framework, therefore, opens a door for a comparative study of these frameworks, which is the goal of [13]'s research. To establish a firm distinction among the numerous frameworks, Sultan [13] identified twelve (12) new characteristic features common to all frameworks of interest, then extensively compare how these features are implemented on each framework. After well performed comparative study of these frameworks, various advantages of using a particular type of framework were discussed. Although, no framework was pronounced the 'winner', otherwise, the pros and cons of using choosing a particular framework for a particular project were clearly stated to help companies, developers, and researchers in the decision-making stage [13].

In order to make sure that long-run development is achieved, old technologies should be updated with new technologies [14]. Utilizing a frontend framework is one of the new technologies that facilitate the separation of frontend and backend development, hence making both development processes independent. Guan, et al. [14], aim to use this research as a medium to achieve the review of the framework (Yahoo UI) used to develop Networked Control System Lab (NCSLab) system.

Redevelop the existing system with the React.js framework, as the support for Yahoo UI is no longer available. Make a comparison among Angular.js, React.js and Vue.js First, Guan, et al. [14] comprehensively explained the current architecture of the NCSLab in order to give a clear insight into its current state. Then, the system was redeveloped in the React.js framework. Also, deep explanations and e=definitions of all components of React.js utilized in the redesign were provided, as well as an emphasis on the backend-frontend separation. In conclusion, Guan, et al. [14] inferred that the exceptional features of the React.js framework brought about a new user experience (UX) on the pages of NCSLab.

As the technology on the web is experiencing exponential growth, HTML is fast becoming the worldwide consortium, as well as leading the frontend development to be standing firm at the forefront of the history of the internet [15]. The goal of [15]'s research is to establish some facts to assist in the selection stage of frameworks or libraries in the development of an e-business platform in order to ensure a satisfied UI/UX (User Interface / User Experience). In Xing, [15]'s research, the most popular frontend frameworks React.js, Angular.js, and Vue.js were comparatively studied in terms of performance focusing on the features that can improve the overall performance of an e-business platform. The frontend solutions such as data processing, volume & performance, language-based, and technical support were examined. A conclusion from this study was that the Angular.js framework is characterized by extensive functions and features capable of the development of large commercial projects such as e-business platforms, while React.js and Vue.js frameworks are a good fit for small-to-medium projects such as streaming applications, blogging system, etc. [15].

2.2 STUDY B – Literature Review

As organizations' information systems grow, Prayogi, et al. [16] were interested in the fact that data exchange within those systems becomes increasingly important. Information systems play crucial roles in business processes in educational institutions, from student and course registration to end-of-semester/session evaluations and graduation processes [16]. The objective of Prayogi, et al. [16]'s research study was to prototype the development of a REST API for academic information systems and to analyze its performance by implementing it using two different server technologies: PHP and Node.js. The prototype that Prayogi, et al. [16] implemented was developed using a database with one sample table representing employees in a college, and two different endpoints were created for the implemented REST API. In the experiment, Apache JMeter was used to simulate a thousand concurrent requests on the database .Node.js consistently outperformed PHP in the experiment, achieving a 100% throughput for all concurrent 1000 requests, while PHP recorded a throughput of only 48.70% under the same conditions [16].

Chaniotis, et al. [17]'s article discusses the implications of developing end-to-end web applications in the social web era and examines a distributed architecture suitable for modern web application development. The aim of this study was to find the most efficient and scalable technology stack for web application development in the current social web era. The authors, Chaniotis, et al. [17] conducted stress tests on popular server-side technologies, including PHP/Apache stack, Nginx, and Node.js, to determine their efficiency and scalability. The study found that the PHP/Apache stack was not efficient in handling increasing demand in network traffic, while Nginx was more than 2.5 times faster in I/O operations than Apache. Node.js outperformed both in I/O operations and resource utilization but lacked in serving static files. Therefore, Nginx was recommended to be used in front of Node.js to proxy static file requests, offering better infrastructure in terms of efficiency and scalability. The study concluded that building cross-platform applications using web technologies is feasible and productive, and Node.js is an excellent tool for developing fast, scalable network applications that offer client-server development integration and aid code reusability [17].

Raharjo [18]'s research explores Node.js, which is an application framework that can construct network server and web applications, and its effectiveness in building dynamic web applications. The authors draw a comparison between Node.js and the PHP/Nginx web development stack and analyze their performance and scalability. The main goal was to assess and compare the performance and scalability of Node.js and PHP/Nginx web applications using a load generator. To conduct the study, the designed mock applications based on the Dijkstra Algorithm, which computes the shortest path between nodes, particularly the Trans Jogia shelters. They utilized a load generator to simulate concurrent user requests and assessed the performance and scalability of Node.js and PHP/Nginx web applications. The study determined that Node.js applications performed better and were more scalable than PHP/Nginx applications [18].

Lei, et al. [19]'s study highlights the significance of large-scale, high-concurrency, and data-intensive web applications in the latest generation of websites. Node.js has become popular for building such applications. Lei, et al. [19] were interested in comparing the performance of Node.js, Python-Web, and PHP in constructing data-intensive web applications using benchmark and scenario tests. Lei, et al. [19] conducted benchmark and scenario tests to compare the performance of Node.js, Python-Web, and PHP in creating data-intensive web applications. The benchmark tests assessed performance data, while the scenario tests simulated realistic user behavior. The study results revealed that Node.js has a much higher capacity than PHP and Python-Web for handling requests within a specific time frame. The authors concluded that Node.js is lightweight and efficient, making it an excellent choice for I/O intensive websites. PHP is only suitable for small and medium-scale applications, while Python-Web is developer-friendly and suitable for large web architectures. This study is the first to assess these web programming technologies using both objective systematic tests and realistic user behavior tests, with Node.js being the primary focus of discussion [19].

The current requirements of web applications, which demand performance and scalability and implement the event model throughout the stack was examined in the research study. It also introduces Node.js, a new web framework that accomplishes both through server-side JavaScript and event-driven I/O. The objective is to compare practical web frameworks for the challenges of the current web and assess the performance and efficiency of Node.js as a server-side language and non-blocking asynchronous model. The experiments will compare Node.js to Apache and evaluate practical web frameworks for the challenges of the current web. Tests was conducted against two comparable frameworks that compare service request times over multiple cores. The results will demonstrate the performance of JavaScript as a server-side language and the efficiency of the non-blocking asynchronous model. The research study concludes that Node.js is an appropriate framework for developing scalable web servers that can be scaled and distributed across multiple nodes using clustering and replication mechanisms. Node.js is efficient in utilizing a multithreading programming approach and its non-blocking asynchronous model, making it an ideal choice for high-performance web applications [20].

To verify portability and flexibility, Dhalla [21] tests native JSON parsers written in 5 different programming languages (Java, Python, PHP, MS.NET Core, and JavaScript). Additionally, the research emphasizes the significance of JSON in IoT, mobile computing, smart grid systems, big data applications, and the requirement for effective data transfer. The goal of this study is to observe how five different programming languages can easily parse JSON files that have fixed nesting depths and key-value pairs. It will assist in choosing the best technological platform for real-time messaging, IoT, and JSON parsing, among other uses [21]. Five JSON parsers were tested by Dhalla [21], and their efficiency in terms of processing time and resource usage was measured. It was done using a laptop running Windows 10 Home, an Intel Core i5-7200U processor clocked at 2.50 GHz, and 16 GB of RAM. Microsoft's System and Oracle's JSON-P library were two of the five parsers employed.JSON.parse() in Node.js, the Text.Json library, the native Python json module, and the json decode function in PHP. Five parsers—JavaScript, Java, PHP, Python, and MS.Net Core—were tested for parsing performance, memory usage, and CPU usage. All parsers' parsing speeds were shown to be positively correlated with file size, with JavaScript being the least efficient for smaller files and Java being the highest. JavaScript has the most RAM for greater sizes, whereas Python had the least. For JSON-formatted data, JavaScript performed the best while Java used less memory and CPU. Less effective than the other three parsers were the PHP and Python parsers. More programming languages and real-world JSON data should be included in future work [21].

The use of server-side JavaScript programming is discussed in Nkenyereye and Jang [22], and Rhino, a Java-based JavaScript interpreter, is contrasted with V8-based alternatives like Node.js and SilkJS. It describes how Node.js enables scalable network applications by avoiding blocking with an event-loop and ensuring that call-backs are handled as rapidly as feasible when events happen. Nkenyereye and Jang [22] suggests using a wearable device for remote healthcare monitoring in rural villages, but it also emphasizes how difficult it will be for the healthcare hub server to manage concurrency problems. It is advised to use Node.js as a cross-platform runtime environment to effectively manage concurrency [22]. According to test results, MongoDB is 40% quicker than Apache Sling and can be used to create an event-driven, high-performance healthcare hubserver. For the Remote Healthcare Monitoring (RHM) to gather patient data and compare it to guidelines established by doctors, an object model is offered. A message is sent to a set of actions to run an object, which carries out tasks that the patient and the doctor had previously agreed to. A message queue and a thread pool are combined with Node.js to provide asynchronous communication. The callback contains the state data for passive objects that completed lengthy, non-blocking tasks [22]. The test plan for evaluating Node.js and Apache Sling's functionality in a healthcare monitoring system is described [22]. The client is the Instant Heart Rate Android app, and JMeter is used to gauge response time and throughput. The findings from Nkenyereye and Jang [22]'s research study indicate that Node.js-MongoDB outperforms Apache-Sling in terms of throughput and response time, especially when there are many concurrent users. It was discovered that Node.js was about 40% quicker than Apache-sling.

Crawford and Hussain [23] discussed the difficulties in selecting a server-side technology for web development, comparing four major scripting languages, including PHP, Django, Ruby on Rails, and Node.js, based on five characteristics: popularity, accessibility to development tools and packages, ease of starting, and availability of help and support. It is predicted that this research will be useful to software engineers, developers, and educators that teach web development courses [23]. Crawford and Hussain [23] compared the selected scripting technologies in terms of getting started, help and support, popularity, development tools and package management systems, integration with database and drivers, and lastly, performance. Node.js, due to its fast-technology nature enables event-driven, non-blocking I/O operations [23]. It is a well-liked development environment that can be used to build console apps, run on desktop computers using Electron, in web browsers, and in embedded systems. It is simple to keep using any database system desired thanks to the interface with five of the most well-liked databases through packages. However, given the limited number of packages, operating systems, and performance levels, its learning curve is constrained [23].

New programming tools are increasingly necessary, particularly for web and mobile apps, as websites and web applications continue to expand [24]. The two most widely used web development languages, PHP and ASP, are contrasted in this study with an emphasis on their most recent iterations. In order to create dynamic, interactive web applications, server-side scripting languages like ASP and PHP as well as databases like MySQL are used. PHP is a well-liked general-purpose scripting computer language for online development, while ASP is an open-source web framework that builds webpages using HTML5, CSS, and JavaScript. Odeh [24] compared the quantity of websites on the internet between 1995 and 2018 using the C# programming language. The purpose of Odeh [24]'s work is to offer a critical analysis and useful advice on how to use a web programming language to create a high-quality product. Odeh [24]'s research technique looks at web developers' experiences and views on PHP, ASP, and web development. Other comparison criteria include cost, performance, readability, understanding, maintainability, editing & deployment tools, platform, database, webservers, core-language, synthetic character, webpage structure. The knowledge, application domain, platform being used, and other considerations all play a role in the decision between ASP and PHP. While ASP is more dependable and effective than PHP, PHP is better suited for developers who are more familiar with Microsoft products. Both are appropriate, but C# is a safer language in terms of server-side web application development [24].

In order to make them simpler for users to understand and learn, programming languages are designed to increase readability and writability [25]. High-level programming languages were created to boost abstraction and simplify coding, including Python, R, and Julia. For both inexperienced and seasoned programmers, readability, writability, and reliability are crucial aspects of programming languages. Reliable programming languages guarantee that the code behaves as intended. The readability, writability, and dependability of six regularly used programming languages are compared in [25] study: C, C++, Java, Python, JavaScript, and R. The study's goal is to assess their competitive advantage in various applications and comprehend the trade-offs. A survey was used to collect data, and a theoretical comparison was performed to analyze the value of these criteria and their impact on the decision-making process of selecting a programming language. Ahmed, et al. [25]'s research devised a novel method for comparing the six programming languages in terms of readability, writability, and dependability. A metric evaluation system was developed to evaluate each language for each indicator using the categories 'Bad,' 'Moderate,' and 'Good.' A poll was also undertaken to validate the appraisal of this metric system, with a group of people answering a series of questions. The most readable and writeable languages were found to be Python and R, with Java having an advantage in terms of reliability [25]. According to a poll, Python is the programming language of choice for beginners and non-programmers, although Java is favored by seasoned programmers due to its dependability. Theoretical analysis and survey findings were complementary, with Python excelling in readability and writability and Java excelling in reliability. Expert programmers might, however, favors the language in which they feel the most at ease.

The evolution of web scripting languages from CGI to PHP and ASP.NET is covered in this research [26]. Open source, PHP is a commonly used server-side scripting language that can be integrated into HTML. Microsoft's ASP.NET web development platform enables programmers to create dynamic web apps using compiled languages like VB.NET and C#. Scripting languages are progressing on many fronts, but businesses still have the difficult but important task of comparing options to determine which one best suits their particular requirements [26]. The focus of the Adebukola and Kazeem [26]'s research study is to analyze and contrast the effectiveness of PHP and ASP.NET, the two most widely used dynamic scripting languages for online development. In Adebukola and Kazeem [26]'s study, WAPT software is used to evaluate PHP and ASP.NET for online application development. The focuses on measuring the response time, which is the amount of time it takes for a client to send and receive a request, to assess the performance of online applications created using PHP and ASP.NET. Performance tests were run to determine the average, minimum, and maximum reaction times while the two technologies remained the same during the development of the program. The most significant performance indicator is thought to be response time. PHP has a faster reaction time under stress and endurance tests, making it a superior choice for online applications [26].

For online applications that require lots of data, Node.js is gaining popularity [27]. Due to its event-driven, non-blocking I/O approach, Node.js is growing in popularity, and Brar, et al. [27]'s study employs objective benchmark testing and accurate user behavior testing to assess its performance. Brar, et al. [27] utilized benchmark tests and scenario tests to evaluate the performance of Node.js, Python-Web, and PHP. The test results provide some insightful enforcement data, demonstrating that Node.js can handle far more requests in a certain period than PHP and Python-Web can. Three key tests were run in Brar, et al. [27]’s experiment to gauge how well Node.js, Python Web, and PHP performed. Node.js fared better than Python Web and PHP, according to the results, in terms of average requests per second and latency per request. Python-Web, Node.js, and PHP are mature frameworks for large-scale websites [27].

Since 1991, websites have developed quickly, giving businesses access to online presences, portfolios, and web-based applications HTML-based website development has given way to Node.js and Python, which provide new difficulties such multi-user requests and high concurrency [28]. Challapalli, et al. [28]’s research paper examines the process of creating websites in the past and present, the evolution of content delivery over time, the uses of websites, websites for mobile devices, and a performance comparison of the two most popular web backend development languages, namely Node.js and Python. In Challapalli, et al. [28]’s research, the server must be started and the program must be directed to the same port on the local host in order to prepare for the test. Only 10 users are simulated by the software because more will increase the server's and benchmark software's resource demands. To prevent high CPU utilization and slow server response, it is best to keep the number of users low. The "Hello World" server model, which is the simplest operational server model, aids in identifying the internal variations in various technologies. Node JS outpaced Python at processing requests, processing about 250 times as many requests as Python in a 30-second period. As the number of users increased, Node JS's requests per second increased rapidly, whereas Python's increase was steady and gradual. The failure rates in both situations were 0%. Python's average response time was roughly 2040ms, compared to 7ms for Node JS. While Python's latency climbed dramatically as the number of users increased, plateauing at 14000ms near the end of the test, with an average delay of 7187ms, Node JS had an average latency of 1.04ms [28].

2.3 STUDY C – Literature Review

Structured Query Language (SQL) representing the Relational database poses itself as a logical choice for data containing fixed or rarely changeable data structure. However, NoSQL databases advent opened the door for fast processing of vast quantities of unstructured data, such as Biomedical data, EEG signal data [29] [30]. As today’s businesses continue to face challenges in the use of different database types, Bjeladinovic, et al. [30], proposes a panacea to explore the use of a database consisting of SQL/NoSQL. In the hybrid database system proposed by Bjeladinovic, et al. [30], when the user initiates a query statement, the system decomposes it and determine the suitable database language to be used in executing the query and adjusts to it. The system then accepts the results, unifies them, translate them into user's language, then display an integrated result [30]. Bjeladinovic, et al. [30] architecture enabled connection of data from different hybrid component as well as centralization of database administration.

Antas, et al. [31]'s study examines the impact of COVID-19, evaluating database systems for storing and mining COVID-19 data, assessing SQL and NoSQL databases along with Data Mining algorithms. The research identifies MongoDB as the most effective database for most tests and highlights the success of the Random Forest algorithm in predicting COVID-19 outcomes. Despite vaccine progress, the paper emphasizes the need for continued efforts and AI-based analysis to address the pandemic's challenges and prevent future crises. By constructing a complete data framework for COVID-19 analysis using a variety of data sources, Antas, et al. [31] solves shortcomings in earlier research. With the use of cutting-edge techniques like Naive Bayes, Decision Tree, Random Forest, and Logistic Regression, the work focuses on choosing appropriate databases for COVID-19 data and establishing a framework for data mining. To conduct performance testing, the study combines data from numerous sources, particularly Brazilian hospitals, and builds relational models for Microsoft SQL Server, MongoDB, and Cassandra databases [31]. The study builds a COVID-19 predictive model with a variety of data variables, attaining good accuracy and insights. It compares databases and discovers that MongoDB outperforms SQL Server in unstructured data without the need for join queries due to its superior performance and scalability. The study emphasizes the value of coordinated efforts in containing the pandemic and highlights MongoDB as an excellent option for COVID-19 data processing [31].

The demand for effective data storage and retrieval has increased due to the expansion of cloud and IoT applications [32]. Because they can manage Big Data from IoT contexts, NoSQL databases like Redis, Cassandra, MongoDB, and Neo4j are overtaking conventional SQL-based databases in popularity [32]. In this situation, security and privacy issues develop that call for solutions for data protection and access management. Sicari, et al. [32]'s study investigates these problems, evaluates the functionalities of several NoSQL databases, and offers potential future paths for strengthening security in IoT database scenarios. The rising popularity of NoSQL databases in the business sector is attributed to their distributed architecture and dynamic management capabilities. Despite mimicking relational databases in certain aspects, they maintain their unstructured nature. The trend of using SQL query engines like Apache Drill to access NoSQL databases is on the rise. Key-value models are not recommended for data relationships or complex transactions, while document-oriented models suit logging events and web analytics, and column-oriented models are useful for indexing, counters, and deadline management [32]. The analysis of the state-of-the-art shows that many literary works make use of NoSQL databases, frequently coupling them with security frameworks rather than directly integrating security capabilities, to ensure the accuracy of data storage and querying procedures.

Both SQL and NoSQL Databases face difficulties in processing Big Data effectively due to their respective emphasis on managing structured data and vertical scalability (SQL) against handling unstructured data (NoSQL) [33]. The different topologies of SQL and NoSQL databases make it difficult to choose the best paradigm for an organization's needs, and different paradigms within cloud platforms make it even more difficult to move data across different cloud service providers. Khan, et al. [33]'s research examines data portability and interoperability issues across various cloud platforms, with an emphasis on SQL and NoSQL Database software architectures. Performance evaluations of SQL and NoSQL databases show that they are both suitable for OLTP databases and large data analytics, respectively. The report makes recommendations for how to overcome problems with data transfer, including the requirement for unified APIs to improve interoperability and streamline data movement across various cloud service providers. This study, which focuses on technical concerns for SQL and NoSQL Databases, uses the systematic literature review approach to choose pertinent articles. Khan, et al. [33] also offers insights into the proper use scenarios, data analysis, data gathering methods, and results for these databases. This comprehensive assessment of the literature looked at 142 papers on the subject, analyzing the distribution of document types and comparing the performance of SQL and NoSQL Document Databases. The study also investigated DBaaS, and unified APIs approaches for data portability and interoperability, highlighting gaps and pointing out that SQL and NoSQL databases offer distinct advantages in particular scenarios while also addressing difficulties in data movement across cloud platforms through DBaaS architecture and unified APIs frameworks.

Khasawneh, et al. [34]'s study emphasizes the significance of documenting events and knowledge throughout human history by focusing on the development of data storage techniques from ancient methods like cutting on stones to contemporary database systems. Relational databases, or SQL databases, use SQL queries to analyze and retrieve data held inside a specified schema that creates relational links between the data, providing an organized method of data storage. A non-relational database management system includes NoSQL databases, such as key-value, column-oriented, document-oriented, and graph-oriented types. They enable the storage of many data kinds without the need for rigid linkages or a predetermined schema. In situations where on-request analytics and high consistency are crucial, NewSQL systems, a modification of conventional SQL, can also be used as an alternative to NoSQL [34].

Kumar and Mohanavalli [35] examine the advent of document-oriented databases as a rival to conventional relational databases, their flexibility in storing semi-structured data, and their use in NoSQL systems. Performance, scalability, and the significance of choosing the proper database type for applications—particularly streaming ones—are all stressed. MongoDB and CouchDB, two well-known document-oriented NoSQL databases, are thoroughly compared [35]. The collections and documents that make up MongoDB's architecture include dynamic schemas and enable BSON for data representation, indexing, and replication. On the other side, CouchDB prioritizes usability, employing multi-version concurrency control, bidirectional synchronization, Erlang for concurrency, and JSON for documents[35]. In order to analyze these databases in the context of a streaming application, Kumar and Mohanavalli [35] highlighted their advantages and disadvantages for various use cases. Kumar and Mohanavalli [35]'s application involves capturing tweets from Twitter based on specified keywords and storing them as JSON documents in both MongoDB and CouchDB for the purpose of performance comparison and potential sentiment analysis. This study focuses on comparing the widely used document-oriented NoSQL databases MongoDB and CouchDB specifically within the context of a streaming application using NodeJS. For performance analysis, the system architecture shows how streaming data travels through NodeJS to JSON documents kept in the two databases. In the context of streaming applications, Kumar and Mohanavalli [35] MongoDB with CouchDB based on a few qualitative features, such as replication, storage type, CAP features, and MapReduce capabilities. The investigation concludes that MongoDB outperforms CouchDB in terms of performance for streaming applications [35].

Unstructured and semi-structured data are commonly handled by NoSQL databases like MongoDB and Oracle NoSQL, which provide adaptable, schema-less storage suited for a variety of data types [36]. Padhy and Kumaran [36]'s research study compares the insert, remove, and update performance of MongoDB with Oracle NoSQL using a range of dataset sizes. The investigation is done on a single node, however next research might look at performance in clustered settings. The research comprises running time-based insert, remove, and update performance tests on MongoDB and Oracle NoSQL databases. Database performance is assessed using test datasets with a range of sizes (from 1,000 to 100,000 records). As the amount of the dataset grows, MongoDB outperforms Oracle NoSQL in terms of insert operations. Like insert operations, MongoDB exhibits improved performance as dataset size increases. Oracle NoSQL, however, offers competitive performance. When it comes to update operations, MongoDB once more outperforms Oracle NoSQL, especially for larger datasets [36].

The BigDAWG polystore idea is used in Kepner, et al. [37]'s study to merge many databases, and it looks at how well SQL, NoSQL, and NewSQL databases perform in financial transactions, internet search, and data analysis. Presenting the SQL relational model using associative arrays and highlighting important mathematical concepts like associativity, commutativity, distributivism, identities, annihilators, and inverses are the main goals of this paper [37]. Additionally, Kepner, et al. [37]'s study tries to quantify how these characteristics affect the efficiency of associative array operations. Kepner, et al. [37]'s study investigates how the relational paradigm, which has its roots in set theory, underpins SQL databases. It looks at the improvements to boost mathematical rigor as well as the mathematical rigor needed for databases. By describing relations as associative arrays, the paper suggests an associative array model that incorporates SQL, NoSQL, and NewSQL. It explains how relations can be thought of as associative arrays that answer queries on sets, tuples, indices, ordered sets, multisets, and sequences. It also provides other relational operations using associative array algebra. The discovery in Kepner, et al. [37]'s research provides fresh insights into the mathematical features of relations by showing that relations can be represented as associative arrays. It emphasizes how crucial distributivism and associativity are for maximizing these processes and accelerating query execution.

Flexible and scalable infrastructures are required as communication technologies advance and converge more and more, to store and analyze the enormous amounts of data generated everyday by mobile applications and social networks. NoSQL databases have become a substitute for storing this type of data [38]. Araujo, et al. [38]'s study compares the effectiveness of the Cassandra and MongoDB databases through two experiments. Using three workloads to assess performance, the first experiment found that Cassandra performed better. The second experiment evaluates real-time performance by simulating actions on a database of investors from the National Treasury. MongoDB outperforms other databases in most of these activities. The YCSB benchmark and Python notebooks in Google Collab for CRUD operations were used in the studies, which were carried out in both cloud and standalone infrastructures. Araujo, et al. [38] contrasted MongoDB, a schema-less document-based storage, and Cassandra, a columnar-based storage. While MongoDB excelled in the second experiment mimicking real-world operations on a database of investors from the National Treasury, Cassandra outperformed it in the first trial using the YCSB tool. With the exception of updates, where Cassandra performed better, MongoDB showed to be more efficient for a number of operations [38].

Mahmood, et al. [39]'s study highlights the necessity for effective data processing in applications like power plants, smart cities, and transportation systems by comparing the performance of MongoDB, Cassandra, and Redis in large-scale sensor log analysis in actual hydraulic power systems. In order to provide insights for IIoT applications, Mahmood, et al. [39] examines three central NoSQL data stores for large-scale sensor log analysis, concentrating on key lookup and range query operations. On more than 100 million sensor log data from hydraulic power systems, key lookup and range query operations are executed as part of the performance evaluation. Mahmood, et al. [39] compare three NoSQL data stores: Redis (distributed key-value store), Cassandra (column store), and MongoDB (document store). Range queries are used to get records with measurements that deviate from predicted values, whereas key lookup queries are used to locate specific sensor log entries. In these data stores, Mahmood, et al. [39]'s study makes use of secondary ordered indexes and composite primary keys to enhance query performance. On a cluster of 64-CPU servers with 384GB of RAM, performance tests are run. All three NoSQL data stores perform comparable for the key lookup query, with query execution times typically within 10 milliseconds. Redis outperforms MongoDB and Cassandra in the range query because of its in-memory architecture, whereas MongoDB performs better than Cassandra mostly because its B-tree index frequently sits in memory, minimizing disc access [39]. The study offers insightful information on the benefits and limitations of various NoSQL data stores for handling fundamental queries in complex IoT applications.

To compare effective data storage and administration in a small-scale Internet of Things application, namely a sprinkler system, Rautmare and Bhalerao [40]'s study compares standard SQL and NoSQL databases, specifically MySQL and MongoDB. The main goal is to determine whether NoSQL databases, notably MongoDB, perform better than SQL databases like MySQL in various IoT application scenarios [40]. The study measures response times for various queries, such SELECT and INSERT, while considering elements like the quantity of threads and records. In this study, Rautmare and Bhalerao [40] deployed a small-scale Internet of Things (IoT) application for a sprinkler system and makes use of sensors to gather temperature, humidity, and soil moisture data. Both MySQL and MongoDB databases hold this data. Response times for various queries executed with various numbers of threads and records are used as the basis for the performance comparison. Apache JMeter is used in the study for load testing and performance evaluation. Interesting results of the investigation are presented. Initially, MongoDB performs comparably to MySQL for SELECT queries, but as the number of threads rises, response times increase, possibly owing to system overload [40]. MySQL, in comparison, keeps its stability. MongoDB outperforms MySQL for INSERT queries, showing quicker response times. However, due to problems like write queue overflow and database locking, MongoDB displays fast response time changes during INSERT operations. In these cases, MySQL's replies are more consistent. The report advises that while deciding between SQL and NoSQL databases for IoT applications, query types and application needs should be considered [40].

CHAPTER 3

STUDY A – JavaScript Frameworks: A Comparative Study Between React.js And Angular.js

3.1 INTRODUCTION

A framework is the technology that offers a software engineer a defined code structure to build applications as fast and completely as possible. Advancing technologies continuously demand advancing functionalities, hence the need for evolving frameworks. Software development frameworks have indistinguishable functionalities but each framework has characteristic features that make them distinct. However, when selecting the framework to use for a project, it is important to consider some key measures such as the technicality of the framework, the overall goal of the project or system, and the developer's skills.

Frameworks facilitate productivity and fast development via reusable codes and design. However, a situation of overhead may occur with regard to development. We propose to embark on research that involves a comparative study of two JavaScript frameworks namely: React.JS and Angular.js. In the course of the study, we will perform a comparative analysis of most attributes of the selected framework. JavaScript allows for fully building an application in it on the client side while the data interface is done on the server side and vice versa. The flexibility of JavaScript gives room to developers from diverse backgrounds of software designs to build distinct JavaScript frameworks. The study would include the comparison of the selected frameworks leveraging on the Document Object Model (DOM) performance as the basis of the study to determine which one of the frameworks is best to be considered for software projects and inferring the strengths and weaknesses of each one of them.

The constant updates in technology require constant updates in the tools used to develop applications needed to access the updates. Since the conception of frameworks, a great shift has been recorded as most developers move from creating web applications using the native stack to the use of frameworks: a sophisticated tool that offers predefined design and code reuse to enhance the development of fast and complete applications. Our aim is to proffer a fair decision-making process as to which framework to choose for specific projects based on our analyzed measures and results obtained from our deep analysis and observations taken from running applications developed using the frameworks selected. The proposed research intends to deliver and outline use cases of the selected JavaScript frameworks, React and Angular. In order to analyze the critical features of the selected frameworks, we will develop two similar web applications, one using React.JS and the other using Angular, and carefully observe and record some critical measures while both applications run on selected operating systems and machines.

Our research follows a thorough process to observe the selected framework while we implement similar applications developed in both frameworks and comparatively assess them. With this, we would be able to proffer a reliable choice of framework to software engineers while embarking on a project in real-life production. The comparative study in our research would be a procedural paradigm from top-level analysis to architectural analysis, ensuring ease of digestion of the process for future researchers.

3.2 BACKGROUND

It is a known fact that some time ago, the front end in the world of application development was meant to majorly function as the building block of the application where the contents of the page were being displayed. However, with the birth of a new technology era where millions of devices ranging from tablets, mobile, and phones to other IoT devices, are now being used to access websites, there comes a rise in the need for cross-platform, easy-to-use, and reusable frameworks to facilitate the efficiency of application development it is important to identify the support of ECMA standards on different modern browsers as this has a significant effect on how JavaScript (or its frameworks) functions.

In order to proffer remedies to challenges pertaining to front-end frameworks in the aspect of web development, several research has been conducted and this research won't be an exception. However, the drive behind this research focuses on the fact that most developers find it difficult in choosing the right framework for their project. At the end of this research, we are confident to proffer an efficient solution to the issue of choosing the right framework leveraging on our results from the analysis of critical metrics and other functionalities and factors of our framework of interest in this research.

3.3 METHOD

The purpose of this study is to compare the two most popular JavaScript frameworks, React.js and Angular.js, and to make the decision-making process on which frontend framework to use easy for software engineers and/or companies while embarking on a project. This approach allowed for in-depth insights into the framework functionalities, performance, and efficiency, as well as other advantages that any selected framework has to offer.

As the focus of this study is to comparatively analyze the performance of two different web application frameworks on the same server, we acknowledged the fact that the network capturing style is widely in use to gather the necessary dataset for frameworks. However, it is a de facto that network data capturing analysis for web applications is characterized by few drawbacks as compared with native applications in the sense that on a mac/Linux and windows operating system, a loopback network adapter can be used to log all pertinent packets of required data restricting all unwanted background network traffic.

This part of our paper will elaborate on the performance of each selected framework, with the aim of giving fairly advice to developers on the choice of framework to choose for projects in the real world. Our research is based on two aspects, one involving the development of two similar web applications using the selected frameworks, then comparing the performance of these applications based on some selected measures to uncover the pros and cons of these frameworks, and what kind of project each framework is suitable for. Both applications would be tested on localhost or a local server and connected to a MySQL database engine from which data would be fetched. Also, we would dynamically load rows into a table with the click of a button. The table below shows a high level of specifications of the environments in our application that would be tested in.

Table 3.1: Parameters and values of environment

Environmental (VM) Parameters	Values
VM RAM size	3GB
VM HDD size	100GB
Host Machine Brand	Windows
VM Operating System	Windows

We would ensure to mimic the real-life project following the universal, representative, and instructive standards for development and production. This project would feature a networked moderately sized project. Furthermore, our analysis would involve a benchmark test that would estimate DOM (Document Object Model) operation individually, as well as a deep-dive study of the architectural layer to explore reasonable facts while drawing our conclusions.

Eventually, our work will assess the two experiments we mentioned above to observe if both validate each other as regards the outcome obtained. A convincing conclusion can be inferred if our results correlate in a consistent manner. However, if the outcome of our experiments contradicts, this is a red flag as it depicts a flaw in our experiment. We have identified the importance of convincing results, hence setting up dual experiments to absolutely ascertain standard conclusions. Inferring conclusions based on DOM operation-based benchmarking would be subject to less accuracy when juxtaposed with a complex web application, such as the amazon e-commerce website [1]. Also, using a real-world web application might impose certain network errors which can restrict us from getting the right metric readings for our analysis. Thus, our two proposed experiments are a good approach as it gives us a standard simulation to verify and complement both applications. We also recognize the fact that the reliability is open to improvement from future researchers.

In this chapter, we would first explain what framework means generally. Then, we would go ahead to discuss the selected frameworks for this study. Next, we would dive into the ecosystem in which our experiment was performed which is a virtual machine and its specifications. Other important thing discussed in this chapter includes the implementation of two similar web applications developed differently in React.js and Angular.js, testing procedures to obtain performance metrics, as well a comparison of the frameworks of study.

3.3.1 Framework

In layman's world, the framework is a generic term used to define a supporting structure on top of which other things are built on. However, in the field of Software Engineering, framework (or software framework) is an abstraction in which software, offering generic functionalities, can be selectively changed by additional user-written code, hence providing application-specific software. Also, offers a standard way to develop and deploy applications leveraging a universal, reusable software environment that offers specific functionality as part of a larger software platform to speed up software application development, solutions, and products. Some components pertinent to a software framework include compilers, code libraries, support programs, toolbox/set, and APIs (Application Programming Interface) which help bond different components to enable the full development of a system.

3.3.1.1 Web Application Framework (WAF)

A Web Application Framework is a software framework developed to provide support for web application development. This framework is embedded with components such as web services, web resources, and web APIs, as well as providing a standard way of building and deploying web applications into the World Wide Web (WWW).

3.3.1.2 React.js

The React.js framework was developed at Facebook as an internal project and was converted to open-source in May 2013 to allow contributions from JavaScript developers worldwide. As React.js is a framework that consists of JavaScript libraries for developing user interfaces, it has the ability to unite multiple independent code pieces into some complex user interface. The framework is characterized by cross-browser compatibilities, as well as high performance and simple code logic. With the React.js framework, developers have the ability to directly combine user interfaces with components such as buttons, dialogs, and many more which results in rich interactive webpages. Also, the introduction of 'JavaScript XML' popularly called JSX syntax in creating the user interface simplifies the reusability of components while ensuring clarity in the internal structure of the components. In the components, the React.js framework uses the DOM rendering capabilities in the browser to differentiate code from real target on the webpage thereby making the development of mobile applications easier.

3.3.2 Angular.js

The Angular.js framework was developed by Misko Hevery in 2009. Primarily, the framework is used in developing single-page applications (SPAs) with the CRUD (Create Read Update Delete) functionalities, leveraging its dynamic page design features. Angular.js is characterized by a lot of features ranging from MVW (Model-View-Whatever: - ability of a framework to use any of Model-View-Controller or Model-View-View-Model approaches) which is considered the core feature, modularity, automated bi-directional data binding, semantic tagging, to dependency injection. In the Angular.js framework, HTML is extended with directives, and data binding to HTML is done via expressions. Its directives can be utilized in HTML in the following ways:

- New HTML elements
- HTML element's attribute
- HTML element classes
- HTML element comments

The Angular.js framework is pronounced a complete framework considering its embedded features such as templates, two-way data binding, routing, modular services, filters, and dependency injection.

3.3.3 Virtual Machine (VM)

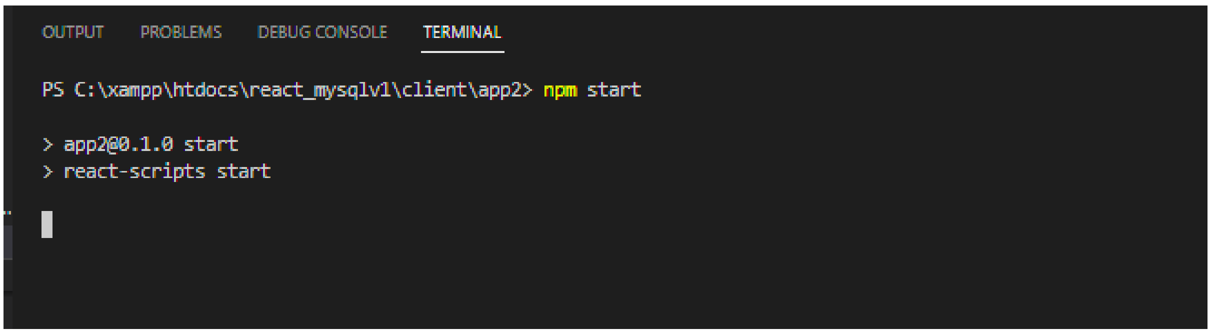
In this project, to ensure enough isolation of the system, we leverage virtual machine functionalities. A Virtual Machine, or VM for short, is defined as a compute resource that uses software in place of a physical computer for program execution and application deployment. These VM's can also be deployed to online cloud platforms such as Microsoft Azure, Google Cloud Platform(GCP), and Amazon Web Service(AWS)[41]. One or more VMs (sometimes referred to as the guest) is able to fully function or run on a physical machine (sometimes referred to as the host), where each VM runs its unique OS (Operating System) and functions separately from other VMs on the same host. The use of a VM in our project enabled us to exercise a platform-independent development as a VM was VMWare Fusion with Windows OS was installed on a MAC OS where all development processes took place. The VM was then exported to evaluate performance on two other host PCs.

3.3.4 Similar Web App Implementation

As the objective of this research project was to measure and compare the performance of two JavaScript frontend web frameworks (React.js and Angular.js), we have created two web applications, one using React.js, and the other using Angular.js. These two applications have similar interfaces as well performing similar tasks. The tasks performed by the web applications developed includes statically fetching 20 rows of data from an SQL database and presenting it in a jQuery Datatable, dynamically adding 1000 rows to a jQuery Datatable on the click of a button.

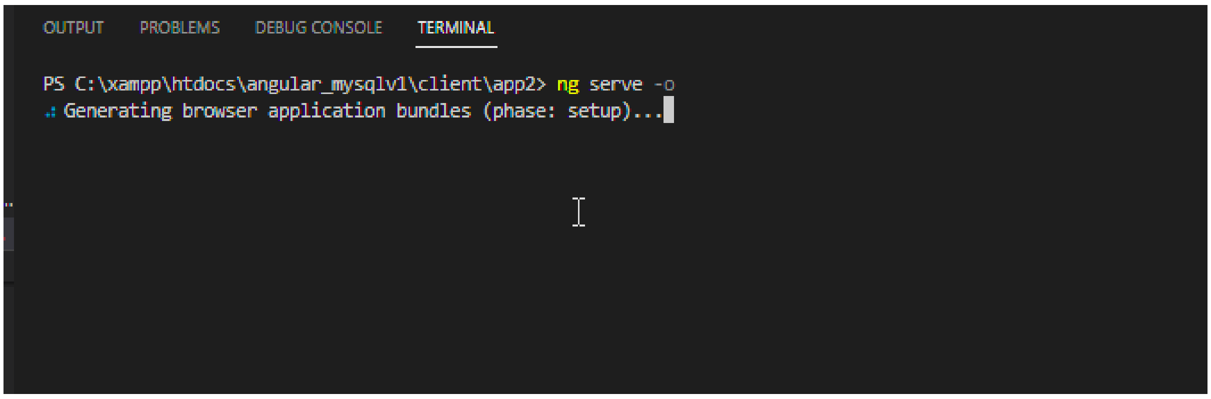
3.3.5 Testing Procedures

The commands used to start and open the applications in a web browser (Google Chrome, in our case) are 'npm start' (for react.js) and 'ng serve -o' (for angular.js). Once these commands are entered in the terminal and we hit the 'enter' button, the app begins to build and eventually opens in a web browser.



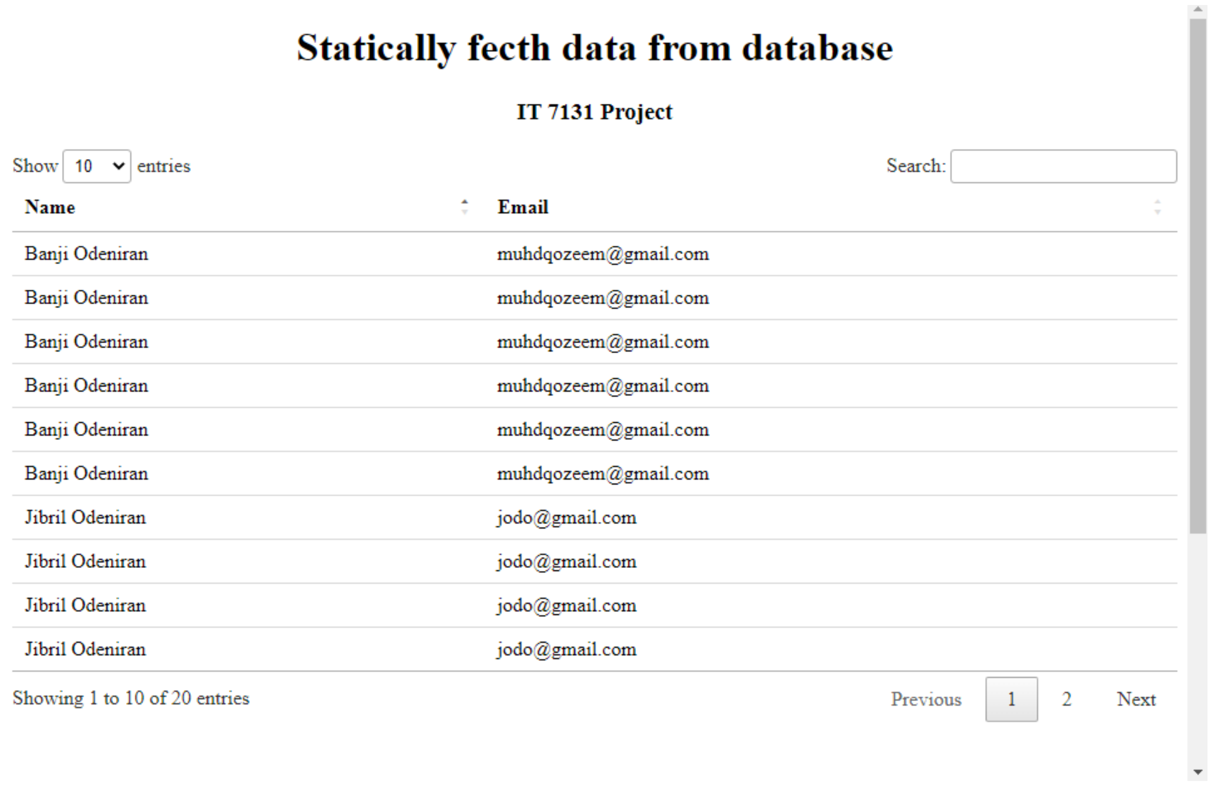
```
PS C:\xampp\htdocs\react_mysqlv1\client\app2> npm start

> app2@0.1.0 start
> react-scripts start
```

Figure 3.1: Command to run React.js App from terminal



```
PS C:\xampp\htdocs\angular_mysqlv1\client\app2> ng serve -o
.: Generating browser application bundles (phase: setup)...
```

Figure 3.2: Command to run Angular.js App from terminal



Statically fecth data from database

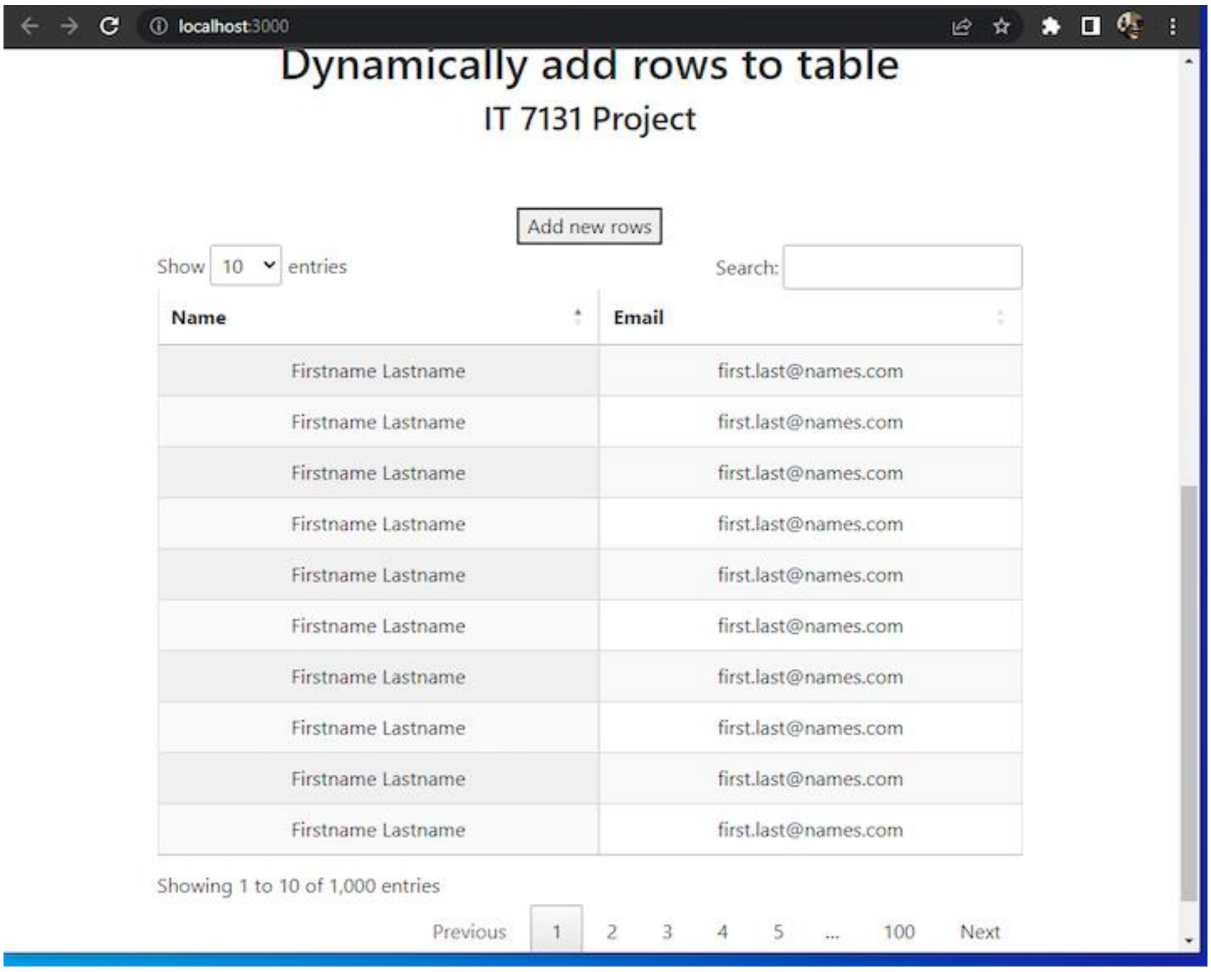
IT 7131 Project

Show 10 entries   Search:

Name	Email
Banji Odeniran	muhdqozeem@gmail.com
Banji Odeniran	muhdqozeem@gmail.com
Banji Odeniran	muhdqozeem@gmail.com
Banji Odeniran	muhdqozeem@gmail.com
Banji Odeniran	muhdqozeem@gmail.com
Banji Odeniran	muhdqozeem@gmail.com
Jibril Odeniran	jodo@gmail.com
Jibril Odeniran	jodo@gmail.com
Jibril Odeniran	jodo@gmail.com
Jibril Odeniran	jodo@gmail.com

Showing 1 to 10 of 20 entries

Figure 3.3: Resulting Application with 20 rows of data loaded from MySQL database



Dynamically add rows to table

IT 7131 Project

Show 10 entries　　Search:

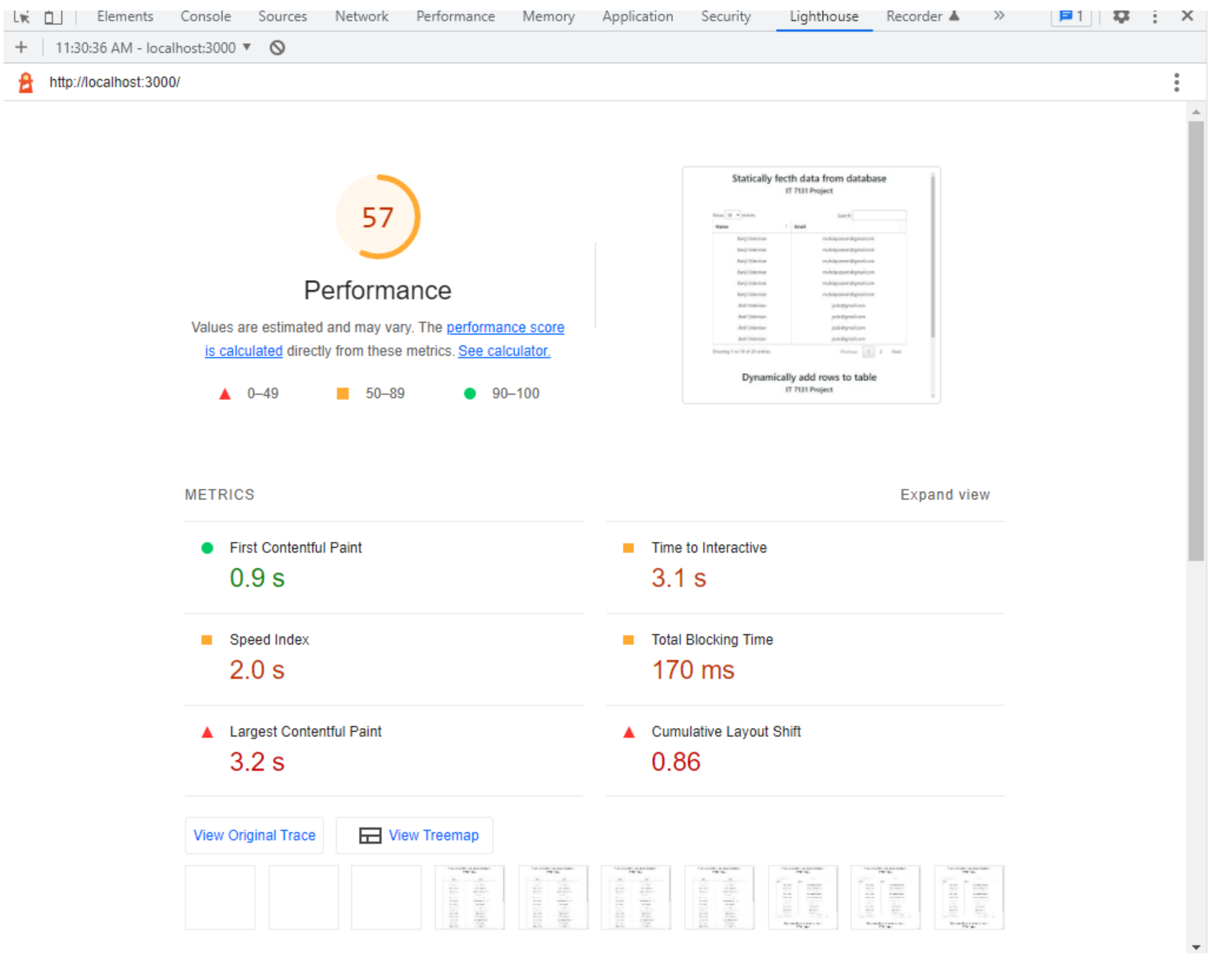
Name	Email
Firstname Lastname	first.last@names.com
Firstname Lastname	first.last@names.com
Firstname Lastname	first.last@names.com
Firstname Lastname	first.last@names.com
Firstname Lastname	first.last@names.com
Firstname Lastname	first.last@names.com
Firstname Lastname	first.last@names.com
Firstname Lastname	first.last@names.com
Firstname Lastname	first.last@names.com
Firstname Lastname	first.last@names.com

Showing 1 to 10 of 1,000 entries

Figure 3.4: Generating 1000 paginated rows on button click

3.3.6 Lighthouse

Next is to check for performance metrics by navigating to lighthouse following the clicks: from google chrome, click the three vertical dots on the top right corner => more tools => developer tools => lighthouse. Set 'mode' to 'Navigation(default)', 'device' to 'Desktop', and check 'Performance' under 'categories. Finally, hit the Analyze page load button to get the performance metrics.



57

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 ■ 50–89 ● 90–100

METRICS Expand view

● First Contentful Paint 0.9 s

■ Time to Interactive 3.1 s

■ Speed Index 2.0 s

■ Total Blocking Time 170 ms

▲ Largest Contentful Paint 3.2 s

▲ Cumulative Layout Shift 0.86

Statically fecth data from database
IT 7131 Project

Dynamically add rows to table
IT 7131 Project

Figure 3.5: Performance metrics from Lighthouse

First Contentful Paint *(FCP) – Records the time taken to paint the first text or image.*

3.3.6.1 **Time to Interactive** (TI) – Gives the time taken for the page to become fully interactive

3.3.6.2 **Speed Index** (SI) – Shows how fast content becomes populated and seen on the web page

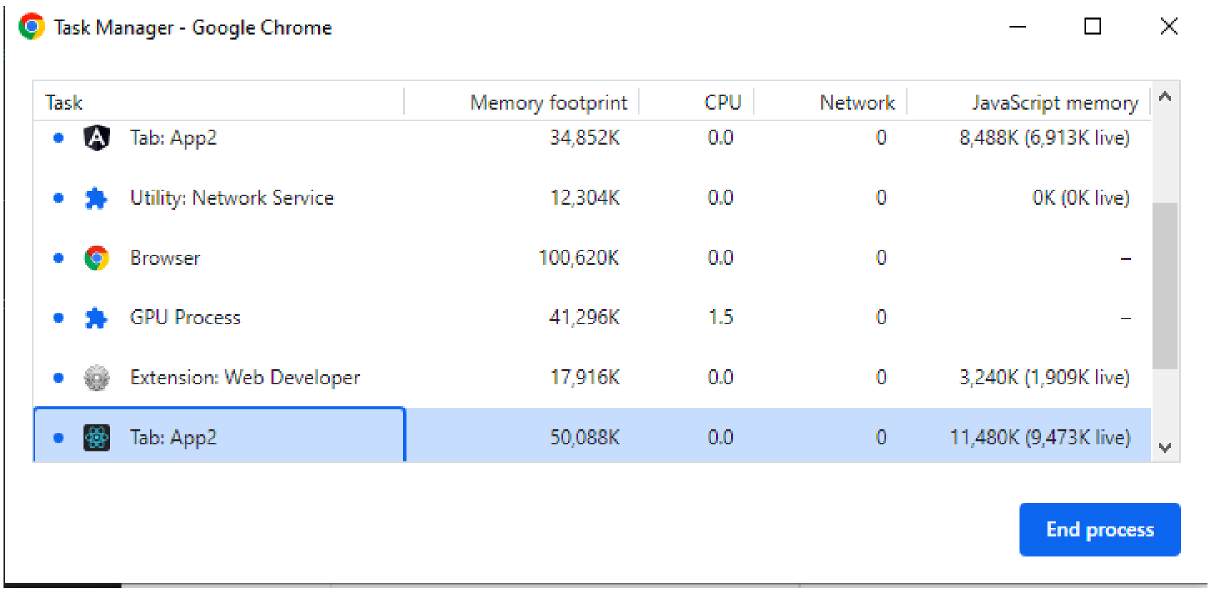
3.3.6.3 **Total Blocking Time** (TBT) – Gives the time period between TI and FCP if activity's length is greater than 50ms.

3.3.6.4 **Largest Contentful Paint** (LCP) – Records time taken to paint the largest text or image

3.3.6.5 **Cumulative Layout Shift** (CLS) – computes the movement of visible elements within viewport

3.3.7 Task Manager (Google Chrome)

Also, we explored google chrome's task manager to obtain further performance metrics and memory usage of the selected frameworks. Below is a screenshot of google chrome's task manager while the web apps were executed. We chose to use google chrome's task manager over windows' task manager as it offers a high level of isolation and gives reports in a cool GUI that was easily captured.



Task	Memory footprint	CPU	Network	JavaScript memory
Tab: App2	34,852K	0.0	0	8,488K (6,913K live)
Utility: Network Service	12,304K	0.0	0	0K (0K live)
Browser	100,620K	0.0	0	–
GPU Process	41,296K	1.5	0	–
Extension: Web Developer	17,916K	0.0	0	3,240K (1,909K live)
Tab: App2	50,088K	0.0	0	11,480K (9,473K live)

Figure 3.6: Google Chrome Task Manager

3.3.7.1 Memory Footprint (MF) – Displays the amount of RAM each process is currently using

3.3.7.2 CPU – Shows the amount of CPU power a process is taking up.

3.3.7.3 Network – Measures the amount of data a process uses in operation

3.3.7.4 JavaScript Memory (JM) – Represents the JavaScript Heap. In this column, we are interested in the value in parenthesis which shows the amount of memory used by reachable objects on the web page. This value is correlated to a number of objects in the project.

3.4 RESULT

The benchmark performance metrics obtained during the extensive test of the applications developed using the React.js and Angular.js frameworks are presented below. The specification and configuration settings of the Virtual Machine (VM) environment used in development are given in the table below:

Table 3.2: Virtual Machine (VM) Configuration and Specification Settings

Device	Summary
Memory	3GB
Processors	2
Hard Disk (NVMe)	100GB
CD / DVD SATA	Auto detect
Network Adapter	NAT

Short explanations and implications of the items in the table above are given:

3GB Memory – This implies that the VM has a total of three (3) gigabytes of Random Access Memory (RAM) which is used by the VM to store and run all its tasks including the operating system and all of its applications.

Two (2) Processors – This indicates that the VM has two (2) virtual processors which in turn results to the increased processing power of the VM, such that multiple tasks can be run simultaneously on the VM.

100GB Hard Disk (NVMe) – This means that the VM has a 100GB non-volatile data storage that holds and maintains data and files when the VM is turned off. However, NVMe (Non-Volatile Memory express) increases the speed of data transfer between enterprise and client systems and solid-state drives over a computer's high-speed Peripheral Component Interconnect Express bus.

Auto Detect CD/DVD SATA – This helps to automatically detect the Serial Advanced Technology Attachment (SATA) which is responsible for how data is transferred between the VM and mass storage devices.

NAT Network Adapter – This facilitates the transport of network data between the virtual machine and the external network. The VM NAT device identifies incoming data packets that are intended for the virtual machine and sends them to the correct destination.

Also, the specifications of the Host Machines that were utilized during testing were given in the respectively attached screenshots. The meaning of all terminologies present in the performance metrics screenshots have been comprehensively explained in the Method section of this paper.

In the tables below, we present the overall performance of both React.js and Angular.js while the developed web applications were executed on two different host machines. The specifications of the host machines were included in each screenshot. We have performed the execution on different days to ensure we obtain metrics that are fit for further analysis, as well as make conclusion.

Table 3.3: Day-1 React.js Performance Readings (A)

Wimlab: Host 1 (Memory: 65GB | OS: Windows)	
Lighthouse	
performance (%)	61
first contentful paint (seconds)	0.8
time to interactive (seconds)	3.4
speed index (seconds)	2

Table 3.4: Day-1 React.js Performance Readings (B)

GA: Host 2 (Memory: 16GB | OS: Windows)	
Lighthouse	
performance (%)	54
first contentful paint (seconds)	0.9
time to interactive (seconds)	3.5
speed index (seconds)	2

total blocking time (meters/seconds)	70
largest contentful paint (seconds)	3.3
cumulative layout shift	0.874

total blocking time (meters/seconds)	200
largest contentful paint (seconds)	3.3
cumulative layout shift	0.605

Table 3.5: Day-1 React.js Performance Readings (C)

Chrome Task Manager (page load with 20 rows)	
memory footprint (K)	51472
CPU	20.3
network (B/s)	1940
JavaScript memory (K)	12551

Table 3.6: Day-1 React.js Performance Readings

Chrome Task Manager (page load with 20 rows)	
memory footprint (K)	23924
CPU	52.6
network (B/s)	1030
JavaScript memory (K)	3496

Table 3.7: Day-1 React.js Performance Readings (E)

Chrome Task Manager (add 1000 rows)	
memory footprint	117956
CPU	103.3
network	0
JavaScript memory	11406

Table 3.8: Day-1 React.js Performance Readings (F)

Chrome Task Manager (add 1000 rows)	
memory footprint	116604
CPU	105.5
network	0
JavaScript memory	12483

Table 3.9: Day-1 Angular.js Performance Readings (A)

Wimlab: Host 1 (Memory: 65GB | OS: Windows)	
Lighthouse	
performance (%)	48
first contentful paint (seconds)	4.9
time to interactive (seconds)	5.7
speed index (seconds)	4.9
total blocking time (meters/seconds)	20
largest contentful paint (seconds)	5.9
cumulative layout shift	0.081

Table 3.10: Day-1 Angular.js Performance Readings (B)

GA: Host 2 (Memory: 16GB | OS: Windows)	
Lighthouse	
performance (%)	49
first contentful paint (seconds)	4.9
time to interactive (seconds)	5.9
speed index (seconds)	4.9
total blocking time (meters/seconds)	10
largest contentful paint (seconds)	6.1
cumulative layout shift	0.06

Table 3.11: Day-1 Angular.js Performance Readings (C)

Chrome Task Manager (page load with 20 rows)	
memory footprint (K)	50048
CPU	3.1
network (KB/s)	8.9
JavaScript memory (K)	10857

Table 3.12: Day-1 Angular.js Performance Readings (D)

Chrome Task Manager (page load with 20 rows)	
memory footprint (K)	15244
CPU	21.4
network (KB/s)	8.9
JavaScript memory (K)	7041

Table 3.13: Day-1 Angular.js Performance Readings (E)

Chrome Task Manager (add 1000 rows)	
memory footprint (K)	52236
CPU	11
network (KB/s)	0
JavaScript memory (K)	7045

Table 3.14: Day-1 Angular.js Performance Readings (F)

Chrome Task Manager (add 1000 rows)	
memory footprint (K)	50312
CPU	9.2
network (KB/s)	0
JavaScript memory (K)	7171

Table 3.15: Day-2 React.js Performance Readings (A)

Wimlab: Host 1 (Memory: 65GB | OS: Windows)	
Lighthouse	
performance (%)	60
first contentful paint (seconds)	0.8
time to interactive (seconds)	3.4
speed index (seconds)	1.8
total blocking time (meters/seconds)	110
largest contentful paint (seconds)	3.2
cumulative layout shift	0.898

Table 3.16: Day-2 React.js Performance Readings (B)

GA: Host 2 (Memory: 16GB | OS: Windows)	
Lighthouse	
performance (%)	54
first contentful paint (seconds)	0.9
time to interactive (seconds)	3.5
speed index (seconds)	2
total blocking time (meters/seconds)	200
largest contentful paint (seconds)	3.3
cumulative layout shift	0.605

Table 3.17: Day-2 React.js Performance Readings (C)

Chrome Task Manager (page load with 20 rows)	
memory footprint (K)	74416
cpu	80.4
network (B/s)	1940
JavaScript memory (K)	18236

Table 3.18: Day-2 React.js Performance Readings (D)

Chrome Task Manager (page load with 20 rows)	
memory footprint (K)	23924
cpu	52.6
network (B/s)	1030
JavaScript memory (K)	3496

Table 3.19: Day-2 React.js Performance Readings (E)

Chrome Task Manager (add 1000 rows)	
memory footprint	141732
cpu	113.5
network	0
JavaScript memory	17832

Table 3.20: Day-2 React.js Performance Readings (F)

Chrome Task Manager (add 1000 rows)	
memory footprint	116624
cpu	118.5
network	0
JavaScript memory	12483

Table 3.21: Day-2 Angular.js Performance Readings (A)

Wimlab: Host 1 (Memory: 65GB | OS: Windows)
Lighthouse

Table 3.22: Day-2 Angular.js Performance Readings (B)

GA: Host 2 (Memory: 16GB | OS: Windows)
Lighthouse

performance (%)	46
first contentful paint (seconds)	4.9
time to interactive (seconds)	5.8
speed index (seconds)	4.9
total blocking time (meters/seconds)	0
largest contentful paint (seconds)	6
cumulative layout shift	0.154

performance (%)	49
first contentful paint (seconds)	4.9
time to interactive (seconds)	5.9
speed index (seconds)	4.9
total blocking time (meters/seconds)	10
largest contentful paint (seconds)	6.1
cumulative layout shift	0.06

Table 3.23: Day-2 Angular.js Performance Readings (C)

Chrome Task Manager (page load with 20 rows)	
memory footprint (K)	47676
CPU	17.2
network (KB/s)	8.9
JavaScript memory (K)	7026

Table 3.24: Day-2 Angular.js Performance Readings (D)

Chrome Task Manager (page load with 20 rows)	
memory footprint (K)	15244
CPU	21.4
network (KB/s)	8.9
JavaScript memory (K)	7041

Table 3.25: Day-2 Angular.js Performance Readings (E)

Chrome Task Manager (add 1000 rows)	
memory footprint (K)	53044

Table 3.26: Day-2 Angular.js Performance Readings (F)

Chrome Task Manager (add 1000 rows)	
memory footprint (K)	50312

CPU	7.8
network (KB/s)	0
JavaScript memory (K)	7161

CPU	9.2
network (KB/s)	0
JavaScript memory (K)	7171

In Table 3.27, we have calculated the average of the overall performance metrics recorded in the two days of our testing. The result, as displayed below shows that the React.js framework has an average performance score of 57%, while the Angular.js framework has an average performance score of 48%.

Table 3.27: Table showing the average of the overall performance of React.js and Angular.js

Day 1	React	Angular
Wimmlab	61	48
GA	54	49
Day 2		
Wimmlab	60	46
GA	54	49
Average	57	48

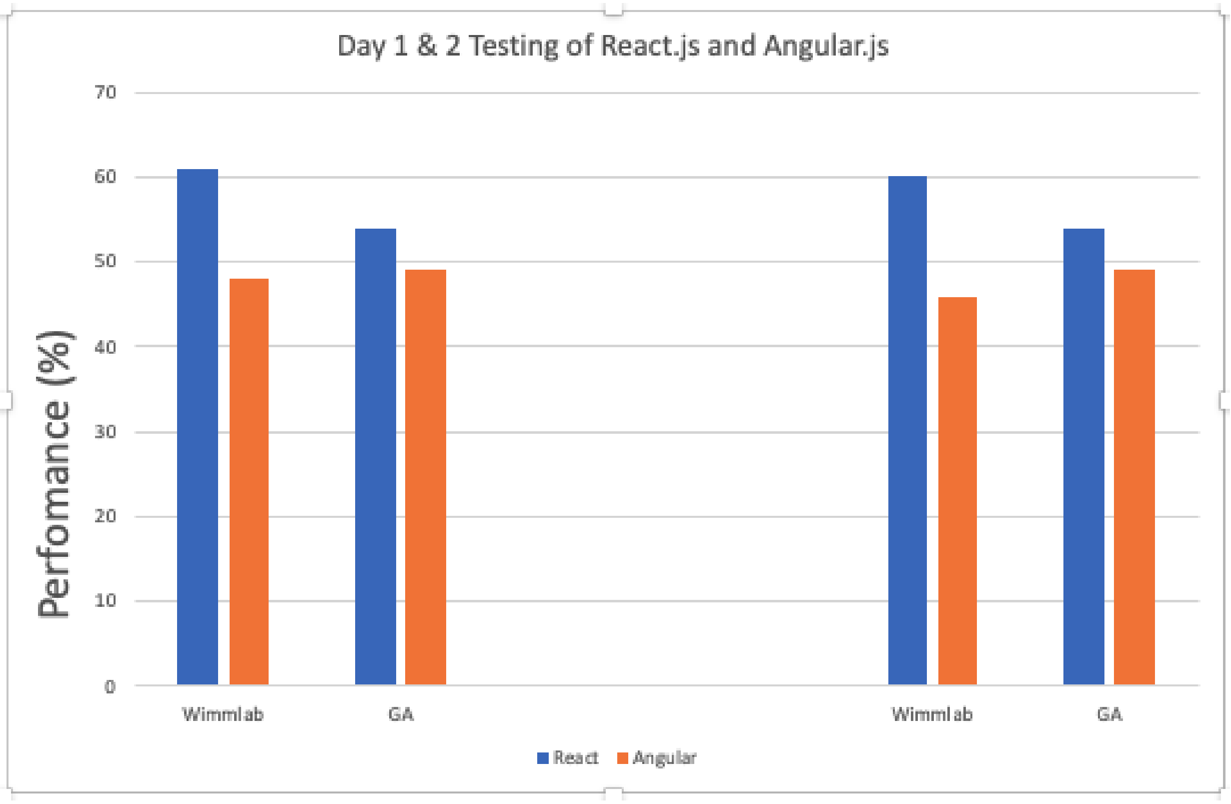


Figure 3.7: Column graph of day 1 & 2 performance of react.js and angular.js frameworks

Finally, we present our result in a graphical format below to ensure a quick grasp of the trend between our framework of interest.

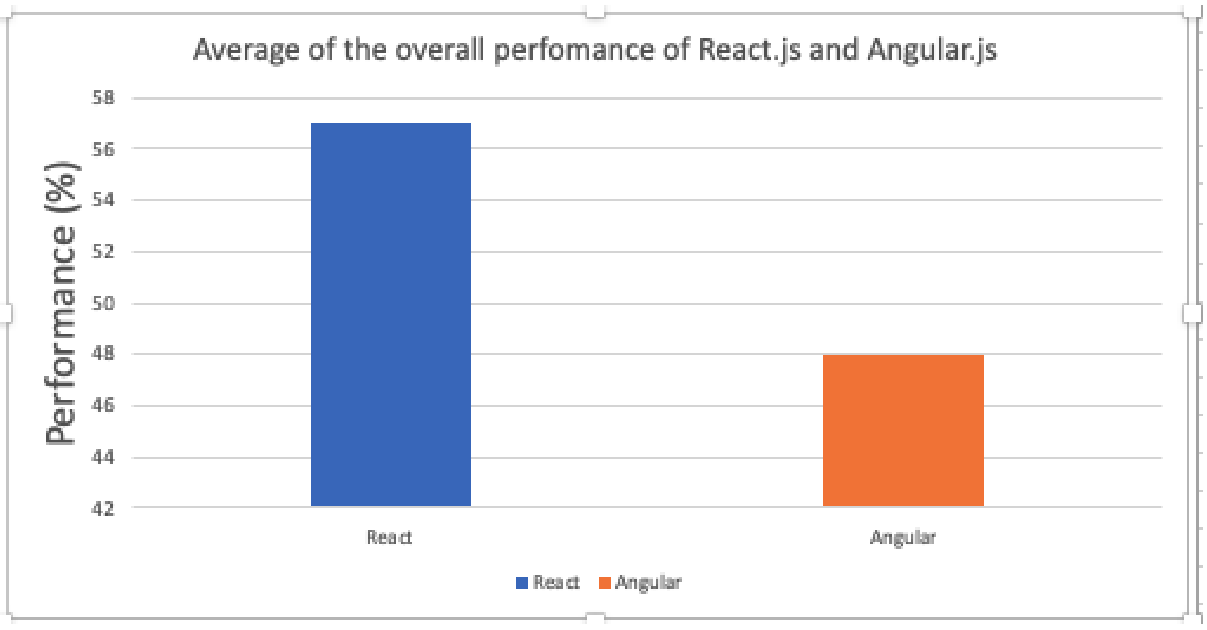


Figure 3.8: Column graph of average performance of React.js and Angular.js

3.5 DISCUSSION

Currently, there are three (3) significant frontend framework which is:

- React.js
- Angular.js
- Vue.js

These frameworks each have their characteristics and features that makes each one of them unique and best for a particular type of a project. However, software engineers need to be aware of some basic features of a framework before jumping to a conclusion in the decision-making stage. Hence, our objective for this research is. In our study, we focused on two most popular JavaScript frameworks (React.js and Angular.js) and we were able to develop from scratch, two similar and simple web applications each with these frameworks. Selected benchmark features of these applications have been extensively tested and performance metrics have been recorded and analyzed. From our qualitative and quantitative analysis, we have pinpointed some top features of each framework and their technical implications, which would in turn help software engineers in the decision-making process of frontend framework selection for a project. Also, to ensure that a production and well-isolated environment is modeled, we have created a standard Virtual Machine (VM) environment to simulate the real-life scenario where our developed applications were tested. Our applications have been extensively run and tested on different machines to make sure a fair conclusion on the performance of both frameworks.

3.6 CONCLUSION

In our research study, we have been able to carry out a comparative analysis of the two most-used JavaScript frontend frameworks, React.js and Angular.js. From our work, we uncover many insights regarding the overall performance of these frameworks by developing basic similar web applications with each of the frameworks. We took further steps to measure benchmark performance metrics leveraging the lighthouse and task manager, which are all part of the google chrome browser. The test environments were isolated enough to simulate the real-life world-wide web. The results show that, although both frameworks have their unique significance with respect to the kind of project it's being used for, React.js still outperforms Angular.js in the overall performance scoring based on our analysis shown in the tables and graphs above. In future works, we intend to improve on these web applications and also consider more robust performance metrics to ascertain more rigid conclusions regarding these selected frameworks.

CHAPTER 4

STUDY B – PHP vs. Node.js: Determining the Better Backend Scripting Language

4.1 INTRODUCTION

Node.js and PHP are two popular backend scripting technologies adopted in building web applications. As both technologies are characterized with unique features, understanding their performance characteristics is important, so that we can determine which language is best suited for a particular type of project.

PHP is a general-purpose scripting language that works well with web development and is mainly used for dynamic webpages, while Node.js developed on chrome's JavaScript runtime for developing fast and scalable network apps [19].

Node.js adopts an event-driven, non-blocking I/O style which efficient handling of large concurrent connections. Whenever a request is received by node.js server, the event is placed on a queue while node.js event loop picks up the event and handles it asynchronously. If this event requires blocking I/O or computation, node.js offloads the task to a separate thread from the internal thread pool in order to ensure that event loop is not blocked, then handle the request once it is completed. After processing the request, it is sent back to the event queue by node.js where the event loo pick it up and handle it. This makes node.js to simultaneously handle multiple requests without event-loop blockage or performance issue. On the other hand, when PHP executes, a PHP interpreter processes it on the server-side which generates a Hypertext Markup Language (HTML), or other output that can be sent to web browser's client [16].

In our research study, we compared both PHP and Node.js in terms of performance using sorting algorithms and a heap algorithm to generate all possible permutations. Our goal is to provide a comprehensive analysis of the performance of these two backend scripting technologies in handling complex algorithms and processing large data. The statistical method we used in analyzing obtained data is T-Test. This statistical method is best used for testing for a significant difference between two given samples and the result obtained from this process is known as t-statistics [42]. We hope to use this research study to provide deep insights that can help organizations and software engineers make a well-informed choice on which backend scripting technologies to adopt for their web development projects.

4.2 BACKGROUND

4.2.1 Sorting Algorithms

Sorting algorithms are procedures for sorting items according to a specific order. Unsorted items are reordered based on a specific criterion, such as alphabetical or numerical. Sorting algorithms are classified based on their space and time complexities, stability, comparison or non-comparison-based nature. To our research study, we have picked only three of the commonly and most efficient used sorting algorithms, namely:

- Binary Sort – is a simple sorting algorithm that works by comparing each element with the elements that precede it and swapping them if they are in the wrong order. If the array is in a sorted order, binary sort has a O(n2) worst-case time complexity and a best-case time complexity of O(n log n). Binary sort is advantageous in situations when the input data is mostly sorted or for tiny lists.
- Bubble Sort – This algorithm repeatedly swaps adjacent elements in a list until the entire list is sorted. It has a worst-case time complexity of O(n^2) and a best-case time complexity of O(n) when the input data is already sorted. Bubble sort is not very efficient and is generally only used for small lists.
- Quick Sort – This algorithm is a divide-and-conquer algorithm that works by selecting a "pivot" element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. The sub-arrays are then sorted recursively. It has a worst-case time complexity of O(n^2) but typically performs much better, with an average time complexity of O(n log n). Quick sort is widely used and is often the preferred algorithm for large data sets.

In addition to the above-mentioned sorting algorithms, we were keen to explore the use of another special type of algorithm ‘Heap Algorithm’ to generate all possible permutations of a list of elements. It works by recursively generating permutations by swapping elements in the list and then calling itself on the remaining elements.

- Heap Algorithm – The heap algorithm is a recursive algorithm that generates all possible permutations of a given set. It works by swapping elements in the set and recursively generating permutations of the remaining elements. It has a worst-case time complexity of O(n!), where n is the number of elements in the list. Heap algorithm for permutation is useful for cases where all possible permutations of a list are required.

Every algorithm has its unique strengths and weaknesses and can be suitable for specific uses depending on the size and nature of the dataset in question.

In Computer Science, sorting algorithms are fundamental and are utilized in a wide range of applications ranging from database management systems, search engines, and computational biology.

4.2.2 Algorithmic Complexity

A measure of an algorithm's effectiveness or efficiency is its algorithmic complexity. It is typically stated in terms of the number of steps needed to run the algorithm in relation to the size of the input. Time complexity and spatial complexity are the two most popular metrics for algorithmic complexity.

Time complexity measures the number of computational steps required to execute an algorithm as a function of the size of the input data. The most common notations used to express time complexity are O, Ω, and Θ. Big O notation is the most used notation to describe time complexity, and it represents an upper bound on the number of steps required by the algorithm. Ω notation represents a lower bound on the number of steps, and Θ notation represents an average or tight bound on the number of steps.

Space complexity, on the other hand, measures the amount of memory required by an algorithm to execute as a function of the size of the input data. It is usually expressed in terms of the number of memory cells required by the algorithm.

The analysis of algorithmic complexity is important because it helps us understand how an algorithm will perform on different input sizes and allows us to make informed decisions about which algorithm to use for a particular task. In general, we prefer algorithms with lower time and space complexity because they are more efficient and faster. However, the choice of algorithm also depends on other factors, such as the problem domain, the input size, and the available resources.

4.2.3 Comparison of Sorting Algorithms

For our research, we have selected the commonly used sorting algorithms (binary, bubble, quick) and an additional algorithm heap algorithm, which is used to generate all permutations of a given list. All our selected algorithms are well known and commonly used algorithms in Computer Science.

In order to effectively communicate the comparison of these sorting algorithms, we would be using the pseudocode of each.

- Pseudocode for Binary Sort

```
function binarySort(array):

    if length of array is less than or equal to 1:

        return array

    else:

        mid = length of array // 2

        left = array[:mid]

        right = array[mid:]

        return merge(binarySort(left), binarySort(right))

function merge(left, right):

    result = empty array

    while left is not empty and right is not empty:

        if left[0] is less than or equal to right[0]:

            append left[0] to result

            remove left[0] from left

        else:
            append right[0] to result

            remove right[0] from right

    if left is not empty:

        append all elements in left to result

    if right is not empty:

        append all elements in right to result

    return result
```

How it works:

I. The function binarySort takes an array as input and checks if the length of the array is less than or equal to 1. If it is, then the function simply returns the array as it is already sorted. This is the base case of the recursive function.

II. If the length of the array is greater than 1, the function calculates the midpoint of the array by dividing the length by 2 using integer division (//). It then splits the array into two sub-arrays: left, containing the elements from the start of the array up to the midpoint, and right, containing the elements from the midpoint to the end of the array.

III. The function then recursively calls itself on the left and right sub-arrays using binarySort(left) and binarySort(right), respectively, and then merges the two sorted sub-arrays using the merge function

IV. The merge function takes two sorted sub-arrays, left and right, and merges them into a single sorted array. It initializes an empty array called result to store the merged array.

V. The merge function takes two sorted sub-arrays, left and right, and merges them into a single sorted array. It initializes an empty array called result to store the merged array.

VI. After the loop completes, the function checks if there are any remaining elements in either the left or right sub-array. If there are, it appends them all to the result array.

VII. Finally, the function returns the merged and sorted array.

- Pseudocode for Bubble Sort

```
function bubbleSort(array):

    for i in range(length of array):

        for j in range(length of array - 1):

            if array[j] is greater than array[j+1]:

                swap array[j] with array[j+1]

    return array
```

How it works:

I. The function bubbleSort takes an array as an input.
II. A nested loop is used to iterate through the array. The outer loop runs length of array times.
III. The inner loop runs length of array - 1 times. This is because we compare each element with the one next to it, so there is no need to compare the last element with anything.
IV. The if statement inside the inner loop checks if the current element (array[j]) is greater than the next element (array[j+1]).
V. If the current element is greater than the next element, we swap the two elements using a temporary variable.

VI. We repeat this process until all elements have been compared and sorted in ascending order.
VII. Finally, the sorted array is returned.

- Pseudocode for Quick Sort

```
function quickSort(array):

    if length of array is less than 2:

        return array

    else:

        # find the median element

        medianIndex = length of array / 2

        median = array[medianIndex]



        left = empty array

        right = empty array

        for each element in array except the median:

            if element is less than or equal to median:

                append element to left

            else:

                append element to right

        return concatenate quickSort(left), median, quickSort(right)
```

How it works:

I. Check if the length of the input array is less than 2. If it is, return the array since it is already sorted.

II. Otherwise, find the median element of the array by dividing the length of the array by 2 and taking the element at that index.
III. Create two empty arrays called "left" and "right" to hold the elements that are less than or equal to the median and greater than the median, respectively.
IV. Iterate through each element in the input array except for the median element. For each element, if it is less than or equal to the median, append it to the "left" array. Otherwise, append it to the "right" array.
V. Recursively call quickSort() on the "left" and "right" arrays to sort them.
VI. Concatenate the sorted "left" array, the median element, and the sorted "right" array into a single sorted array and return it.

- Pseudocode for Heap Algorithm

```
function heapPermutation(array, size):

    if size is equal to 1:

        print array

    else:

        for i in range(size):

            heapPermutation(array, size-1)

            if size is odd:

                swap array[0] with array[size-1]

            else:

                swap array[i] with array[size-1]
```

How it works:

I. The function heapPermutation takes in an array and its size as input parameters.

II. If the size of the array is equal to 1, it prints the array and returns.

III. `If the size of the array is greater than 1, it enters a loop that iterates through each element of the array.`

IV. `For each element in the array, the function recursively calls itself with the array and size-1 as parameters.`

V. `After the recursive call, if the size of the array is odd, it swaps the first element with the last element of the array.`

VI. `If the size of the array is even, it swaps the ith element with the last element of the array (where i is the current iteration of the loop).`

VII. `The function terminates after the loop has iterated through all the elements in the array.`

Table 4.1: Advantages and Disadvantages of Sorting Algorithms

	Advantages	Disadvantages
Binary	It has a time complexity of O(log n) which makes it much faster than other search algorithms.	Requires that the array is sorted in advance, so it's not useful for sorting unsorted arrays.
	It can be implemented iteratively or recursively, which makes it more flexible in terms of implementation.	Has a worst-case time complexity of O(log n) which is slower than some other sorting algorithms.
Bubble	It is easy to understand and implement, which makes it a good choice for small data sets or as a teaching tool.	Has a worst-case time complexity of O(n^2), which makes it inefficient for large arrays.

	It has a low memory footprint, as it only requires a single additional memory space to store a temporary variable during swaps.	Bubble sort is not very efficient for arrays that are already sorted or nearly sorted.
Quick	It has a very high average-case performance and is one of the fastest sorting algorithms in practice.	Worst-case time complexity is O(n^2) which occurs when the pivot is chosen poorly, making it less efficient than some other sorting algorithms in the worst-case scenario.
	It is an "in-place" sorting algorithm, which means it requires very little additional memory to run, making it useful in memory-constrained situations.	Quick sort requires additional memory space for the stack to keep track of recursive calls which can be a disadvantage for large arrays.

In terms of efficiency, the ranking from most efficient to least efficient is:

Table 4.2: Time Complexity Table of Sorting Algorithms

Algorithm / Time Complexity	Best Case	Average Case	Worst Case
Quick Sort	O(n log n)	O(n log n)	O(n2)
Binary Sort	O(1)	O(n log n)	O(n log n)
Bubble Sort	O(n)	O(n2)	O(n2)

A simple scenario to explain the implications of Table 2 above is as follows:

Suppose we have a dataset of unsorted list/array containing 10 elements [5, 2, 8, 1, 9, 7, 3, 6, 4, 10] , the sorting algorithms work as follows:

⇒ Quick Sort – with a time complexity of O(n log n), the list will be sorted in $10 * log10(10)$ which gives a 30 unit time to sort the array.

- ⇒ Binary Sort – with a time complexity of O(n log n), the list will be sorted in $10 * log10(10)$ which gives a 30 unit time to sort the array. Just like the quick sort above.
- ⇒ Bubble Sort - with a time complexity of O(n2), the list will be sorted in 10^2 which gives a 100 unit time to sort the array.

Note: Although, Quick and Binary sorts have the same time complexity, but the former has a good performance over the later due to:

- ⇒ Cache Efficiency – In contrast to Binary Sort, which needs additional memory space to keep track of indexes or pointers to the sorted sub-lists, Quick Sort is an in-place sorting method.
- ⇒ Partitioning Strategy – Binary Sort divides the array into two half and sorts them recursively before merging the sorted halves, whereas Quick Sort separates the array so that all the elements smaller than the pivot element are transferred to the left side of the pivot. This is slower in practice and requires more steps to sort the same array.

4.2.4 PHP and Node.js

Dynamic web pages and web applications are created using PHP (Hypertext Preprocessor), an open-source programming language. The language is often combined with HTML, CSS, and JavaScript for creating websites. After PHP executes on the server, the HTML it produces is transferred to the client's browser.

A simple PHP code used to define a variable $name, then dynamically generating an HTML heading using the echo is provided below:

```
<html>

<head>

     <title>My PHP Page</title>

     <meta charset="UTF-8">

     <meta name="viewport" content="width=device-width, initial-
scale=1.0">

</head>

<body>

<?php
```

```
      // This is a PHP code block
      $name = "Qozeem Odeniran";
      echo "<h1>Hello, $name!</h1>";
?>
</body>
</html>
```

On the other hand, scalable network applications can be created using Node.js, a JavaScript runtime. Building real-time applications that need a lot of I/O operations, such chat applications, streaming services, and online games, is made possible by Node.js' event-driven, non-blocking I/O paradigm. Together with other web technologies like Angular, React, and Vue.js, Node.js is frequently utilized.

A sample code that creates a simple web server that listens on port 3000 and responds with a "Hello, Qozeem Odeniran!" message when a request is made to the server.

```
const http = require('http');
const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  const name = "Qozeem Odeniran";
  res.end(`Hello, ${name}`);
});
const port = 3000;
server.listen(port, () => {
  console.log(`Server running at http://localhost:${port}/`);
});
```

4.3 METHODOLOGY

4.3.1 Research Problem and Purpose of Study

Comparing the efficiency of two backend programming languages and technologies, PHP (old) and Node.js (new), using a variety of sorting algorithms, including binary sort, bubble sort, quick sort, and heap algorithm to generate all possible permutation of array element, is the research objective and goal of the study in this research. Our goal is to identify the most optimal backend language or scripting technology that can provide a maximum performance with regards to latency – time taken for a request to reach the server and receive a response. This research study aims to offer useful insights to software engineers and organizations seeking to enhance their web application performance and make well-informed choices when selecting the backend scripting technologies.

Back-end scripting technologies are crucial in web development, as they greatly enhance the development of web applications and services. The popularity of backend scripting technologies is constantly evolving, with new backend scripting technologies emerging on a regular basis. As such, developers as well as organizations need to evaluate the performance of existing and new backend scripting technologies to determine which ones to use for their projects.

4.3.2 Performance Goal, Statement of Research Objectives and Research Questions

4.3.2.1 Performance Goal:

There is a significant difference between the performance of PHP (old) and Node.js (new) backend scripting technologies.

4.3.3 Research Objectives:

⇒ To compare the performance of PHP and Node.js using binary sort, bubble sort, quick sort, and heap algorithm.

⇒ To determine which algorithm performs better in PHP and Node.js
⇒ To provide recommendations for choosing the appropriate backend scripting technology.

4.3.4 Research Questions:

⇒ RQ 1: Is node.js faster than PHP when it comes to Sorting Algorithms?
⇒ RQ 2: Is node.js is faster than PHP when it comes to heap algorithm?
⇒ RQ 3: In overall, is node.js faster than PHP for backend scripting?

4.3.5 Description of Research Design, Apache JMeter, PHP & Node.js, Sorting Algorithms, and Performance Metrics.

4.3.6 Research Design:

Our design for this research involved comparing the performance of two prominent backend scripting technologies – PHP (old) and Node.js (new) using three sorting algorithms (bubble, binary, and quick) to sort data as we exponentially increased the array size, and heap algorithm to generate all possible permutations of the elements of the array. Each of the algorithms were run thirty (30) times for both PHP and Node.js. Our performance metrics were collected using the Apache JMeter software – an open-source load testing tool used to simulate different types of load testing scenarios for web applications. The selection of these algorithms was based on popularity, relevance and effectiveness in data sorting as used in the field of Computer Science. The key performance metric used in this research was LATENCY which is discussed extensively in later paragraph.

4.3.7 Apache JMeter & Settings:

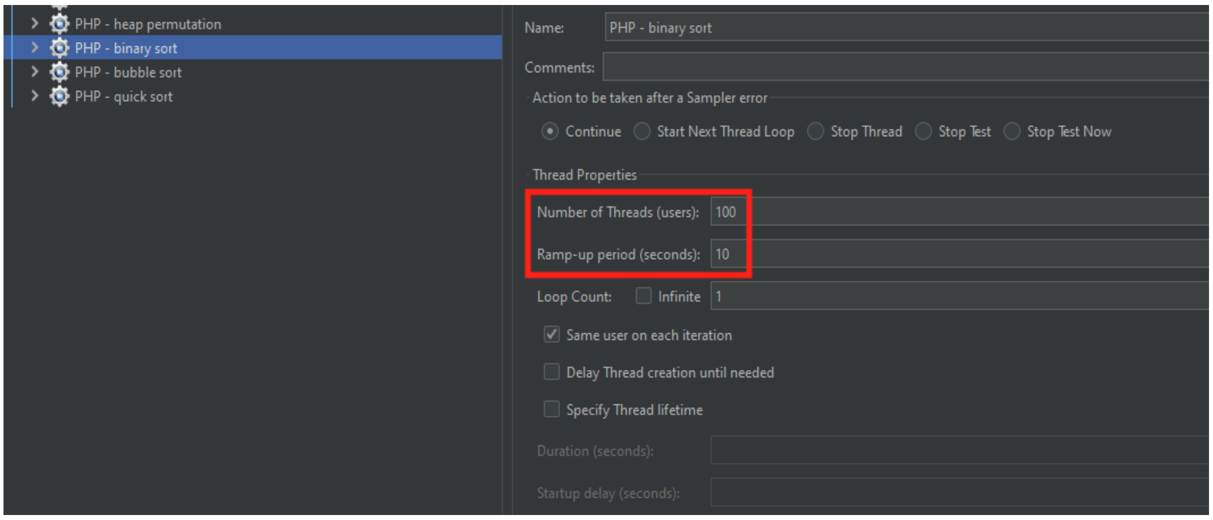
⇒ Ramp Up
time taken by JMeter to start all the virtual users specified in the test plan. It determines the rate at which users are added during the test execution. We used a ramp-up time of 10.
⇒ Thread – Thread refers to a single virtual user that simulates user activity on the application being tested. Each thread executes the test script independently of all other threads. We simulated 100 virtual users.

Although, there is no one-size-fits-all answer to what the standard settings for ramp-up and threads (number of simulated users) should be in Apache JMeter. The ideal values depend on various factors such as the size and complexity of the application being tested, the available hardware resources, and the performance goals. However, as a rule of thumb, the ramp-up time should be set to a value that allows a gradual increase in the number of virtual users, instead of an abrupt spike. This helps simulate real-world scenarios where traffic gradually increases over time. A good starting point for ramp-up time could be 5-10 seconds. The number of threads, or virtual users, should also be chosen based on the hardware resources available for the load testing. In general, it's recommended to start with a low number of threads and gradually increase it to find the optimal number that the system can handle without performance degradation or failures. A starting point could be 50-100 threads, and then gradually increase to several hundred or even thousands depending on the available resources. A ramp-up of 10 seconds is a reasonable choice for a 100 number of threads and a typical test scenario as allows the load to be gradually increased (our array size), giving the server time to warm up and stabilize before reaching the maximum load. It also provides enough time for JMeter to start all the threads and for the server to respond to the initial requests. In Apache JMeter, the Ramp-up period represents the time taken to spin up all the threads in the thread group.

For instance, if we have 100 threads and a ramp-up period of 10 seconds, then JMeter will create a new thread every 100 milliseconds (10,000 milliseconds / 100 threads). In other words, the ramp-up period determines how quickly the threads will be created and started. If the ramp-up period is short, like 1 second, then all 100 threads will be created almost simultaneously, which could overload the server being tested.

On the other hand, if the ramp-up period is too long, like 100 seconds, then it will take too long to start all the threads, and the test may not be representative of real-world scenarios.

Therefore, the ramp-up period of 10 seconds and 100 threads mean that JMeter will create a new thread every 100 milliseconds over the course of 10 seconds, so all threads will be active after the ramp-up period. This setting strikes a balance between quickly starting the threads and not overloading the server with too many requests at once.



Name: PHP - binary sort

Comments:

Action to be taken after a Sampler error

[✓ Continue] Start Next Thread Loop Stop Thread Stop Test Stop Test Now

Thread Properties

Number of Threads (users): 100

Ramp-up period (seconds): 10

Loop Count: ☐ Infinite 1

☑ Same user on each iteration

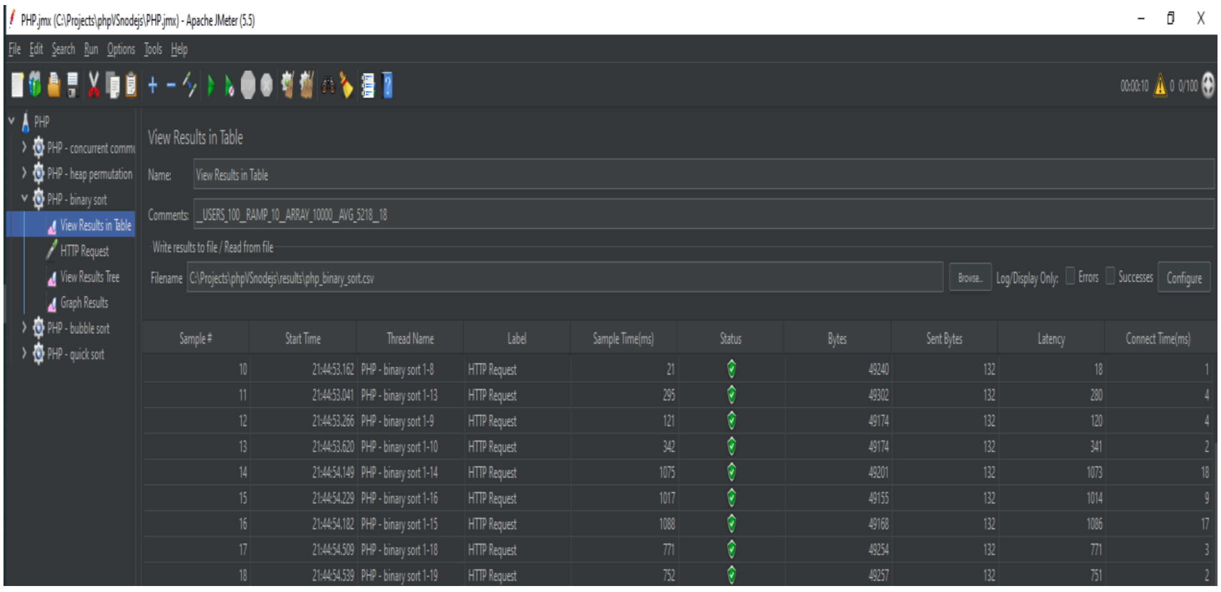
☐ Delay Thread creation until needed

☐ Specify Thread lifetime

Duration (seconds):

Startup delay (seconds):

Figure 4.1: Apache JMeter showing Number of Threads & Ramp-up period



View Results in Table

Name: View Results in Table

Comments: _USERS_100_RAMP_10_ARRAY_10000_AVG_5218_18

Write results to file / Read from file

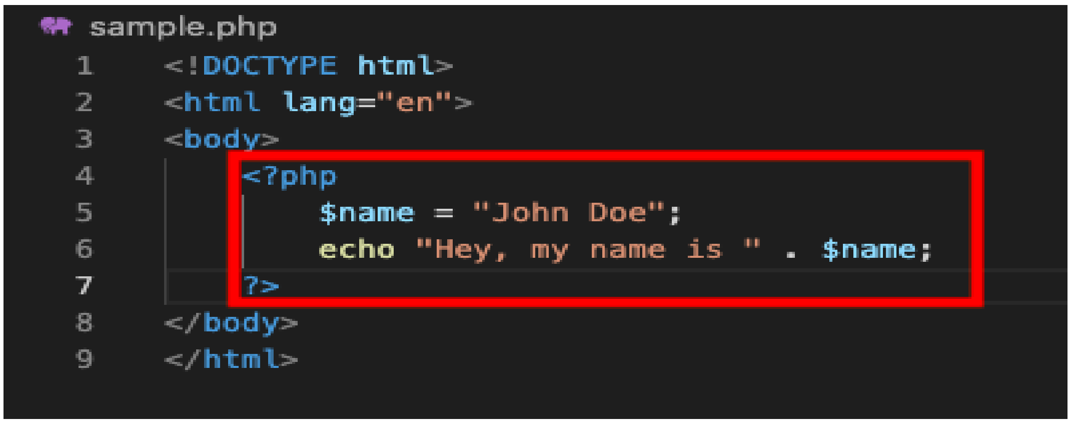
Filename C:\Projects\phpVSnodejs\results\php_binary_sort.csv Browse... Log/Display Only: ☐ Errors ☐ Successes Configure

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
10	21:44:53.162	PHP - binary sort 1-8	HTTP Request	21	✓	49240	132	18	1
11	21:44:53.041	PHP - binary sort 1-13	HTTP Request	295	✓	49302	132	280	4
12	21:44:53.266	PHP - binary sort 1-9	HTTP Request	121	✓	49174	132	120	4
13	21:44:53.620	PHP - binary sort 1-10	HTTP Request	342	✓	49174	132	341	2
14	21:44:54.149	PHP - binary sort 1-14	HTTP Request	1075	✓	49201	132	1073	18
15	21:44:54.229	PHP - binary sort 1-16	HTTP Request	1017	✓	49155	132	1014	9
16	21:44:54.182	PHP - binary sort 1-15	HTTP Request	1088	✓	49168	132	1086	17
17	21:44:54.509	PHP - binary sort 1-18	HTTP Request	771	✓	49254	132	771	3
18	21:44:54.539	PHP - binary sort 1-19	HTTP Request	752	✓	49257	132	751	2

Figure 4.2: Apache JMeter showing test run results

4.3.7.1 PHP

Hypertext Pre-processor (PHP), commonly referred to as a scripting language, is a server-side programming language used to create dynamic web pages. Rasmus Lerdorf created it in 1994 to include some dynamic components on his page. It supports object-oriented programming, is open source, simple to learn, and integrates well with HTML. Even now, PHP is still used as the back-end scripting technology for many millions of websites. In order to execute it successfully, a server and PHP editor are required. The most recent version is PHP-8.

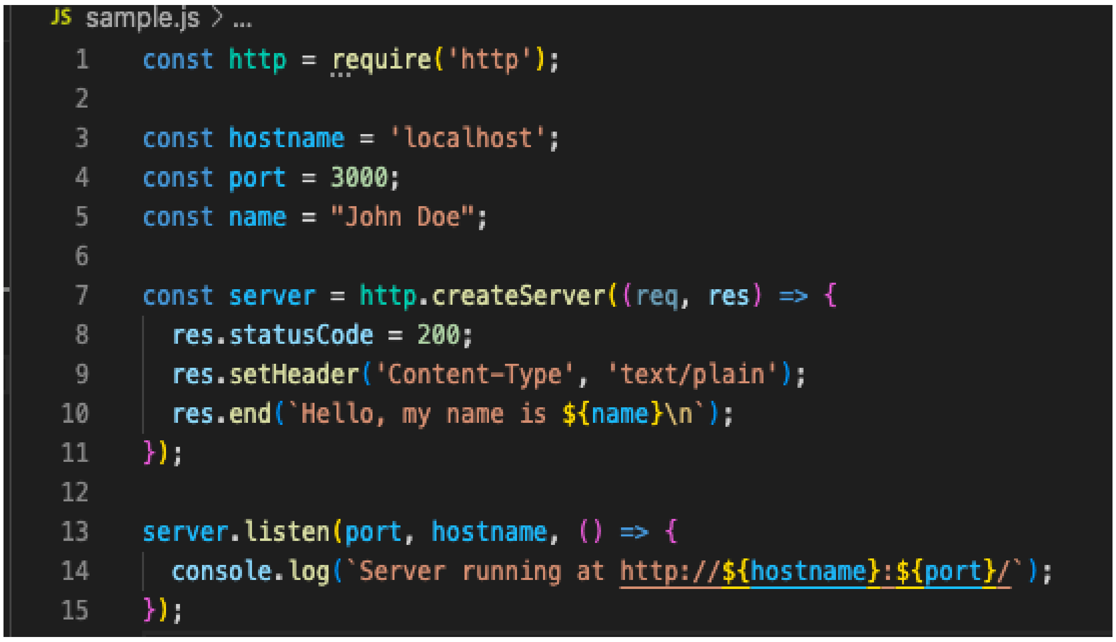


```
sample.php
<!DOCTYPE html>
<html lang="en">
<body>
    <?php
        $name = "John Doe";
        echo "Hey, my name is " . $name;
    ?>
</body>
</html>
```

Figure 4.3: A basic PHP code inside HTML page.

4.3.7.2 Node.js:

Since its release in 2009, Node.js, a well-known JavaScript technology, has been installed over a billion times. It is a cross-platform runtime environment with an event-driven, non-blocking I/O strategy for quick, scalable online apps based on Chrome's JavaScript runtime. With over 800,000 GitHub repositories, businesses are seeing the advantages of Node.js more and more. On GitHub, JavaScript is now the most widely used language.



```
sample.js > ...
const http = require('http');

const hostname = 'localhost';
const port = 3000;
const name = "John Doe";

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end(`Hello, my name is ${name}\n`);
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

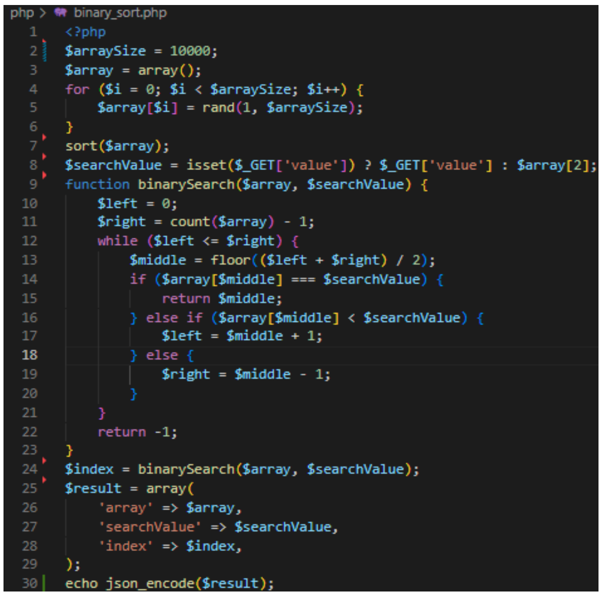
Figure 4.4: A basic node.js code

4.3.7.3 Sorting Algorithms

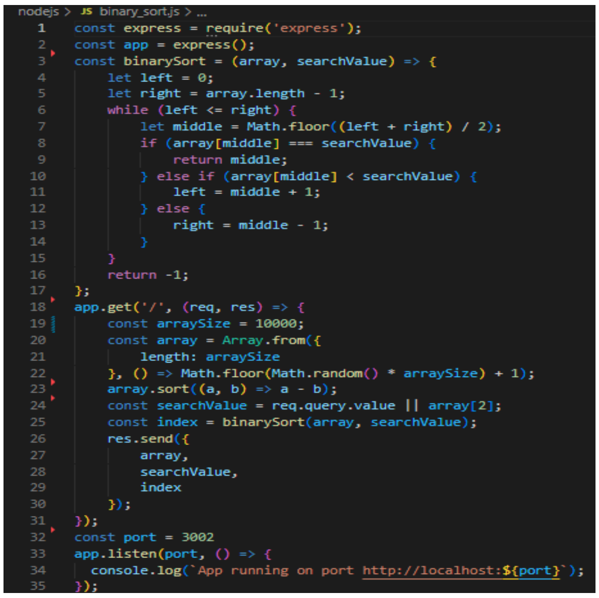
In Computer Science, sorting algorithms refer to procedures for sorting elements in an array or list. Sorting algorithms implement the process of rearrangement by sorting items according to a predetermined order.

In our research, we implemented the following sorting algorithms.

⇒ Binary Sort: This algorithm uses a divide-and-conquer style whereby it divides the array into two equal halves, sorts each half individually, and merges them back together. Elements in the left and right sub-arrays are compared by the merge operation, which then merges them into sorted order. Until the full array is sorted, this procedure is repeated.



```
php > binary_sort.php
<?php
$arraySize = 10000;
$array = array();
for ($i = 0; $i < $arraySize; $i++) {
    $array[$i] = rand(1, $arraySize);
}
sort($array);
$searchValue = isset($_GET['value']) ? $_GET['value'] : $array[2];
function binarySearch($array, $searchValue) {
    $left = 0;
    $right = count($array) - 1;
    while ($left <= $right) {
        $middle = floor(($left + $right) / 2);
        if ($array[$middle] === $searchValue) {
            return $middle;
        } else if ($array[$middle] < $searchValue) {
            $left = $middle + 1;
        } else {
            $right = $middle - 1;
        }
    }
    return -1;
}
$index = binarySearch($array, $searchValue);
$result = array(
    'array' => $array,
    'searchValue' => $searchValue,
    'index' => $index,
);
echo json_encode($result);
```

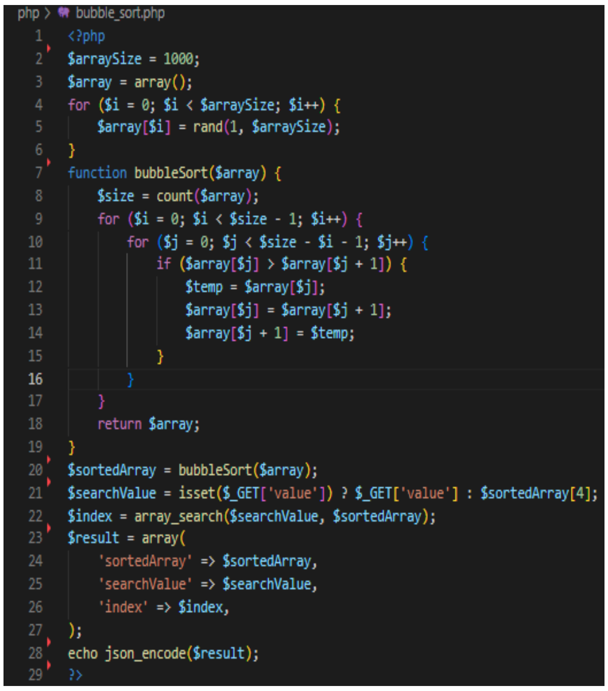


```
nodejs > JS binary_sort.js > ...
const express = require('express');
const app = express();
const binarySort = (array, searchValue) => {
    let left = 0;
    let right = array.length - 1;
    while (left <= right) {
        let middle = Math.floor((left + right) / 2);
        if (array[middle] === searchValue) {
            return middle;
        } else if (array[middle] < searchValue) {
            left = middle + 1;
        } else {
            right = middle - 1;
        }
    }
    return -1;
};
app.get('/', (req, res) => {
    const arraySize = 10000;
    const array = Array.from({
        length: arraySize
    }, () => Math.floor(Math.random() * arraySize) + 1);
    array.sort((a, b) => a - b);
    const searchValue = req.query.value || array[2];
    const index = binarySort(array, searchValue);
    res.send({
        array,
        searchValue,
        index
    });
});
const port = 3002
app.listen(port, () => {
  console.log(`App running on port http://localhost:${port}`);
});
```

Figure 4.5: Binary Sort in PHP

Figure 4.6: Binary Sort in Node.js

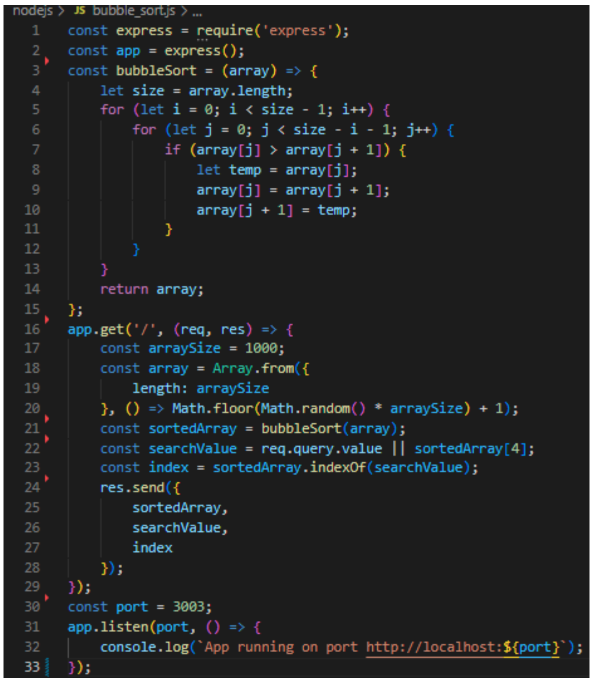
⇒ Bubble Sort: This sorting algorithm analyses adjacent elements and swaps them if they are in the wrong order as it iteratively moves through the list to be sorted. Until the list is sorted, this trip through the list is repeated. The smaller components "bubble" is passed to the top of the list with each run, hence this technique is known as bubble sort. The worst-case and average time complexity of bubble sort, where n is the total number of elements to be sorted, is O(n2).



php > bubble_sort.php

```php
<?php
$arraySize = 1000;
$array = array();
for ($i = 0; $i < $arraySize; $i++) {
    $array[$i] = rand(1, $arraySize);
}
function bubbleSort($array) {
    $size = count($array);
    for ($i = 0; $i < $size - 1; $i++) {
        for ($j = 0; $j < $size - $i - 1; $j++) {
            if ($array[$j] > $array[$j + 1]) {
                $temp = $array[$j];
                $array[$j] = $array[$j + 1];
                $array[$j + 1] = $temp;
            }
        }
    }
    return $array;
}
$sortedArray = bubbleSort($array);
$searchValue = isset($_GET['value']) ? $_GET['value'] : $sortedArray[4];
$index = array_search($searchValue, $sortedArray);
$result = array(
    'sortedArray' => $sortedArray,
    'searchValue' => $searchValue,
    'index' => $index,
);
echo json_encode($result);
?>
```

Figure 4.7: Bubble Sort in PHP

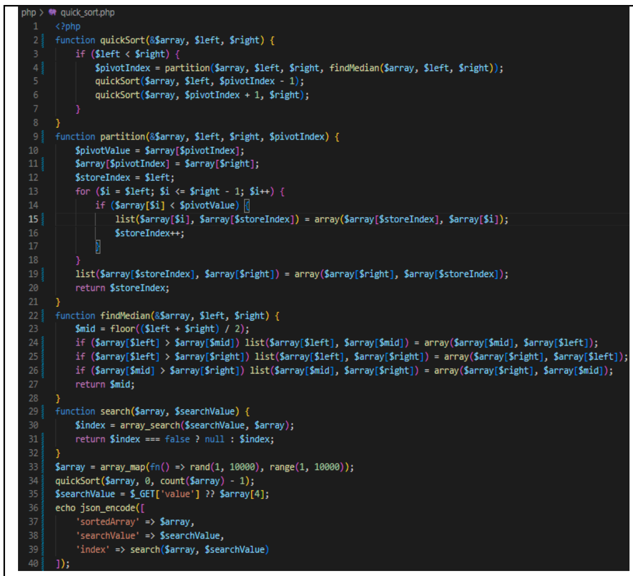


nodejs > JS bubble_sort.js > ...

```javascript
const express = require('express');
const app = express();
const bubbleSort = (array) => {
    let size = array.length;
    for (let i = 0; i < size - 1; i++) {
        for (let j = 0; j < size - i - 1; j++) {
            if (array[j] > array[j + 1]) {
                let temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
        }
    }
    return array;
};
app.get('/', (req, res) => {
    const arraySize = 1000;
    const array = Array.from({
        length: arraySize
    }, () => Math.floor(Math.random() * arraySize) + 1);
    const sortedArray = bubbleSort(array);
    const searchValue = req.query.value || sortedArray[4];
    const index = sortedArray.indexOf(searchValue);
    res.send({
        sortedArray,
        searchValue,
        index
    });
});
const port = 3003;
app.listen(port, () => {
    console.log(`App running on port http://localhost:${port}`);
});
```

Figure 4.8: Bubble Sort in Node.js

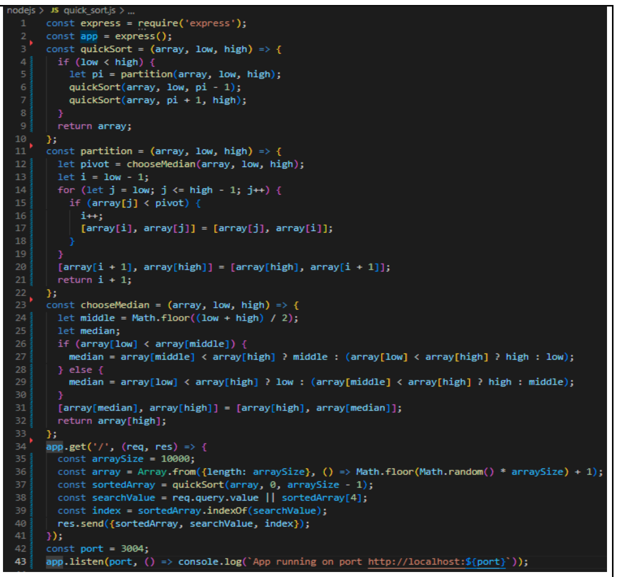
⇒ Quick Sort: This sorting technique divides an array or list of elements into two sub-arrays, one of which contains elements less than a pivot value and the other of which contains elements greater than or equal to the pivot value. After that, it sorts the two sub-arrays recursively.



php > quick_sort.php

```php
<?php
function quickSort(&$array, $left, $right) {
    if ($left < $right) {
        $pivotIndex = partition($array, $left, $right, findMedian($array, $left, $right));
        quickSort($array, $left, $pivotIndex - 1);
        quickSort($array, $pivotIndex + 1, $right);
    }
}
function partition(&$array, $left, $right, $pivotIndex) {
    $pivotValue = $array[$pivotIndex];
    $array[$pivotIndex] = $array[$right];
    $storeIndex = $left;
    for ($i = $left; $i <= $right - 1; $i++) {
        if ($array[$i] < $pivotValue) {
            list($array[$i], $array[$storeIndex]) = array($array[$storeIndex], $array[$i]);
            $storeIndex++;
        }
    }
    list($array[$storeIndex], $array[$right]) = array($array[$right], $array[$storeIndex]);
    return $storeIndex;
}
function findMedian(&$array, $left, $right) {
    $mid = floor(($left + $right) / 2);
    if ($array[$left] > $array[$mid]) list($array[$left], $array[$mid]) = array($array[$mid], $array[$left]);
    if ($array[$left] > $array[$right]) list($array[$left], $array[$right]) = array($array[$right], $array[$left]);
    if ($array[$mid] > $array[$right]) list($array[$mid], $array[$right]) = array($array[$right], $array[$mid]);
    return $mid;
}
function search($array, $searchValue) {
    $index = array_search($searchValue, $array);
    return $index === false ? null : $index;
}
$array = array_map(fn() => rand(1, 10000), range(1, 10000));
quickSort($array, 0, count($array) - 1);
$searchValue = $_GET['value'] ?? $array[4];
echo json_encode([
    'sortedArray' => $array,
    'searchValue' => $searchValue,
    'index' => search($array, $searchValue)
]);
```

Figure 4.9: Quick Sort in PHP

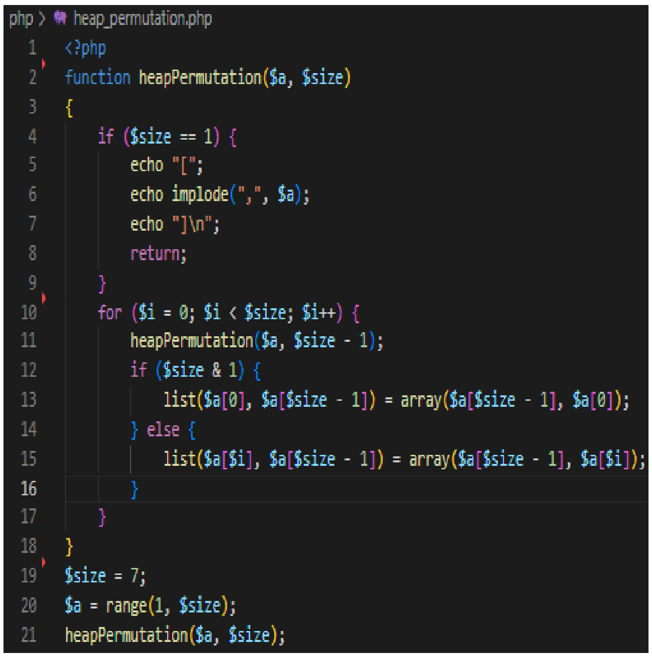


nodejs > JS quick_sort.js > ...

```javascript
const express = require('express');
const app = express();
const quickSort = (array, low, high) => {
  if (low < high) {
    let pi = partition(array, low, high);
    quickSort(array, low, pi - 1);
    quickSort(array, pi + 1, high);
  }
  return array;
};
const partition = (array, low, high) => {
  let pivot = chooseMedian(array, low, high);
  let i = low - 1;
  for (let j = low; j <= high - 1; j++) {
    if (array[j] < pivot) {
      i++;
      [array[i], array[j]] = [array[j], array[i]];
    }
  }
  [array[i + 1], array[high]] = [array[high], array[i + 1]];
  return i + 1;
};
const chooseMedian = (array, low, high) => {
  let middle = Math.floor((low + high) / 2);
  let median;
  if (array[low] < array[middle]) {
    median = array[middle] < array[high] ? middle : (array[low] < array[high] ? high : low);
  } else {
    median = array[low] < array[high] ? low : (array[middle] < array[high] ? high : middle);
  }
  [array[median], array[high]] = [array[high], array[median]];
  return array[high];
};
app.get('/', (req, res) => {
  const arraySize = 10000;
  const array = Array.from({length: arraySize}, () => Math.floor(Math.random() * arraySize) + 1);
  const sortedArray = quickSort(array, 0, arraySize - 1);
  const searchValue = req.query.value || sortedArray[4];
  const index = sortedArray.indexOf(searchValue);
  res.send({sortedArray, searchValue, index});
});
const port = 3004;
app.listen(port, () => console.log(`App running on port http://localhost:${port}`));
```

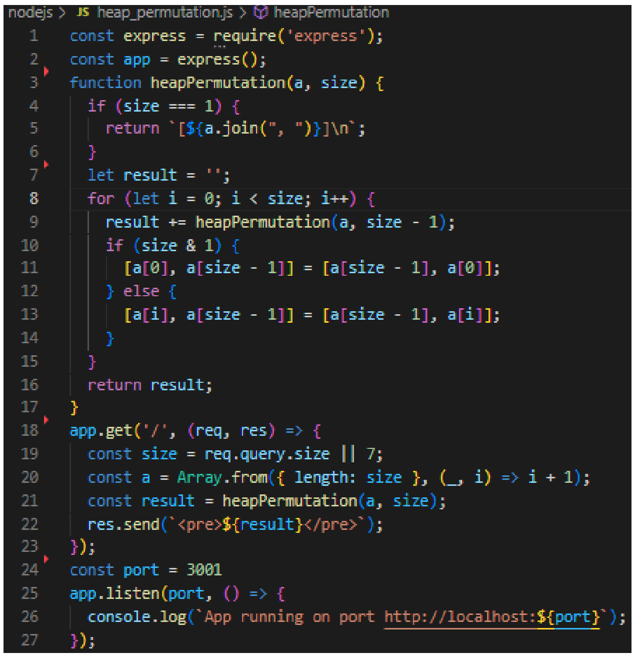
Figure 4.10: Quick Sort in Node.js

⇒ Heap Algorithm: This algorithm generates all possible permutations of a given array or list. The for loop iterates over each element of the array, and for each element it calls heapPermutation with a reduced array size of $size-1. The swap operation is performed depending on whether $size is odd or even. If $size is odd, the first element of the array is swapped with the last element, otherwise the $ith element is swapped with the last element.



php > heap_permutation.php

```php
<?php
function heapPermutation($a, $size)
{
    if ($size == 1) {
        echo "[";
        echo implode(",", $a);
        echo "]\n";
        return;
    }
    for ($i = 0; $i < $size; $i++) {
        heapPermutation($a, $size - 1);
        if ($size & 1) {
            list($a[0], $a[$size - 1]) = array($a[$size - 1], $a[0]);
        } else {
            list($a[$i], $a[$size - 1]) = array($a[$size - 1], $a[$i]);
        }
    }
}
$size = 7;
$a = range(1, $size);
heapPermutation($a, $size);
```



nodejs > heap_permutation.js > heapPermutation

```javascript
const express = require('express');
const app = express();
function heapPermutation(a, size) {
  if (size === 1) {
    return `[${a.join(", ")}]\n`;
  }
  let result = '';
  for (let i = 0; i < size; i++) {
    result += heapPermutation(a, size - 1);
    if (size & 1) {
      [a[0], a[size - 1]] = [a[size - 1], a[0]];
    } else {
      [a[i], a[size - 1]] = [a[size - 1], a[i]];
    }
  }
  return result;
}
app.get('/', (req, res) => {
  const size = req.query.size || 7;
  const a = Array.from({ length: size }, (_, i) => i + 1);
  const result = heapPermutation(a, size);
  res.send(`<pre>${result}</pre>`);
});
const port = 3001
app.listen(port, () => {
  console.log(`App running on port http://localhost:${port}`);
});
```

Figure 4.11: - Heap Algorithm in PHP

Figure 4.12: - Heap Algorithm in Node.js

4.3.7.4 Performance Metrics

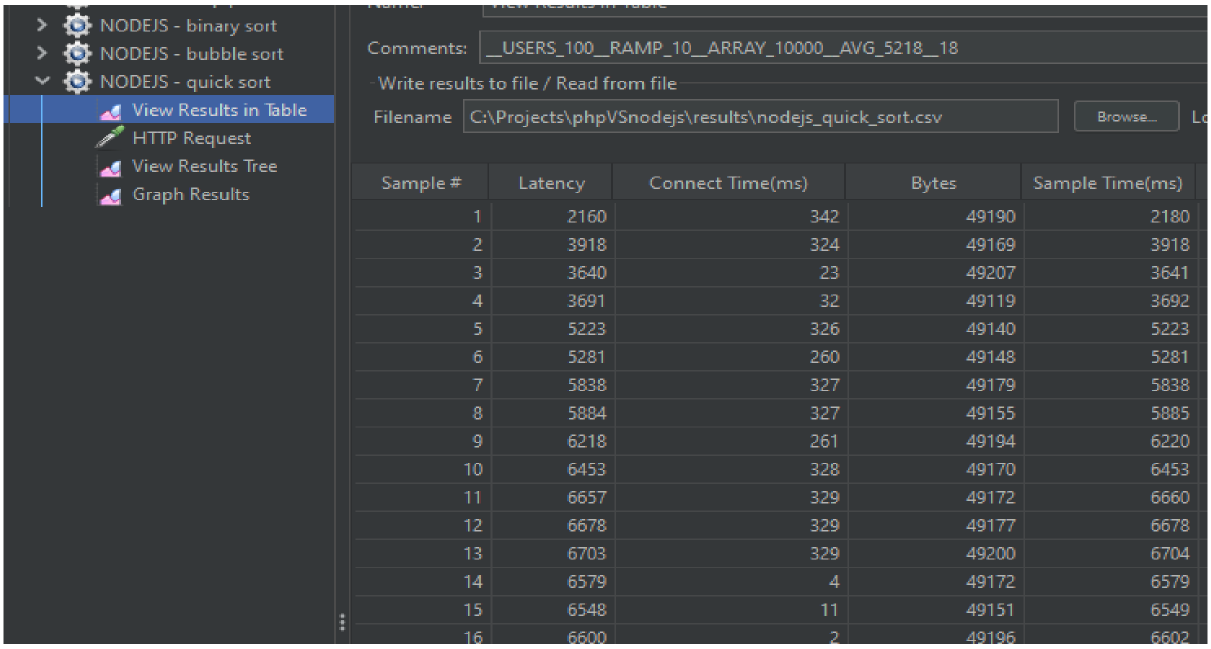
Latency (milliseconds) - The term "latency" describes how long it takes a server, usually measured in milliseconds, to reply to a client request. It includes any processing time required on the server side as well as the time it takes for a request to go from the client to the server and back again. A server that is quick and responsive has a low latency, while one that is slow and unresponsive has a high latency. Latency is an important metric for evaluating the performance of any system, especially in the context of computer networks and the internet. Latency refers to the time delay between a request for data and the response to that request. In other words, it's the time it takes for data to travel from its source to its destination. Low latency is important for a number of reasons. Firstly, it affects the user experience of interactive applications such as online gaming, video conferencing, and real-time communication tools. In these applications, high latency can cause delays and interruptions in communication, leading to a poor user experience. Secondly, latency is critical for high-speed trading applications, where even a small delay can have a significant impact on the outcome of a trade. In this context, low latency can give traders a competitive advantage by allowing them to make decisions and execute trades faster than their competitors.

- Sample Time (milliseconds) - Sample time is the duration of time for which a test run is executed. It can help identify whether the system can provide consistent performance over a period of time, especially when multiple requests are made.
- Connect Time (milliseconds) - Connect time is the time taken to establish a connection between the client and the server. A low connect time is crucial for applications that require frequent connections, such as real-time applications.
- Sent Bytes (bytes) - Bytes refer to the amount of data transferred during a request/response cycle. The amount of data transferred can have a significant impact on the overall performance of a system, especially when handling large volumes of data.

4.3.8 Data Collection Process, Testing Procedures

4.3.8.1 Collection Process

As Apache JMeter provides several ways to collect data during load testing, our adopted collection process involves the use of TABLE LISTENER – which provides listener to collect data in real-time while the load test is running. With the table listener, we were able to capture a wide range of data including latency, connect times, bytes, sample time etc. Apache JMeter enables us export this table as a csv file which we then used for our statistical analysis.



Comments: __USERS_100__RAMP_10__ARRAY_10000__AVG_5218__18

Write results to file / Read from file

Filename C:\Projects\phpVSnodejs\results\nodejs_quick_sort.csv

Sample #	Latency	Connect Time(ms)	Bytes	Sample Time(ms)
1	2160	342	49190	2180
2	3918	324	49169	3918
3	3640	23	49207	3641
4	3691	32	49119	3692
5	5223	326	49140	5223
6	5281	260	49148	5281
7	5838	327	49179	5838
8	5884	327	49155	5885
9	6218	261	49194	6220
10	6453	328	49170	6453
11	6657	329	49172	6660
12	6678	329	49177	6678
13	6703	329	49200	6704
14	6579	4	49172	6579
15	6548	11	49151	6549
16	6600	2	49196	6602

Figure 4.13: Apache JMeter showing the Table Listener

4.3.8.2 Procedure

To ensure that we are only running targeted load test, our testing procedure involves first right-clicking on the intended algorithm, then clicking on 'Start' to begin the test load run.

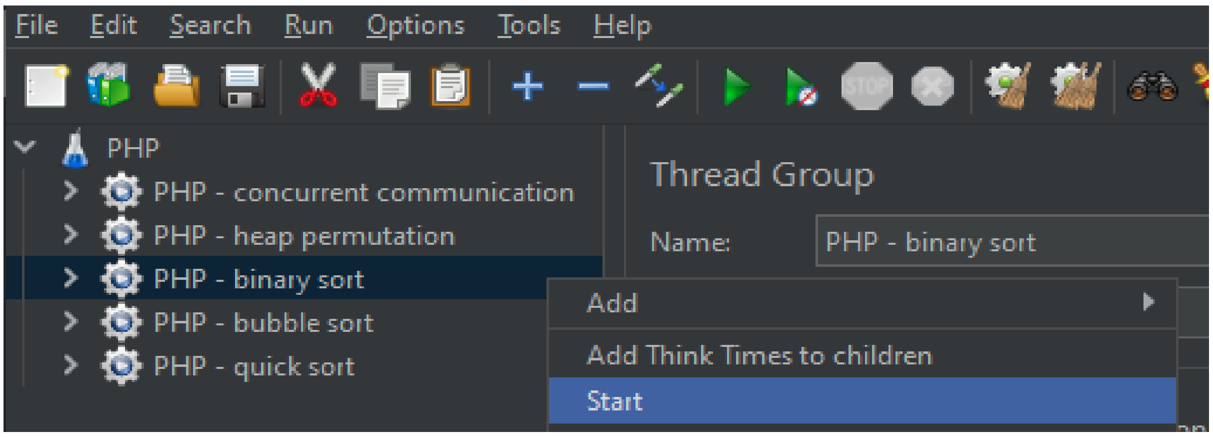


Figure 4.14: Running targeted algorithm in Apache JMeter

4.3.8.3 Data Analysis Process & Statistical Techniques, Hypothesis Testing

4.3.8.4 Analysis Process:

The analysis process of performance testing in our research involves thoroughly examining the data collected during the load test to gain insights into the behavior of the backend scripting technologies, as well as the sorting algorithms under different loads (array sizes). Our main goal for analyzing remains the fact we want to see which backend scripting performs better. We will break down the analysis processes in the next paragraphs.

- Step 1 – Data Preparation: - we gathered the data collected during the test and organize them in a format that we can easily analyze.
- Step 2 – Performance Metrics: - once our data has been organized and formatted, we calculated the average latency of each simulated user for the 30 runs.

4.3.8.5 Statistical Technique

Since our research involves a moderately large data, and we aimed to analyze the data to draw conclusions so that we can make reasonable inferences, we adopted one of the commonly used statistical technique, T-TEST which is a type of hypothesis test used to determine whether two groups are significantly different.

Hypothesis testing is an effective technique for analyzing the significance of the difference when comparing populations, such as when assessing or testing the difference between the means from two samples of data [42].

4.3.8.6 Hypothesis Testing:

Hypothesis testing is the process of making a statement and verifying it with data. During this process, it is typically assumed that two samples are not different from each other. This assumption helps to create two hypotheses: a null hypothesis (H0) and an alternative hypothesis (Ha). The outcomes of a hypothesis test can either reject the null hypothesis and accept the alternative hypothesis (indicating that there is a difference between two samples), or not reject the null hypothesis. In a scenario where the null hypothesis states that the population means of two unrelated groups are equal (H0: u1=u2) and the alternative hypothesis states that the population means are not equal (Ha: u1≠u2), it is determined whether to reject or accept the alternative hypothesis based on a significance level of $p<0.05$.

4.4 RESULTS

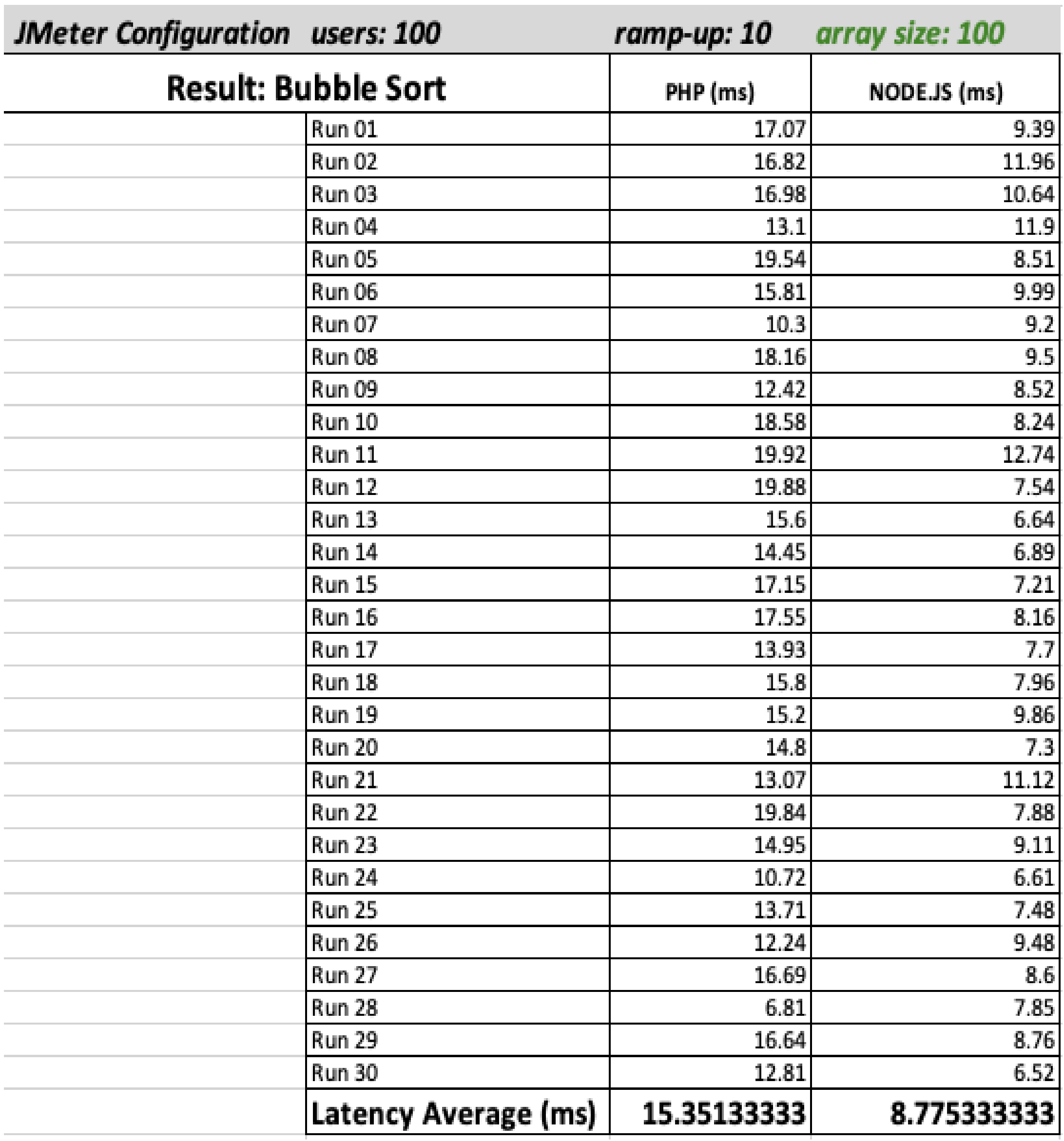
4.4.1.1 Overview

Our research question for this study involves determining if there is difference in performance between PHP (old backend technology) and Node.js (new backend technology) is significant. After writing the programs to implement various sorting algorithms, and heap algorithm to generate all possible permutations, we gathered our data from the performance test we ran on Apache JMeter. We adopted the T-Test statistical method in analyzing collected data, since it is best used to determine if there is a statistically significant difference between the means of two groups, in our case PHP and Node.js.

4.4.1.2 Statistics

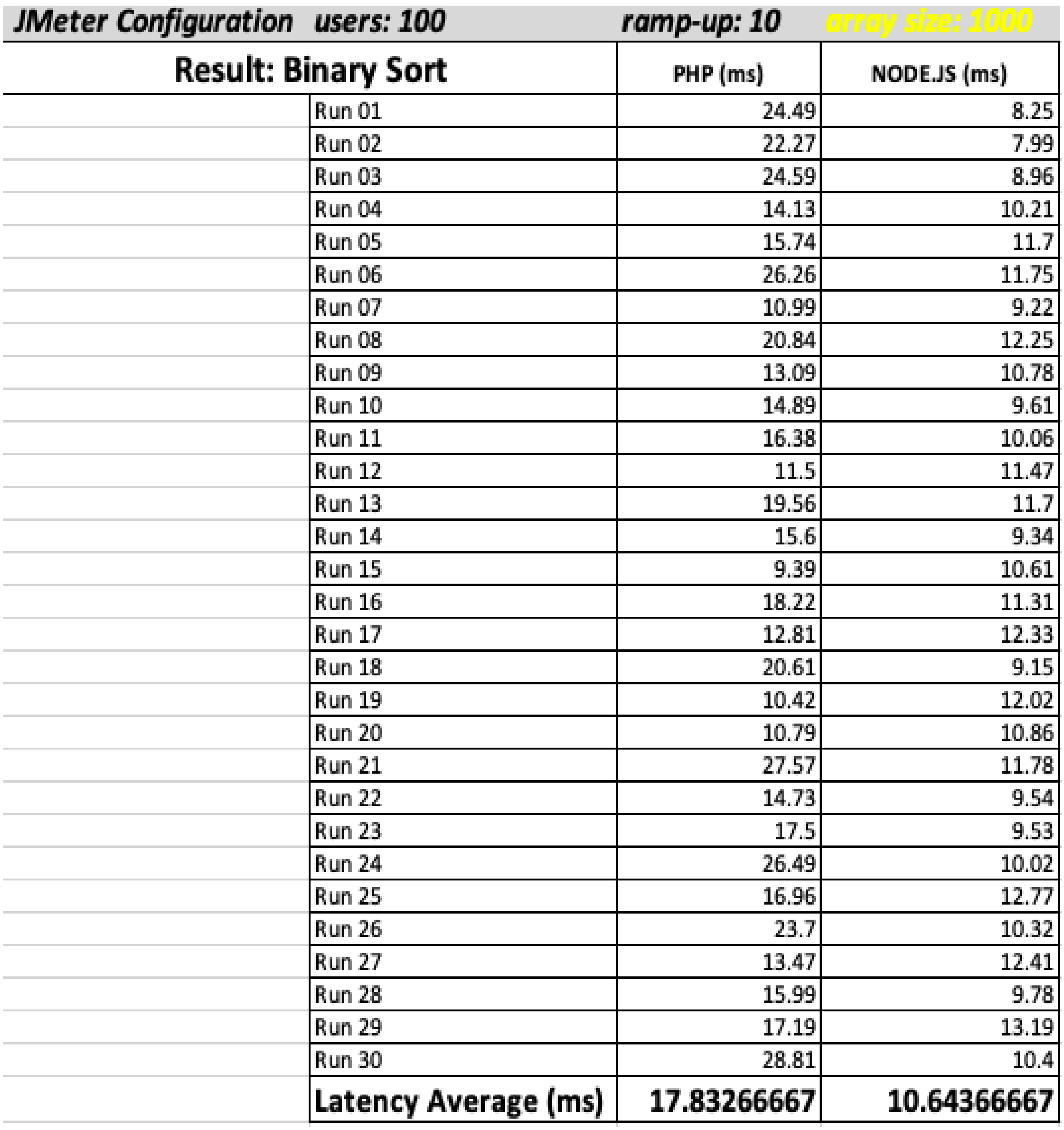
4.4.1.3 Descriptive:

We summarized and described the collected data in terms of central tendency in order to provide a clear and concise picture of the performance of PHP and Node.js such a mean.



JMeter Configuration	*users: 100*	*ramp-up: 10*	*array size: 100*
Result: Bubble Sort		**PHP (ms)**	**NODE.JS (ms)**
	Run 01	17.07	9.39
	Run 02	16.82	11.96
	Run 03	16.98	10.64
	Run 04	13.1	11.9
	Run 05	19.54	8.51
	Run 06	15.81	9.99
	Run 07	10.3	9.2
	Run 08	18.16	9.5
	Run 09	12.42	8.52
	Run 10	18.58	8.24
	Run 11	19.92	12.74
	Run 12	19.88	7.54
	Run 13	15.6	6.64
	Run 14	14.45	6.89
	Run 15	17.15	7.21
	Run 16	17.55	8.16
	Run 17	13.93	7.7
	Run 18	15.8	7.96
	Run 19	15.2	9.86
	Run 20	14.8	7.3
	Run 21	13.07	11.12
	Run 22	19.84	7.88
	Run 23	14.95	9.11
	Run 24	10.72	6.61
	Run 25	13.71	7.48
	Run 26	12.24	9.48
	Run 27	16.69	8.6
	Run 28	6.81	7.85
	Run 29	16.64	8.76
	Run 30	12.81	6.52
	Latency Average (ms)	**15.35133333**	**8.775333333**

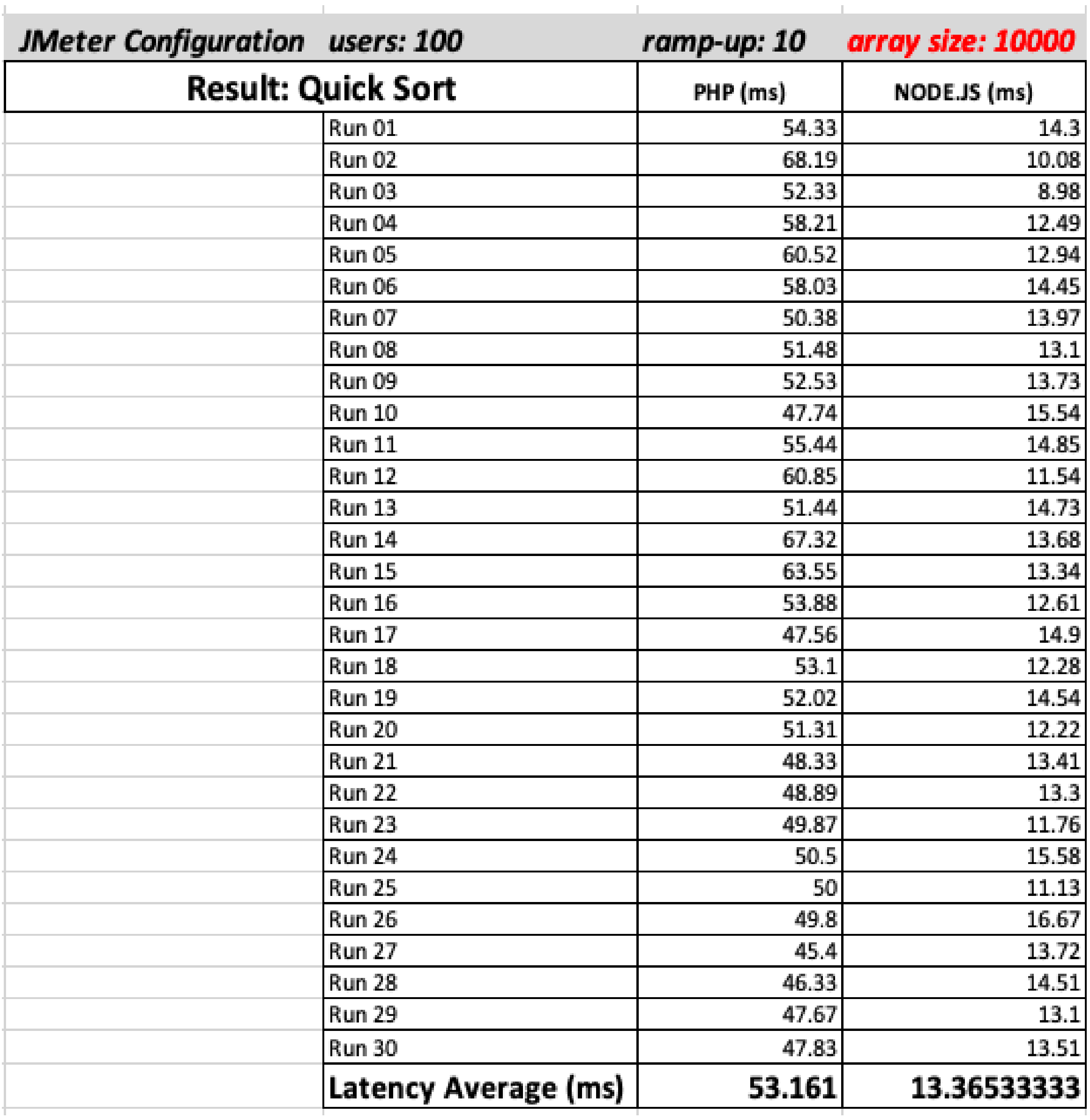
Figure 4.15: Bubble Sort result for 30 runs and Latency Average



JMeter Configuration *users: 100* *ramp-up: 10* *array size: 1000*

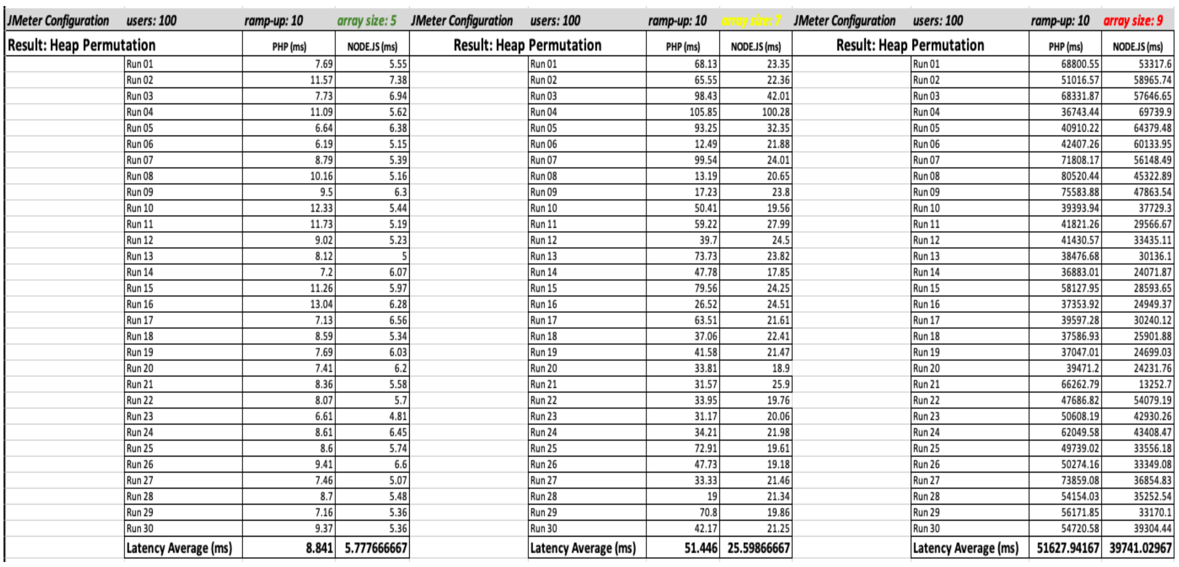
Result: Binary Sort	**PHP (ms)**	**NODE.JS (ms)**
Run 01	24.49	8.25
Run 02	22.27	7.99
Run 03	24.59	8.96
Run 04	14.13	10.21
Run 05	15.74	11.7
Run 06	26.26	11.75
Run 07	10.99	9.22
Run 08	20.84	12.25
Run 09	13.09	10.78
Run 10	14.89	9.61
Run 11	16.38	10.06
Run 12	11.5	11.47
Run 13	19.56	11.7
Run 14	15.6	9.34
Run 15	9.39	10.61
Run 16	18.22	11.31
Run 17	12.81	12.33
Run 18	20.61	9.15
Run 19	10.42	12.02
Run 20	10.79	10.86
Run 21	27.57	11.78
Run 22	14.73	9.54
Run 23	17.5	9.53
Run 24	26.49	10.02
Run 25	16.96	12.77
Run 26	23.7	10.32
Run 27	13.47	12.41
Run 28	15.99	9.78
Run 29	17.19	13.19
Run 30	28.81	10.4
Latency Average (ms)	**17.83266667**	**10.64366667**

Figure 4.16: Binary Sort result for 30 runs and Latency Average



JMeter Configuration	*users: 100*	*ramp-up: 10*	*array size: 10000*
Result: Quick Sort		PHP (ms)	NODE.JS (ms)
	Run 01	54.33	14.3
	Run 02	68.19	10.08
	Run 03	52.33	8.98
	Run 04	58.21	12.49
	Run 05	60.52	12.94
	Run 06	58.03	14.45
	Run 07	50.38	13.97
	Run 08	51.48	13.1
	Run 09	52.53	13.73
	Run 10	47.74	15.54
	Run 11	55.44	14.85
	Run 12	60.85	11.54
	Run 13	51.44	14.73
	Run 14	67.32	13.68
	Run 15	63.55	13.34
	Run 16	53.88	12.61
	Run 17	47.56	14.9
	Run 18	53.1	12.28
	Run 19	52.02	14.54
	Run 20	51.31	12.22
	Run 21	48.33	13.41
	Run 22	48.89	13.3
	Run 23	49.87	11.76
	Run 24	50.5	15.58
	Run 25	50	11.13
	Run 26	49.8	16.67
	Run 27	45.4	13.72
	Run 28	46.33	14.51
	Run 29	47.67	13.1
	Run 30	47.83	13.51
	Latency Average (ms)	**53.161**	**13.36533333**

Figure 4.17: Quick Sort result for 30 runs and Latency Average



JMeter Configuration	*users: 100*	*ramp-up: 10*	*array size: 5*
Result: Heap Permutation		PHP (ms)	NODE.JS (ms)
	Run 01	7.69	5.55
	Run 02	11.57	7.38
	Run 03	7.73	6.94
	Run 04	11.09	5.62
	Run 05	6.64	6.38
	Run 06	6.19	5.15
	Run 07	8.79	5.39
	Run 08	10.16	5.16
	Run 09	9.5	6.3
	Run 10	12.33	5.44
	Run 11	11.73	5.19
	Run 12	9.02	5.23
	Run 13	8.12	5
	Run 14	7.2	6.07
	Run 15	11.26	5.97
	Run 16	13.04	6.28
	Run 17	7.13	6.56
	Run 18	8.59	5.34
	Run 19	7.69	6.03
	Run 20	7.41	6.2
	Run 21	8.36	5.58
	Run 22	8.07	5.7
	Run 23	6.61	4.81
	Run 24	8.61	6.45
	Run 25	8.6	5.74
	Run 26	9.41	6.6
	Run 27	7.46	5.07
	Run 28	8.7	5.48
	Run 29	7.16	5.36
	Run 30	9.37	5.36
	Latency Average (ms)	**8.841**	**5.777666667**

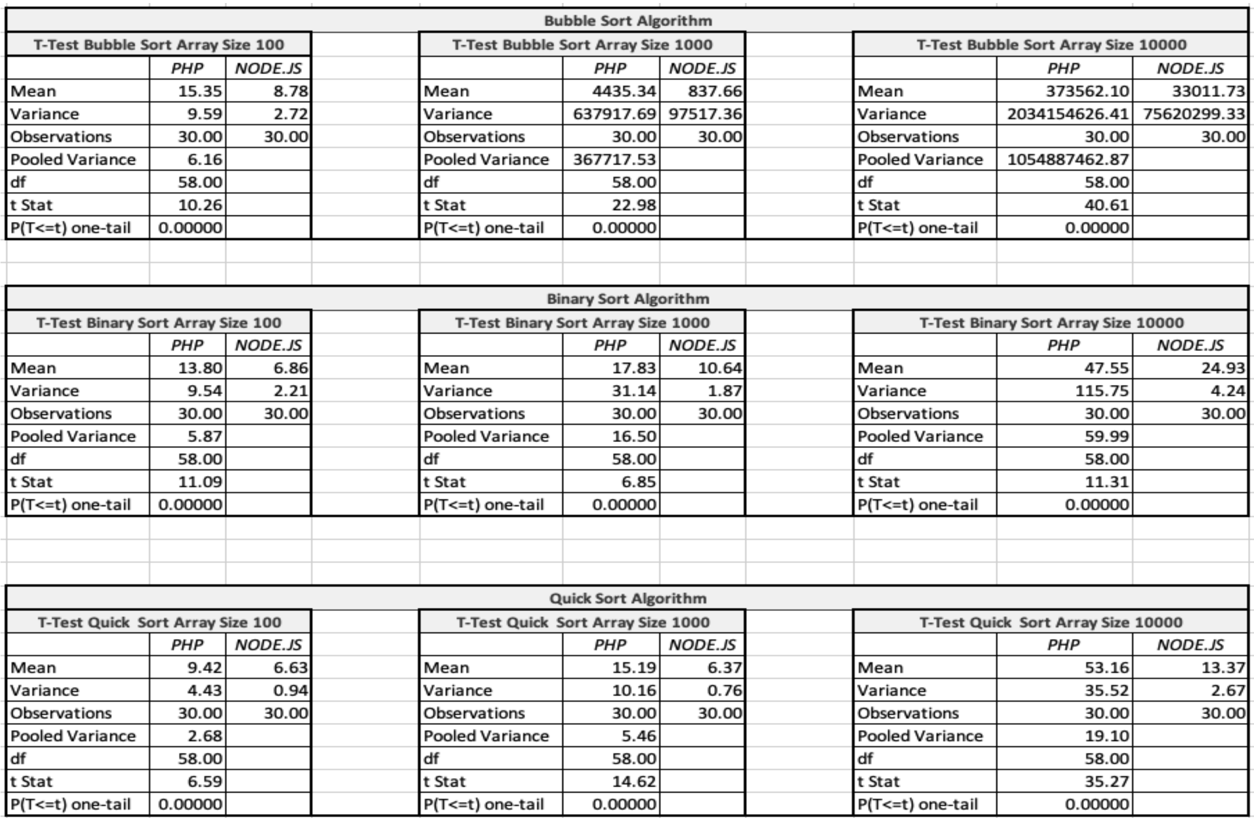
JMeter Configuration	*users: 100*	*ramp-up: 10*	*array size: 7*
Result: Heap Permutation		PHP (ms)	NODE.JS (ms)
	Run 01	68.13	23.35
	Run 02	65.55	22.36
	Run 03	98.43	42.01
	Run 04	105.85	100.28
	Run 05	93.25	32.35
	Run 06	12.49	21.88
	Run 07	99.54	24.01
	Run 08	13.19	20.65
	Run 09	17.23	23.8
	Run 10	50.41	19.56
	Run 11	59.22	27.99
	Run 12	39.7	24.5
	Run 13	73.73	23.82
	Run 14	47.78	17.85
	Run 15	79.56	24.25
	Run 16	26.52	24.51
	Run 17	63.51	21.61
	Run 18	37.06	22.41
	Run 19	41.58	21.47
	Run 20	33.81	18.9
	Run 21	31.57	25.9
	Run 22	33.95	19.76
	Run 23	31.17	20.06
	Run 24	34.21	21.98
	Run 25	72.91	19.61
	Run 26	47.73	19.18
	Run 27	33.33	21.46
	Run 28	19	21.34
	Run 29	70.8	19.86
	Run 30	42.17	21.25
	Latency Average (ms)	**51.446**	**25.59866667**

JMeter Configuration	*users: 100*	*ramp-up: 10*	*array size: 9*
Result: Heap Permutation		PHP (ms)	NODE.JS (ms)
	Run 01	68800.55	53317.6
	Run 02	51016.57	58965.74
	Run 03	68331.87	57646.65
	Run 04	36743.44	69739.9
	Run 05	40910.22	64379.48
	Run 06	42407.26	60133.95
	Run 07	71808.17	56148.49
	Run 08	80520.44	45322.89
	Run 09	75583.88	47863.54
	Run 10	39393.94	37729.3
	Run 11	41821.26	29566.67
	Run 12	41430.57	33435.11
	Run 13	38476.68	30136.1
	Run 14	36883.01	24071.87
	Run 15	58127.95	28593.65
	Run 16	37353.92	24949.37
	Run 17	39597.28	30240.12
	Run 18	37586.93	25901.88
	Run 19	37047.01	24699.03
	Run 20	39471.2	24231.76
	Run 21	66262.79	13252.7
	Run 22	47686.82	54079.19
	Run 23	50608.19	42930.26
	Run 24	62049.58	43408.47
	Run 25	49739.02	33556.18
	Run 26	50274.16	33349.08
	Run 27	73859.08	36854.83
	Run 28	54154.03	35252.54
	Run 29	56171.85	33170.1
	Run 30	54720.58	39304.44
	Latency Average (ms)	**51627.94167**	**39741.02967**

Figure 4.18: Heap Algorithm result for 30 runs and Latency Average

4.4.1.4 Inferential

We utilized this to make inferences and drew conclusions on our collected data. We used t-test as our inferential statistical method to determine if there is a significant difference in the performance between PHP and Node.js. As seen in our t-test results in figures 4.19 – 4.20, p-value is less than 0.05, hence we reject the null hypothesis and concluded that a significant difference occurs between the performance of PHP and Node.js.



Bubble Sort Algorithm

T-Test Bubble Sort Array Size 100

	PHP	NODE.JS
Mean	15.35	8.78
Variance	9.59	2.72
Observations	30.00	30.00
Pooled Variance	6.16	
df	58.00	
t Stat	10.26	
P(T<=t) one-tail	0.00000	

T-Test Bubble Sort Array Size 1000

	PHP	NODE.JS
Mean	4435.34	837.66
Variance	637917.69	97517.36
Observations	30.00	30.00
Pooled Variance	367717.53	
df	58.00	
t Stat	22.98	
P(T<=t) one-tail	0.00000	

T-Test Bubble Sort Array Size 10000

	PHP	NODE.JS
Mean	373562.10	33011.73
Variance	2034154626.41	75620299.33
Observations	30.00	30.00
Pooled Variance	1054887462.87	
df	58.00	
t Stat	40.61	
P(T<=t) one-tail	0.00000	

Binary Sort Algorithm

T-Test Binary Sort Array Size 100

	PHP	NODE.JS
Mean	13.80	6.86
Variance	9.54	2.21
Observations	30.00	30.00
Pooled Variance	5.87	
df	58.00	
t Stat	11.09	
P(T<=t) one-tail	0.00000	

T-Test Binary Sort Array Size 1000

	PHP	NODE.JS
Mean	17.83	10.64
Variance	31.14	1.87
Observations	30.00	30.00
Pooled Variance	16.50	
df	58.00	
t Stat	6.85	
P(T<=t) one-tail	0.00000	

T-Test Binary Sort Array Size 10000

	PHP	NODE.JS
Mean	47.55	24.93
Variance	115.75	4.24
Observations	30.00	30.00
Pooled Variance	59.99	
df	58.00	
t Stat	11.31	
P(T<=t) one-tail	0.00000	

Quick Sort Algorithm

T-Test Quick Sort Array Size 100

	PHP	NODE.JS
Mean	9.42	6.63
Variance	4.43	0.94
Observations	30.00	30.00
Pooled Variance	2.68	
df	58.00	
t Stat	6.59	
P(T<=t) one-tail	0.00000	

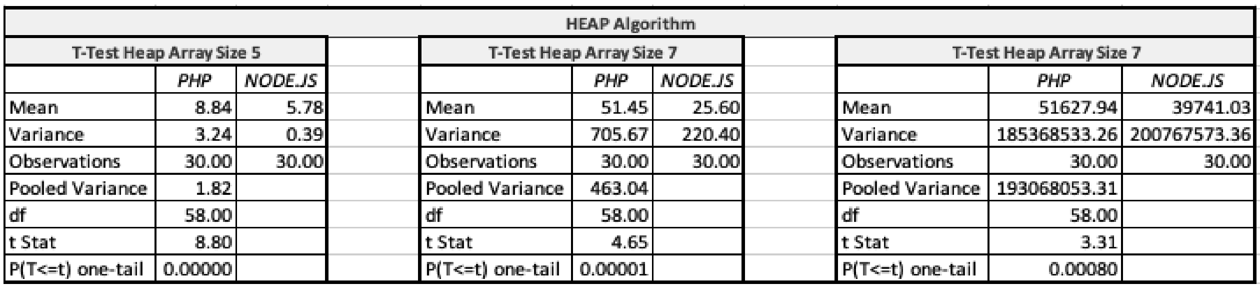
T-Test Quick Sort Array Size 1000

	PHP	NODE.JS
Mean	15.19	6.37
Variance	10.16	0.76
Observations	30.00	30.00
Pooled Variance	5.46	
df	58.00	
t Stat	14.62	
P(T<=t) one-tail	0.00000	

T-Test Quick Sort Array Size 10000

	PHP	NODE.JS
Mean	53.16	13.37
Variance	35.52	2.67
Observations	30.00	30.00
Pooled Variance	19.10	
df	58.00	
t Stat	35.27	
P(T<=t) one-tail	0.00000	

Figure 4.19: Sorting Algorithm results for t-test



HEAP Algorithm

T-Test Heap Array Size 5

	PHP	NODE.JS
Mean	8.84	5.78
Variance	3.24	0.39
Observations	30.00	30.00
Pooled Variance	1.82	
df	58.00	
t Stat	8.80	
P(T<=t) one-tail	0.00000	

T-Test Heap Array Size 7

	PHP	NODE.JS
Mean	51.45	25.60
Variance	705.67	220.40
Observations	30.00	30.00
Pooled Variance	463.04	
df	58.00	
t Stat	4.65	
P(T<=t) one-tail	0.00001	

T-Test Heap Array Size 7

	PHP	NODE.JS
Mean	51627.94	39741.03
Variance	185368533.26	200767573.36
Observations	30.00	30.00
Pooled Variance	193068053.31	
df	58.00	
t Stat	3.31	
P(T<=t) one-tail	0.00080	

Figure 4.20: Heap Algorithm results for t-test

4.4.2 Limitations

In this study, we had couples of limitations as discussed below:

I. Variability in performance – for the sorting algorithms, we limited our array to 100, 1000, and 10000. While for the Heap algorithm, our array size was limited to 5, 7 and 9. In both PHP and Node.js cases, the implemented algorithms behaved differently as we vary the array size.

II. Scope of study – we have limited our performance to sorting and heap algorithms which may not be generalizable to other types of software or applications. Also, we limited performance metric to latency which may not capture all aspects of system performance.

4.4.3 Future Research

Regarding our accomplished results and limitations of the study, there are several areas for future research which include considering additional sorting algorithms and permutation algorithms to determine performance difference in PHP and Node.js. Include other backend technologies and compare their performance to that of PHP and Node.js. Furthermore, researchers can examine the impact of various hardware configurations on the performance of PHP and Node.js. Lastly, Investigating the relationship between the performance and other important factors like development time, scalability, and maintainability.

4.4.4 Discussion

Our research study was focused on comparing the performance of PHP (old backend technology) and Node.js (new backend technology) through the implementation of different sorting algorithms and a heap permutation to generate all possible permutation of a given array size. We measured performance using a load testing software, Apache JMeter and analyzed collected data using t-test statistical method. Our results showed that a significant difference occur in the performance of PHP and Node.js. To be precise, Node.js outperformed PHP in terms of latency and other performance metrics. These findings proposed that the new backend technology, Node.js may be a better option for developing backend systems which required high performance. It is however important we state that this study had some limitations as we only considered relatively small sample size and focused on specific sorting algorithms and a single heap algorithm. Future researcher could expatiate on this study by testing a larger sample size, considering other sorting and heap algorithms, and explore other backend technology performance. Lastly, our research proffered valuable insights into the performance of PHP and Node.js and can be helpful in the decision-making process when selecting a backend scripting technology for developing a high-performance system.

4.5 CONCLUSION

In conclusion, our research study aimed at performance comparison between PHP and Node.js, which are two prominent backend technologies. We implemented various complex sorting algorithms and a heap algorithm to generate all possible permutations with varying array sizes. Our tests were ran using Apache JMeter and the T-Test statistical method was used in analyzing the collected data. Our results showed that Node.js performed better than PHP in all the tests, with notable statistically significant difference between the two backend scripting technologies.

Our study will contribute to the existing literature on backend technologies and will provide valuable insights not just for developers but also organizations in choosing the suitable backend scripting technology on project needs. The newer technology, Node.js portrayed a superior performance compared to the PHP, the old one. However, it is important we state that this study has few limitations as we could only consider small sample size and specific algorithms tested. Future researchers are advised to expatiate on this study by considering more algorithms and a larger sample size. Despite that, our study provides valuable contribution to the field of backend technologies and highlights the significance of considering performance while choosing a technology.

CHAPTER 5

STUDY C – Database Technology Performance Evaluation: SQL vs. NoSQL

5.1 INTRODUCTION

Data management has seen considerable changes because of technology over time, including the introduction of databases to streamline the procedure. These databases, which can range from straightforward text documents to complex ones, need to be periodically fine-tuned to get rid of information that is redundant, inconsistent, or inaccurate [43]. The decision between SQL and NoSQL, is critical in deciding the effectiveness and performance of various database operations in the constantly changing world of database systems. Both SQL and NoSQL databases have distinctive feature and functionality sets that make them suitable for various use cases and scenarios. It is crucial to figure out which of these two database technologies shines in terms of speed and performance across different database activities. Big data, which consists of a sizable number of heterogeneous data, is expanding quickly, needing new methods for data storage, organisation, performance, and analysis. Due to the difficulty traditional relational databases have managing this enormous volume of data, Big Data Analytics and NoSQL systems have been developed for better decision-making and business value [44].

The mainstays of structured data storage for years have been SQL databases, which provide high data consistency and powerful querying capability. Conversely, NoSQL databases, with MongoDB at the forefront, have emerged as dynamic, flexible alternatives capable of managing significant amounts of unstructured or semi-structured data while offering horizontal scalability. In our study, we compared the performance of SQL and NoSQL (MongoDB) database systems in terms of time spent (elapsed time) performing a variety of typical database operations, including insert, select, update, delete, and aggregate (average). Our study's main goal is to provide a thorough assessment of these two different database technologies' performance capabilities when managing massive datasets. We used the T-Test, a statistical approach well known for determining significant differences between two supplied samples, to analyse the data obtained from Apache JMeter. T-statistics are the results of this statistical process. We hope that the research's findings will provide insightful information that will help businesses, data analysts, and software developers make wise adoption decisions.

5.2 METHODOLOGY

5.2.1 Research Problem and Purpose of Study

By implementing and comparing five fundamental database operations (Insert, Select, Update, Delete, and Aggregate, specifically average) in both SQL and NoSQL environments, the research problem aims to understand the subtle efficiency differences between SQL and NoSQL (MongoDB) database management systems (DBMS). This research study's main purpose is to examine the performance and efficiency differences between SQL and NoSQL (MongoDB) DBMS, with a particular emphasis on five crucial database operations: Insert, Select, Update, Delete, and Aggregate (average).

5.2.2 Performance Goal, Statement of Research Objectives and Research Questions

5.2.2.1 Performance Goal

The goal of the study is to provide empirical data and thorough insights into the performance differences between SQL and NoSQL databases, especially regarding important database operations. This research equips decision-makers to make well-informed decisions about database technology selection and setup, thereby improving operational efficiency and resource utilization. It does this by identifying when and why one DBMS performs better than the other in particular scenarios.

5.2.3 Research Objectives: In this research, our objectives are as follows

I. To thoroughly compare the effectiveness of SQL and NoSQL (MongoDB) DBMS in carrying out the essential database operations, revealing differences in latencies, elapsed times, execution times, resource usage, and general effectiveness.
II. To compare the relative performance of SQL and NoSQL databases across various activities, it is necessary to consider variations in workload size, query complexity, and concurrent user access patterns.
III. To examine the responsiveness and efficiency of both SQL and NoSQL databases as data volume and user concurrency grow to determine their respective scalability constraints.

IV. To provide database managers and developers with useful advice, it is necessary to define optimization techniques and best practices for improving the performance of SQL and NoSQL databases for processes.
V. To assist practitioners in making intelligent technological decisions by identifying which database system (SQL or NoSQL) displays superior performance for specific database activities and under which contextual settings.
VI. To offer organizations, developers, and data professionals' useful information and suggestions for selecting the best DBMS for their unique operational needs, enhancing system performance, and maximizing resource allocation.

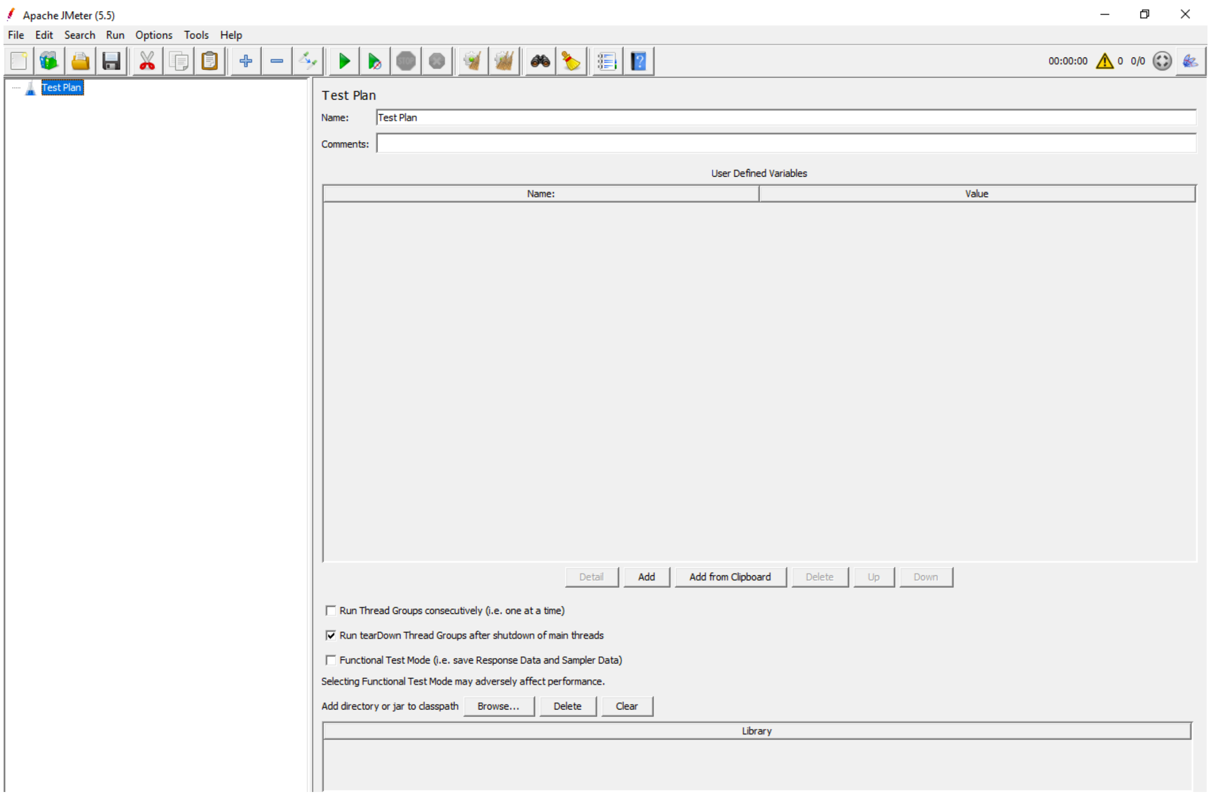
5.2.3.1 Research Questions

What are the performance differences for the key database operations (Insert, Select, Update, Delete, and Aggregate - average) in the context of database management systems, specifically SQL and NoSQL (MongoDB), and under what circumstances does one outperform the other, thus providing crucial insights into the choice between SQL and NoSQL for specific database operations in various real-world scenarios?

- RQ1 – which of SQL and NoSQL is faster in Insert, Select, Delete, Update and Aggregate database operations?
- RQ2 – Under which conditions is SQL vs NoSQL experience higher performance?

5.2.3.2 Description of Research Design, SQL & NoSQL, DBMS used, and Performance Metrics.

The dispute between SQL and NoSQL has persisted in the world of database management systems (DBMS), with each having certain benefits and limitations. This study used a well-structured design, Apache JMeter as the performance testing tool, and important performance metrics to reveal their relative efficacy. We give a thorough explanation of the research methodology, DBMS selection, and performance indicators used in the study below.



Test Plan

Name: Test Plan

Comments:

☐ Run Thread Groups consecutively (i.e. one at a time)

☑ Run tearDown Thread Groups after shutdown of main threads

☐ Functional Test Mode (i.e. save Response Data and Sampler Data)

Selecting Functional Test Mode may adversely affect performance.

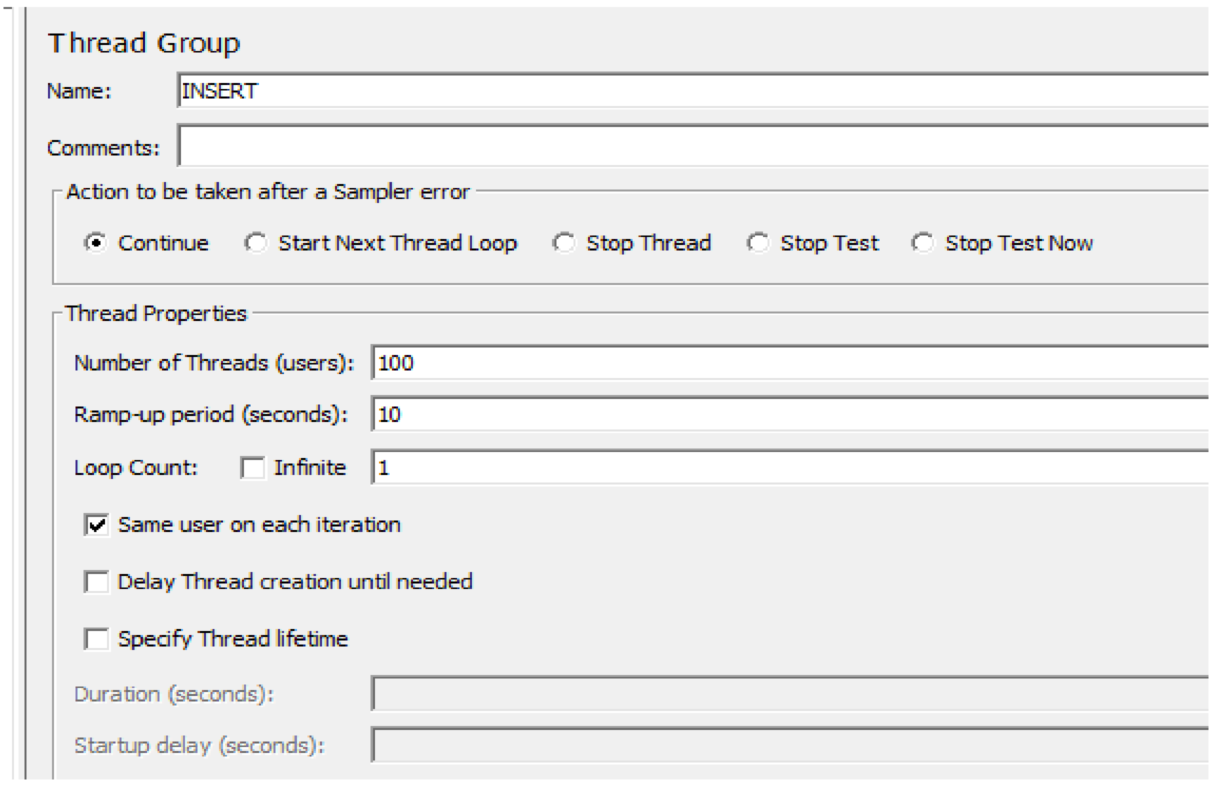
Add directory or jar to classpath

Figure 5.1: Apache JMeter

5.2.3.3 Research Design

The research methodology was carefully developed to compare the performance of SQL and NoSQL databases for five essential database operations: Insert, Select, Update, Delete, and Aggregate (more particularly, computing the average). To facilitate a thorough comparison, these operations were carried out under carefully monitored circumstances.

Insert Operation: Apache JMeter was used in the research with a setup of 100 users and a ramp-up time of 10 seconds for this operation. 10,000 rows of data were inserted into each database. To verify statistical reliability, this procedure was repeated 30 times.


Thread Group

Name: INSERT

Comments:

Action to be taken after a Sampler error

[✓ Continue] Start Next Thread Loop Stop Thread Stop Test Stop Test Now

Thread Properties

Number of Threads (users): 100

Ramp-up period (seconds): 10

Loop Count: ☐ Infinite 1

☑ Same user on each iteration

☐ Delay Thread creation until needed

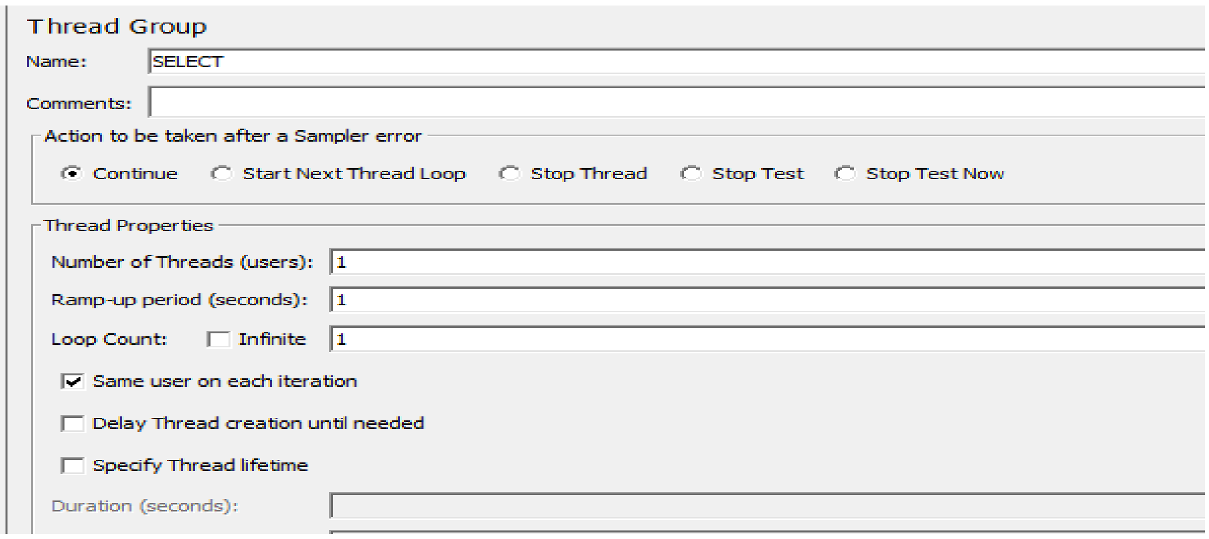
☐ Specify Thread lifetime

Duration (seconds):

Startup delay (seconds):

Figure 5.2: Apache JMeter showing the INSERT OPERATION thread group

Select Operation: Apache JMeter was set up for the Select operation with a single user and a one-second ramp-up time. To choose 300,000 rows of data from each database was the goal. This was performed 30 times, much as the Insert procedure, for full examination.


Thread Group

Name: SELECT

Comments:

Action to be taken after a Sampler error

[✓ Continue] Start Next Thread Loop Stop Thread Stop Test Stop Test Now

Thread Properties

Number of Threads (users): 1

Ramp-up period (seconds): 1

Loop Count: ☐ Infinite 1

☑ Same user on each iteration

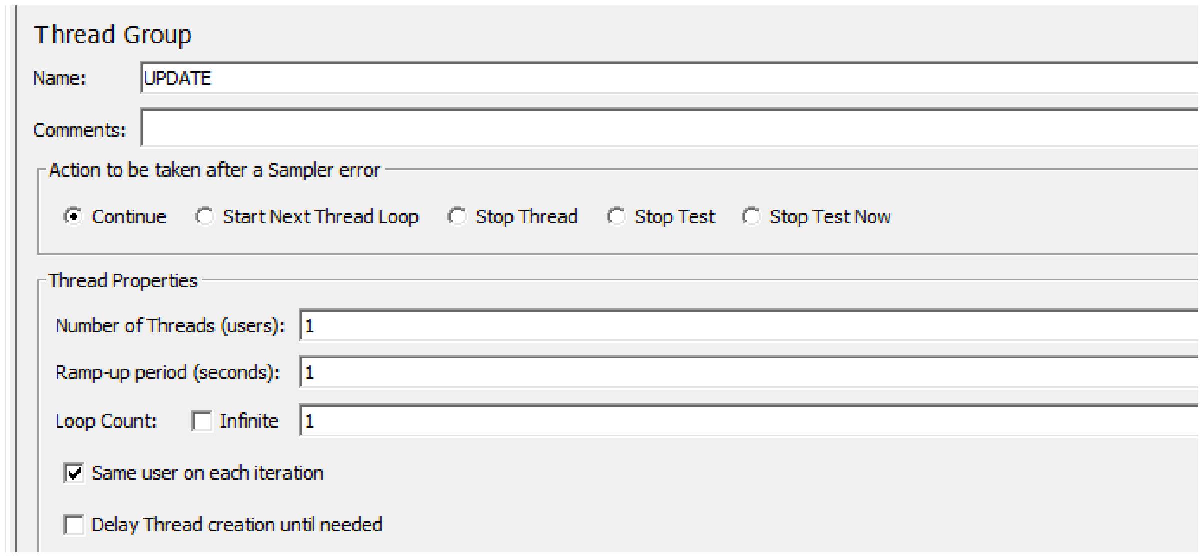
☐ Delay Thread creation until needed

☐ Specify Thread lifetime

Duration (seconds):

Figure 5.3: Apache JMeter showing the SELECT OPERATION thread group.

Update Operation: JMeter was set up with a single user and a ramp-up time of one second for updating data. 50 rows of data were added to the databases at once. To identify performance trends, this procedure was performed 30 times.



Thread Group

Name: UPDATE

Comments:

Action to be taken after a Sampler error

[✓ Continue] Start Next Thread Loop Stop Thread Stop Test Stop Test Now

Thread Properties

Number of Threads (users): 1

Ramp-up period (seconds): 1

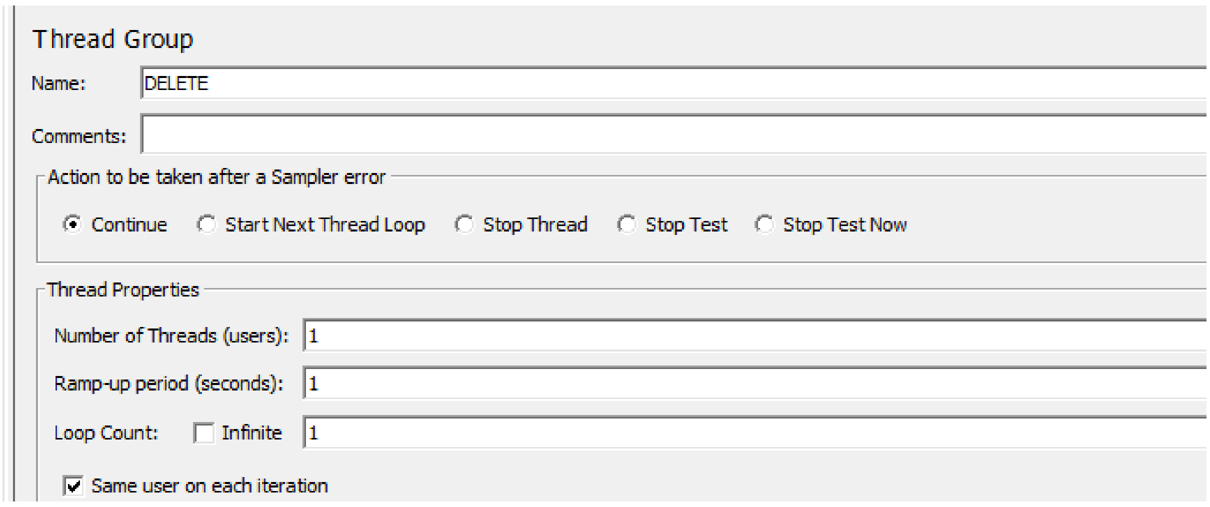
Loop Count: ☐ Infinite 1

☑ Same user on each iteration

☐ Delay Thread creation until needed

Figure 5.4: Apache JMeter showing the UPDATE OPERATION thread group

Delete Operation: JMeter was used for the delete operation, which had a ramp-up time of one second and only one user. It was designed to remove a sizable dataset—300,000 entries—from each database all at once.



Thread Group

Name: DELETE

Comments:

Action to be taken after a Sampler error

[✓ Continue] Start Next Thread Loop Stop Thread Stop Test Stop Test Now

Thread Properties

Number of Threads (users): 1

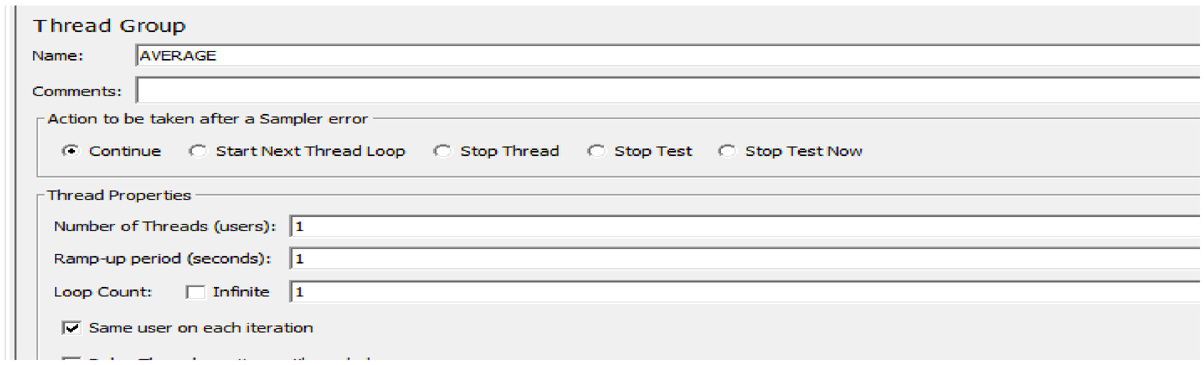
Ramp-up period (seconds): 1

Loop Count: ☐ Infinite 1

☑ Same user on each iteration

Figure 5.5: Apache JMeter showing the DELETE OPERATION thread group

Aggregate (Average) Operation: Lastly, JMeter was set up with a single user and a ramp-up time of one second for the aggregate operation (calculating the Population Average). The aim was to calculate the average for each database's 3,000,000 cities.



Thread Group

Name: AVERAGE

Comments:

Action to be taken after a Sampler error

[✓ Continue] Start Next Thread Loop Stop Thread Stop Test Stop Test Now

Thread Properties

Number of Threads (users): 1

Ramp-up period (seconds): 1

Loop Count: Infinite 1

[✓ Same user on each iteration]

Figure 5.6: Apache JMeter showing the AGGREGATE OPERATION thread group

5.2.3.4 Choice of SQL & NoSQL DBMS

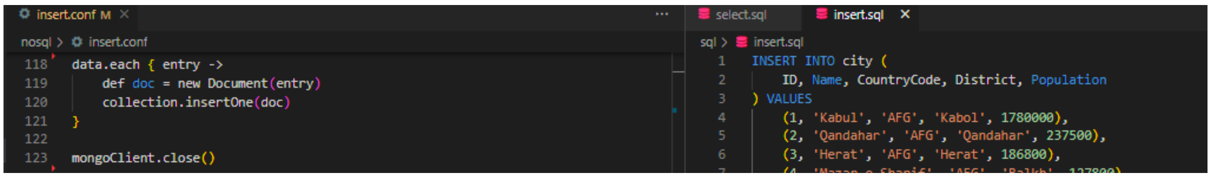
The relational and document-oriented database paradigms were represented in the study by the SQL and NoSQL (MongoDB) databases, respectively. SQL was represented by a well-liked DBMS, which is notable for its ability to store structured and tabular data. Because of its famed flexibility and scalability, MongoDB, a leading example of NoSQL databases, is an excellent choice for the comparison study.

5.2.3.5 Performance Metrics with an Emphasis on "Elapsed"

Performance metrics are essential for evaluating a DBMS's effectiveness across a range of operations. The "Elapsed" metric, which measures the overall amount of time required to complete an operation, was the focus of this study. Elapsed time, which includes the time needed for query processing, data retrieval, and system response, is a crucial metric of system effectiveness. Better performance is indicated by a shorter elapsed time.

5.2.3.5.1 Elapsed Time's Relevance to DBMS Performance

Insertion Operation: A faster elapsed time shows that the DBMS handles data insertion effectively, reducing the time used to add new records to the database. Lower elapsed periods suggest greater effectiveness in this situation for both SQL and NoSQL.

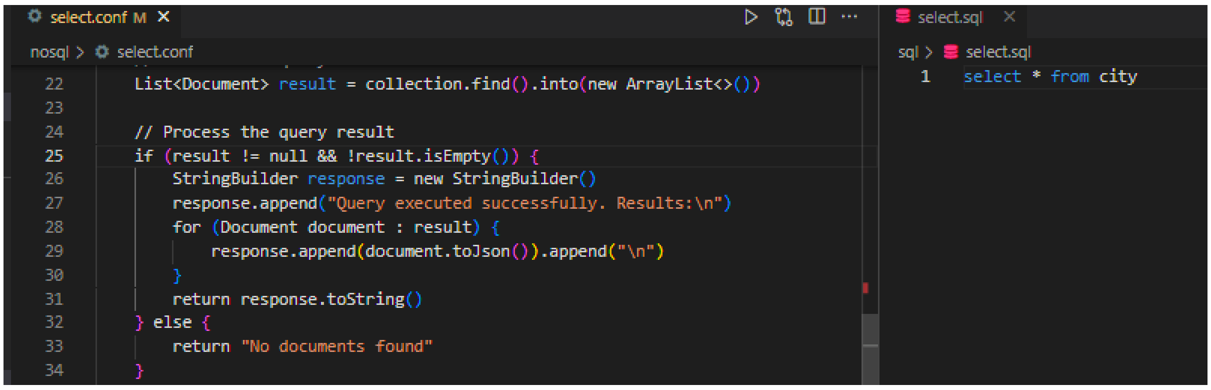


```
data.each { entry ->
    def doc = new Document(entry)
    collection.insertOne(doc)
}

mongoClient.close()
```

```
INSERT INTO city (
    ID, Name, CountryCode, District, Population
) VALUES
    (1, 'Kabul', 'AFG', 'Kabol', 1780000),
    (2, 'Qandahar', 'AFG', 'Qandahar', 237500),
    (3, 'Herat', 'AFG', 'Herat', 186800),
```

Figure 5.7: VS Code showing NoSQL and SQL queries for INSERT operations

Select Operation: Lower elapsed durations for the Select procedure indicate that the DBMS can quickly retrieve data from substantial datasets. Better database performance is indicated by faster query response times.

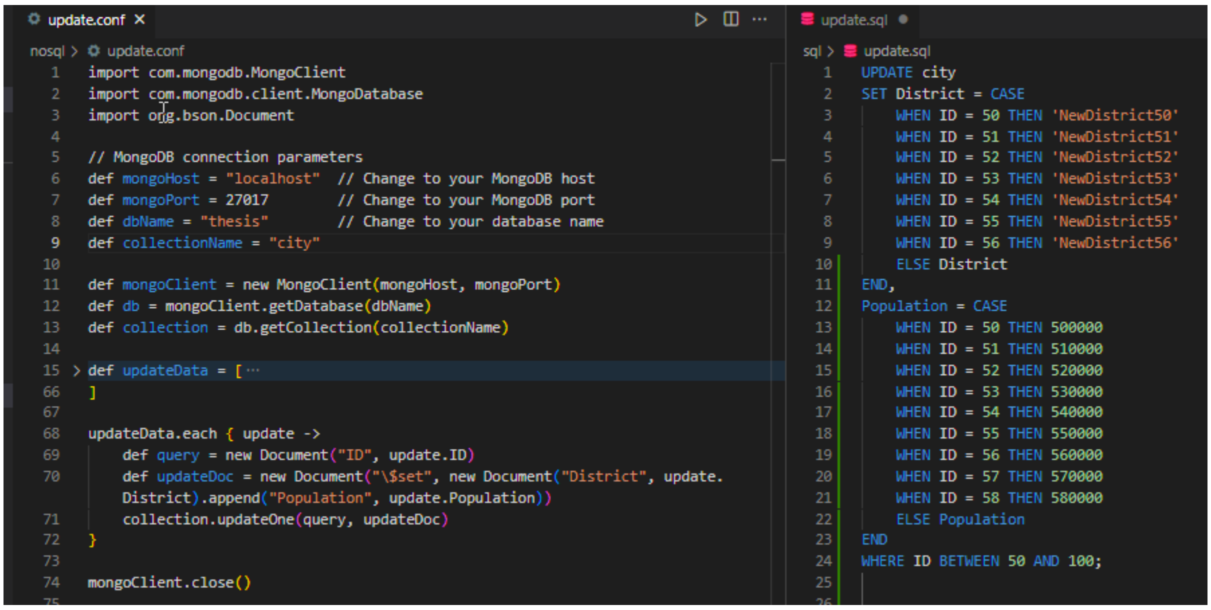


```
    List<Document> result = collection.find().into(new ArrayList<>())

    // Process the query result
    if (result != null && !result.isEmpty()) {
        StringBuilder response = new StringBuilder()
        response.append("Query executed successfully. Results:\n")
        for (Document document : result) {
            response.append(document.toJson()).append("\n")
        }
        return response.toString()
    } else {
        return "No documents found"
    }
```

```
select * from city
```

Figure 5.8: VS Code showing NoSQL and SQL queries for SELECT operations.

Update Operation: Shorter update times show that the DBMS can efficiently change existing entries. Lower times indicate that the system properly manages data revisions.



update.conf

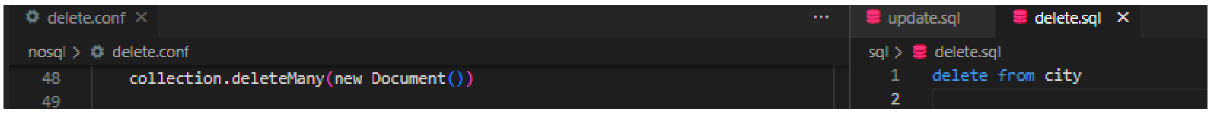
```
import com.mongodb.MongoClient
import com.mongodb.client.MongoDatabase
import org.bson.Document

// MongoDB connection parameters
def mongoHost = "localhost"  // Change to your MongoDB host
def mongoPort = 27017        // Change to your MongoDB port
def dbName = "thesis"        // Change to your database name
def collectionName = "city"

def mongoClient = new MongoClient(mongoHost, mongoPort)
def db = mongoClient.getDatabase(dbName)
def collection = db.getCollection(collectionName)

def updateData = [⋯
]

updateData.each { update ->
    def query = new Document("ID", update.ID)
    def updateDoc = new Document("\$set", new Document("District", update.
    District).append("Population", update.Population))
    collection.updateOne(query, updateDoc)
}

mongoClient.close()
```

update.sql

```
UPDATE city
SET District = CASE
    WHEN ID = 50 THEN 'NewDistrict50'
    WHEN ID = 51 THEN 'NewDistrict51'
    WHEN ID = 52 THEN 'NewDistrict52'
    WHEN ID = 53 THEN 'NewDistrict53'
    WHEN ID = 54 THEN 'NewDistrict54'
    WHEN ID = 55 THEN 'NewDistrict55'
    WHEN ID = 56 THEN 'NewDistrict56'
    ELSE District
END,
Population = CASE
    WHEN ID = 50 THEN 500000
    WHEN ID = 51 THEN 510000
    WHEN ID = 52 THEN 520000
    WHEN ID = 53 THEN 530000
    WHEN ID = 54 THEN 540000
    WHEN ID = 55 THEN 550000
    WHEN ID = 56 THEN 560000
    WHEN ID = 57 THEN 570000
    WHEN ID = 58 THEN 580000
    ELSE Population
END
WHERE ID BETWEEN 50 AND 100;
```

Figure 5.9: VS Code showing NoSQL and SQL queries for UPDATE operations.

Delete Operation: Shorter elapsed periods during the Delete operation signify effective data removal, reducing system downtime during bulk deletion operations.



delete.conf

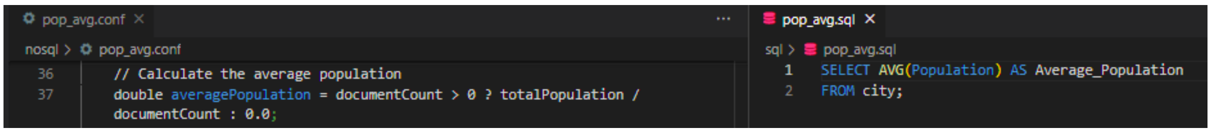
```
    collection.deleteMany(new Document())
```

delete.sql

```
delete from city
```

Figure 5.10: VS Code showing NoSQL and SQL queries for UPDATE operations.

Aggregate (Average) Operation: For the Aggregate operation, shorter elapsed times signal efficient data aggregation and calculation of the average. Lower times indicate that the DBMS can process large volumes of data swiftly.



pop_avg.conf

```
    // Calculate the average population
    double averagePopulation = documentCount > 0 ? totalPopulation /
    documentCount : 0.0;
```

pop_avg.sql

```
SELECT AVG(Population) AS Average_Population
FROM city;
```

Figure 5.11: VS Code showing NoSQL and SQL queries for Aggregate (Average) operations.

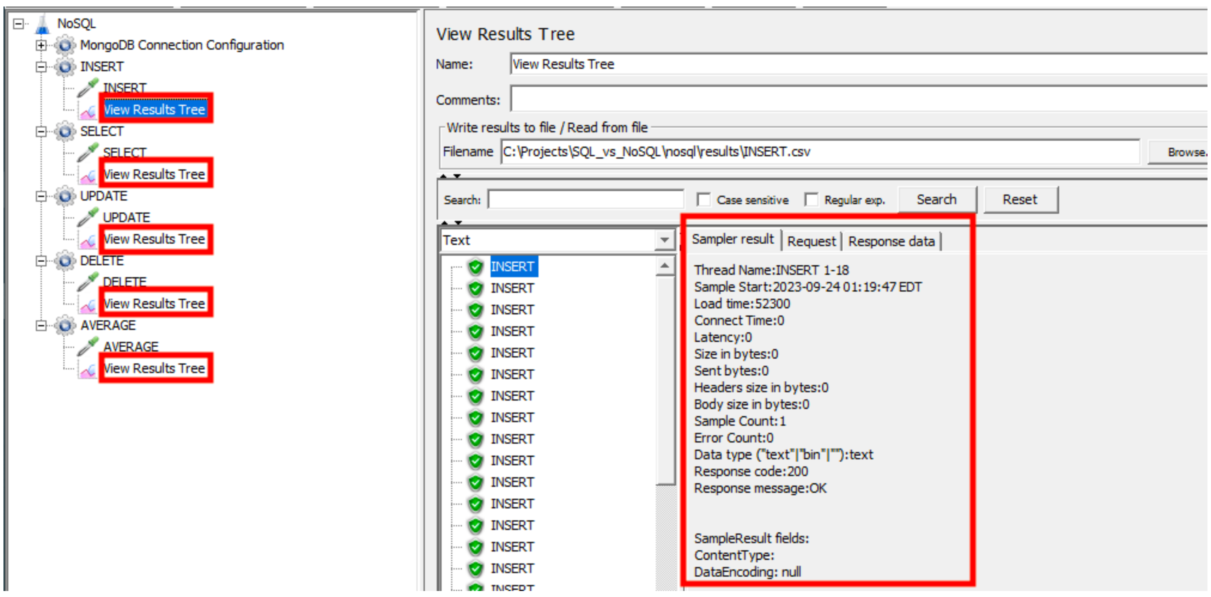
5.2.4 Data Collection Process, Testing Procedures

5.2.4.1 Collection Process

We successfully collected data in real-time as the load test was being run by using the "View Results Tree" listener in our load testing with Apache JMeter. This listener gave us the opportunity to concentrate on important performance data, particularly "elapsed time."

The "View Results Tree" listener offered a thorough view of each sample request that was executed, showing specific details about response times, response data, response headers, and other pertinent information. We focused primarily on the "elapsed time" metric, which counts the time between sending a request and receiving a response.

This method gave us the ability to carefully monitor and evaluate each request's performance in terms of "elapsed time," which is an important sign of a system's responsiveness and effectiveness during load testing. Focusing on this measure under the "View Results Tree" listener allowed us to obtain important understanding of how our system handled various loads. By enabling us to export this data, including the "elapsed time," into a CSV file, Apache JMeter significantly aided our investigation. The detailed statistical analysis we conducted using this CSV file afterwards allowed us to base our decisions and optimizations on the real-time performance indicators acquired during our load testing.



View Results Tree

Name: View Results Tree

Comments:

Write results to file / Read from file

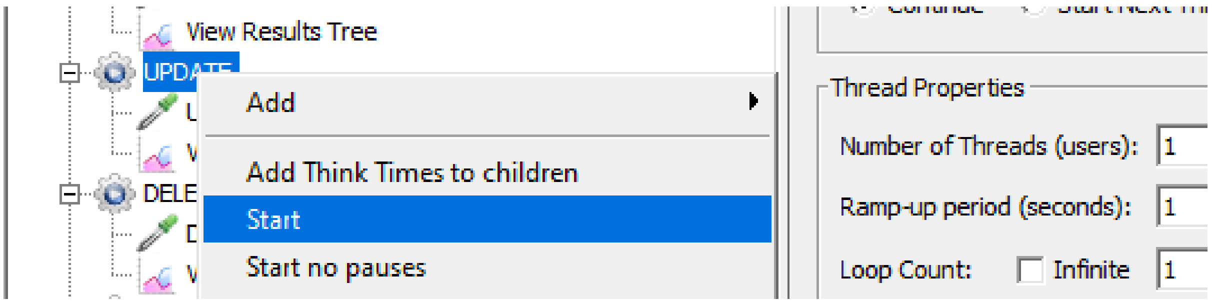
Filename C:\Projects\SQL_vs_NoSQL\nosql\results\INSERT.csv

```
Thread Name:INSERT 1-18
Sample Start:2023-09-24 01:19:47 EDT
Load time:52300
Connect Time:0
Latency:0
Size in bytes:0
Sent bytes:0
Headers size in bytes:0
Body size in bytes:0
Sample Count:1
Error Count:0
Data type ("text"|"bin"|""):text
Response code:200
Response message:OK


SampleResult fields:
ContentType:
DataEncoding: null
```

Figure 5.12: Apache JMeter showing the 'View Results Tree' Listener

5.2.4.2 Procedure

Our testing approach involves right-clicking on the chosen database operation (insert, update, select, delete, or average) first, then selecting 'Start' to start the test load run. This ensures that we are only doing targeted load tests.



Thread Properties
Number of Threads (users): 1
Ramp-up period (seconds): 1
Loop Count: Infinite 1

Figure 5.13 Running targeted algorithm in Apache JMeter

5.2.4.3 Data Analysis Process & Statistical Techniques, Hypothesis Testing

5.2.4.4 Analysis Process

In our research, the performance testing analysis procedure entails carefully evaluating the data gathered during the load test to learn more about how the sorting algorithms and backend scripting technologies behave under various loads (array sizes). The primary purpose of our analysis is still to determine which backend programming is more effective. In the paragraphs that follow, we'll break down the analysis procedures. The data gathered during the test were organized so that we could simply analyze them. After organizing and formatting our data, we determined the average 'elapsed time' of each simulated user throughout the course of the 30 test runs.

5.2.4.5 Statistical Technique

The T-TEST, a type of hypothesis test used to determine whether two groups are significantly different, was adopted because our research involves a moderately large amount of data, and we aimed to analyze the data to draw conclusions so that we can make reasonable inferences. See equation below:

Equation 5.1: T-Test Equation

$$T = \frac{\mu 1 - \mu 2}{{}^{sp}\sqrt{\frac{1}{n1} + \frac{1}{n2}}}$$

Where:

T represents the t-statistics

$\mu 1, \mu 2$ represents the means of distributions.

n1, n2 represents the degrees of freedom

5.2.4.6 Hypothesis Testing

Hypothesis testing is the process of making a statement and verifying it with data. During this process, it is typically assumed that two samples are not different from each other. This assumption helps to create two hypotheses: a null hypothesis (H0) and an alternative hypothesis (Ha) for each of our five tests. The outcomes of a hypothesis test can either reject the null hypothesis and accept the alternative hypothesis (indicating that there is a difference between two samples), or not reject the null hypothesis. In a scenario where the null hypothesis states that the population means of two unrelated groups are equal (H0: u1=u2) and the alternative hypothesis states that the population means are not equal (Ha: u1≠u2), it is determined whether to reject or accept the alternative hypothesis based on a significance level of $p<0.05$.

In relation to database management systems (DBMS), we hypothesized that SQL databases perform better in reporting operations like SELECT and aggregate operations like AVERAGE, whereas NoSQL databases perform better in data operations, especially in the areas of data creation, updates, and deletions. According to this theory, we believe that NoSQL databases are purposefully made to maximize speedy data updates.

This hypothesis is supported by our observation of statistically significant decreases in the amount of time (elapsed time) it takes to perform operations like create, update, and delete in NoSQL databases as well as read and aggregate (average) operations in SQL databases. Our theoretical perspective was that NoSQL databases are optimized for fast insert and markup operations whereas SQL-based databases are optimized for general operations such as select and aggregate functions. NoSQL is employed to support rapid insertion of data and SQL is utilized to support all data operations for RDBMS. Based on this, we expect data markup to be faster on NoSQL (Create, Insert, Update, Delete) while Read/Select and Aggregate operations will be faster on SQL.

The following paragraph shows all our five null hypotheses (H0-H4) and the corresponding alternative hypotheses (H0-a – H4-a)

- H0: There is no significant difference in the elapsed time for INSERT operations between SQL and NoSQL
- H0-a: The elapsed time of INSERT operations will be less (faster) in NoSQL than in SQL
- H1: There is no significant difference in the elapsed time for UPDATE operations between SQL and NoSQL
- H1-a: The elapsed time of UPDATE operations will be less (faster) in NoSQL than in SQL
- H2: There is no significant difference in the elapsed time for DELETE operations between SQL and NoSQL
- H2-a: The elapsed time of DELETE operations will be less (faster) in NoSQL than in SQL
- H3: There is no significant difference in the elapsed time for SELECT operations between SQL and NoSQL
- H3-a: The elapsed time of SELECT operations will be more (slower) in NoSQL than in SQL
- H4 : There is no significant difference in the elapsed time for AGGREGATE(AVERAGE) operations between SQL and NoSQL
- H4-a: The elapsed time of AGGREGATE(AVERAGE) operations will be more (slower) in NoSQL than in SQL.

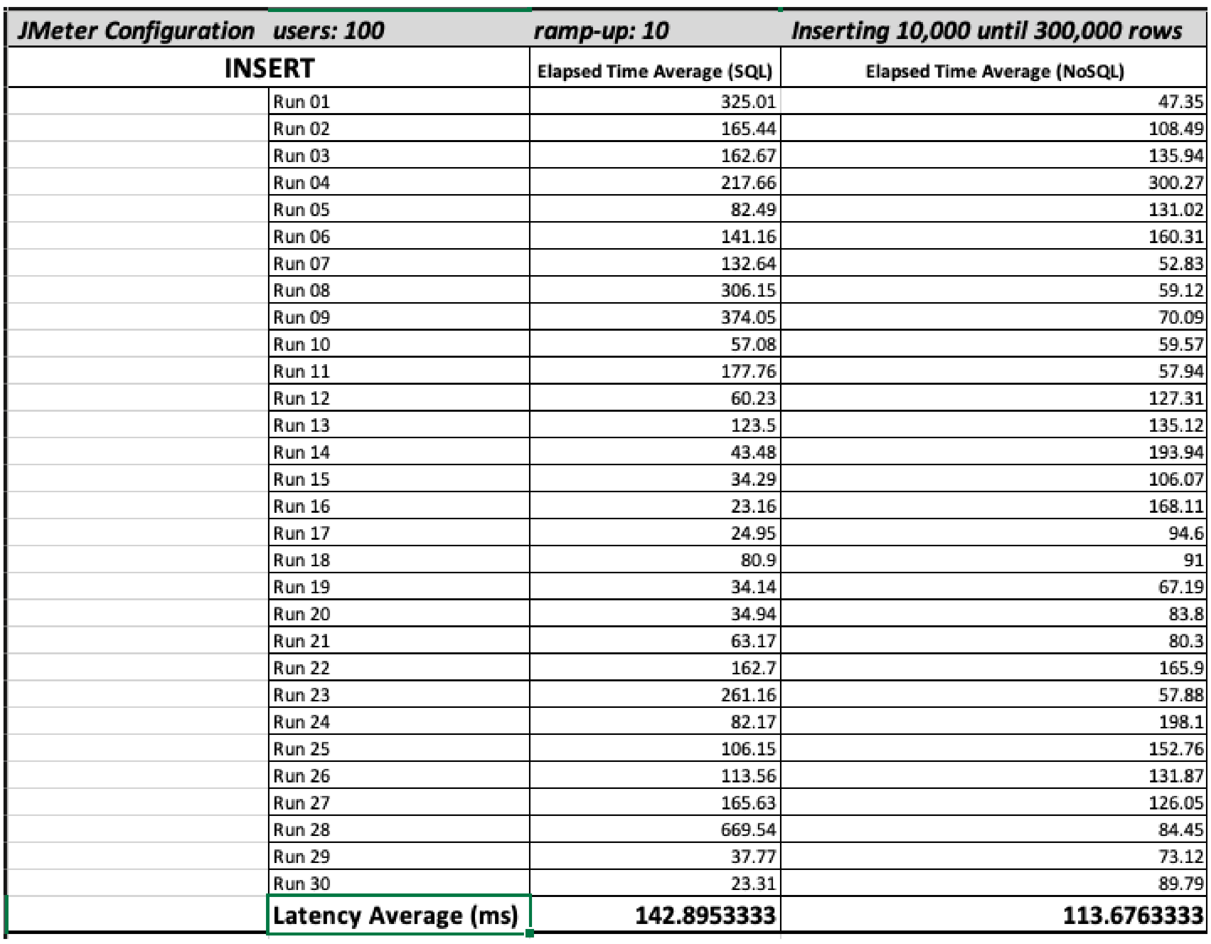
5.3 RESULT

5.3.1.1 Overview

In this study, our main research question is whether there is a significant performance difference between SQL (a relational database management system) and NoSQL (a non-relational database management system) for a variety of database operations, such as Insert, Select, Update, Delete, and Aggregate (Average). We implemented these operations in both SQL and NoSQL, and then we ran tests with Apache JMeter to get performance information. As the T-Test statistical method is best suited for determining whether there is a statistically significant difference between the means of two groups, in our instance, SQL and NoSQL, it was used to analyze the data that had been gathered.

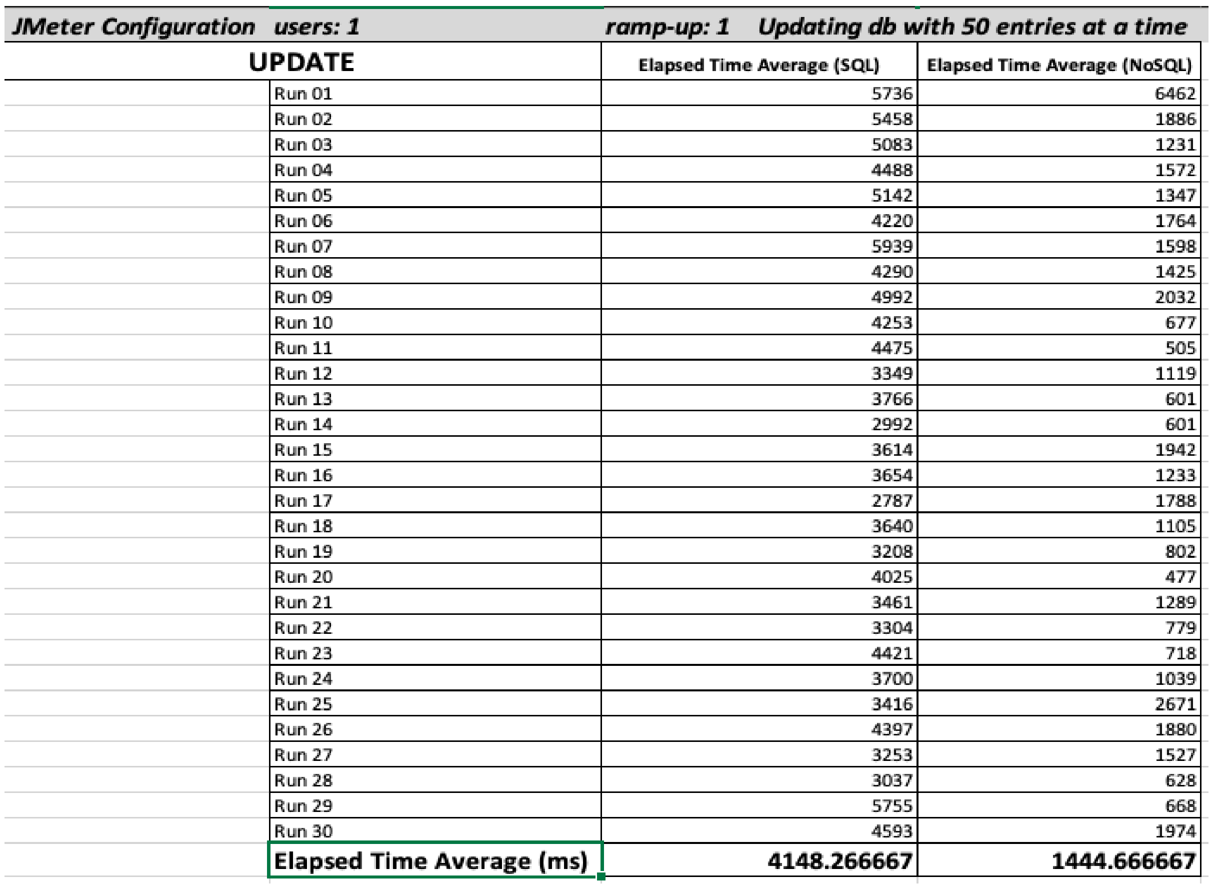
5.3.1.2 Descriptive Statistics

To offer a clear and comprehensive picture of the performance of SQL and NoSQL, including metrics like the mean, we compiled and described the obtained data in terms of central tendency.



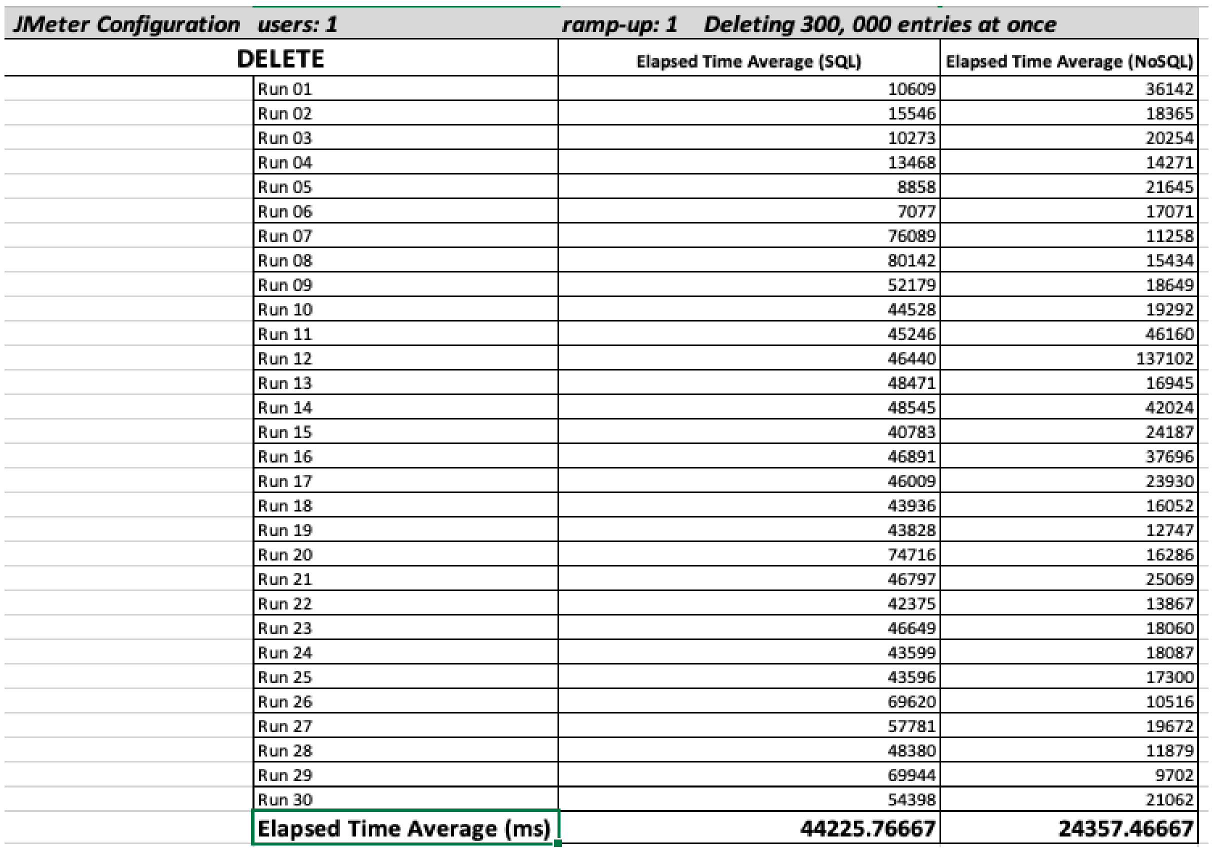
JMeter Configuration	***users: 100***	***ramp-up: 10***	***Inserting 10,000 until 300,000 rows***
INSERT		**Elapsed Time Average (SQL)**	**Elapsed Time Average (NoSQL)**
	Run 01	325.01	47.35
	Run 02	165.44	108.49
	Run 03	162.67	135.94
	Run 04	217.66	300.27
	Run 05	82.49	131.02
	Run 06	141.16	160.31
	Run 07	132.64	52.83
	Run 08	306.15	59.12
	Run 09	374.05	70.09
	Run 10	57.08	59.57
	Run 11	177.76	57.94
	Run 12	60.23	127.31
	Run 13	123.5	135.12
	Run 14	43.48	193.94
	Run 15	34.29	106.07
	Run 16	23.16	168.11
	Run 17	24.95	94.6
	Run 18	80.9	91
	Run 19	34.14	67.19
	Run 20	34.94	83.8
	Run 21	63.17	80.3
	Run 22	162.7	165.9
	Run 23	261.16	57.88
	Run 24	82.17	198.1
	Run 25	106.15	152.76
	Run 26	113.56	131.87
	Run 27	165.63	126.05
	Run 28	669.54	84.45
	Run 29	37.77	73.12
	Run 30	23.31	89.79
	Latency Average (ms)	**142.8953333**	**113.6763333**

Figure 5.14: INSERT operation result for 30 runs showing the elapsed time average.



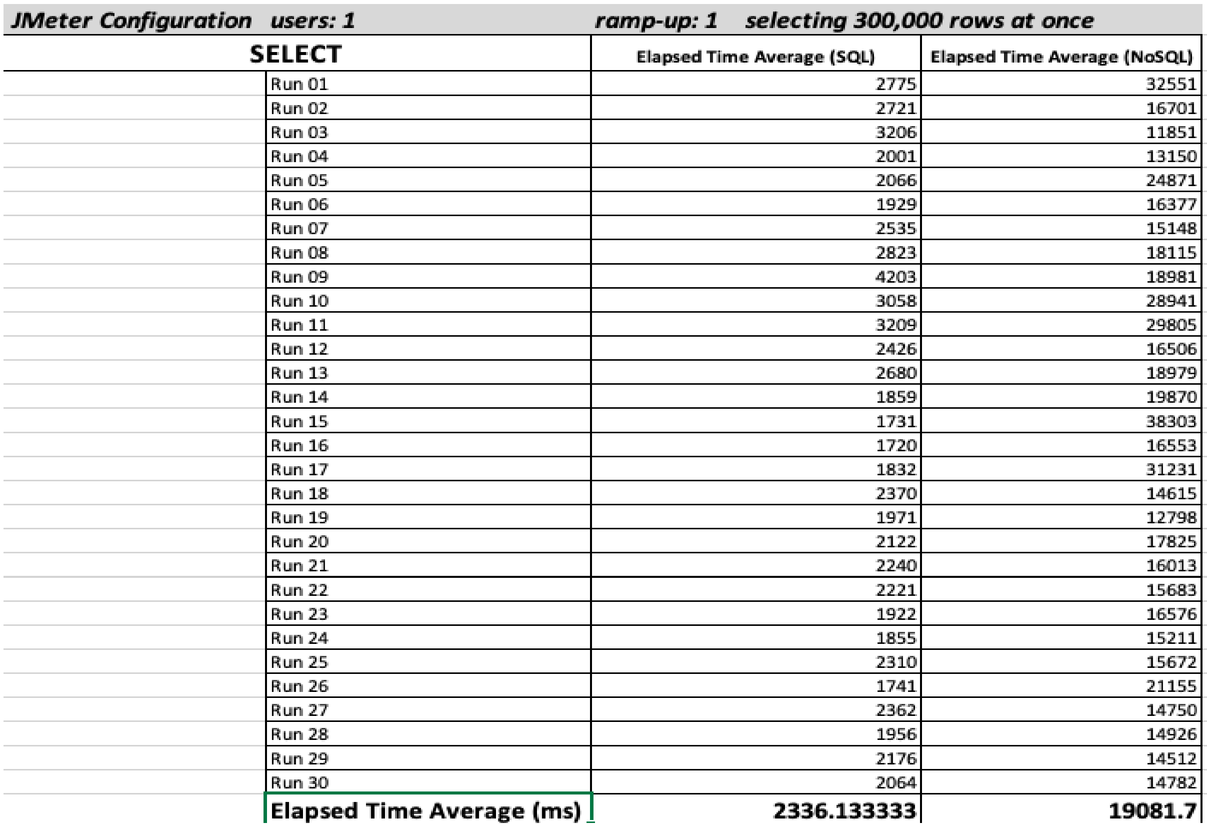
JMeter Configuration	*users: 1*	*ramp-up: 1*	*Updating db with 50 entries at a time*
UPDATE		**Elapsed Time Average (SQL)**	**Elapsed Time Average (NoSQL)**
	Run 01	5736	6462
	Run 02	5458	1886
	Run 03	5083	1231
	Run 04	4488	1572
	Run 05	5142	1347
	Run 06	4220	1764
	Run 07	5939	1598
	Run 08	4290	1425
	Run 09	4992	2032
	Run 10	4253	677
	Run 11	4475	505
	Run 12	3349	1119
	Run 13	3766	601
	Run 14	2992	601
	Run 15	3614	1942
	Run 16	3654	1233
	Run 17	2787	1788
	Run 18	3640	1105
	Run 19	3208	802
	Run 20	4025	477
	Run 21	3461	1289
	Run 22	3304	779
	Run 23	4421	718
	Run 24	3700	1039
	Run 25	3416	2671
	Run 26	4397	1880
	Run 27	3253	1527
	Run 28	3037	628
	Run 29	5755	668
	Run 30	4593	1974
	Elapsed Time Average (ms)	**4148.266667**	**1444.666667**

Figure 5.15: UPDATE operation result for 30 runs showing the elapsed time average.



JMeter Configuration	*users: 1*	*ramp-up: 1*	*Deleting 300, 000 entries at once*
DELETE		**Elapsed Time Average (SQL)**	**Elapsed Time Average (NoSQL)**
	Run 01	10609	36142
	Run 02	15546	18365
	Run 03	10273	20254
	Run 04	13468	14271
	Run 05	8858	21645
	Run 06	7077	17071
	Run 07	76089	11258
	Run 08	80142	15434
	Run 09	52179	18649
	Run 10	44528	19292
	Run 11	45246	46160
	Run 12	46440	137102
	Run 13	48471	16945
	Run 14	48545	42024
	Run 15	40783	24187
	Run 16	46891	37696
	Run 17	46009	23930
	Run 18	43936	16052
	Run 19	43828	12747
	Run 20	74716	16286
	Run 21	46797	25069
	Run 22	42375	13867
	Run 23	46649	18060
	Run 24	43599	18087
	Run 25	43596	17300
	Run 26	69620	10516
	Run 27	57781	19672
	Run 28	48380	11879
	Run 29	69944	9702
	Run 30	54398	21062
	Elapsed Time Average (ms)	**44225.76667**	**24357.46667**

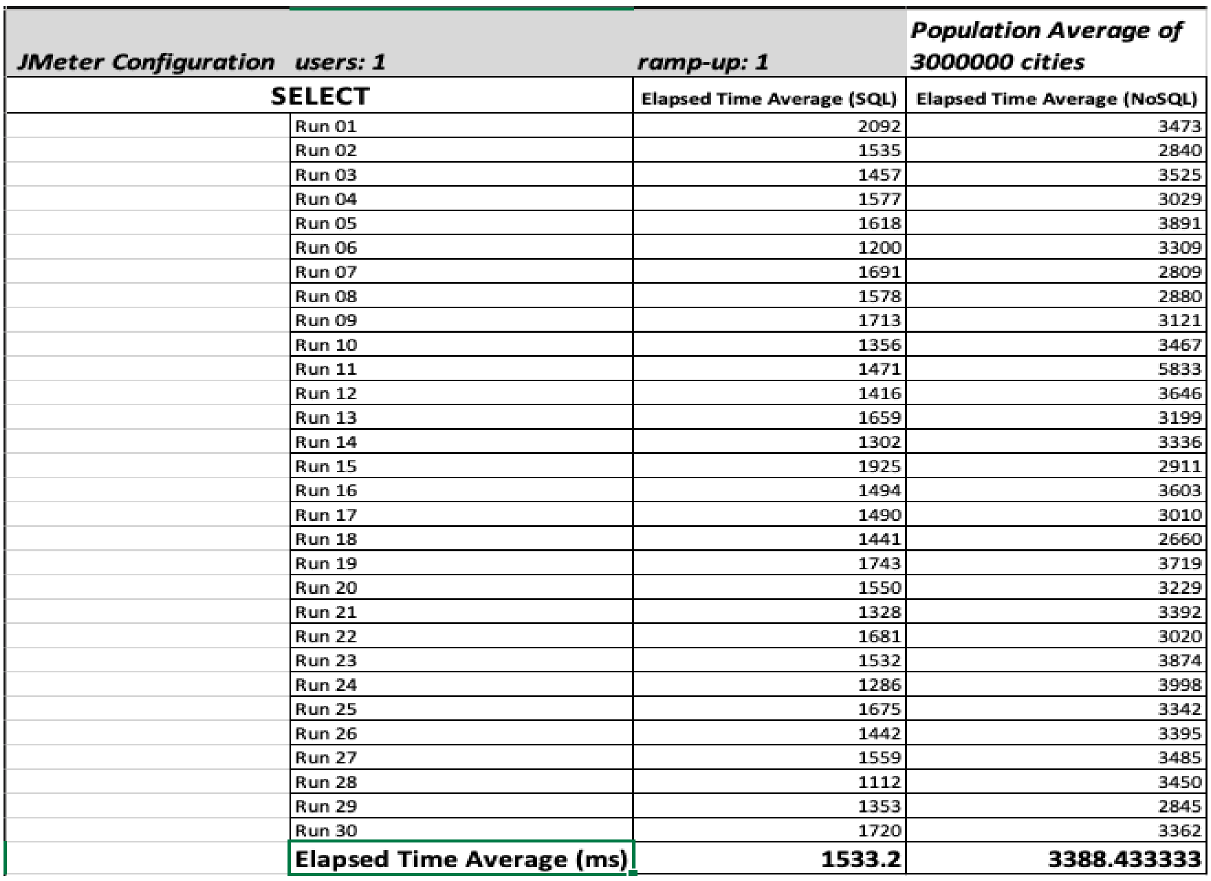
Figure 5.16: DELETE operation result for 30 runs showing the elapsed time average.



JMeter Configuration users: 1 ramp-up: 1 selecting 300,000 rows at once

SELECT	Elapsed Time Average (SQL)	Elapsed Time Average (NoSQL)
Run 01	2775	32551
Run 02	2721	16701
Run 03	3206	11851
Run 04	2001	13150
Run 05	2066	24871
Run 06	1929	16377
Run 07	2535	15148
Run 08	2823	18115
Run 09	4203	18981
Run 10	3058	28941
Run 11	3209	29805
Run 12	2426	16506
Run 13	2680	18979
Run 14	1859	19870
Run 15	1731	38303
Run 16	1720	16553
Run 17	1832	31231
Run 18	2370	14615
Run 19	1971	12798
Run 20	2122	17825
Run 21	2240	16013
Run 22	2221	15683
Run 23	1922	16576
Run 24	1855	15211
Run 25	2310	15672
Run 26	1741	21155
Run 27	2362	14750
Run 28	1956	14926
Run 29	2176	14512
Run 30	2064	14782
Elapsed Time Average (ms)	**2336.133333**	**19081.7**

Figure 5.17: SELECT operation result for 30 runs showing the elapsed time average.

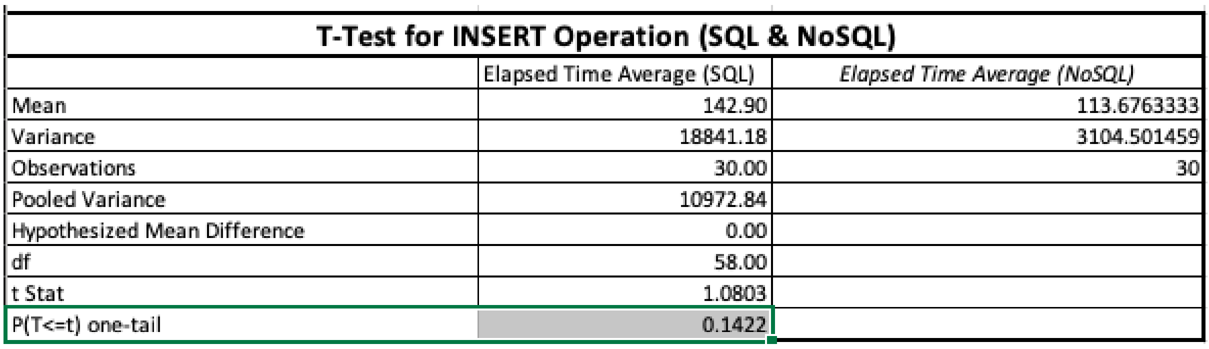


JMeter Configuration	*users: 1*	*ramp-up: 1*	*Population Average of 3000000 cities*
SELECT		**Elapsed Time Average (SQL)**	**Elapsed Time Average (NoSQL)**
	Run 01	2092	3473
	Run 02	1535	2840
	Run 03	1457	3525
	Run 04	1577	3029
	Run 05	1618	3891
	Run 06	1200	3309
	Run 07	1691	2809
	Run 08	1578	2880
	Run 09	1713	3121
	Run 10	1356	3467
	Run 11	1471	5833
	Run 12	1416	3646
	Run 13	1659	3199
	Run 14	1302	3336
	Run 15	1925	2911
	Run 16	1494	3603
	Run 17	1490	3010
	Run 18	1441	2660
	Run 19	1743	3719
	Run 20	1550	3229
	Run 21	1328	3392
	Run 22	1681	3020
	Run 23	1532	3874
	Run 24	1286	3998
	Run 25	1675	3342
	Run 26	1442	3395
	Run 27	1559	3485
	Run 28	1112	3450
	Run 29	1353	2845
	Run 30	1720	3362
	Elapsed Time Average (ms)	**1533.2**	**3388.433333**

Figure 5.18: AGGREGATE(Average) operation result for 30 runs showing the elapsed time average.

5.3.1.3 Inferential Statistics

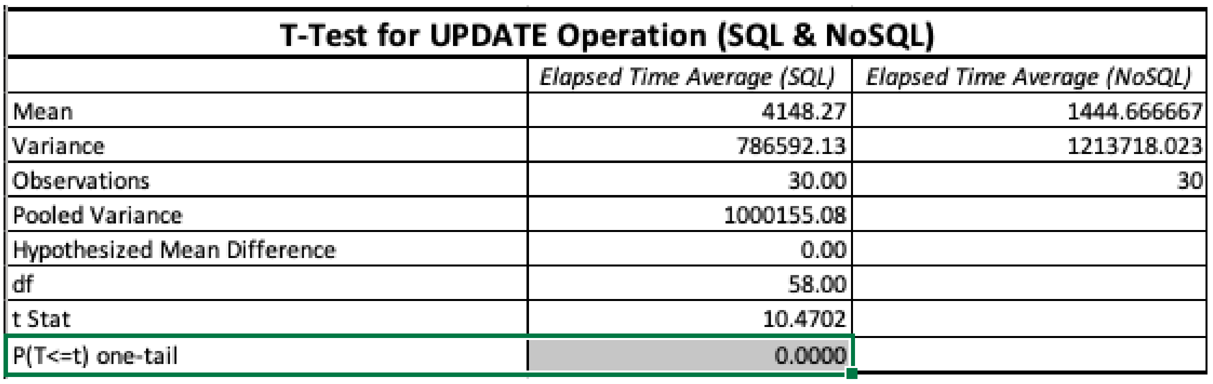
Based on the data we had collected; we used these techniques to build inferences and come to conclusions. Our inferential statistical method for determining whether there is a significant performance difference between SQL and NoSQL was the t-test.



T-Test for INSERT Operation (SQL & NoSQL)

	Elapsed Time Average (SQL)	*Elapsed Time Average (NoSQL)*
Mean	142.90	113.6763333
Variance	18841.18	3104.501459
Observations	30.00	30
Pooled Variance	10972.84	
Hypothesized Mean Difference	0.00	
df	58.00	
t Stat	1.0803	
P(T<=t) one-tail	0.1422	

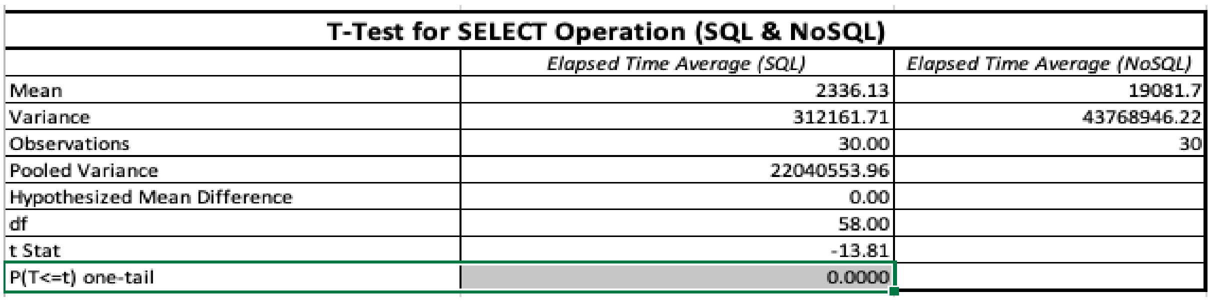
Figure 5.19: INSERT operation results for t-test



T-Test for UPDATE Operation (SQL & NoSQL)		
	Elapsed Time Average (SQL)	*Elapsed Time Average (NoSQL)*
Mean	4148.27	1444.666667
Variance	786592.13	1213718.023
Observations	30.00	30
Pooled Variance	1000155.08	
Hypothesized Mean Difference	0.00	
df	58.00	
t Stat	10.4702	
P(T<=t) one-tail	0.0000	

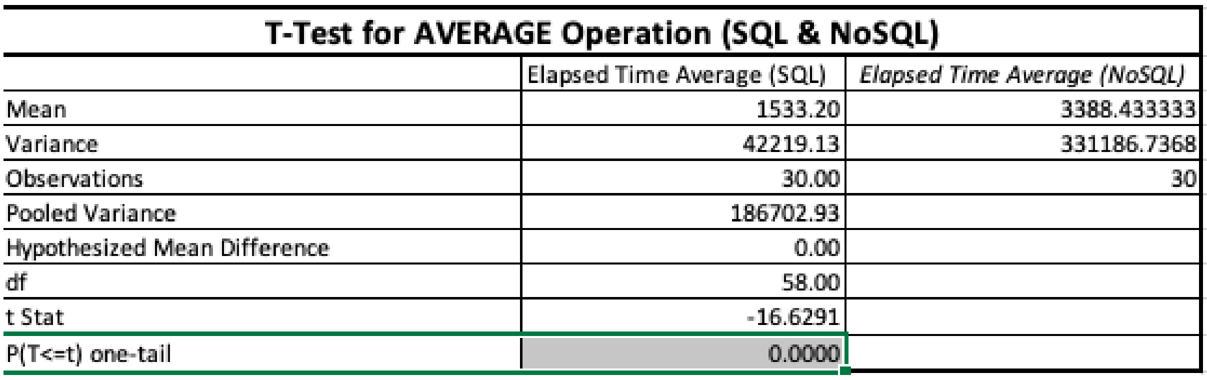
Figure 5.20: UPDATE operation results for t-test

Figure 5.21: DELETE operation results for t-test



T-Test for SELECT Operation (SQL & NoSQL)		
	Elapsed Time Average (SQL)	*Elapsed Time Average (NoSQL)*
Mean	2336.13	19081.7
Variance	312161.71	43768946.22
Observations	30.00	30
Pooled Variance	22040553.96	
Hypothesized Mean Difference	0.00	
df	58.00	
t Stat	-13.81	
P(T<=t) one-tail	0.0000	

Figure 5.22: SELECT operation results for t-test



T-Test for AVERAGE Operation (SQL & NoSQL)		
	Elapsed Time Average (SQL)	*Elapsed Time Average (NoSQL)*
Mean	1533.20	3388.433333
Variance	42219.13	331186.7368
Observations	30.00	30
Pooled Variance	186702.93	
Hypothesized Mean Difference	0.00	
df	58.00	
t Stat	-16.6291	
P(T<=t) one-tail	0.0000	

Figure 5.23: AGGREGATE (AVERAGE) operation results for t-test

As seen in figure 5.19 above, the p-value (p=0.1422) which is greater than 0.05. We there accept the null hypothesis H0 which states that there is no significant difference in the elapsed time for INSERT operations between SQL and NoSQL. However, looking at the figures 5.20-5.23, we can see that the p-value for each of our t-test is less than 0.05 which statistically give us substantial reasons to reject the null hypotheses H1, H2, H3 and H4 and accept the alternative hypotheses H1-a, H2-a, H3-a and H4-a which states that the elapsed time of UPDATE, DELETE, SELECT and AGGREGATE(AVERAGE) operations will be less (faster) in NoSQL than in SQL.

5.3.1.4 Limitations

Although the research on SQL and NoSQL databases in the report is admirable, it has certain drawbacks. Future research should include a wider range of databases because this study concentrated on just one SQL and NoSQL database. Although the study used simplified workloads, real-world circumstances are more complex, necessitating future research into resource restrictions, varied data quantities, and various benchmarking techniques. Although the study focused mostly on "elapsed time," other factors also affect database performance, and real-time data monitoring and operational aspects should be included in subsequent studies for a more thorough analysis.

5.3.1.5 Future Research

The performance differences between SQL and NoSQL databases will be further investigated in future studies, with an emphasis on key database operations including but not limited to Insert, Select, Update, Delete, and Aggregate (Average). Our goal is to discover the situations in which one of these two categories of database management systems (DBMS) surpasses the other and to provide empirical insights into the efficiency differences between them.

Along with these fundamental research tasks, we understand how critical it is to solve the shortcomings identified in our earlier work. Future studies will consider a wider variety of databases, challenging workloads, resource limitations, varying data sizes, and various benchmarking methodologies. To conduct a more thorough analysis, we will additionally investigate performance measures besides "Elapsed Time" and take operational aspects and real-time data monitoring into account. Our future research will focus on these areas to offer useful recommendations for choosing the best DBMS for particular use cases, ultimately optimizing system performance and resource allocation.

5.4 CONCLUSION

In conclusion, our study compares SQL and NoSQL performance across five key database operations. To compare the performance of SQL and NoSQL, this study used Apache JMeter for performance testing and real-time data collecting. The results of our analysis provide valuable insights into the performance differences between SQL and NoSQL for specific database operations. While NoSQL exhibited superior performance in Insert, Update, Delete operations compared to SQL, the study uncovered that SQL databases excel in reporting operations like Select and Aggregate (Average). Our findings support the notion that NoSQL databases are optimized for rapid data manipulation, whereas SQL databases excel in data retrieval and complex querying.

Despite the useful knowledge this study has provided, it is important to recognize its limits. Future studies should look at a wider range of databases, considering different workloads, resource limitations, different data quantities, and benchmarking techniques. Additionally, a more thorough analysis should consider elements other than "Elapsed Time" to provide a comprehensive picture of database performance. Finally, this research offers empirical evidence to guide judgements about database technology and adds to the ongoing discussion over SQL vs NoSQL databases. Our study establishes the groundwork for more informed decisions as organizations strive for maximum performance in a constantly changing data landscape and opens the door for further research into improving database management system effectiveness.

CHAPTER 6

CONCLUSIONS

We examined and contrasted several aspects of full-stack development, including frontend frameworks, backend scripting technologies, and database management systems, in this extensive research study. Our research provides insightful information that developers and organizations may use to guide important technological decisions. In order to better understand the JavaScript frontend frameworks React.js and Angular.js, we painstakingly created comparable web apps and put them through a rigorous performance testing process. Using Google Chrome's Task Manager and Lighthouse, among other technologies, we were able to replicate real-world online situations in our test environments. Our findings highlight the distinct advantages of both frameworks and show that React.js performs better than Angular.js in terms of overall efficiency. Our work provides developers and organizations looking for help in choosing the best frontend framework for their projects with a solid base, and we continue to strive for improvement in future research, including more robust performance measures.

Turning our attention to backend scripting tools, we thoroughly compared Node.js and PHP's performance. utilizing Apache JMeter and the T-Test statistical approach, we collected and analyzed data utilizing heap and complicated sorting algorithms across different array sizes. Based on statistically significant performance differences between these two backend scripting systems, our results clearly favor Node.js. In order to help developers and organizations match the best backend scripting technology for their projects, this research adds to the body of knowledge already available about backend technologies.

Furthermore, we examined the continuous discussion between SQL and NoSQL databases in the context of data management, comparing and contrasting their effectiveness in five essential database functions. By employing Apache JMeter for both performance evaluation and real-time data gathering, we were able to highlight the various advantages of different database systems. NoSQL databases—MongoDB in particular—performed exceptionally well when it came to data processing, whereas SQL databases performed better when it came to reporting procedures like Select and Aggregate (Average). In addition to offering a useful tool for making informed selections regarding database technology, our analysis also identifies the particular use cases for each.

REFERENCES

[1] W. Xu, "Benchmark Comparison of JavaScript Frameworks React, Vue, Angular and Svelte," 2021.

[2] M. Levlin, "DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue, and Svelte," 2020.

[3] D. Verma, "A comparison of web framework efficiency: performance and network analysis of modern web frameworks," 2022.

[4] Vukelić, "Comparison of front-end frameworks for web applications development," *Zbornik Veleučilišta u Rijeci,* vol. 6, no. 1, pp. 261-282, 2018.

[5] Yorulmaz, "JavaScript Frameworks A qualitative evaluation and comparison of the dominant factors in Angular and React Abdul Kadir Yorulmaz," 2020.

[6] E. Wohlgethan, "Supportingweb development decisions by comparing three major javascript frameworks: Angular, react and vue. js," Hochschule für Angewandte Wissenschaften Hamburg, 2018.

[7] R. Mohammadi, "Performance, and Efficiency based Comparison of Angular and React in a case study of Single page application (SPA)."

[8] J. KEPLER, "Comparing Modern Front-End Frameworks," 2022.

[9] S. Mousavi, "Maintainability evaluation of single page application frameworks: Angular2 vs. react," ed, 2017.

[10] E. Saks, "JavaScript Frameworks: Angular vs React vs Vue," 2019.

[11] A. Kumar and R. K. Singh, "Comparative analysis of angularjs and reactjs," *International Journal of Latest Trends in Engineering and Technology,* vol. 7, no. 4, pp. 225-227, 2016.

[12] K. Peguero, N. Zhang, and X. Cheng, "An empirical study of the framework impact on the security of javascript web applications," in *Companion Proceedings of the The Web Conference 2018*, 2018, pp. 753-758.

[13] M. Sultan, "Angular and the Trending Frameworks of Mobile and Web-based Platform Technologies: A Comparative Analysis," in *Proc. Future Technologies Conference*, 2017, pp. 928-936.

[14] S. Guan, W. Hu, and H. Zhou, "Front-end and back-end separation-react based framework for networked remote control laboratory," in *2018 37th Chinese Control Conference (CCC),* 2018: IEEE, pp. 6314-6319.

[15] Y. Xing, J. Huang, and Y. Lai, "Research and analysis of the front-end frameworks and libraries in e-business development," in *Proceedings of the 2019 11th International Conference on Computer and Automation Engineering*, 2019, pp. 68-72.

[16] A. Prayogi, M. Niswar, and M. Rijal, "Design and implementation of REST API for academic information system," in *IOP Conference Series: Materials Science and Engineering*, 2020, vol. 875, no. 1: IOP Publishing, p. 012047.

[17] I. K. Chaniotis, K.-I. D. Kyriakou, and N. D. Tselikas, "Is Node. js a viable option for building modern web applications? A performance evaluation study," *Computing*, vol. 97, pp. 1023-1044, 2015.

[18] W. S. Raharjo, "Performance and Scalability Analysis of Node. js and PHP/Nginx Web Application," *Informatika: Jurnal Teknologi Komputer dan Informatika,* vol. 9, no. 2, p. 67762, 2014.

[19] K. Lei, Y. Ma, and Z. Tan, "Performance comparison and evaluation of web development technologies in php, python, and node. js," in *2014 IEEE 17th international conference on computational science and engineering*, 2014: IEEE, pp. 661-668.

[20] R. Jain, A. Sen, and K. Paliwal, "A Comparative Analysis of New Cow Concept of Node. Js With Php."

[21] H. K. Dhalla, "A Performance Analysis of Native JSON Parsers in Java, Python, MS. NET Core, JavaScript, and PHP," in *2020 16th International Conference on Network and Service Management (CNSM)*, 2020: IEEE, pp. 1-5.

[22] L. Nkenyereye and J.-W. Jang, "Performance evaluation of server-side javascript for healthcare hub server in remote healthcare monitoring system," *Procedia Computer Science,* vol. 98, pp. 382-387, 2016.

[23] T. Crawford and T. Hussain, "A comparison of server side scripting technologies," in *Proceedings of the International Conference on Software Engineering Research and Practice (SERP),* 2017: The Steering Committee of The World Congress in Computer Science, Computer …, pp. 69-76.

[24] A. H. Odeh, "Analytical and Comparison Study of Main Web Programming Languages–ASP and PHP," *TEM Journal,* vol. 8, no. 4, pp. 1517-1522, 2019.

[25] Z. Ahmed, F. J. Kinjol, and I. J. Ananya, "Comparative Analysis of Six Programming Languages Based on Readability, Writability, and Reliability," in *2021 24th International Conference on Computer and Information Technology (ICCIT),* 2021: IEEE, pp. 1-6.

[26] O. M. Adebukola and O. B. Kazeem, "Performance Comparison of dynamic Web scripting Language: A case Study of PHP and ASP .NET," *International Journal of Scientific & Engineering Research,* 2014.

[27] H. Brar, T. Kaur, and Y. Rajoria, "The better comparison between PHP, python-web & Node. js," *Int. J. Res. Eng. Sci.,* vol. 9, no. 7, pp. 29-37, 2021.

[28] S. S. N. Challapalli, P. Kaushik, S. Suman, B. D. Shivahare, V. Bibhu, and A. D. Gupta, "Web Development and performance comparison of Web Development Technologies in Node. js and Python," in *2021 International Conference on Technological Advancements and Innovations (ICTAI),* 2021: IEEE, pp. 303-307.

[29] S. Jamal, M. V. Cruz, S. Chakravarthy, C. Wahl, and H. Wimmer, "Integration of EEG and Eye Tracking Technology: A Systematic Review," *SoutheastCon 2023,* pp. 209-216, 2023.

[30] S. Bjeladinovic, Z. Marjanovic, and S. Babarogic, "A proposal of architecture for integration and uniform use of hybrid SQL/NoSQL database components," *Journal of Systems and Software,* vol. 168, p. 110633, 2020.

[31] J. Antas, R. Rocha Silva, and J. Bernardino, "Assessment of SQL and NoSQL Systems to Store and Mine COVID-19 Data," *Computers,* vol. 11, no. 2, p. 29, 2022.

[32] S. Sicari, A. Rizzardi, and A. Coen-Porisini, "Security&privacy issues and challenges in NoSQL databases," *Computer Networks,* p. 108828, 2022.

[33] W. Khan, T. Kumar, Z. Cheng, K. Raj, A. Roy, and B. Luo, "SQL and NoSQL Databases Software architectures performance analysis and assessments—A Systematic Literature review. arXiv 2022," *arXiv preprint arXiv:2209.06977.*

[34] T. N. Khasawneh, M. H. AL-Sahlee, and A. A. Safia, "Sql, newsql, and nosql databases: A comparative survey," in *2020 11th International Conference on Information and Communication Systems (ICICS),* 2020: IEEE, pp. 013-021.

[35] K. S. Kumar and S. Mohanavalli, "A performance comparison of document oriented NoSQL databases," in *2017 International Conference on Computer, Communication and Signal Processing (ICCCSP),* 2017: IEEE, pp. 1-6.

[36] S. Padhy and G. M. M. Kumaran, "A quantitative performance analysis between Mongodb and Oracle NoSQL," in *2019 6th international conference on computing for sustainable global development (INDIACom),* 2019: IEEE, pp. 387-391.

[37] J. Kepner *et al.*, "Associative array model of SQL, NoSQL, and NewSQL Databases," in *2016 IEEE High Performance Extreme Computing Conference (HPEC),* 2016: IEEE, pp. 1-9.

[38] J. M. A. Araujo, A. C. E. de Moura, S. L. B. da Silva, M. Holanda, E. de Oliveira Ribeiro, and G. L. da Silva, "Comparative performance analysis of NoSQL Cassandra and MongoDB databases," in *2021 16th Iberian Conference on Information Systems and Technologies (CISTI),* 2021: IEEE, pp. 1-6.

[39] K. Mahmood, K. Orsborn, and T. Risch, "Comparison of nosql datastores for large scale data stream log analytics," in *2019 IEEE International Conference on Smart Computing (SMARTCOMP),* 2019: IEEE, pp. 478-480.

[40] S. Rautmare and D. Bhalerao, "MySQL and NoSQL database comparison for IoT application," in *2016 IEEE international conference on advances in computer applications (ICACA),* 2016: IEEE, pp. 235-238.

[41] S. Jamal and H. Wimmer, "Performance Analysis of Machine Learning Algorithm on Cloud Platforms: AWS vs Azure vs GCP," in *International Scientific and Practical Conference on Information Technologies and Intelligent Decision Making Systems*, 2022: Springer, pp. 43-60.

[42] S. Cuk, H. Wimmer, L. M. Powell, and C. M. Rebman Jr, "ELECTRONIC EMERGENCY MEDICAL TECHNICIAN REPORTS-TESTING A PERCEPTION OF A PROTOTYPE," *Issues in Information Systems*, vol. 19, no. 3, 2018.

[43] A. Gupta, S. Tyagi, N. Panwar, S. Sachdeva, and U. Saxena, "NoSQL databases: Critical analysis and comparison," in *2017 International conference on computing and communication technologies for smart nation (IC3TSN)*, 2017: IEEE, pp. 293-299.

[44] B. Jose and S. Abraham, "Performance analysis of NoSQL and relational databases with MongoDB and MySQL," *Materials today: PROCEEDINGS*, vol. 24, pp. 2036-2043, 2020.