Comparative Analysis of RESTful, GraphQL, and gRPC APIs: Perfomance Insight from Load and Stress Testing

Steven Chandra[1]*, Ahmad Farisi[2]
Faculty of Computer Science and Engineering [1], [2]
Universitas Multi Data Palembang
Palembang, Indonesia
stevenchandrafei@mhs.mdp.ac.id[1], ahmadfarisi@mdp.ac.id [2]

Backend constitutes a critical component of digital infrastructure, responsible for processing business logic, managing data, and facilitating communication between software systems. APIs serve as the interface that enables software interaction and plays a pivotal role in backend operations. This study investigates the performance of three API architectures: RESTful, GraphQL, and gRPC. The experimental approach involves the implementation of Load Testing and Stress Testing to assess the performance of these architectures. The experiment utilizes a dedicated server and client hardware to simulate real-world conditions, with parameters such as CPU usage, memory usage, response time, load time, latency, success rate, and failure rate evaluated using a dataset comprising 1,000 rows of student-related records. Result show that RESTful achieves the highest total request but exhibit greater resource consumption and a higher failure rate. GraphQL demonstrated better CPU and memory efficiency with strong stability, though it has higher latency and slower response times. gRPC strikes a balance with a moderate latency and resource usage, albeit with slightly higher memory consumption under stress. By presenting a comprehensive analysis of each API architecture, this study contributes a comprehensive performance analysis under practical testing scenarios giving developers and system architect with data-driven guidance for selecting API architecture to their application needs. RESTful is well suited for high-throughput scenarios with less critical operations, GraphQL excels in resource efficiency and stability, and gRPC offers balanced performance across diverse workloads.

Keywords— API Architecture, gRPC, GraphQL, Restful, Load Testing, Stress Testing

I. INTRODUCTION

One of the most critical elements in digital infrastructure is the backend. The backend is a system component that works behind the scenes, focusing on business logic processing, data management, and is responsible for managing servers and databases [1], [2]. An efficient and reliable backend is essential to support complex operations, such as data processing and real-time updates, especially for applications that handle large volumes of information or need to respond directly to user interactions [3]. In its operations, the backend is supported by APIs that function as interfaces enabling two software components to communicate with each other.

Application Programming Interface (API) is an interface comprising a set of instructions organized in a library [4]. According to [5], an API is code that connects one application to another, providing all the necessary permissions for two software programs to communicate. APIs allow various systems, whether desktop or web, to interact and exchange data with servers or databases without requiring an additional backend, thus simplifying application integration and development. Additionally, API architecture is a critical aspect in determining how an API is organized and implemented.

In recent years, several API architectures have been developed to meet the needs of applications and developers in various scenarios, such as REST, GraphQL, and gRPC. REST (Representational State Transfer) is an architectural style for distributed systems that separates the interface on the client side and business logic on the server side to achieve anarchic scalability in line with internet growth [6]. GraphQL, developed by Facebook in 2012 and publicly released in 2015, is a dynamic single-endpoint query language for interacting with APIs [7]. gRPC is an open-source Remote Procedural Call (RPC) framework developed by Google [8].

With the growing demand for applications to handle user requests and large data volumes quickly and responsively, it is crucial for developers to choose the right API architecture according to application requirements. Each API architecture has its functions and use cases that influence application performance. Several previous studies have compared the performance and effectiveness of API architectures in various scenarios, such as the study Evaluation of Microservices Communication while Decomposing Monoliths [9], which focused on evaluating microservices technologies like HTTP REST, RabbitMQ, Kafka, gRPC, and GraphQL. This study used criteria such as latency, throughput, message size, and memory consumption to test the performance of these technologies. The results showed that each technology has its strengths and weaknesses, such as HTTP REST being simpler and more efficient for direct communication needs, while RabbitMQ and Kafka, which use message brokers for asynchronous communication, are better suited for architectures requiring high availability and loose coupling between services. However, this study had limitations in terms of parameter scope, device types, testing tools, programming

p-ISSN 2301-7988, e-ISSN 2581-0588
DOI : 10.32736/sisfokom.v14i1.2375, Copyright ©2025
Submitted : January 11, 2025, Revised : January 23, 2025, Accepted : January 26, 2025, Published : January 31, 2025

languages, and the complexity of the data used.

Another relevant study, Implementation Comparison of GraphQL and REST API Methods on Node.js Technology by [10], compared two commonly used API architectures in application development: REST API and GraphQL on Node.js. GraphQL was found to be more flexible and efficient than REST API as it allows clients to customize the data they need to display. This study compared the performance of the two API architectures using parameters such as response time and scalability. However, the testing in this study had some shortcomings, including limited testing parameters, the use of simulated data that did not represent real-world complexities, and a lack of repeated testing to ensure consistency of results.

Another study, Analysis of the Effectiveness Comparison Between RESTful and gRPC Architectures in Web Service Implementation [6], used parameters such as response time, response size, CPU usage, throughput, and load time. The study indicated that gRPC has more stable and faster response times as data volume increases compared to RESTful. It also stated that gRPC has lower CPU usage and response sizes compared to RESTful.

Previous studies often focus on limited research parameters, use of simulated data does not represent real-world complexity, and unequal comparison such as [6], [9], [10] [11], and [12] making it difficult to draw definite conclusions for real-world applications. To addresses these gaps, this study aims to comprehensively comparing RESTful, GraphQL, and gRPC in Load Testing and Stress Testing scenario by using seven parameters such as CPU usage, response time, latency, memory usage, loading time, success rate, and failure rate. These seven parameters overlap with those from previous studies which typically examines only two or three parameters, thus enabling a more thorough and realistic evaluation of API architectures.

II. RESEARCH METHODS

This study uses quantitative True-Experimental to compare the performance between three API architectures, namely RESTful, GraphQL, and gRPC. This method was chosen because it allows full control with research variables and ensures high internal validity in performance testing. This study aims to analyze the performance of RESTful, GraphQL, and gRPC API architectures.

A. Research Flow

The flow of the research experiment can be seen in Figure 1 below.

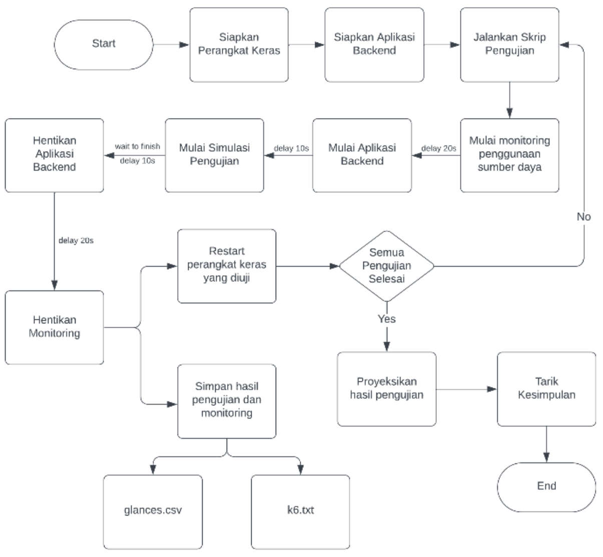

Fig. 1. Research Flow

The experiment begins with preparing the necessary hardware. Afterward, the backend application is configured with required settings. The test script is then executed to start the simulation and data collection process. First, Glances which is be used for monitoring resource usage is started. After a 20-second dela, the backend application is launched, followed by another 10-second delay before starting the K6 simulation script. The simulations are carried out until completion, with a duration of 6 minutes for Load Testing and 23 minutes for Stress Testing. Once the simulation is finished there is a 10-second delay before shutting down the backend application, followed by a 20-second delay to stop the resource monitoring. The results of the test and monitoring are stored in files glances.csv for resource monitoring logs and k6.txt for load test result logs generated by K6. This data is then analyzed to project test results and draw conclusions regarding system performance. The experimentation process is considered complete once all tests are finished and conclusions are drawn based on the collected data.

B. Research Scope

This research uses Load Testing and Stress Testing to compare the performance between three API architectures namely RESTful, GraphQL, and gRPC. This method was chosen because it allows full control with the research variables and ensures high internal validity in performance testing such as CPU usage, response time, latency, memory usage, load time, success rate, and failure rate. data used for testing consists of student records joined with tables for students, accounts, institutions, and study programs, totaling 1,000 rows of data. This choice of 1,000 rows of data balances complexity and resource constraints, providing a manageable sample that reveals key performance behaviors without overwhelming the test environment. It also provided a solid baseline for assessing how each API architecture handle typical loads before scaling up to a larger dataset if needed. By maintaining a controlled dataset size this research can focus on the architectural differences on performance, ensuring that trends remain attributable to the API design rather than external factors introduced by excessive data volumes. The evaluation are conducted by using two dedicated hardware to act as server and client for testing to simulate a real world scenario, the hardware specification can be seen in Table I.

TABLE I. SPESIFICATION TABLE

Spesification	Hardware	
	Server	Client
CPU	Intel Core i7-7700HQ	AMD Ryzen 5 7535HS
RAM	20 GB DDR4	16 GB DDR5
Operating System	Ubuntu Server 24.04.1 LTS	Windows 11 23H2

III. RESULT AND DISCUSSION

The following experimental results that have been carried out can be seen in the following 3.1 and 3.2.

3. 1. Load Test

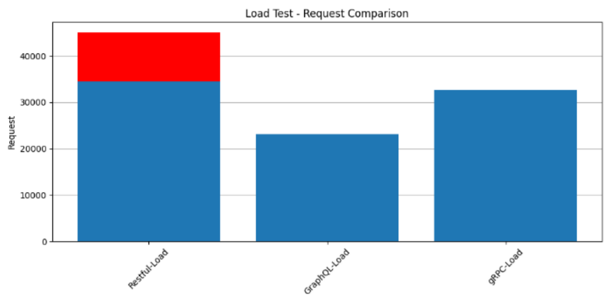

Fig. 2. Load Testing Results – Total Comparison

In Figure 2, the results of Load Testing total requests show that Restful architecture has the largest total requests compared to GraphQL and gRPC during the same testing period. However, Restful has a higher request failure rate than the other two architectures. GraphQL processes fewer requests than Restful, while gRPC performance is close to Restful with no failed requests like Restful.

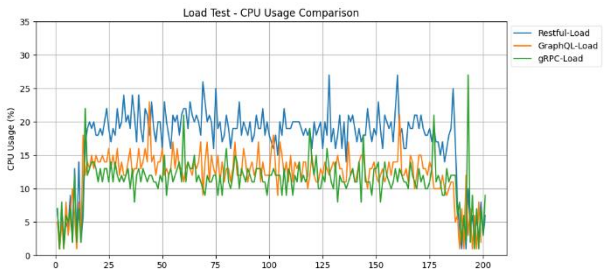

Fig. 3. Load Testing Results – CPU Usage

In Figure 3, the results of Load Testing CPU usage show that Restful uses higher CPU resources than other architectures. GraphQL shows a lower and stable CPU usage with a range of 15%. While gRPC has lower CPU usage than GraphQL and Restful it has some spikes in CPU usage especially in the few seconds before the test ends, this may possibly due to protocol buffer encoding/decoding and connection handling toward the end of the test.

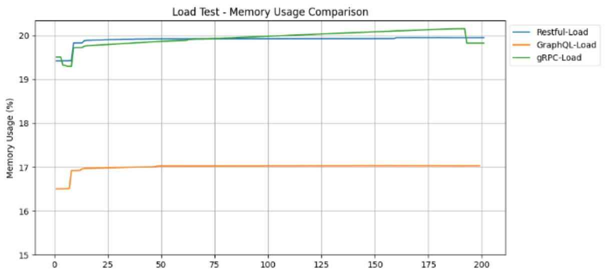

Fig. 4. Load Testing Results – Memory Usage

In Figure 4, the simulated Load Testing results of memory usage show GraphQL has lower memory usage at 17%. While Restful and gRPC have identical memory usage at around 20%. This shows GraphQL is more efficient in memory usage than Restful and gRPC.

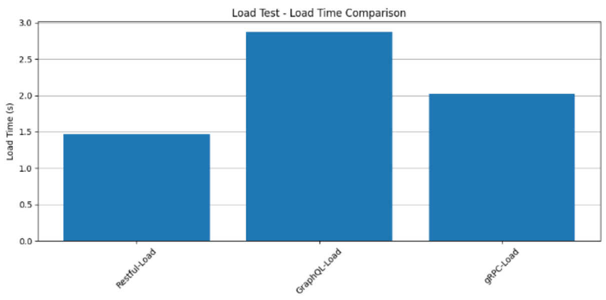

Fig. 5. Load Testing Results – Load Time Comparison

In Figure 5, shows a comparison of the average load time of the API architectures during Load Testing. GraphQL has the highest average load time at around 2.8 seconds, followed by gRPC with an average of around 2 seconds. While Restful has the lowest average load time, which is 1.4 seconds. These results show that Restful tends to have a faster load time response than GraphQL and gRPC.

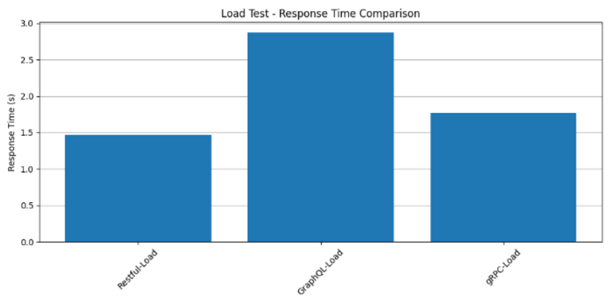

Fig. 6. Load Testing Results – Response Time Comparison

In Figure 6, the results of the average response time in Load Testing show a similar pattern to the average load time. GraphQL stands out with an average response time of 2.8 seconds, while gRPC has an average response time of 1.7 seconds. Restful shows the lowest response time of under 1.6 seconds compared to the other two architectures.

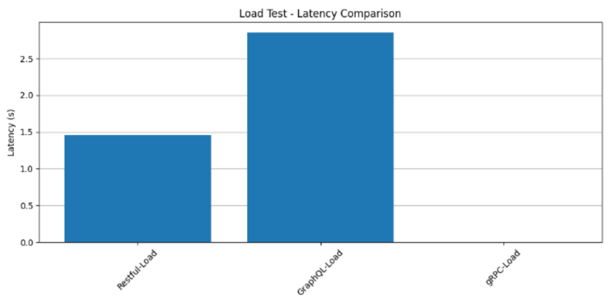

Fig. 7. Load Testing Results – Latency Comparison

In Figure 7, shows the average latency results on Load Testing. The graph shows that GraphQL shows the highest latency by reaching 2.85 seconds followed by Restful with a latency of about 1.5 seconds. Meanwhile, gRPC latency is written as 0 because it cannot be measured using the k6 testing tool.

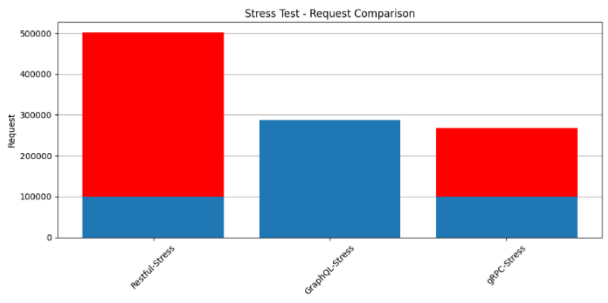

Fig. 8. Load Testing Results – Total Comparison

In Figure 8, the total request Stress Testing results show Restful architecture has higher performance with 500,000 total requests, but has a low success rate at 20%. gRPC also follows a similar pattern with 267,000 total requests with 37% success rate. In contrast, GraphQL showed higher total requests than gRPC, having a 100% success rate out of 287,000 requests.

To facilitate easier comparison, Table II summarize key perfomance metrics for RESTful, GraphQL, and gRPC under Load Testing.

TABLE II. LOAD TEST PERFOMANCE SUMMARY

Metric	RESTful	GraphQL	gRPC
Request Processed	44,976	23,055	32,668
Succes Rate	76.64%	100%	100%
Failure Rate	23.36%	0%	0%
Avg CPU Usage	Highest	Moderate	Lowest Average, occasional spike
Memory Usage	~20%	17%	~20%
Load Time	1,47ss	2,87s	2,02s
Response Time	1,47s	2,87s	1,77s
Latency	1,46s	2,85s	Not measured by K6

3. 2. Stress Test

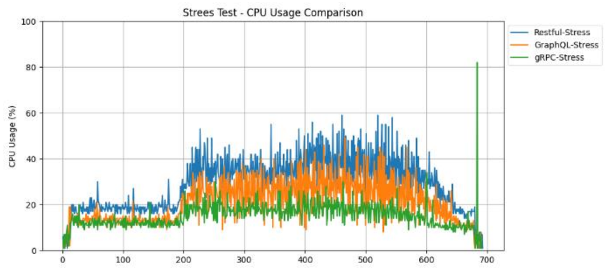

Fig. 9. Stress Testing Results – CPU Usage

In Figure 9, Stress Testing CPU usage also shows a similar trend to Load Testing. Restful tends to use higher CPU than GraphQL and gRPC. While GraphQL has relatively lower CPU usage than Restful. gRPC shows lower CPU usage than the other two architectures, with some spikes in CPU usage at some periods this is especially noticeable in the moments before the test is completed.

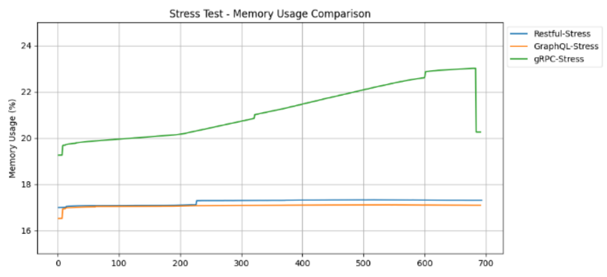

Fig. 10. Stress Testing Results – Memory Usage

In Figure 10, shows the results of Stress Testing memory usage shows a different pattern compared to the previous Load Testing. gRPC shows higher memory usage than Restful and GraphQL memory usage which tends to be stable and almost the same. This indicates that gRPC utilizes more memory than Restful and GraphQL in handling stress testing conditions.

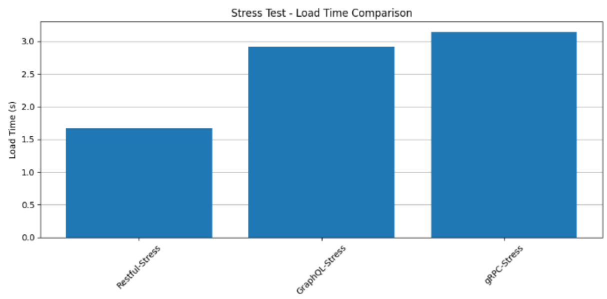

Fig. 11. Stress Testing Results – Load Time Comparison

In Figure 11, shows the average load time results for Restful, GraphQL, and gRPC. Restful has the lowest average load time at around 1.5 seconds. While GraphQL and gRPC show high average load times around 3 seconds, with gRPC having the highest load time of 3.14 seconds.

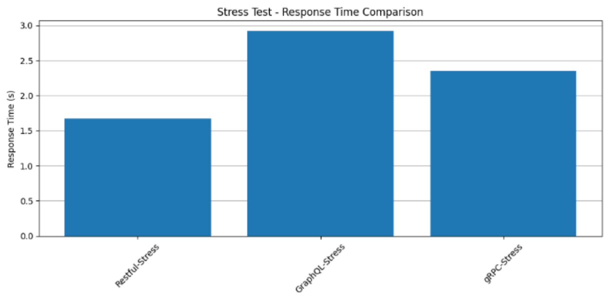

Fig. 12. Stress Testing Results – Response Time Comparison

In Figure 12, shows the results of the average response time during stress testing. Restful has the lowest response time at 1.6 seconds, while GraphQL has the highest response time at 2.9 seconds. gRPC has an intermediate response time of 2.3 seconds.

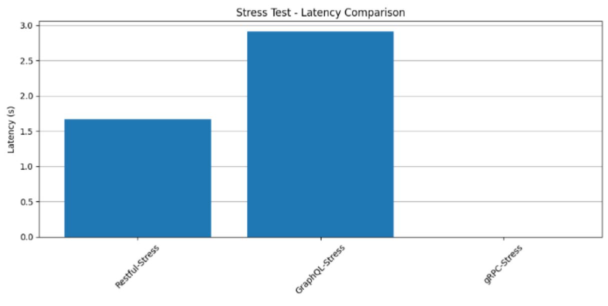

Fig. 13. Stress Testing Results – Latency Comparison

In Figure 13, shows the average latency results of Restful, GraphQL, and gRPC during Stress Testing. The test results are consistent with the Load Testing test, where GraphQL shows the highest average latency at 2.9 seconds while Restful shows an average latency of 1.6 seconds. Meanwhile, gRPC latency could not be measured by the k6 testing tool.

To facilitate easier comparison, Table III summarize key perfomance metrics for RESTful, GraphQL, and gRPC under Stress Testing.

TABLE III. STRESS TEST PERFOMANCE SUMMARY

Metric	**RESTful**	**GraphQL**	**gRPC**
Request Processed	501,460	287,603	267,594
Succes Rate	80.21%	100%	37.27%
Failure Rate	19.79%	0%	62.73%
Avg CPU Usage	Highest	Moderate	Lowest Average, occasional spike
Memory Usage	~20%	17%	~20%
Load Time	1,67ss	2,92s	3,14s
Response Time	1,67s	2,92s	2,35s
Latency	1,67s	2,91s	Not measured by K6

IV. CONCLUSION

The research result of both types of tests shows consistent results, Restful architecture shows the ability to execute more requests in the same period of time, but the success rate is lower than GraphQL and gRPC. GraphQL shows better CPU and memory utilization efficiency and is stable in both tests, while gRPC offers a balance between performance and resources, although under sustained stress conditions gRPC memory utilization is higher than other architectures. It is important to note however that these experiments arise from limited a relatively limited dataset (1,000 rows) and a specific hardware configuration, which could affect generalizability to real-world applications with larger databases or different infrastructure. Future research and experiment therefore could investigate this API architecture with more substantial datasets, potentially in tens of thousands of records, and explore variations in programming language and API architectures. Additionally future research could integrate more comprehensive gRPC monitoring tools, detailed time-series monitoring, and granular profiling to pinpoint the exact cause of gRPC resource spike.

REFERENCES

[1] D. A. Hutomo Putra, E. Darwiyanto, and R. Nurtantyana, “Development of Backend Admin Dashboard for Business Project Monitoring using Scrum Method,” *Indonesia Journal On Computing (Indo-JC)*, vol. 9, no. 2, pp. 118–133, 2024, doi: 10.34818/indojc.2024.9.2.969.

[2] Humdiana and Julieca, “Implementation of Full Stack Web Development for Data Admin on Social Media Buzzbuddies,” *International Journal of Social Science (IJSS)*, vol. 2, no. 2, pp. 1535–1544, Aug. 2022, doi: 10.53625/ijss.v2i2.3093.

[3] M. J. Jamshed o’g’li, “The Significance of Backend Development in Modern Web Applications,” *International Journal of Scientific Researchers*, vol. 8, no. 1, pp. 737–741, 2024.

[4] N. K. Dwi Sabrina, D. Pramana, and T. M. Kusuma, “Implementation of Golang and ReactJS in the COVID-19 Vaccination Reservation System,” *ADI Journal on Recent Innovation (AJRI)*, vol. 5, no. 1, pp. 1–12, Feb. 2023, doi: 10.34306/ajri.v5i1.877.

[5] A. F. Rochim, T. N. Wijaya, and D. Eridani, “A Citation Data Collector Tool of Author’s Profiles in Scopus Based on Web and Application Programming Interface (API),” *IOP Conf Ser Mater Sci Eng*, vol. 1077, no. 1, p. 012017, Feb. 2021, doi: 10.1088/1757-899x/1077/1/012017.

[6] M. I. Yanuardi, Aminudin, and M. Faiqurahman, “Analisis Perbandingan Efektivitas Arsitektur RESTful dan Arsitektur gRPC pada Implementasi Web Service,” *Jurnal Edukasi dan Penelitian Informatika (JEPIN)*, vol. 10, no. 2, pp. 333–341, Aug. 2024.

[7] S. Guha and S. Majumder, “A Comparative Study between Graph-QL & Restful Services in API Management of Stateless Architectures,” *International Journal on Web Service Computing*, vol. 11, no. 2, pp. 1–16, Jun. 2020, doi: 10.5121/ijwsc.2020.11201.

[8] N. Jagnik, “Highly Performant Python Services using gRPC and AsyncIO,” *Journal of Scientific and Engineering Research*, vol. 8, no. 1, pp. 225–229, 2021, [Online]. Available: www.jsaer.com

[9] J. Kazanavičius and D. Mažeika, “Evaluation Of Microservice Communication While Decomposing Monoliths,” *Computing and Informatics*, vol. 42, pp. 1–36, 2023, doi: 10.31577/cai.

[10] Muntahanah, Y. Darmi, and K. Pinandita, “Implementasi Perbandingan Metode GraphQL dan REST API pada Teknologi Nodejs,” *Journal of Information Technology and Computer Science (INTECOMS)*, vol. 7, no. 1, pp. 25–34, 2024.

[11] M. N. Hedelin, “Benchmarking and Performance Analysis of Communication Protocols: A Comparative Case Study of gRPC, REST, and SOAP,” KTH Royal Institute of Technology, 2024. Accessed: Nov. 03, 2024. [Online]. Available: https://kth.diva-portal.org/smash/record.jsf?pid=diva2%3A1887929&dswid=-8588

[12] Goriparthi, “Streamlining API Development: A Comparative Analysis of GraphQL and RESTful Web Services,” *International Journal of Data Analytics Research and Development*, vol. 2, no. 1, pp. 59–71, 2024, [Online]. Available: https://iaeme.com/Home/issue/IJDARD?Volume=2&Issue=1