JCSI 38 (2026) 26-31
Received: 16 September 2025
Accepted: 12 November 2025

A performance comparison of web programming interfaces: GraphQL, gRPC and Thrift

Piotr Rożek*, Mariusz Dzieńkowski

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

This article presents a comparative performance comparison of three technologies commonly used to implement web programming interfaces: GraphQL, gRPC and Thrift. For the purposes of this study, three applications were developed, each utilizing one of the selected technologies. The applications were tested using JMeter. The execution time and data volume returned during operations performed on a relational database were measured. The results showed that gRPC was the fastest choice for the shortest execution time for large-scale read operations, while GraphQL and Thrift were faster for operations transporting small volumes of data.

Keywords: communication protocols; API; performance testing; JMeter

*Corresponding author

Email address: s95548@pollub.edu.pl (P. Rożek)

1. Introduction

Application programming interfaces (APIs) are a key elements in the context of communication between applications. APIs define a set of rules and protocols thanks to which different software components can interact with each other [1]. This allows the exchange of data and the use of functions of other programs, without the need to know their implementation. Application programming interfaces are therefore a kind of middleman in the exchange of data, thanks to which it is possible to create programs based on the functionality or data provided by other software, that allows the creation of more multilevel applications using already built components. There are several ways to implement an application programming interface. The most popular is REST - Representational State Transfer, a software architecture style introduced by Roy Fielding in 2000, based on the HTTP protocol [2]. Another approach is Simple Object Access Protocol - SOAP, a somewhat outdated, 1998-era XML-based communications protocol used to exchange information between systems over computer networks, especially web services [3]. In 2007, an interface appeared that allowed defining APIs in a neutral interface language - Apache Thrift [4]. Then, in 2015, the gRPC programming interface was created - remote procedure call protocol framework created by Google based on the HTTP/2 communication protocol [5]. That same year, GraphQL also saw the light of day - an alternative to the popular REST, an API query language created for the Facebook platform, which, according to the programmers of the popular social networking site, ensures more effective cooperation between the client and the server [6].

1.1. Motivation

The comparative analysis of web programming interfaces such as GraphQL, gRPC and Thrift is justified by the diversity of needs of modern IT systems and the dynamic development of API technology. Each of these interfaces offers a different approach to handling communication and data exchange between applications, adapted to specific requirements, such as performance, flexibility or ease of integration. Choosing the right solution is crucial for optimizing the work of programming teams and the quality of created applications, especially in the face of the growing popularity of micro services and multiplatform applications. Comparison of gRPC, Thrift and GraphQL allows to understand their advantages and limitations, which helps you choose the best tool for a specific application.

1.2. The aim of the work and research hypotheses

The aim of this work is to compare and analyze API implementations – gRPC, GraphQL and Thrift in terms of performance. The analysis will concern the comparison of the performance of the above technologies in the context of the time needed to handle requests and size of transferred data. The scope of research conducted for the needs of the work includes the selection of a diagnostic tool, development of test applications, execution of tests, analysis of the obtained results and discussion.

The following hypothesis was put forward in the work: *gRPC is more efficient communication technology than GraphQL and Thrift, as provides the shortest requests handling time under comparable conditions*.

1.3. Related works

Before starting the research, an analysis of selected publications was made that discuss the topic of Application Programming Interfaces such as gRPC, GraphQL and Thrift. Due to the limited number of articles on Thrift technology, the focus was primarily on works that concern the performance and technical aspects of GraphQL and gRPC, as well as comparisons of different technologies like REST or SOAP.

Many articles focus on comparing the performance of REST and GraphQL, analyzing the differences in response time, size of transferred data, and number of requests served. Articles such as those by Margański and Pańczyk [7], Oggier [8], and Diyasa et al. [9] confirm that

REST is more efficient for retrieving large amounts of data or in traditional applications, but GraphQL prevails in scenarios requiring limited and precise access to data. The studies by Śliwa and Pańczyk [10] show that in terms of performance measured in transactions/sec, GraphQl obtains better results for large amounts of data, but with smaller data load REST is better. Goriparthi [11] on the other hand noted that GraphQL copes better with dynamically changing application requirements and has lower CPU usage, but when it comes to response time, REST comes out as faster option. Brito and Valente [6], on the other hand, conducted an experiment to evaluate and compare the effort required to implement queries in GraphQL versus REST in the context of service-based architectures. They conducted a study, that required 22 students with varying degrees of familiarity with data technologies, who were asked to implement eight queries to access a web service using GraphQL and REST. Based on the results of the study, they proved that GraphQL requires less effort to implement API queries.

Brito, Mombach, and Valente [12] conducted a practical evaluation of an API migration from REST to GraphQL, showing a significant reduction in the size of data transferred between server and client (by 94% in terms of fields and 99% in bytes). This migration brings benefits in terms of data optimization and performance. Ambasht [13] emphasizes that GraphQL revolutionizes API development, improving collaboration between teams and precision in data delivery. However, he notes challenges related to performance management and security.

Hartig and Pérez [14] attempted to formalize the GraphQL language by studying its semantics and computational complexity. The results indicate that GraphQL enables efficient query handling, and the size of results can be estimated in polynomial time, which allows for better control over potentially large responses in web scenarios.

Articles such as those by Niswar et al. [15] and Erlandsson and Remes compare REST, GraphQL, and other API technologies (SOAP, gRPC). They show that gRPC is the most efficient in microservice architectures due to its fast response time and efficient resource utilization. REST maintains its advantage in simpler systems, while GraphQL proves to be suitable in environments requiring flexibility. Aydemir and Basçiftçi [16] analyzed the performance and availability of three modern APIs (gRPC, Thrift, and REST). Test results showed that a prototype API gateway implementation based on gRPC was better than the other solutions in terms of performance. Kamiński et al. compared the most commonly used protocols and data transfer concepts: REST, WebSocket, gRPC GraphQL, and SOAP. Tests showed that gRPC is the fastest in data transfer between client and server [17].

Cederlund [18] and Vesić, Kojić [19] drew attention to the specific uses of APIs in client and web applications. Relay+GraphQL is better at handling parallel queries, but REST remains more efficient for simple operations. GraphQL excels in poor Internet connection conditions, reducing the number of HTTP requests due to its structure.

GraphQL has been presented as a technology with great potential for the future, especially in the articles by Ambasht [13] and Lawi et al. [20]. It enables flexible data management, reducing redundant queries and offering a better user experience. However, it requires a mature approach to design and education of development teams. Meanwhile, the article by Raghav [21] explores the reasons for the need to move from traditional REST APIs to gRPC in the global payments ecosystem. gRPC has emerged as a promising alternative to traditional REST APIs, offering several advantages in terms of performance, scalability, and strong typing.

2. Research methodology

2.1. Test applications

For the purpose of conducting tests on the performance of web-based programming interfaces, three test applications with the same functionalities were implemented using GraphQL, gRPC and Thrift technologies respectively. The created applications are used to provide and manage order information. They were created in Java, using the Spring Boot development framework, and connected to a MySQL relational database. The database schema is shown in Figure 1. The applications allow basic operations on the database, such as creating, reading, updating and deleting records.

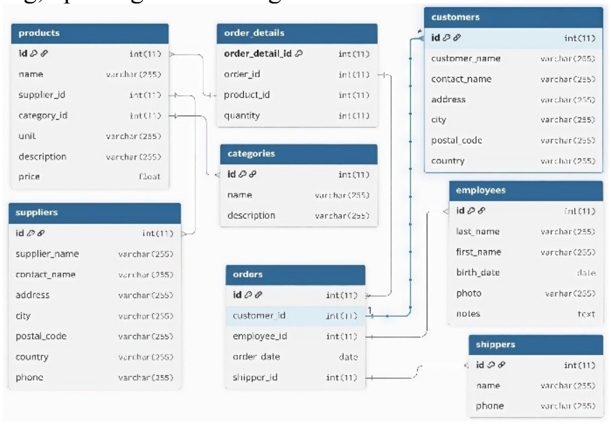

Figure 1: Database schema.

The order_details table combines quantity and IDs of both orders and products. The order table consists of the date when the order was placed along with IDs of customers, employees and shippers. Every order_detail and order records are related to records in the other tables in the database, so when retrieved, they join data to produce a complete view of who ordered what, who handled it, who shipped it, and which products were included in a single order.

2.2. Testing environment

The JMeter tool was used to conduct the tests. It is an open-source tool used to test the performance of web applications and other services [22]. It allows the simulation of user traffic and observation of application performance under heavy load. To conduct the tests, it was necessary to determine the number of threads and the test parameters, i.e. the duration and the number of requests sent per second. To eliminate possible factors that might affect the test results, such as network latency, the tests were conducted on a single machine. The parameters of the test environment, including computer specifications, are shown in Table 1.

Table 1: Computer specifications

Component	Specification
CPU	Intel(R) Core(TM) i5-6300HQ, 2.30 GHz
RAM	16 GB
OS	Windows 10 Home, 64-bit

2.3. Research scenarios

Four research scenarios were developed for the study and are presented in Table 2.

Table 2: Test scenarios

No	Scenario
1.	Measuring the time and size of the server's response when reading all the records from order_details table linked with other tables in the database (1000 records).
2.	Measuring the time and the size of the server's response when reading a single record based on id from order_details table linked with other tables in the database.
3.	Measuring the time and the size of the server's response when inserting a single order record into the database.
4.	Measuring the time and the size of the server's response when updating a single order record.

The time it required to process a request was measured in milliseconds, and the size of the downloaded data was measured in bytes.

Each group of virtual users was tested with different ramp up time period. For the smaller groups that consisted of 50 and 100 users, the ramp up time was 5 seconds, which was equivalent of 10 and 20 requests per second. For the group of 500 and 1000 users, the ramp up time was respectively 15 and 30 seconds, which resulted in 33.3 requests per second. Every user was in charge of sending single request. The number of users and ramp up times were the same for all performed tests. Each test was repeated 5 times, and gathered results were averaged. The 'Loop Count' parameter in the JMeter configuration was set to 1. Table 3 summarizes the configurations for the three tested technologies.

Table 3: Comparative summary of configuration

Tech.	Protocol	Sampler Type	Payload Format	Plugin
GraphQL	HTTP/1.1	HTTP Request	JSON	No
gRPC	HTTP/2	gRPC Request	JSON	Yes
Thrift	HTTP/1.1	HTTP Request	JSON (TJSON Protocol)	No

3. Research results

3.1. Reading multiple records

As a part of first test, a series of read requests was executed for the previously defined user groups.

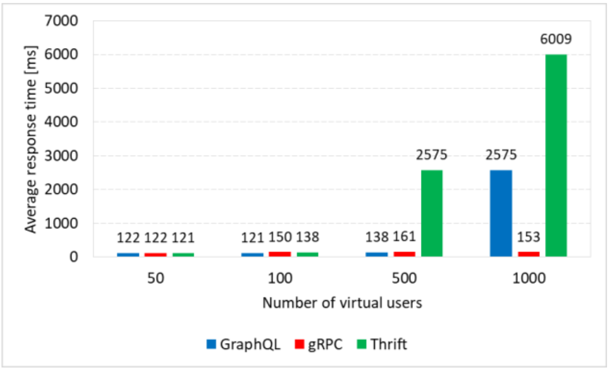

Figure 2: Average response time for reading all order detail records.

The query was designed to retrieve all data from a table containing 1000 records with order detail information in a single request. Figure 2 shows that for groups of 50 users, response times were almost identical across all technologies, differing by less than 1%. For the 100-user group, GraphQL maintained strong performance, responding 19% faster than gRPC and 12% faster than Thrift. At 500 users, the differences became even more pronounced: GraphQL responded 14% faster than gRPC and 94,6% faster than Thrift. However, for 1000-user group, gRPC achieved the best results, responding 94% faster than GraphQL and suprising 97% faster than Thrift.

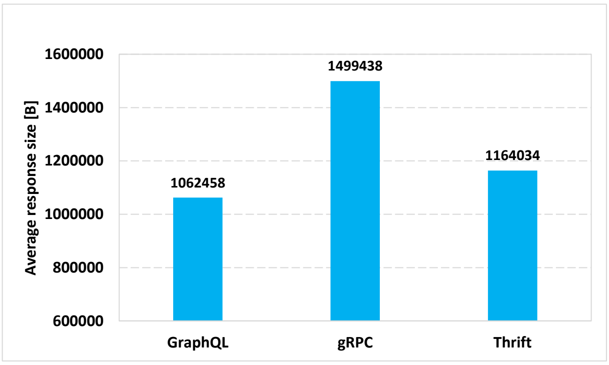

Figure 3: Response size for reading all order detail records.

3.2. Reading a single record

In the second test, each virtual user submitted a request and received a single record identified by the provided identifier as the result of its processing. The records were retrieved from a table containing specific order information. The results of this test are presented in Figure 4.

Across all groups defined by the number of virtual users, Thrift and GraphQL achieved the best performance, with Thrift being slightly more efficient. For small groups of 50 and 100 users, the average response times were nearly identical across all three technologies tested. However, as the number of virtual users increased, the average gRPC response time rose significantly. With 500 users, the response times for Thrift and GraphQL were more than four times faster than for gRPC. A similar trend was observed with 1,000 users, where GraphQL and Thrift were 3.6 times faster than gRPC.

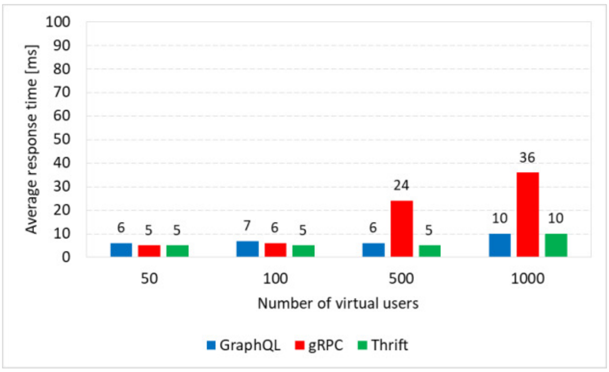

Figure 4: Average response time for reading a single order record.

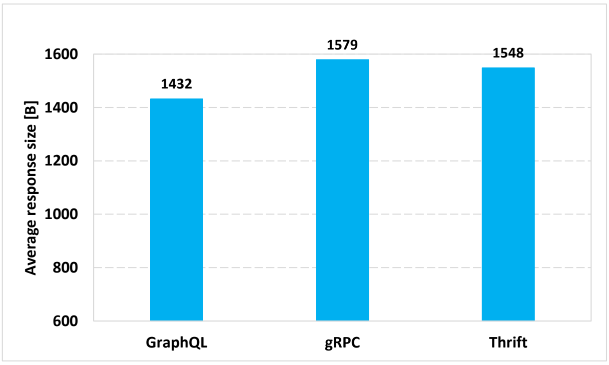

Figure 5: Average response size for retrieving a single order detail record.

Figure 5 shows the average response size for each test application executed using the compared technologies. Consistent with the previous study, responses returned by GraphQL had the smallest data volume, being 10% and 8% smaller than those returned by gRPC and Thrift, respectively.

3.3. Creating a new record

Test 3 involved creating a new database record containing order data. The test was conducted with the same virtual user groups as in the previous test. The results of this test are presented in Figure 6.

For requests to create a new database record, Thrift achieved the best performance in terms of request handling speed across the various virtual user group sizes. As the number of users (load) increased, its average request handling time remained nearly constant, ranging from 8 to 12 ms. In contrast, the response times for GraphQL and gRPC increased significantly under heavier loads. With 500 users, Thrift was 5.3 times faster than gRPC and 1.2 times faster than GraphQL. With 1,000 users, Thrift was 4.6 times faster than GraphQL and 3.2 times faster than gRPC.

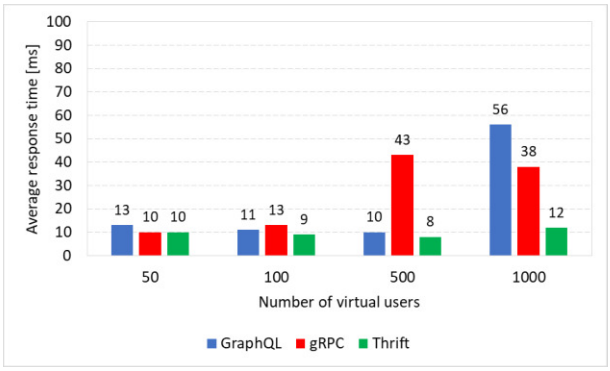

Figure 6: Average response time for creating a new order record.

Figure 7 presents the results for the amount of data transferred in response to the POST request. In terms of response size, gRPC performed best, producing the smallest data volume compared to GraphQL and Thrift. The response size for gRPC was about 9% smaller than that of GraphQL and almost 15% smaller than that of Thrift.

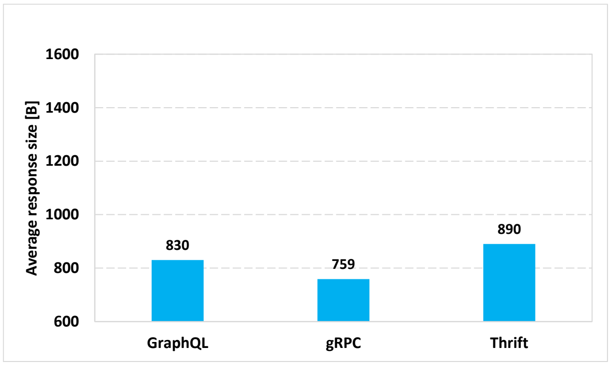

Figure 7: Response size for creating new order record.

3.4. Updating a single record

Test 4 involved updating a selected database record containing order data. The test was conducted with the same virtual user groups as in the previous experiments. The results are presented in Figure 8. In this study, GraphQL consistently outperformed the other two web APIs, achieving significantly lower average response times across all user workloads. With 50 virtual users, GraphQL handled requests 74% faster than gRPC and 80% faster than Thrift.

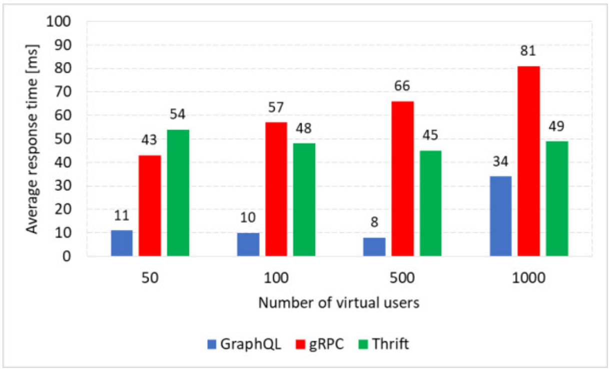

Figure 8: Average response time for updating an order record.

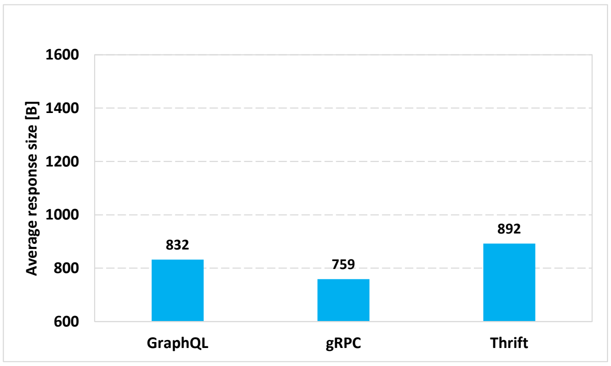

Figure 9: Average response size for updating an order record.

Even with 1,000 users, where request handling times increased, GraphQL maintained a performance advantage, responding 58% faster than gRPC and 31% faster than Thrift. gRPC demonstrated the lowest efficiency under both light and heavy workloads, while Thrift performed slightly better than gRPC across comparable levels regardless of the number of virtual users. GraphQL remained the most efficient web APIs overall, though it exhibited a greater performance degradation in the 1,000-user scenario.

Figure 9 shows the results regarding the average volume of requests returned by the server during the test. Since the add and update operation returns the same data, the response sizes are almost identical.

4. Conclusions

Comparative performance testing of GraphQL, gRPC, and Thrift across various database operations and user workloads revealed that no single technology consistently outperformed the others in all scenarios. Instead, each web API demonstrated strengths and weaknesses depending on the type of operation and workload.

For multiple-record reads, gRPC performed best under heavy load, significantly outperforming both GraphQL and Thrift. For single-record reads, Thrift and GraphQL performed best, maintaining low response times regardless of load, while gRPC showed poor scalability with large virtual user groups.

When creating a new record, Thrift demonstrated the best performance regardless of the number of virtual users, making it the most reliable choice for write-intensive scenarios. Interestingly, gRPC produced slightly smaller response sizes in this case, even though its response times deteriorated with increasing load. For record updates, GraphQL achieved the best response times, especially with smaller virtual user groups. However, gRPC performed worst for record modification operations under both light and heavy load. Comparative performance testing of GraphQL, gRPC, and Thrift across various database operations and user workloads revealed that no single technology consistently outperformed the others in all scenarios. Instead, each web APIs demonstrated strengths and weaknesses depending on the type of operation and workload.

In terms of response size, GraphQL consistently generated the smallest payloads when reading data, which can reduce bandwidth consumption. gRPC, although often slower in higher load scenarios, excelled in minimizing response size when creating and updating records.

Overall, the results show that:

- Thrift is most efficient for create operations, with stable performance under load.
- GraphQL is advantageous for read operations on small to medium scales and for updates, thanks to its low response times and compact payloads.
- gRPC scales better for bulk read operations but struggles with single record queries and high-concurrency writes, despite its efficiency in minimizing payload size for certain operations.

Thus, the choice of interface depends strongly on the expected workload characteristics: Thrift for stable and scalable writes, GraphQL for flexible and efficient reads/updates with lower payload sizes, and gRPC for high-throughput bulk reads.

The results help partially confirm the hypothesis - gRPC is more efficient communication technology than GraphQL and Thrift, as provides the shortest requests handling time under comparable conditions. The hypothesis can be confirmed in the case of reading multiple records.

In conclusion, it is impossible to point to a clear answer as to which technology has proven to be universally the best choice. When designing an application, one should pay attention to issues associated with each technology, such as the number of users, amount of data transferred between the server and the client or request throughput. An important factor when choosing a technology is also its popularity and comfort of use. The most common choice for developers is REST, but the research conducted prove that it will not always provide the best solution. Therefore, the above study presents the possibilities offered by other technologies.

References

[1] What is an API (Application Programming Interface)?, https://www.ibm.com/think/topics/api, [12.09.2025].

[2] R. T. Fielding, Architectural Styles and the Design of Network-based Software Architectures, PhD dissertation, University of California, Irvine, 2000.

[3] SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), W3C Recommendation, https://www.w3.org/TR/soap12-part1, [30.10.2025].

[4] Apache Thrift, https://thrift.apache.org, [12.09.2025].

[5] Introduction to gRPC, https://grpc.io/docs/what-is-grpc/introduction, [12.09.2025].

[6] The query language for modern APIs, https://graphql.org/, [12.09.2025].

[7] P. Margański, B. Pańczyk, REST and GraphQL comparative analysis, Journal of Computer Sciences Institute 19 (2021) 89–94, https://doi.org/10.35784/jcsi.2473.

[8] C. Oggier, How fast GraphQL is compared to REST APIs, Bachelor thesis, Haaga-Helia University of Applied Sciences, Helsinki, 2020.

[9] G. S. M. Diyasa, G. S. Budiwitjaksono, H. A. Ma'rufi, I. A. W. Sampurno, Comparative Analysis of Rest and GraphQL Technology on Nodejs-Based Api Development, Nusantara Science and Technology Proceedings (2021) 43–52, https://doi.org/10.11594/nstp.2021.0908.

[10] M. Śliwa, B. Pańczyk, Performance comparison of programming interfaces on the example of REST API, GraphQL and gRPC, Journal of Computer Sciences Institute 21 (2021) 356–361, https://doi.org/10.35784/jcsi.2744.

[11] S. Goriparthi, Streamlining API development: A comparative analysis of GraphQL and restful web services, International Journal of Data Analytics Research and Development 2(1) (2024) 59–71, https://doi.org/10.34218/IJDARD.

[12] G. Brito, M. T. Valente, REST vs GraphQL: A Controlled Experiment, In IEEE International Conference on Software Architecture (2020) 81–91, https://doi.org/10.48550/arXiv.2003.04761.

[13] A. Ambasht, API Integration using GraphQL, International Journal of Computer Trends and Technology (2023) 28–33, https://doi.org/10.14445/22312803/IJCTT-V71I8P104.

[14] O. Hartig, J. Pérez, Semantics and Complexity of GraphQL, In International World Wide Web Conferences Steering Committee, Geneva, Switzerland (2018) 1155–1164, https://doi.org/10.1145/3178876.3186014.

[15] M. Niswar, R. Safruddin, A. Bustamin, I. Aswad, Performance evaluation of microservices communication with REST, GraphQL, and gRPC, International Journal of Electronics and Telecommunications 70(2) (2024) 429 – 436, https://doi.org/10.24425/ijet.2024.149562.

[16] F. Aydemir, F. Başçiftçi, Performance and availability analysis of API design techniques for API gateways, Arabian Journal for Science 50 (2025) 11485–11498, https://doi.org/10.1007/s13369-024-09474-9.

[17] Ł. Kamiński, M. Kozłowski, D. Sporysz, K. Wolska, P. Zaniewski, R. Roszczyk, Comparative review of selected Internet communication protocols, Foundations of Computing and Decision Sciences 49 (2023) 39–56, https://doi.org/10.2478/fcds-2023-0003.

[18] M. Cederlund, Performance of frameworks for declarative data fetching: An evaluation of Falcor and Relay+GraphQL, Master thesis, Skolan för Informations, Stockholm, 2016.

[19] M. Vesić, N. Kojić, Comparative analysis of web application performance in case of using REST vs GraphQL, In 4th International Scientific Conference on Recent Advances in Information Technology, Tourism, Economics, Management and Agriculture – ITEMA (2020), https://doi.org/10.31410/ITEMA.2020.17.

[20] A. Lawi, B. L. E. Panggabean, T. Yoshida, Evaluating GraphQL and REST API services performance in a massive and intensive accessible information system, Computers 10(11) (2021) 138, https://doi.org/10.3390/computers10110138.

[21] M. Raghav, The future of payment systems: Transitioning from REST to Grpc for improved efficiency and security, Zenodo (2023), https://doi.org/10.5281/zenodo.7765404.

[22] JMeter, https://jmeter.apache.org, [30.10.2025].