International Journal of Science, Strategic Management and Technology
Volume **02** Issue **05** May-**2026** | ISSN: **3108-1762 (Online)** | Impact Factor: 3.8
An International, Peer-Reviewed, Open Access Scholarly Journal Indexed in recognized academic databases

# Performance Comparison of React and Angular

**Amit Kumar**[1]
B.Tech (Information Technology)
NIET, Greater Noida

**Mr. Ram Kumar Sharma**[2]
Assistant Professor (IT Department)
NIET, Greater Noida

https://doi.org/10.55041/ijsmt.v2i5.194

**Cite this Article**: Kumar, A. (2026). Performance Comparison of React and Angular. International Journal of Science, Strategic Management and Technology, *02*(05). https://doi.org/10.55041/ijsmt.v2i5.194



**Abstract**

A fast change in how we create and maintain single-page applications (SPAs) means you need a framework (front-end) that can scale alongside your application as well as provide efficiently designed UIs and manage interactions with large amounts of dynamic data. This performance comparison study is structured around using the benchmarking methods from experimental studies to compare React and Angular frameworks. The performance comparison will use performance indicators available for measurement, including load time, time until the application is ready for interaction (TII), efficiency of the DOM manipulation, rendering speed, memory usage, and CPU usage at load. To improve consistency, the same types of applications will be implemented through each framework in a controlled environment (aka benchmarking environment). Prior studies have shown that architectures built using the virtual DOM typically perform better on both creating & deleting operations; whereas, incremental DOM styled frameworks typically perform well under updating states within the UI. The scalability characteristics under increasing workload are also impacted by the architectural complexity of the framework compared to its efficiencies once implemented in a runtime environment. Using the above information, this paper provides the reader with a clearer understanding of performance characteristics and behaviour for applications built on different rendering paradigms. The conclusions drawn can be used as evidence for the decision making of developers & researchers when selecting the best framework to use based upon their application size, amount of interaction within the application, and scalability needs.

## 1. INTRODUCTION

Since the beginning of the Internet, web development has changed drastically. In the early

days of the internet, web pages were simply static HTML documents where people would simply read information. These systems were all about documentation and had very few possibilities of contributing back to the website. There was no automatic way to update a document; it had to be done by logging into the server and manually changing the source code for each update to the content of that page. At that time, websites were focused on being accessible and having basic navigation between documents through hyperlinks; they were not focused on being interactive, nor on having the best performance.

Starting in the late 1990s and early 2000s when server-side scripting technologies like PHP, ASP and JSP emerged, web application systems shifted from being static document delivery systems to being delivered as dynamic web pages with content generated at the time they were requested by the user. Database systems were added into these systems, which provided a mechanism to create personalized content for individual users, to authenticate the identity of users,

and to enable e-commerce on the web. At this time, we were starting to see a shift in how we viewed the purpose of a webpage; the purpose of a webpage started to change from being an informational document to being a functional web application.

The next significant change occurred with the introduction of asynchronous communication technologies such as AJAX, which allowed an application to update small sections of the content of a page without having to reload the complete webpage each time the data changed. This created a faster response time for users and contributed to a much better user experience. Along with the evolution of AJAX, client-side scripting languages like JavaScript evolved from being a simple client-side enhancement tool, to becoming a fundamental part of the development process.

Finally, the emergence of Single Page Applications (SPAs), has further increased the need for developing client-side architectures that are structured and scalable.

The ongoing evolution of the web ecosystem is focused on performance, scalability, maintainability, and a user experience-driven design approach. Therefore, applications today are expected to accommodate high user traffic, dynamically update content, and offer seamless cross-platform compatibility. In support of this evolution, frameworks such as React and Angular have emerged to provide systematic methods for developing efficient, scalable user interfaces.

Web applications have become more interactive, leading to limitations in performance and usability found in traditional multi-page architectures. In a typical web application, each user interaction that requires new data will usually force a complete page reload at the browser, leading to noticeable latency, higher bandwidth usage, and a disrupted flow of the user's experience. The demand for faster, more fluid interactions (similar to those available with desktop software) led to the development of a new architectural model known as the Single Page Application (SPA).

Single-page applications are built on the principle of loading only one HTML page initially and then changing the content displayed on that page dynamically, based on user interaction. SPAs request that the web server provide only the data that has changed, not an entire page of rendered HTML content. Communication with backend services occurs asynchronously, allowing the SPA to retrieve only the changes and update the user interface without needing to refresh the complete page. This model greatly reduces unnecessary data transfer and significantly improves responsiveness.

The introduction of AJAX was one of the most important factors that lead to the development of SPA architecture by providing a method for background communication between customers and servers regarding requests for information from both sides. However, as applications become more complex; it became harder to manage state, do routing and update the User Interface (UI) solely through use of raw JavaScript. The need for frameworks that had structured front-end components, provided for declarative rendering and allowed for efficient state management became apparent as a result of having to solve these problems.

SPAs provided significant advantages, including the ability to reduce user disruption caused by page reloads (or lack of), the ability to provide real-time updates and to facilitate smooth transitions between views, and the ability to create highly interactive applications (e.g., dashboards, collaboration tools, streaming services and social media interfaces). Performance-wise, SPAs frequently result in lower server loads because they place most of the rendering load on the client side (the server).

SPAs also present a new set of technical challenges. First, they often result in higher initial load times (because of larger JavaScript bundles), and require additional strategies in order to achieve search engine optimization (SEO). Second, efficient updates to the DOM (Document Object Model) and consistency of state must be taken into consideration when designing for performance. These challenges in turn have had a significant impact on the architecture of contemporary frameworks, with the introduction of such techniques as the virtual DOM and incremental DOM.

As a result, SPAs are now a turning point for web development, as they alter how apps are developed and

evaluated in their performance. Understanding this change will help us evaluate modern frameworks such as React and Angular where both were specifically developed to meet the requirements that SPAs generate.

With the many technologies developed to implement modern client-side applications, React and Angular have driven most technology growth and most developers to adopt what they both have to offer. They both were built to address the ever-increasing complexity of single-page application development, yet they are architecturally, functionally, and developmentally quite different.

React is a JavaScript library that Facebook introduced. It is designed around building user interfaces through a component architecture and is not considered a complete framework. Instead, React focuses on developing the "view" portion of a web application, allowing developers to have flexibility in determining what other tools to use for routing, state management, and data handling. Notable for its use of a virtual DOM, React creates a copy (or representation) of the user interface in memory, so that it can efficiently update the actual DOM by only modifying the elements of the UI that have changed, rather than completely rebuilding the actual DOM from its source. This selective way of rendering updates will help to ensure that React applications are responsive, especially if they are subject to continuous state changes. React employs a top-to-bottom approach for the flow of data through its components.

Angular, which is being maintained/developed by Google, offers a complete framework solution. Angular provides a combined platform for creating apps with routing, dependency injection, form handling and communication to/from HTTP requests as part of a unified system (unlike React, where these services must be integrated separately). Angular is built on Component-based architecture, too. Still, it is based on a structured design pattern that utilises TypeScript for strong typing. Angular uses an incremental DOM structure along with a Change Detection mechanism to update the view when there are changes to the application state. As a result, Angular promotes structured scalability and maintainability, especially when used within enterprise-level applications.

In contrast to React, Angular is designed to provide maximum completeness and built-in functionality without requiring third-party libraries for full-scale development. In comparison, React's modular nature typically results in fewer components being needed to build an initial product (due to its lighter configuration), but often necessitates the addition of additional libraries for larger-scale production use. Meanwhile, Angular typically supplies extensive built-in features and functions, resulting in potentially larger bundle sizes and more difficult to learn. While both React and Angular provide improved application performance, better user experience and easier development of complex Uls, their different internal render strategies, state management methods and architectural designs will provide measurable differences in how they behave during runtime. These differences will allow for systematic performance comparisons and evaluations of both techniques.

Most of the research that has already been done typically presents confounding descriptive comparisons based (mostly) on architectural characteristics, types of users, or experience with them. Even though some studies are based on empirical evaluation of frameworks, even these types of analyses are typically limited to a few isolated application use cases, very small implementations, or single performance metrics. Given these types of differences in performance test environment, application design, and measurement technique, consistently evaluating development frameworks' performance becomes very difficult for any developer/organization to make an informed selection regarding a proper framework based on objective performance criteria versus simply relying on community preference or familiarity with the framework's ecosystem.

The most modern application usage requires a very responsive application design with extensive and efficient DOM manipulations, efficiently managed memory, as well as scalable performance in response to increased workloads. The rendering strategies (or architectural designs) used by frameworks (for example, virtual DOM vs. direct manipulation of DOM or Incremental vs. complete change detection) will likely result in producing varying levels of runtime efficiency, initial loading times, and system resource

consumption. Literature currently lacks sufficient standardisation to adequately assess the impact that architectural differences have on real-world performance.

As such, the core research problem that is being investigated is the need to perform an organised and controlled performance comparison (volume testing) between the specified frameworks (React and Angular) on consistent and defined benchmarking metrics to determine and report on the measurable behaviour of both frameworks under like conditions. Some of the specific types of measurable performance indicators to be analysed are: load time, rendering speed, memory usage, CPU utilisation, and DOM manipulation efficiency. By addressing this gap, the study aims to provide evidence-based insights that support informed decision-making in frontend framework selection.

## 2. RELATED WORK

This section presents the related studies about the performance comparison of React and Angular.

The work [1] studies evaluated three prominent frameworks (React, Angular, & Vue.js) for performance and scalability and included a detailed examination of their implementation in terms of workload characteristics especially related to those factors common among large and complex web applications. Through the creation of equivalent CRUD-based benchmarks in each framework, comparisons were made across both frameworks through standard benchmarks that measured key areas including: initial and subsequent load time, rendering speed, DOM update efficiency (in terms of how efficiently were updates made to the visual representation of an application), peak memory consumption, frequency of memory leaks, and CPU utilization. Evaluation of all frameworks was achieved using browser-based profiling tools such as Chrome DevTools that enabled real-time visualization of runtime characteristics. The findings of this study highlight the advantages of utilizing React due to its virtual DOM mechanism that provided efficiency in terms of rendering and competitive scalability. However, in addition to providing the aforementioned advantages in relation to use, React also suffers from the possibility of performance degradation when working on highly complex state transitions. Although Angular has the benefit of being a rich feature set and a highly structured approach to development, it demonstrated generally slower initial load times than the other two frameworks, as well as higher overall memory consumption (in part due to its extensive framework architecture and also its use of a two-way data-binding mechanism). Finally, Vue.js provided balanced levels of performance, and demonstrated generally lower memory consumption than either of the other two frameworks, as well as providing consistent levels of rendering speed regardless of the testing methodology utilized.

The work [2] researchers presented a structured method to measure rendering performance using Angular, React, and Vue with an emphasis on metrics rather than qualitative data from controlled experiments (using the virtual DOM vs incremental DOM paradigms). Through a standardized web application, the researchers had consistent conditions to compare and evaluate each of the frameworks. Random software (Selenium) was used to automate testing within a Docker based system. The researchers could obtain accurate data on how long it took to render when manipulating the DOM by monitoring the lifecycle hooks provided by each framework. By collecting data in this way, the researchers found that Vue provided the best performance with respect to manipulating large numbers of DOM elements, particularly during add and delete operations. Angular (incremental DOM), was very good at performing update operations because of how it processes nodes directly. React was able to achieve fairly competitive Times to Interactive, and was generally ready for user interaction faster than the other frameworks, but experienced some performance hits when adding an extremely large number of elements to the DOM. The study also highlighted differences in production bundle size, noting that Angular generated comparatively larger builds, which may influence download and initialisation time. While the research provides a well-defined benchmarking methodology and detailed performance metrics, it focuses primarily on rendering efficiency within a controlled toy application.

The study referenced in [3] is primarily about measuring front-end website performance metrics. The researchers created two functionally identical websites

using both frameworks for the practical implementation of this study design. The performance evaluation used measurable indicators (e.g., code size, response time, and page load times for both frameworks) as the basis for conducting this evaluation. In contrast to purely architectural research, the authors conducted their research through real deployments of the website using GTMetrix to record multiple performance metrics: DNS lookup time, connection time, wait time, and load full page time. Results show that AngularJS will produce a smaller rendered file size than ReactJS; however, even though the size of the code used by ReactJS is larger, it consistently outperformed AngularJS in run time performance when accessed through a browser. ReactJS had consistently lower response times than AngularJS and less time to completely load a web page than AngularJS. The data in the response time table is also consistent with this finding where the total wait and receive time for ReactJS were less than for AngularJS. The metrics for fully loaded time for creating web pages and load time for creating DOMs also supports the fact that ReactJS had better client-side rendering performance than AngularJS.

The work [4] conducted a detailed comparative study of Angular, React, and Vue within the context of full web application development, focusing primarily on the user interface layer while also considering server-side integration. The authors developed three functionally identical applications - an online car-selling system, to ensure objective evaluation across frameworksEach implementation was created using a different scripting environment and styling methodology: Angular actually made use of TypeScript and SCSS, React used JavaScript and styled-components and Vue used JavaScript with standard CSS. The way that each technology worked mechanically provided a way to make a systematic comparison of all aspects such as structural complexity (the form in which the application has been created), code organisation (where files live), rendering performance (how well or fast the application functions) and development effort (how long it took to create and develop the implementation). Several technical items were analysed in this entire process including file and directory structure, number of lines of code, total character count, compilation time, time to render the component in a browser, and time required to build an individual component among other items. The results show that Angular has the longest compilation time of 25 seconds when compared to 15 for React and 12 for Vue. Additionally, even though Angular created the largest number of total characters used in the HTML and scripting, it also had the most structural complexity as indicated by the amount of HTML and scripting created. React typically has the lowest amount of code used, while Vue lies between both React and Angular when comparing to implementation size. Browser developer tools were used for rendering performance evaluation, and it was determined Vue has a slightly longer time to generate components in the browser than either React or Angular. The time frame to develop an individual component is yet another area where the technologies differ from one another. Angular demanded the most development time, largely due to TypeScript usage and its structured configuration requirements. Although the study provides a comprehensive comparison of development complexity and structural characteristics, its primary emphasis lies in code volume, compilation time, and implementation effort rather than detailed runtime performance metrics such as memory usage, CPU utilisation, or high-load scalability. Therefore, additional experimental research focusing on measurable execution performance under controlled workload conditions remains necessary to further clarify performance trade-offs between React and Angular.

The work [5] presented a structured comparative analysis of Angular and React within the context of single-page application development. The study evaluated both technologies from multiple angles by comparing the technologies based on performance, scalability, usability, speed of development, support from the community, flexibility, ability to integrate with APIs and security. This project adopted a wider scope of evaluation than other benchmarking studies which have been based on only one parameter. This has been achieved through the use of a combination of the literature review method, empirical test methods and actual assessment of either technology's capabilities. The definition of both technologies' architecture was included in this study. While Angular has been defined here as an all-encompassing framework (meaning a framework that is very prescriptive) complete with several tools already included (e.g., routing; the ability

to inject dependencies; & the ability to create and manage forms), React has been defined as a 'lightweight' (very non-prescriptive) library using a component-based architecture with a virtual DOM rendering method. In terms of performance; the speed at which an application will load when first accessed by a user or the efficiency with which an application responds to a user's actions on it; React out-performed Angular primarily because of improvements made to the virtual DOM whereas Angular was heavier than React in terms of memory usage due to two-way data binding and its very broad feature set because Angular is ideal for large enterprise-scale applications that are going to be developed in a structured manner. When evaluating scalability and maintainability; Angular was considered to have a more simplified code base due to the fact that it used the same predefined architecture throughout; and; while React provided more flexibility in developing applications than Angular; it required that developers plan for the architectural decisions that would be made throughout the course of developing the application. From the standpoint of the Developer Experience; React was considered to be easier to learn than Angular; however; the amount of time that would be spent learning all the tools associated with Angular's (comprehensive) ecosystem would require a developer to spend more time learning than they would spend with React. Community and ecosystem analysis indicated that both technologies benefit from strong industry backing, though React possesses a broader third-party ecosystem. Although the research offers a multi-dimensional comparison and identifies practical trade-offs between the two frameworks, its performance analysis primarily relies on general benchmarking and qualitative interpretation rather than detailed runtime profiling under controlled experimental conditions. Consequently, further empirical investigation focusing on measurable metrics such as DOM rendering time, memory consumption under load, and scalability stress testing would strengthen the understanding of performance differences between React and Angular.

## 3. METHODOLOGY

In this experiment, three popular JavaScript Frameworks (React, and Angular) were tested using a lightweight experimental approach to compare their performance. The methodology of this research is-to ensure that the testing process was conducted in a fair manner while still providing the same opportunity for all frameworks to be compared equally; thus, it allows for reproducibility, i.e., someone else can repeat the experiment and get the same results as another person does, and to ensure simplicity when conducting an experiment in an environment with very few resources available to conduct research at an academic institution.

The experiments used all low-cost/free software applications and were performed using a basic standard desktop computer rather than using high end equipment or a sophisticated automation tool (automated testing tool). The application structure across all three frameworks was the same. Therefore, the methodology for this study will focus on using common practical measurements (without requiring either expensive or unique measurement equipment) to evaluate performance differences across the three frameworks. By providing a detailed methodology along with practical measurement results for each framework, students/researchers will have an easier time replicating these experiments in their own academic setting.

### 3.1 Experimental Environment Setup

All of the experiments for this study were carried out on a personal computer and not using any virtualisation or cloud services (containers). The point of using the same environment in which a student or smaller developer would normally work, was to test how the frameworks perform under the real-world conditions a student or small developer would be using.

The system used in these experiments can be described as having a simple processor, limited RAM, and having a standard operating system. The frameworks were installed using Node.js, and the Node Package Manager (NPM) was used to manage the packages. Visual Studio Code was the code editor used to develop the frameworks.

In order to measure the performance of the frameworks, all testing was conducted using Google Chrome as the web browser. Google Chrome DevTools and the Lighthouse performance auditing tool were used to analyse various performance metrics including load time and Time to Interactive. In addition, the performance of the DOM manipulation of the web

pages was measured using performance API (performance.now()), and the time as measured by performance.now was compared before and after DOM manipulation.

### 3.2 Application Design and Implementation

To ensure a fair comparison between Angular and React, a simple web app was created with both Javascript frameworks using an identical structure. The app's function (to keep things as simple as possible) and visual design (to eliminate visually distracting factors that might affect performance) were intentionally kept to a minimum thereby removing numerous complex elements and allowing us to compare the two platforms solely on the way they render and manipulate the DOM as opposed to any advanced or architectural features of either framework.

The app has a basic List-type structure that allows four primary functions to be performed.

- Creating multiple elements
- Modifying an element
- Modifying all elements
- Deleting all elements

The two versions of the app created with each framework have the same layout and logic. They have only a button to interact with (creating, modifying, and deleting elements) and an area to display elements created by the user. When the user interacts with the user interface, the DOM is modified based on the framework used to create the app.

The same number of elements (100, 500, and 1000) were used in both framework versions of the app. All four functions of creating, modifying and/or deleting elements in the app were performed using the core state management provided by each framework. No third-party libraries, UI component libraries or optimization plugins were utilized. This ensures that the performance data obtained from the test reflects the frameworks' core capabilities and not third-party influences on their performance. The app was styled in a minimalistic way.

### 3.3 Performance Metrics

To evaluate how each of the frameworks performs in terms of their performance, some simple performance metrics were created, that relate to frontend performance, and can be obtained using standard, free browser tools, focusing on metrics that represent the user experience and also how efficiently the framework renders.

#### 3.3.1 Build Bundle Size

The sum of all the JavaScript files created after creating a production build of a web application is the total size of the build. Smaller bundles generally will lead to faster download speeds and less time spent to load the application, which is especially important when working with slow connections. Production builds of the frameworks in the study were created using the default building commands of their respective CLI's. This metric shows how much additional "weight" is added by the framework, even for a very simple application.

#### 3.3.2 Page Load Time

Page load time is defined as how long it takes for an application to display in the browser after it has been requested. Page load times were obtained by using the Google Chrome DevTools and the Lighthouse auditing tool to record page load times for when the user started the request until the page displays all the main contents fully loaded and rendered. The applications used in this study are simple applications that do not require access to external APIs.

#### 3.3.3 Total Blocking Time (TBT)

As part of measuring page load responsiveness, TBT was chosen as an important indicator of page responsiveness during the loading phase. While visual metrics measure how fast content appears on the screen, TBT provides information about how responsive an application is while scripts are executing. TBT will be measured using the Lighthouse performance reporting tool available within Google Chrome. The application will be tested in desktop mode under controlled conditions and the TBT value will be extracted from the generated report. Each

framework will be tested several times and the average reported value will be recorded to reduce variance between test runs.

### 3.3.4 DOM manipulation time

DOM manipulation time is a measure of how efficiently the framework performs dynamic updates to the webpage. DOM manipulation time was subdivided into four separate types of operation performed on the DOM:

- Creating multiple elements
- Editing one element
- Editing all elements
- Deleting all elements

To assess the efficiency of the framework with respect to the above types of operation, timestamps will be recorded before and after each type of operation, using the Performance Measurement API in the browser, in order to get the elapsed execution time. High levels of efficient handling of the DOM will significantly improve an application's smoothness and responsiveness.

### 3.3.6 CPU Utilisation

To determine the processing overhead (CPU Utilization) introduced by each framework during runtime execution. In terms of performance, we can predict how responsive each framework or user interface is based on load time and DOM manipulation time measurements. However, it can provide further insight into the amount of work necessary to perform rendering and state updates through CPU profiling. CPU activity was measured utilizing the Performance Profiling tool found in Google Chrome DevTools. Upon conducting each test scenario, recording would take place before executing the DOM operation and immediately after the completion of the operation. Scripting time acts as an indirect measure of how much work the CPU has done as JavaScript is executed on the main processor of the browser to run the application. Because of this, scripting time is an additional metric to consider for runtime efficiency beyond visual performance measurement metrics.

### 3.3.7 Memory Consumption

The measurement of memory consumption allows us to evaluate the impact of each framework on browsers' memory allocations during the performance of dynamic operations. In order to maintain application stability over long periods of time (especially with single-page applications), memory management is an essential factor. Memory consumption assessment was measured in Google Chrome DevTools using the Memory tab. A heap snapshot was taken before and after executing the selected DOM manipulation operation. The difference of those two measurements showed how much JavaScript heap memory usage increased in performing the DOM manipulation operation. The primary metrics collected were the heap size differences. This number shows how much additional memory the DOM manipulation consumed.

## 4. RESULT

Each metric was recorded across five independent test runs, and the average value was calculated to ensure consistency and reduce experimental variation. All measurements were conducted using production builds under identical testing conditions.

### 4.1 Bundle Size Comparison

The production build output shows that React generated a bundle size of 191 KB, while Angular produced a slightly larger bundle of 198 KB.

Table 1: Bundle Size Comparison

| Framework | Bundle Size (KB) |
|---|---|
| React | 191 |
| Angular | 198 |

The difference of 7 KB is relatively small and does not represent a significant overhead for small-scale applications. However, Angular's marginally larger bundle size may be attributed to its integrated architecture, which includes built-in dependency injection and change detection mechanisms.

### 4.2 Load Performance Evaluation

Load performance was measured using Lighthouse in desktop mode. The average results are presented below.

Table 2: Load Performance Metrics

| Metric | React | Angular |
|---|---|---|
| First Contentful Paint (sec) | 0.20 | 0.40 |
| Largest Contentful Paint (sec) | 0.38 | 0.40 |
| Total Blocking Time (ms) | 0 | 4 |
| Performance Score | 100 | 100 |

React demonstrated a faster First Contentful Paint, rendering initial visible content in approximately half the time required by Angular. The Largest Contentful Paint values were nearly identical, indicating comparable rendering of primary content elements.

Total Blocking Time was negligible for both frameworks; however, Angular recorded a small average blocking time of 4 ms. Despite these differences, both frameworks achieved a perfect Lighthouse performance score of 100, indicating optimised performance under the tested conditions.

### 4.3 DOM Manipulation Performance

DOM operations were evaluated for element creation, update, and deletion. The average results are shown below.

Table 3: DOM Manipulation Performance (Average in ms)

| Operation | React | Angular |
|---|---|---|
| Create 100 elements | 0.14 | 0.06 |
| Create 500 elements | 0.16 | 0.08 |
| Create 1000 elements | 0.16 | 0.16 |
| Edit 1 element | 0.10 | 0.04 |
| Edit all elements | 0.60 | 0.84 |
| Delete all elements | 0.06 | 0.00 |

For element creation tasks, Angular performed slightly faster for smaller datasets (100 and 500 elements), while both frameworks showed identical performance at 1000 elements.

In single-element editing, Angular demonstrated lower execution time. However, for bulk updates (edit all elements), React showed better performance compared to Angular.

Deletion performance differences were minimal due to extremely small execution times, making practical differences negligible.

Overall, DOM manipulation performance was comparable, with slight advantages alternating between frameworks depending on the operation type.

### 4.4 CPU Utilisation (Scripting Time)

Scripting time was measured using Chrome DevTools Performance profiling.

Table 4: Average Scripting Time

| Framework | Scripting Time (ms) |
|---|---|
| React | 58.2 |
| Angular | 28.4 |

Angular required significantly less scripting time compared to React during execution. This indicates that Angular's runtime operations consumed fewer CPU cycles under the tested workload. React's higher scripting time may be influenced by internal virtual DOM reconciliation processes.

### 4.5 Memory Consumption

Memory usage was evaluated by comparing heap snapshots before and after DOM operations.

Table 5: Memory Usage Comparison

| Framework | Average Heap Increase (MB) |
|---|---|
| React | 3.0 |
| Angular | 1.9 |

Angular demonstrated lower memory growth during execution, while React exhibited a higher heap increase. The additional memory usage in React may be associated with its internal state management and virtual DOM structures.

## 5. CONCLUSION

In this study, an evaluation of the performance of the Angular and React frameworks was conducted using an experimental methodology suitable for a low-resource academic setting. The primary aim of the research is to describe performance characteristics associated with practical, "real-world" systems. The value of the study was to establish controllable measurements in order to provide researchers and developers with reliable data about the performance characteristics of both frameworks, including bundle size, load performance, efficiency of DOM manipulation, CPU utilisation, and memory usage under tested workloads.

Both frameworks are found to perform well for small to medium sized applications within the parameters of the study. React has an advantage in the initial visual rendering speed, especially with respect to the First Contentful Paint (FCP). React also showed a good performance for bulk update functionality involving multiple DOM elements. Angular has lower average scripting times and less growth in memory while executing. Both of these attributes suggest that Angular is able to perform more conservatively with respect to the amount of resources used as measured by the study parameters.

Relative to each other, the bundle size for both frameworks is approximately the same, as is the performance score on Lighthouse. The measured performance when manipulating elements in the DOM is minimal and the results are dependent upon the specific operation being performed within each framework -- neither framework dominates in all cases.

When comparing the performance of React and Angular in a controlled local test with low levels of complexity, their differences were found to be inconsequential. Consequently, decisions about which framework to use should not be based only on absolute measures of performance; other considerations, such as the specific requirements of the project, architecture, the economic context of the project, scalability and dependability of the frameworks, and the developers' level of expertise, should also be taken into account when making such choices.

While the experiments were conducted on a basic application with very few external calls made to an API, and with no routing complexity included, as well as not including any simulation of the latency that might be experienced over a network, the results relate to the performance characteristics of the frameworks under controlled, academic conditions and not in large, production-type environments. Future research in this area could expand upon this research by including real-world scenarios, more comprehensive datasets, testing on mobile devices, and testing in network-throttled conditions; therefore, providing further insight into the performance of the frameworks in real-world deployment scenarios.

In summary, both React and Angular can provide efficient, reliable options for building modern web applications; however, due to the fact that their performance was comparable but not markedly different, between the two frameworks for the purposes of this study, it cannot be said that one framework would have an overall advantage over the other in any way, including building modern web applications.

## 6. References

[1] Mikita Piastou, "Performance and Scalability for Three Front End Frameworks: React, Angular, and Vue.js," will be published in the World Journal of Advanced Engineering Technology and Sciences, 2023, 09(02), 366-376 DOI: https://doi.org/10.30574/wjaets.2023.9.2.0153.

[2] R.N.V. Diniz-Junior et al., "Evaluating the Performance of Web Rendering Technologies via javaScript - Angular, React and Vue," 2022 XVLIII Latin America Computer Conference (CLEI), Armenia, Colombia, 2022, pp. 1-9, doi: 10.1109/CLEI56649.2022.9959901.

[3] Ilham Yusron and Antoni Wibowo "Performance Compare of ReactJs and AngularJs on the Front End of the Website" International Journal of Advanced Science and Information Technology, Vol. 9 No. 4 pp. 1456 - 1463 year 2020.

[4] P. Dymora, M. Mazurek, M. Nycz "Comparison of Angular, React and Vue3 Technologies in the Process of Web Applications Creation on the UI Side." Journal of Education, Technology and Computer Science No. 4(34), pp. 210-222, year 2023.

[5] P.R. Rao, P.Priyanshi and S. Vashishtha "Angular Versus React - A Comparative Study of Single Page Applications" International Journal of Current Science (IJCSPUB) Vol. 13 No. 1 pp. 875 - 894 January 2023.

[6] "Performance optimization techniques for reactjs," in 2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), 2019, pp. 1–5.

[7] Tarun Sharma, Shruti Gupta, Uday Raj Singh, "Analyzing the difference between ReactJS and AngularJS", 2023 International Conference on Computational Intelligence, Communication Technology and Networking (CICTN), pp.37-42, 2023.

[8] S. Delcev and D. Draskovic, "Modern javascript frameworks: A survey study," in 2018 Zooming Innovation in Consumer Technologies Conference (ZINC), 2018, pp. 106–109.

[9] 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.