Comparative Performance Analysis of JAMstack and Monolithic Web Architectures

Khushi R. Shah
Department of ICT
Veer Narmad South Gujarat University
Surat

Payal D. Joshi
Department of ICT
Veer Narmad South Gujarat University
Surat

ABSTRACT
User experience and engagement are the most important factors in web development, as they greatly influence the speed and reliability of a website. In recent times, with the emergence of full-stack development, JAMstack architecture has gained importance. That said, it does not mean that monolithic architecture is less useful and should fail to be used at all. Both quantitative and qualitative analysis of the client-side performance of two web architectures, monolithic and JAMstack, help bridge the research gap in existing studies by providing A clear, data-driven comparison of their efficiency in handling dynamic content across varying volumes. It shows how performance metrics such as Time to Interactive (TTI), Time to First Byte (TTFB), and memory consumption are sensitive based on architectural characteristics. The monolithic architecture often suffers from slow initial load times, as it is heavily dependent on server-side processing, regardless of the data amount. JAMstack provides a swift supply of static assets using a CDN, where the performance of this architecture mainly depends on the amount of dynamic data being fetched and rendered from the client side. JAMstack is mostly used when we need to reuse the backend code in multiple frontend apps. It also investigates how the memory usage of each architecture differs across browser engines. More importantly, such information is capable of helping organizations make better decisions about their web infrastructure that suit their specific project objectives.

General Terms

Performance, Measurement, Experimentation, Design, Web Architecture

Keywords
JAMstack, Monolithic Architecture, Web Performance, Time to Interactive (TTI), Time to First Byte (TTFB), Memory Usage, Google Lighthouse, GTmetrix, Content Delivery Network (CDN), Single-Page Application (SPA).

1. INTRODUCTION
The responsiveness of a website strongly influences user interactions and engagement. Web developers face a big challenge in designing a site that meets user aesthetics, is accessible, and is reliable anywhere in the world. This holds true also for the User Interface (UI) and User Experience (UX) domains of web design because performance and responsiveness of a site directly impact how users perceive and interact with it. Despite the site having good content, a slow user interface can lead to a poor user experience. This has led to the emergence of different website architectures, each with its own strengths and weaknesses.

Monolithic and JAMstack are the two most popular architectures for building modern Web applications. The monolithic model is known as the traditional approach, where the components of website are closely interconnected, both the frontend and backend operate as a single unit. In contrast, the JAMstack model separates the components of a website, where the frontend, backend, and reusable parts are separated using prebuilt static files and APIs to render content more quickly. This helps in reusing the backend code in multiple frontend applications.

Even though there is a lot of discussion about the pros and cons of both models, there is still a need for quantitative practical based research that shows how they actually perform in terms of speed, memory use, SEO, and user experience. Understanding the benefits and potential limitations of both architectures is important for:

- Better decision-making during infrastructure design in organizations.
- Gaining practical insights into how each architecture performs under varying conditions.
- Evaluating their influence on key performance metrics, such as load time, memory usage, SEO, and Time to First Byte (TTFB).

The aim of the study is to fill that gap by creating identical web applications using both architectures and then comparing their performance in different scenarios.
To explore and compare important factors influencing the performance of both the architectures is the primary objective of this research. This practical and data driven approach can help provide valuable insights to the developers and organizations in better understanding which architecture best suits their needs.

The rest of the paper is organized as follows. The past researches and studies are discussed in the Related Work section. The research objectives, the specifics of developing the web application, the data collection methods and tools for the performance analysis are discussed in the Research Methodology section. The Analysis of the metrics from various tools in detail is elaborated in the Results and Findings section. The key insights gained from the study are discussed in the Conclusion section.

2. RELATED WORK
[1] Sam Whitley (2023) carried out a comparison of the JAMstack and monolithic architectures by using WordPress to demonstrate the monolithic stack, and a modern JAMstack stack with React, serverless functions and Netlify. His outcomes are based on load speed, scalability, and deployment. Whitley's study convincingly demonstrated JAMStack's advantages, as far as load speed and capacity for traffic with load increases go. However, it was limited as it utilized a pre-built CMS platform, asset delivery solely using CDNs, and static asset usage. [2] A mixed-method study by Marković et al. (2022) evaluated the maturity, adoption, and future potential

of the JAMstack architecture, where they reviewed 77 studies and narrowed them down to six key publications. They also conducted an online survey with 44 web developers and held semi-structured interviews with 4 experienced professionals. While the research offered valuable insights into trends in JAMstack adoption and the views of practitioners, it mainly relied on surveys. No experimental performance assessments or considerations of dynamic application scenarios were included. [3] Both a JAMstack and a WordPress implementation was done in the Nguyen study, in order to analyze and compare JAMstack architecture and monolithic architectures. It was founded that JAMstack provided enhanced performance, security, and scalability than monolithic architecture. However, it was limited to a CMS-based monolithic implementation and lacked a JAMstack evaluation of dynamic content rendering and real-time capabilities. [4] A mix of literature review, surveys, and interviews was used for studying JAMstack architecture in Orosz (2020). This approach showed better performance in terms of scalability, performance, and improved security. It also offered the added benefits of a lower learning curve and less maintenance effort but it only focused on qualitative insights and did not involve practical performance testing. [5] CI/CD pipelines for JAMstack applications were set up using CircleCI and Netlify in Hoang et al. (2020). A Gatsby-based JAMstack project was created and deployed where it showcased the automation of build, test, and deployment workflows. Lighthouse data showed performance improvements of about 75%, with a perfect SEO score of 100%. Nevertheless, its scope was confined to JAMstack, excluding any comparison with monolithic or database-centric dynamic applications. [6] A JAMstack based e-commerce website was developed using Next.js, Sanity, and Tailwind to showcase the applicability of JAMstack architecture in the study by Nguyen (2022). They achieved high performance, improved security, scalability, and readiness for SEO through their approach. However, their evaluation focused only on one e-commerce case study. There was no quantitative comparison or direct analysis with monolithic systems. Furthermore, they did not explore real-time dynamic content or extensive user interaction. [7] In W. Ruoxuan and M. Uehara study, a React-based JavaScript Development Environment (ReJDE) was developed as a single-page application (SPA) for programming education on smartphones. The main aim was to address the original JDE's inability to support multiple curricula and save the learning history. A headless CMS (microCMS) was used for implementing a multitenant system and a "notebook" feature, inspired by Project Jupyter, was also introduced. However, this study is fundamentally limited by its exclusive focus on a single architecture and the absence of a direct side-by-side comparison with a traditional monolithic system. [8] In practice, a modern Single-Page Application (SPA) architecture was used in a 2019 case study by Gavrilă, Băjenaru, and Dobre. The design features client-side rendering, asynchronous data loading, and API-based content delivery with a deliberate separation between the frontend and backend. The design led to faster loading times, better user experience, and reduced maintenance. No direct comparison with monolithic architectures was made, but as mentioned, the benefits in some ways resemble those of scalability and flexibility related to the latter with reference to Jamstack-based approaches. [9] Kowalczyk and Szandala (2021) investigated the SEO performance of Single Page Applications (SPAs) and Multi Page Applications (MPAs). When techniques such as prerendering, enhanced metadata, and performance improvements are applied, it is found that SPAs exhibit similar SEO performance to MPAs, based on the research conducted. This study primarily focuses on SEO in SPAs vs MPAs and its rendering strategies and lacks broader architectural insights and raw performance metrics. [10] In his 2015 investigation, Nygard (2015) analyzed how the Single- Page Application (SPA) architecture can be a basis for creating modern web applications to be both scalable and responsive. Three SPA prototypes were created using HTML, CSS, JavaScript, AJAX, and API-driven data retrieval. The results were that SPAs can provide a more consistent and smoother user experience (less page reload and quick transitions) in a unified UI experience. Yet issues lingered about SEO, semantic HTML support, and the extensive use of JavaScript.

2.1 Research Gap

These studies uncover and explore many dimensions of various architectures. Most of the related work looks at strengths of JAMstack in a qualitative way, often focusing on single architecture evaluations or comparing it with pre-built CMS platforms like WordPress, which are not true monolithic environments. A major gap in the existing research is the lack of direct, data-driven comparisons between a custom-built monolithic application and a JAMstack application, especially when it comes to raw performance under varying data loads and how resources are consumed across different browsers. This study addresses that gap by building two identical web applications and carrying out a practical, data driven analysis of performance indicators and memory usage, providing a clearer understanding of the differences between monolithic and JAMstack architecture.

3. METHODOLOGY

3.1 Research Design

This paper is an analysis of the effectiveness of two versions of the same web application: a hotel website in a monolithic architecture and JAMstack architecture. Although the two websites utilize the same back-end framework including PHP, HTML, CSS, and a MySQL database, their front-end architecture as well as the delivery approach is very different. The website runs in several parts but is built on the data of 200 cities and their corresponding hotels (see Table 1).

• Monolithic Website: This follows a tightly coupled structure with PHP as the backend framework, HTML as the frontend, and a MySQL database. The entire page is rendered on the server with each request. The images are served through CDN.

• JAMstack Website: This decoupled approach leverages APIs and Ajax for the frontend, with PHP serving as the backend API and MySQL as the database. The frontend uses a single-page application (SPA) model, where the navbar, slideshow, footer, and left navbar remain fixed, and the content is loaded dynamically within a window frame. The images are served through CDN.

Table 1. Research Methodology Details

Aspect	Details
Study Type	Experimental study comparing two versions of a same website based on Monolithic and JAMstack architectures.
Platform	A common back-end platform built with Core PHP and MySQL, while the front-end components and data delivery methods differ.
Features	• 5 website sections (Home, FAQ, Contact, Gallery, Hotels) • Image serving from CDN

	• Dynamic hotel data based on a dataset of 200 cities • 4 cities have 100 hotels and rest 196 cities have 5 hotels
Website A	**Monolithic Version:** Tightly coupled architecture. Backend: PHP, Frontend: HTML/CSS, Database: MySQL. All pages are server-side rendered (SSR).
Website B	**JAMstack Version:** Decoupled architecture. Frontend: JavaScript/Ajax with PHP serving as the backend API. It uses Single-Page Application (SPA) model where key elements are fixed, with content loaded dynamically in a window frame.
Hosting Platform	InfinityFree
CDN Service Provider	Cloudinary

3.2 Research Objectives

• Evaluate and compare key performance indicators (KPIs) and throughput between monolithic and JAMstack based website.
• Analyze memory usage/heap size by both websites on different browsers.
• To understand the best situations for using each architecture.

3.3 Website Development and Setup

Both websites are designed to be identical in content, structure, and functionality, with the only difference being the CDN integration, hosted on the same server to ensure independent and fair analysis of performance. Details of the hosting and CDN platforms are as given below:
• Hosting platform: infinityfree.com
• CDN Integration Service: Cloudinary

InfinityFree is a free web hosting provider that helps in managing websites based on PHP and MySQL. It operates on a cloud-based infrastructure that ensure a high uptime, unlimited storage space, free subdomains, SSL and DNS services with many other tools providing effortless deployment. Cloudinary is a SaaS based media asset delivery platform that handles media upload, storage, optimization and delivery via CDN services. It has high performance media processing servers, free tier storage and supports dynamic asset URLs. The primary purpose of choosing this CDN provider was for the easy SDK integration with multiple tech stacks which generate dynamic media URLs after upload. However, Cloudinary has not publicly disclosed the exact number of edge servers, but they claim to utilize many strategically placed edge servers worldwide.

3.3.1 Performance Measurement Tools

To measure website performance, the following tools were used to track the key metrics:
• Memory tab in browser developer tools
• GTmetrix
• Google Lighthouse

These tools provided quantitative results on performance, including load times, request handling, Largest Contentful Paint (LCP) and load times and overall page speed metrics for both websites.

4. RESULTS AND FINDINGS

The following results were recorded from tools such as GT Metrix, Google Lighthouse and browser developer tools.

4.1 Memory Usage Analysis

The data from the Memory Tab of browser Developer Tools were compared for both Websites on the following browsers:
1. Google Chrome
2. Microsoft Edge
3. Mozilla Firefox

On Google Chrome, the Monolithic website occupies significantly more memory on load. The monolithic site's heap size is 17.7 MB, which is approximately 153% more than the JAMstack site's heap size of 7.0 MB (see Figure 1, Figure 2, Table 2).

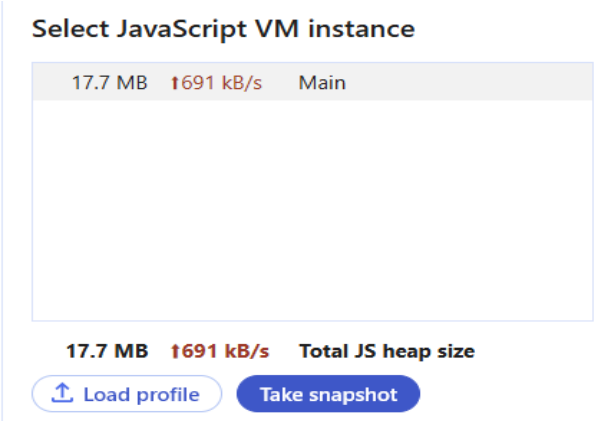

Fig 1: Monolithic Website: Google Chrome - Memory Panel Developer Tools

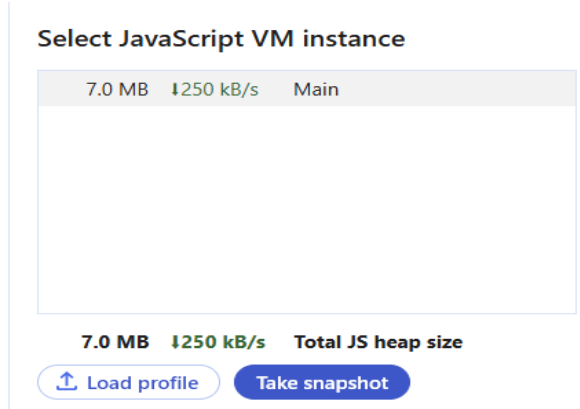

Fig 2: JAMstack Website: Google Chrome - Memory Panel Developer Tools

Similarly, on Microsoft Edge, the monolithic website uses more memory. The monolithic heap size is 11.5 MB, which is about 60% more than the JAMstack site's heap size of 7.2 MB (see Figure 3, Figure 4, Table 2).

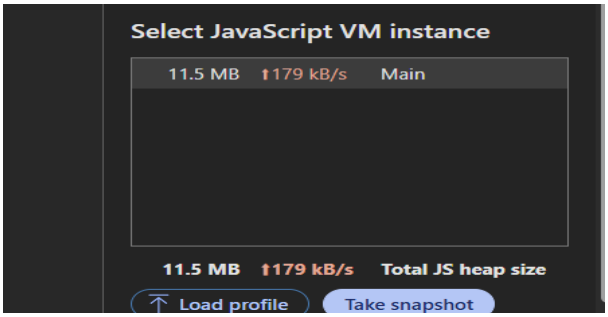

Fig 3: Monolithic Website: Microsoft Edge - Memory Panel Developer Tools

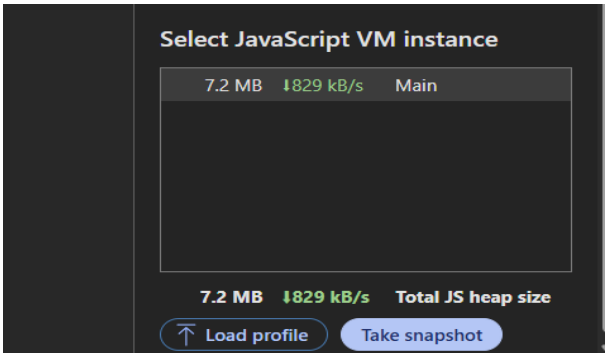

Fig 4: JAMstack Website: Microsoft Edge - Memory Panel Developer Tools

In contrast, the results on Mozilla Firefox are completely opposite. The JAMstack site consumes 4.77 MB of RAM and the monolithic site 3.83 MB. The memory footprint of the JAMstack site is almost 25% larger on Firefox, the opposite trend as Chrome and Edge (Fig. 5., Fig. 6.). This variation of results can be ascribed to the contrast of the browser engines, as Chrome and Edge are Chromium-based while Firefox utilizes the Gecko engine (Table 2.).

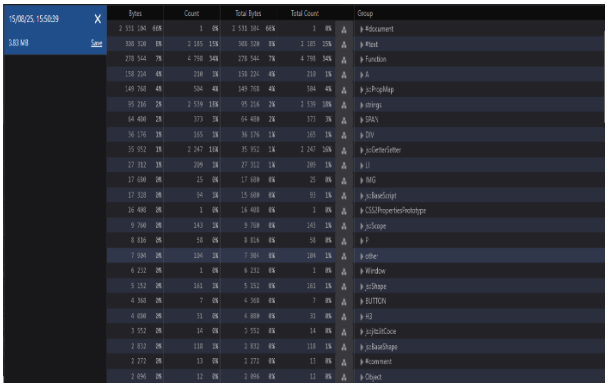

Fig 5: Monolithic Website: Mozilla Firefox - Memory Panel Developer Tools

Fig 6: JAMstack Website: Mozilla Firefox - Memory Panel Developer Tools

Monolithic and JAMstack architectures exhibit fundamentally different performance characteristics. JAMstack excels in initial user experience and backend efficiency, while the monolithic site shows some scalability with data reduction, but at the cost of a much slower start.

Table 2. Memory Analysis Results

Browser	**Monolithic** Site	**JAMstack Site**
Google Chrome	17.7 MB	7.0 MB
Microsoft Edge	11.5 MB	7.2 MB
Mozilla Firefox	3.83 MB	4.77 MB

4.2 GTmetrix Comparison Report

4.2.1 Performance (%)

The Jamstack site has a higher performance score of 65%, compared to the monolithic site's 55%. This reflects the faster initial load times and more efficient delivery (see Figure 7).

4.2.2 Structure (%)

The Jamstack site has a much higher structure score of 96%, compared to the monolithic site's 73%. This indicates a more optimized and well-organized front-end code and asset delivery for the Jamstack architecture (see Figure 7).

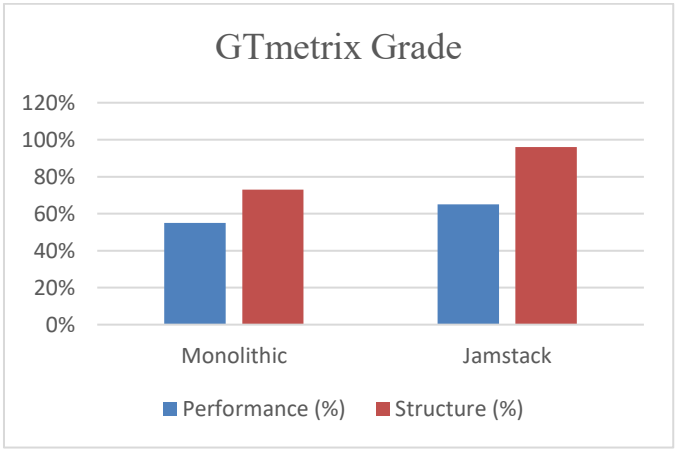

Fig 7: GTmetrix Grade Comparison

4.2.3 Web Vitals

These metrics are used for determining the User Experience (UX).

- First Contentful Paint (FCP): How quickly the first piece of content is rendered on the screen is indicated by FCP. The FCP of Jamstack site is approximately 0.8-0.9 seconds, indicating an almost instant display of the first piece of content. Whereas, the FCP of the monolithic site is around 9.3 seconds, showing a user waits longer for the content to be rendered (see Figure 8).
- Time to Interactive (TTI): TTI shows when a website becomes fully interactive. The Jamstack site becomes fully interactive in 0.8-0.9 seconds. Whereas, the monolithic site shows a TTI of 14 seconds (see Figure 8) which is much longer than Jamstack site.
- Speed Index(s): This metric indicates how quickly the content is visually populated. The Speed Index of Jamstack site is 12.9s, which is better than the monolithic site having Speed Index of 13.9s (see Figure 8).
- Largest Contentful Paint (LCP): The time taken for the largest visible element to load is shown by LCP. For Jamstack site the LCP ranges from 6.6 to 7.6 seconds, while for the monolithic

site it is around 9.2s showing that the largest element on the Jamstack page loads faster, contributing to a better perceived performance (see Figure 8).
• Total Blocking Time (TBT): Both architectures have a Total Blocking Time of 0, indicating that there is no significant blocking of the main thread from user interaction for either site (see Figure 8).

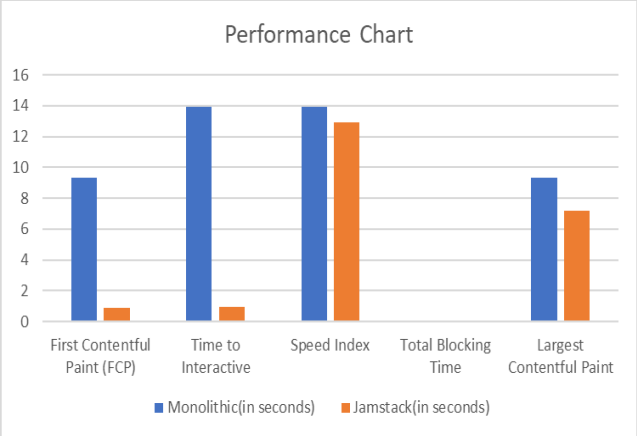

Fig 8: Performance Comparison Chart

4.2.4 Browser Timing Analysis

• Redirect Duration: The monolithic site shows a noticeable redirect duration of about 3 seconds, whereas the Jamstack site completes the redirect at 0.001 seconds (see Figure 9).
• Connection Duration: It indicates the time taken to establish a connection between the server and client. The monolithic site is 0.151 seconds, which is almost twice as fast as the Jamstack 0.307 seconds (see Figure 9).
• Backend Duration: Jamstack demonstrates an extremely efficient backend with durations ranging from 0.15s to 0.17s. This is an order of magnitude faster than the monolithic site, which has a backend duration of 4.9s to 5.7s. This is a key bottleneck for the monolithic architecture (see Figure 9).
• Time to First Byte (TTFB): The time taken for the first byte of data to reach the user is called TTFB. Jamstack has a consistently low TTFB of 0.468 seconds, a direct benefit of serving static content from a CDN. The monolithic site's TTFB is much higher at 8.8 seconds due to extensive server-side processing (see Figure 9).
• DOM Content Loaded Time: The data provided does not contain DOM Content Loaded Time values for a direct comparison (see Figure 9).
• DOM Interactive Time: DOM Interactive Time measures the time the browser takes to first become interactive. The monolithic site takes a long 13.9 seconds for the browser to become interactive, whereas the Jamstack site is ready in just 0.916 seconds (see Figure 9).
• Onload Time: Onload time measures the time the browser takes for the page to load fully. The monolithic site completes the page load in 13.9 seconds, while the Jamstack site finishes in just 0.92 seconds. This is a significant difference (see Figure 9).
• Fully Loaded Time: Fully Loaded Time measures the time when the entire page, including additional scripts and resources, has fully loaded. It turns out that the monolithic site runs a bit faster than that, since Fully Loaded Time varies from 13.9s to 14.8s in the monolithic site, while in the Jamstack it comes in at 17.7s to 18.9s. While the monolithic site starts slow, it may perform some server-side optimizations and finishes the entire page load a little faster. The initial load is fast for the Jamstack site, but this time around it takes a longer time to fetch and render many assets from the client-side, so overall loading times of the site are slow (see Figure 9).

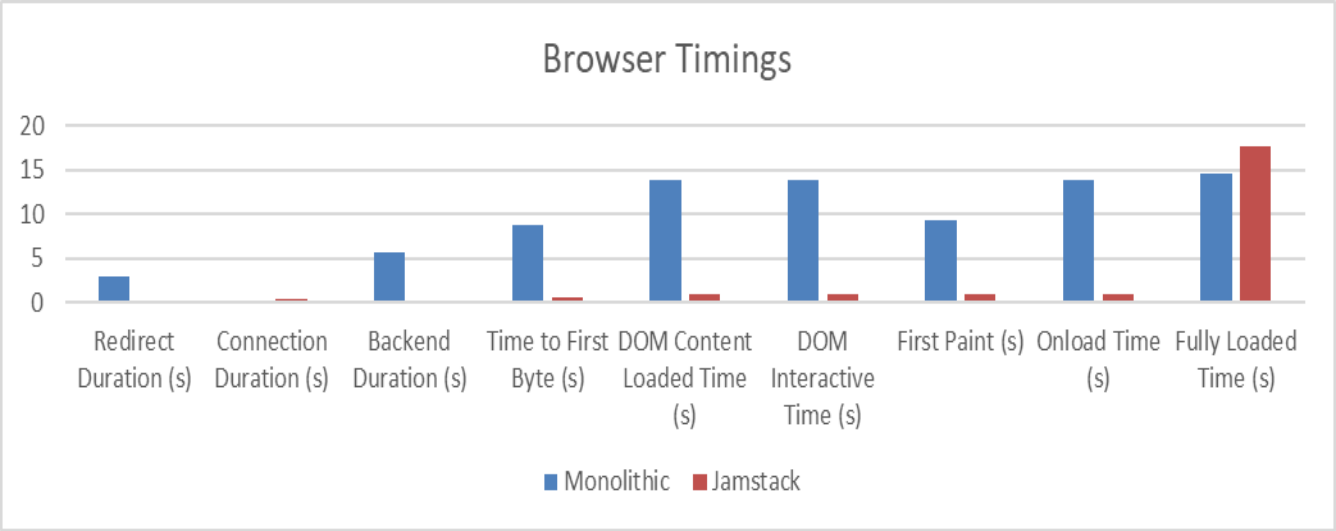

Fig 9: Browser Timings Comparison

4.3 Google Lighthouse Comparison Report

• Performance (%): The monolithic site scores 79, which is slightly higher than the Jamstack's 75 (see Figure 11, Table 3).
• Accessibility (%): The monolithic site scores 75, which is significantly higher than the Jamstack score of 57 (see Figure 11, Table 3).
• Best Practices (%): The Jamstack site scores 83, which is significantly higher than the monolithic score of 57 (see Figure 11, Table 3).
• SEO (%): SEO score of monolithic site is 91 which is higher compared to the Jamstack score of 80 (see Figure 11, Table 3).
• First Contentful Paint (s): The Jamstack site has a faster FCP time (1.1s) compared to the monolithic site (1.6s). Here, the initial content is rendered more quickly in the Jamstack architecture (see Figure 10, Table 3).
• Largest Contentful Paint (s): Both architectures have the same LCP time (1.7s). This means the largest element on the page takes an equal amount of time to render for both the sites (see Figure 10, Table 3).

• Total Blocking Time (s): As both sites get a TBT of 0s, meaning they execute JavaScript efficiently without blocking the main thread (Figure 10).
• Cumulative Layout Shift: Similar to TBT, here also both the sites have a CLS of 0, their layout is stable and there are no unexpected shifts (see Figure 10, Table 3).
• Speed Index (s): Monolithic site has a Speed Index of 5.1s which is slightly faster than that of the Jamstack site (5.6s), meaning that its visual content is populated more quickly than the Jamstack site (see Figure 10, Table 3).
• Initial Server Response Time (s): The Initial Server Response Time of Jamstack site (0.23s) is notably faster than the monolithic site having 6.37s (see Figure 10, Table 3).
• Avoid Multiple Page Redirects (s): The estimated savings of monolithic site from avoiding redirects is higher (0.6s) than the Jamstack site (0.39s) (see Figure 10, Table 3).
• Eliminate Render-Blocking Resources (s): The monolithic site has estimated savings (0.48s) which are higher than the Jamstack site (0.37s) from eliminating render-blocking resources (see Figure 11, Table 3).
• Enable Text Compression (KiB) & Serve Static Assets with Efficient Cache Policy: The Jamstack site has a higher potential for savings from text compression (197 KiB) compared to the monolithic site (118 KiB). The monolithic site has more resources (9) that could benefit from an efficient cache policy, while the Jamstack site has fewer (6) (see Table 3).
• Reduce Unused JavaScript (KiB) & Properly Size Images (KiB): Both sites passed the audit for properly sizing images. The monolithic site has a slightly higher estimated savings from reducing unused JavaScript (21 KiB) than the Jamstack site (20 KiB) (see Table 3).

Table 3. Google Lighthouse Metrics

Metric	Monolithic	Jamstack
Performance (%)	79	75
Accessibility (%)	75	57
Best Practices (%)	57	83
SEO (%)	91	80
First Contentful Paint (s)	1.6	1.1
Largest Contentful Paint (s)	1.7	1.7
Total Blocking Time (s)	0	0
Cumulative Layout Shift	0	0
Speed Index (s)	5.1	5.6
Initial Server Response Time (s)	6.37	0.23
Avoid Multiple Page Redirects (s)	0.6	0.39
Eliminate Render-Blocking Resources (s)	0.48	0.37
Enable Text Compression (KiB)	118	197
Serve Static Assets with Efficient Cache Policy (No. of Resources)	9	6
Reduce Unused JavaScript (KiB)	21	20

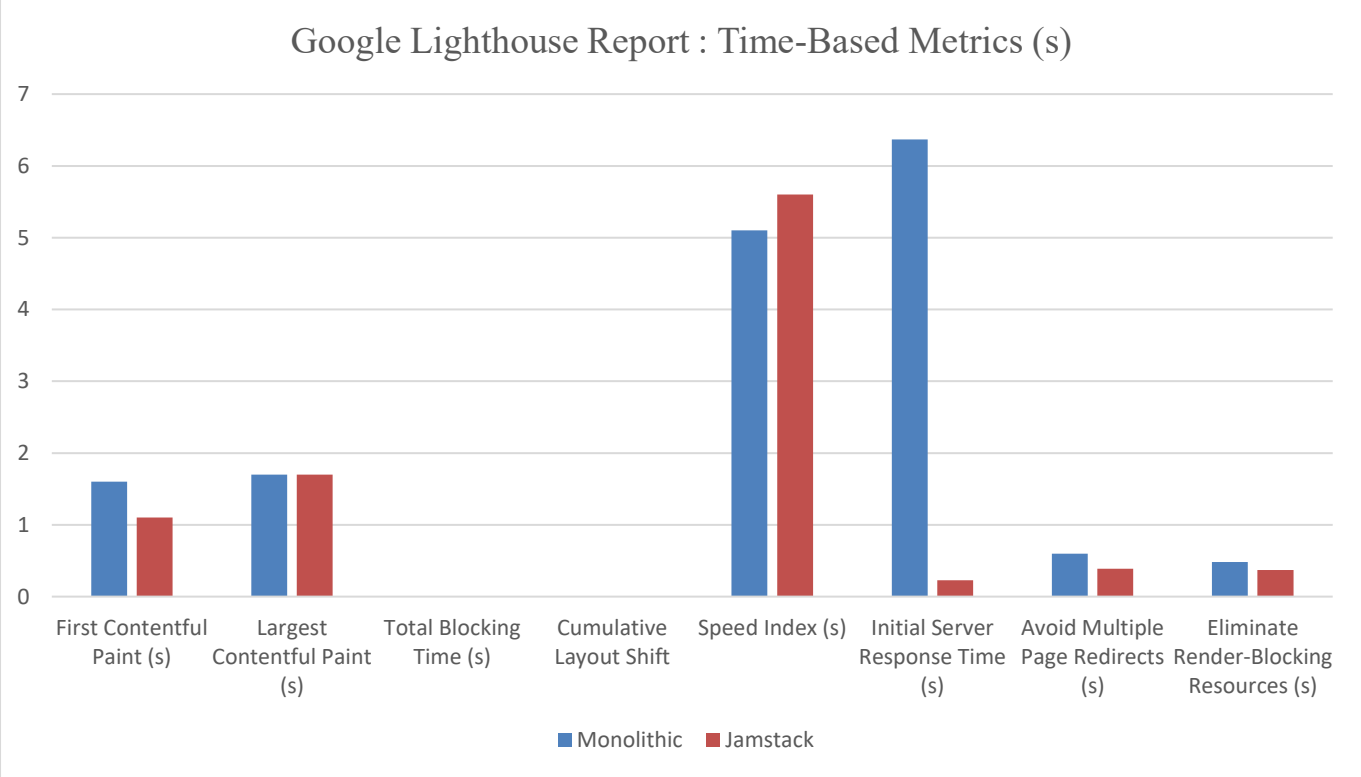

Fig 10: Google Lighthouse Timing Report

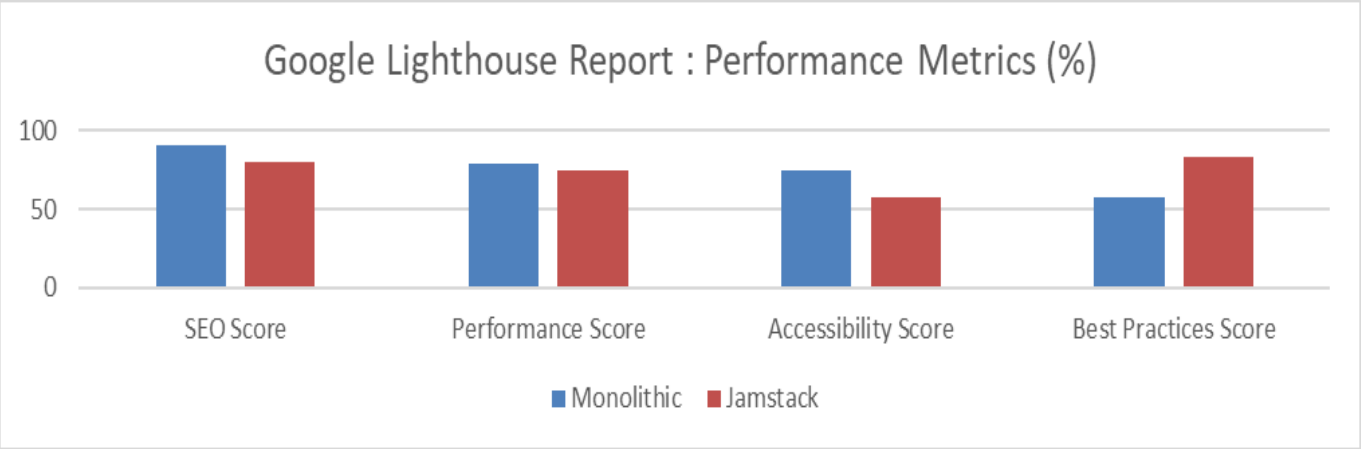

Fig 11: Google Lighthouse Performance Report

4.4 The Impact of Amount of Data Fetched from Backend in Monolithic vs JAMstack Architectures

4.4.1 Impact of Increasing Number of Cities on Backend Data Fetching Performance

The performance of the monolithic architecture remains consistently stable, showing minimal change with metrics such as First Contentful Paint (FCP), Time to Interactive (TTI), Speed Index (SI), and Largest Contentful Paint (LCP) as the number of cities is decreased. The FCP and LCP consistently measure around 9 seconds, while the TTI and SI stay steady at approximately 13 to 14 seconds. It is thus indicated that the performance of the monolithic architecture is influenced more by the fundamental server-side processing overhead than by the size of the dataset. The primary bottleneck is the time required to process and send the initial response on the server, irrespective of the amount of data displayed (see Figure 12, Table 4).

Whereas, the JAMstack architecture shows a high degree of responsiveness to changes in data volume. Initial load metrics such as FCP and TTI, are consistently fast and low (around 0.8 seconds). This result is because of serving pre-rendered static content from a CDN. The most significant change is seen in the Largest Contentful Paint (LCP) and Speed Index (SI) where the LCP drops from approximately 7.5 seconds (at 200 cities) to 6.5 seconds (at 40 cities) and the SI also shows improvement from around 13 seconds to 12.5 seconds. After the initial quick load, the overall performance of JAMstack site is primarily determined by the volume of data that needs to be fetched via APIs and rendered on the client side. If the number of cities is reduced means fewer API calls have to be made and thus less data to process, resulting in a faster LCP and more fast visual population of the page (see Figure 13, Table 5).

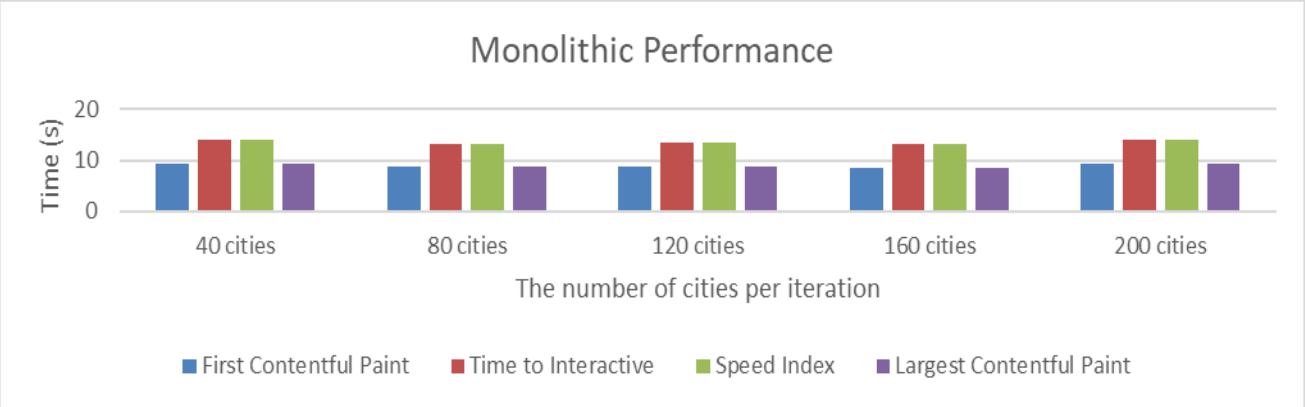

Fig 12: Monolithic Performance Comparison across number of cities

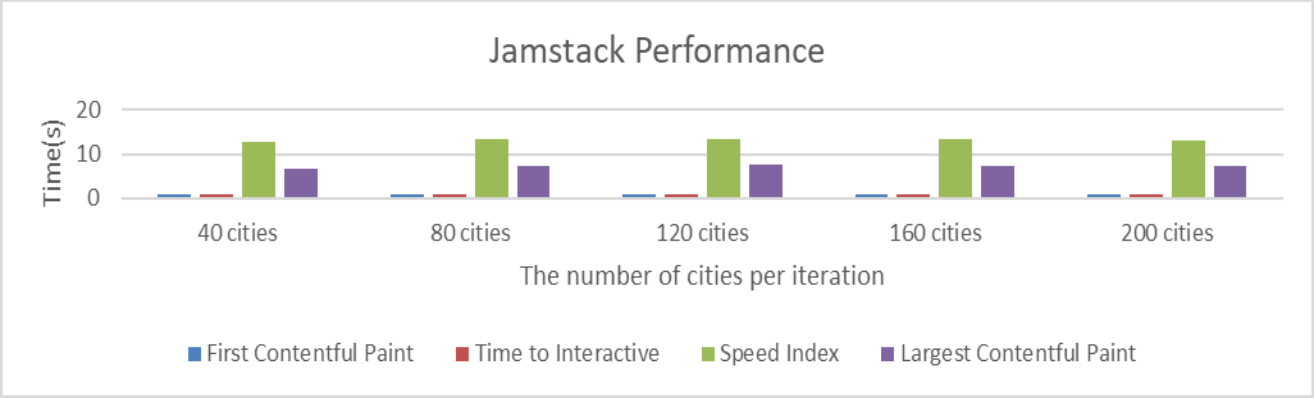

Fig 13: JAMstack Performance Comparison across number of cities

Table 4. Monolithic Cities Data

Cities	First Contentful Paint	Time to Interactive	Speed Index	Largest Contentful Paint
40 cities	9.2	14.1	14	9.2
80 cities	8.7	13.2	13.3	8.7
120 cities	8.7	13.5	13.4	8.7
160 cities	8.6	13.2	13.2	8.6
200 cities	9.3	13.9	13.9	9.3

Table 5. JAMstack Cities Data

Cities	First Contentful Paint	Time to Interactive	Speed Index	Largest Contentful Paint
40 cities	0.826	0.918	12.6	6.6
80 cities	0.906	0.994	13.3	7.3
120 cities	0.835	0.93	13.4	7.6
160 cities	0.857	0.916	13.4	7.3
200 cities	0.853	0.92	12.9	7.2

4.4.2 Impact of Increasing Hotel Data Volume on Backend Data Fetching Performance

As the number of hotels in 4 cities are increased the monolithic architecture suffers from a significant server-side performance bottleneck. The Initial Server Response Time degrades sharply from 4.727 s at 150 images to 5.949 s at 250 images as the number of hotel increases (see Figure 15). This shows that backend processing in the monolithic website does not scale efficiently with data volume. As a result, overall performance is directly impacted. Therefore, as the load increases the performance score drops from 88 to 72 (see Figure 14).

On the contrary, the Jamstack architecture provides extremely fast and stable server responses, even under higher hotel loads. The maximum Initial Server Response Time recorded in our analysis is just 0.289 s, which is almost 20 times faster than the monolithic process. This is mainly due to Jamstack's decoupled design and dependence on Content Delivery Network (CDN) to serve pre-rendered assets. Hence, the user-perceived loading remains smooth, while the First Contentful Paint (FCP) is as low as 0.9 seconds even with a growing number of hotels (see Figure 16).

However, Jamstack has its own limitations. While it eliminates server-side delays, performance challenges begin to appear on the client side as the number of hotels increases. The Fully Loaded Time is 18.8 seconds when the image count reaches 250, as opposed to 11.2 seconds for the monolithic website (see Figure 18). This means that the browser is the new bottleneck as it must download, decode and render a large volume of image assets. Therefore, Largest Contentful Paint (LCP) also increases from 7.3 seconds to 7.8 seconds (see Figure 17).

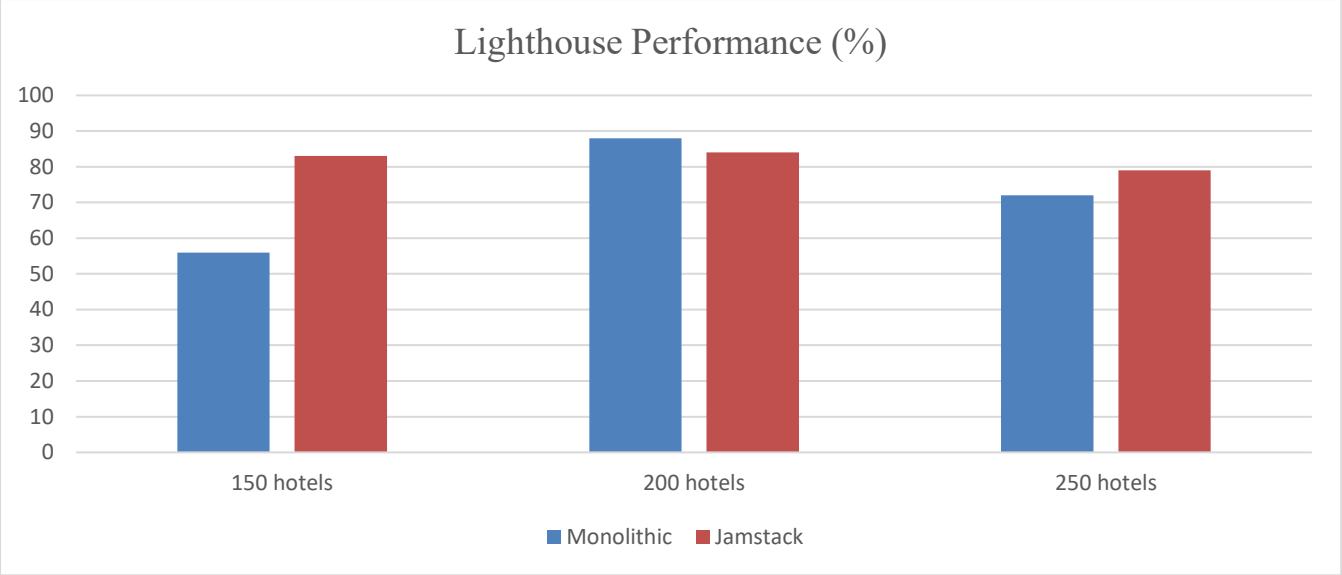

Fig 14: Lighthouse Performance Comparison Data for Series of Hotels

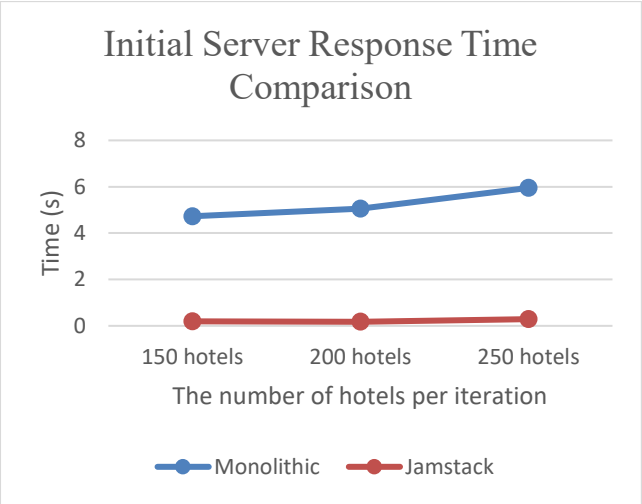

Fig 15: Initial Server Response Time Comparison

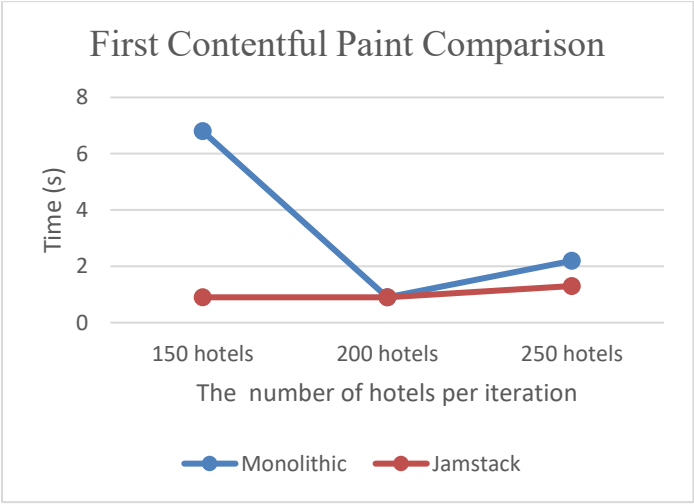

Fig 16: First Contentful Paint Comparison

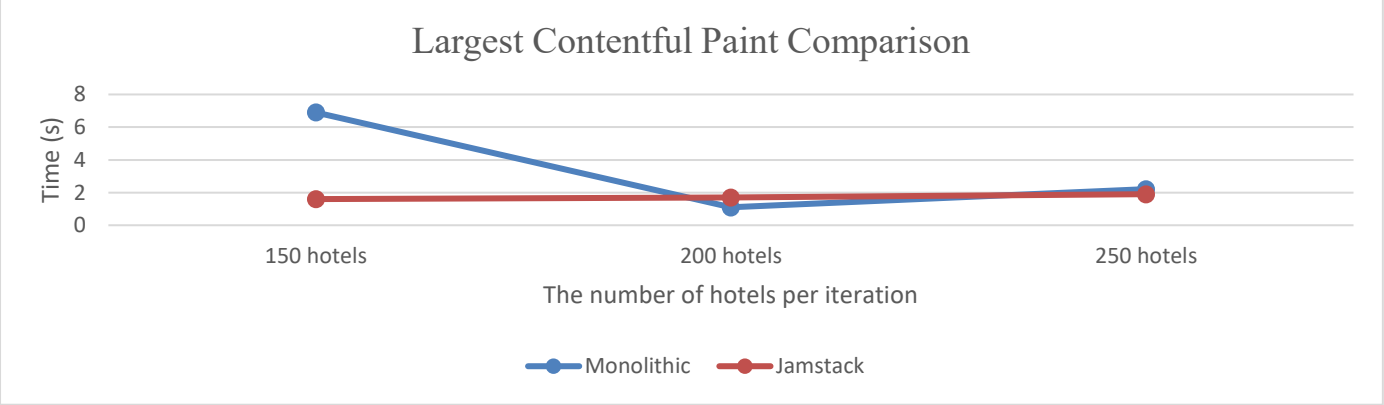

Fig 17: Largest Contentful Paint Comparison

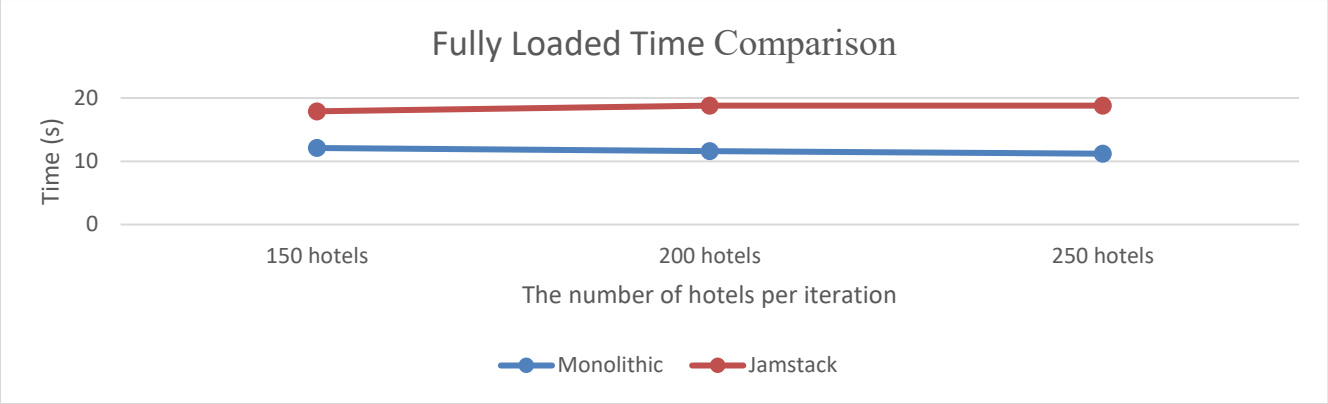

Fig 18: First Contentful Paint Comparison

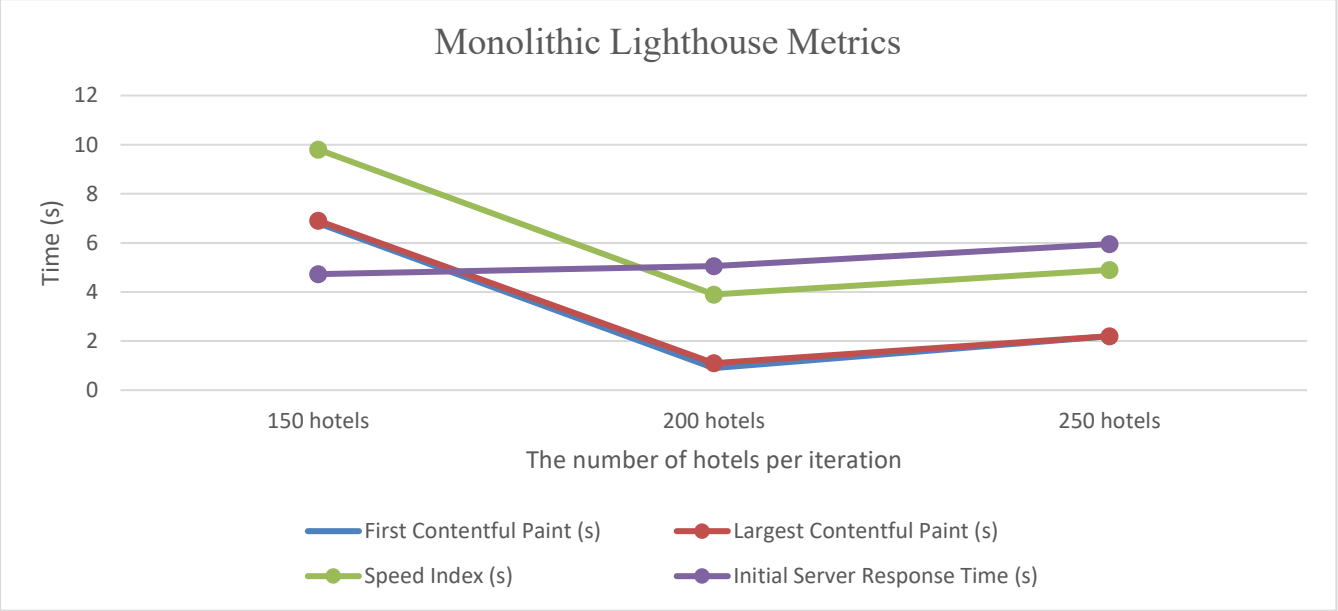

Fig 19: Monolithic Lighthouse Metrics Data for Series of Images

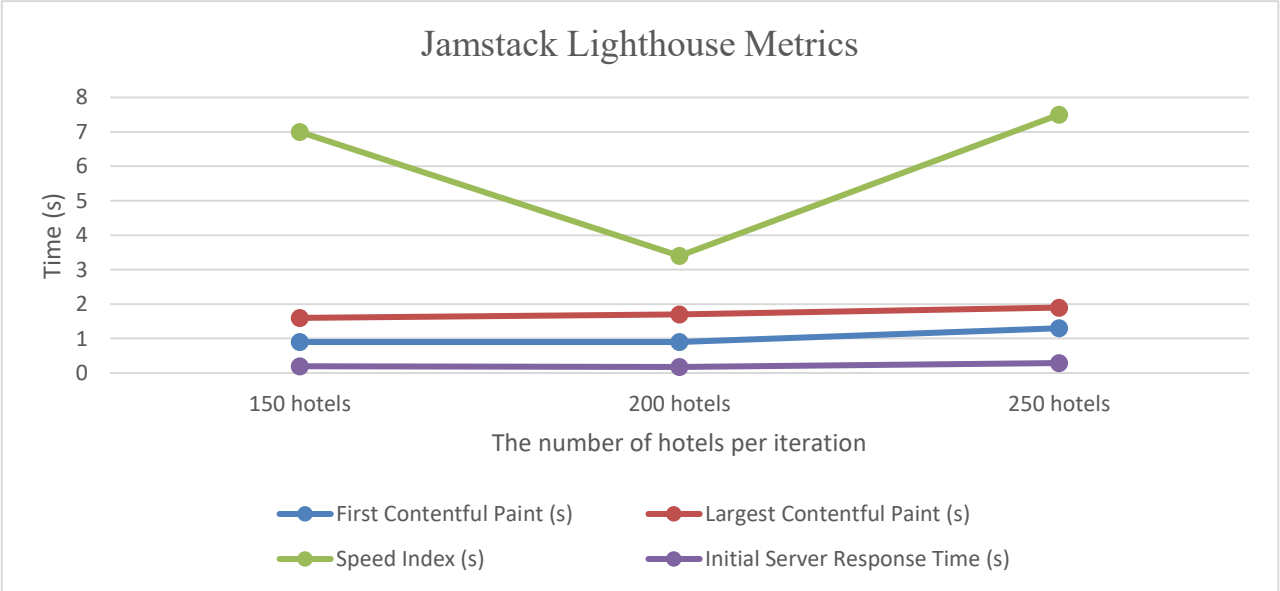

Fig 20: JAMstack Lighthouse Metrics Data for Series of Hotels

4.6 Key Findings

• Given its use of a CDN to serve pre-rendered, static content, JAMstack has a clear advantage in initial performance, with faster First Contentful Paint (FCP) and Time to Interactive (TTI).
• The monolithic site which is slow initially is capable of slightly quicker Fully Loaded Time, which also shows that it may use some server-side optimizations to be more efficient at loading the page.
• The two architectures differ in how their data volumes affect performance. Since, monolithic site is still bottlenecked by server-side processing its browser performance remains the same regardless of the number of cities.
• Whereas, the Jamstack site showed a noticeable improvement with reduced data as less data needs to be fetched and rendered on the client browser.
• In key Google Lighthouse metrics such as SEO, Performance, and Accessibility the monolithic site scored higher than Jamstack site indicating that each architecture has its own advantages and limitations.

5. CONCLUSION

The web architecture performance analysis domain has to be revisited as the shift between traditional monolithic and modern JAMstack models becomes increasingly critical in the current era. The performance metrics enhance user experience as they play a vital role in determining the quality of a web application. The two websites when analyzed revealed that both architectures demonstrated significant strengths. However, their core performance characteristics remain fundamentally different. It has been observed that JAMstack provided fast and responsive initial experience by making use of CDN for pre-rendered content. They are ideal for high-performance, scalable, API-driven applications such as marketing sites, documentation portals, headless e-commerce, and multi-client platforms where reusable backend services are required. They eliminated server-side processing bottlenecks, but its performance was influenced by the volume of dynamic data rendered on the client side. Although, JAMstack apps rely heavily on pre-rendered static content and APIs, which is indeed great for many cases but it is not always the most ideal case especially when projects are small and SEO-critical. On the contrary, the monolithic architecture, though limited by server-side processing and slower initial load, achieved a slightly faster fully loaded time and higher scores in Lighthouse categories such as SEO and accessibility. Monolithic apps typically use server-side rendering, which improves SEO automatically but do not benefit from a separate backend layer. It works best for small-scale, SEO-focused applications such as blogs, local business sites, admin dashboards, or internal tools where tight coupling and simplicity are preferred. The data also revealed that memory usage varied depending on browser engines. The monolithic site consumed more memory on Chrome and Edge, whereas the JAMstack site consumed more on Firefox. It can be inferred that the choice of architecture must be aligned with project specific priorities, as both architectures present unique trade-offs in terms of speed, scalability, and resource utilization.

6. REFERENCES

[1] Whitley, S. (2023). A quantitative study on the performance and scalability of Jamstack in comparison to a monolithic web architecture [Bachelor's thesis]. In Häme University of Applied Sciences (HAMK), Bachelor's Programme in Information and Communication Technology (pp. 52–53). https://www.theseus.fi/bitstream/handle/10024/802108/Whitley_Sam.pdf?sequence=2&isAllowed=y.

[2] Markovic, D., Scekic, M., Bucaioni, A., & Cicchetti, A. (2022). Could jamstack be the future of web applications architecture? Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing. https://doi.org/10.1145/3477314.3506991

[3] Nguyen, D. P. (2021). How JavaScript ecosystem and open-source tooling enable a modern era of Single-Page Applications. Theseus. https://www.theseus.fi/handle/10024/495352

[4] Orosz, E. (2020). Modern Web Development with JAMsatck. Theseus. https://www.theseus.fi/handle/10024/341469.

[5] Hoang, T. (2020). JAMStack Continuous Integration and Continuous Deployment with CircleCI and Netlify. In Metropolia University of Applied Sciences, Bachelor of Engineering (p. 31) [Thesis]. https://www.theseus.fi/bitstream/handle/10024/342452/Hoang_Tri.pdf?sequence=2

[6] Nguyen, T. (2022). JAMSTACK: A MODERN SOLUTION FOR E-COMMERCE [Thesis]. VAASAN AMMATTIKORKEAKOULU UNIVERSITY OF APPLIED SCIENCES. https://www.theseus.fi/bitstream/handle/10024/751804/Thang%20Nguyen%20e1800955.pdf?sequence=2

[7] A multitenant Single-Page application for programming education. (2023, November 27). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/10409809

[8] Gavrilă, V., Băjenaru, L., & Dobre, C. (2019). Modern Single Page Application Architecture: A case study. Studies in Informatics and Control, 28(2). https://doi.org/10.24846/v28i2y201911

[9] Enhancing SEO in Single-Page web applications in contrast with Multi-Page applications. (2024). IEEE Journals & Magazine | IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/10403891

[10] International Journal of Recent Research Aspects ISSN 2349-7688. (2020). An architectural style for single page scalable modern web application. www.academia.edu. https://www.academia.edu/41538297/An_Architectural_Style_for_Single_Page_Scalable_Modern_Web_Application