JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 37 (2025) 451–456
Received: 24 June 2025
Accepted: 12 October 2025

Performance analysis of the GraphQL API creation technologies using Spring Boot and NestJS

Jakub Maciej Tkaczyk*, Beata Pańczyk

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

Abstract
Multitude of requirements for internet applications has led to creation of countless technologies. Goal of this article is to compare performance of server-side applications providing GraphQL API. Using frameworks NestJS and Spring Boot applications with the same business logic were created. Applications utilize the same data source - database "airport" developed for the experiment. In order to verify formulated hypotheses stress test was conducted. Each test set consisted of following number of requests: 1000, 2000, 4000. Tests showed that NestJS performance exceeds Spring Boot in GraphQL queries. However, GraphQL mutation results point out opposite.

Keywords: performance analysis; NestJS; Spring Boot; GraphQL

*Corresponding author
Email address: s95595@pollub.edu.pl (J. M.Tkaczyk)

1. Introduction

Web applications are widely used by all groups of people. Beginning with regular users, they enable easy access to information and some functionalities. Moreover, they displace often complicated installation process of desktop apps. The intuitive, fast, and reliable applications ensures that the end user does not need to be aware of its internal complexity. Today web applications are often complex systems that can coordinate operations of the entire businesses. The multitude of requirements for applications has given rise to a myriad of technologies, paradigms, and system development patterns. However, the common goal of all solutions is to achieve fast and reliable data exchange between system components as well as between the applications and the user.

For a long time, applications used the REST (Representational State Transfer) standard as a style of client-server communication. The approach is still valued for its simplicity, both in development and usage. However, over the years, a number of disadvantages of this approach have been revealed. Redundant data transfer and the need to perform several queries directly affected the execution time. Additionally, further development of existing APIs also proved to be problematic. In 2015, Facebook proposed new solution as an alternative to REST. GraphQL assumed, that query shape would be defined not by the server, but by the client. Utilization of DSL (Domain-specific Language) directly affected the complexity of queries. Tasks requiring multiple REST queries, in GraphQL could be done with a single query, which shape exactly matches client's requirements.

Goal of this paper is to test performance of selected frameworks, that can be used to implement server-side application, utilizing GraphQL API.

2. Literature review

Comparison of frameworks is rarely done based on GraphQL. Based on literature review, papers addressing this task could not be found. However, following articles prove, that GraphQL has features, that can be useful in software development. Article [1] demonstrates that REST and GraphQL present similar performance in load testing metrics. GraphQL significantly reduces the problem of underfetching and overfetching. It results in the optimalisation of API calls. Additionally, author conducted a survey, in which respondents say that popularity of GraphQL is expected to increase over the next five years. In article [2], researchers conducted an experiment involving calling equivalent REST and GraphQL APIs. It turned out, that responses sizes from GraphQL server were significantly smaller, up to 99%. Furthermore, by using Client Specific Queries, number of requests sent to the server can be reduced. Typically, while solving a task using REST API may take several queries, the same task can be done with just one GraphQL query. Authors of article [3] showed, that for simple queries, which returns single values, solutions based on REST and GraphQL achieved similar response time. However, for nested GraphQL queries turned out to be much more efficient.

In article [4] authors compared performance of several communication protocols in a system based on microservice architecture. Experiment included protocols: gRPC, GraphQL and REST. It was shown that system based on GraphQL had longest time of request handling, moreover it presented highest CPU and RAM utilization. In articles [2, 5] authors point out that developers find implementing GraphQL API significantly harder than similar REST API. Article [5] shows, that some of the most frequently discussed topic on the StackOverflow forum are challenges of implementation and deployment of GraphQL servers.

User experience while using existing GraphQL API is discussed in article [6]. Researchers conducted experiment on students with varying level of experience with working with API. They were asked to complete several tasks using REST and GraphQL APIs provided by GitHub. It turned out, that in general students required

smaller amount of time while using GraphQL API compared to REST. In later survey, respondents valued other GraphQL features, including embedded graphical interface, and user-friendly syntax of GraphQL queries. In summary, GraphQL an alternative style of communication with the server to REST. It works best in complex systems, where underfetching and overfetching are significant issues. GraphQL can limit both amount of data received from server and number of queries required to obtain required data. For smaller systems creation of GraphQL API required more work, compared to REST, and GraphQL does not benefit from the performance of query handling.

Articles [7-10] value Spring Boot by features other than request handling speed. In article [7] authors focused mainly on comparing Spring Boot performance with ASP.NET. Conducted test shows that Spring Boot handles queries longer. Authors of article [8] compared Spring Boot with Express and Django frameworks. After experiment they state that under load of 8000 virtual users Spring Boot has the highest request handling speed, but has significantly more failed request in comparison to Express. Additionally, authors point out that Spring Boot has embedded tools supporting software development, which is a particular advantage for more experienced developers. Similar conclusions were reached by authors of paper [9]. They conducted literature review in order to study developers' satisfaction, while working with frameworks: Django, Rails, Spring Boot and Laravel. A study was carried out, looking at features such as code generation, developer experience and business trends. The multi-criteria evaluation placed Spring Boot in first place ex aequo with Django.

Authors of articles [12, 13] focused on comparing Spring Boot and Node.JS ecosystems. They both created application that utilize REST API, with matching business logic and tested performance. Authors showed, that advantage of one tool over another is not straightforward.

In [12] researchers showed that requests in application made in Node.JS ecosystem handles request faster but difference was so small, that authors considered it insignificant. Paper [13] describes similar comparison, but authors specify that NodeJS application was implemented using Express framework. They conducted experiment comparing time of executing tasks with different complexity. For simple tasks (e.g. logging into application) NodeJS application turned out to be a winner. Spring Boot, on the other hand, performs complex tasks faster. Thes second part required extensive calculations and data from multiple database tables.

Utilization of GraphQL API can significantly impact performance. Results of articles [1, 2, 4] conclude, that GraphQL major advantage is prevention of underfetching and overfetching. GraphQL can outperform REST alternative in terms of speed. Moreover, it can reduce response size by up to 99% [2]. Although simple requests tend to be executed faster in REST. In [9] authors conducted paper reviews, which proved that Spring Boot tends to be slower, compared to NodeJS solutions. Authors of [7] reported consistent results with this statement. Spring Boot application required more time to process a request than its competitor. Conversely, some papers indicate the opposite. In article [8] researchers compared Spring Boot with Express framework. While they also assumed that Express will be faster, Spring Boot turned out to be winner. Based on this trend NestJS was expected to exceed Spring Boot in terms of speed in all test cases.

Article [11] showed that NestJS outrun other TypeScript frameworks in stress test. In experiment, for smaller response data differences was clear. Still, the differences narrowed with increasing response size. Author of [12] compared performance of applications created using NodeJS and Spring Boot framework. They showed that although results favour Node.JS application, difference is negligible. In article [13] there are similar results, however, authors divide test cases as "light" and "heavy" operations. Results presented in this paper are most aligned with findings from performed experiment. GraphQL query, which is equivalent to method GET in REST is performed faster in applications created with NodeJS ecosystem, which includes NestJS. Nevertheless, GraphQL mutations, corresponding to other REST methods, are executed faster in Spring Boot.

Based on literature review, it is possible to conclude, that direct comparison between NestJS and Spring Boot has not yet been performed. Furthermore, no comparison was found between frameworks, where compared applications used GraphQL API. However, based on other studies, it is possible to identify some trends in the performance differences between Node.JS and Java solutions.

3. Scope of work and hypotheses

Main objective of this paper is comparison of the NestJS v11.0.10 and Spring Boot v3.4.0 in terms of performance of API GraphQL. Created applications will be subjected to load tests by sending large number of queries. The following hypotheses have been formulated, testing of which is object of this paper.

H1. Framework NestJS has lower average time of handling simple request, compared to Spring Boot.

H2. GraphQL API created using NestJS handles nested queries faster than corresponding API implemented using Spring Boot.

H3. GraphQL mutations are performed faster in NestJS application that in its Spring Boot counterpart.

4. Materials and methods

Applications were created according to newest guidance available in documentations at the time of conducting this study. Implemented APIs have identical features and operate using the same database. Each application was tested using the same test suite, with different amount of request sent in given intervals. Results were essential to determine request handle time for each framework.

4.1. Technologies and tools

In order to verify formulated statements, two applications with same functionalities were created. Applications

were subjected to number of load tests. Using Apache JMeter requests were sent, and response time measured.

- GraphQL – alternative method of data exchange to REST API. Query language which allows for more versatile data access – client may define requested shape of response. It is possible to nest queries, which usually allows to obtain required data by sending only one request.
- Apache JMeter – open source tool for performing load tests of API. It is mainly used for testing the REST API, but it can be also utilized to test GraphQL. Application has many available configuration options, including number of virtual users, time intervals, and tests conditions. It allows for setting up complex test scenarios and generating reports. The tool can provide important information on the performance or stability of the API.
- NestJS – Node.js framework used for building server-side applications. It is widely used in full-stack systems because of possibility to use a single programming language for the entire project. Unlike most of the Node.js frameworks, it forces codebase structure, which facilitates teamwork and application scaling. Developers value it for modular structure, supporting multiple communication protocols and massive library resources.
- Spring Boot – framework that simplifies building Java enterprise applications. It has been used in professional systems for years, proving its stability and scalability. Spring Boot has an extensive ecosystem with solutions for many development tasks. It has been on the market much longer than NestJS. However, Spring Boot is still being developed and used in projects.

4.2. Test cases

Test cases are divided into three main groups:

- measurement of the execution time of GraphQL query without nested entities,
- measurement of the execution time of GraphQL query with nested entities (Listing 1),
- measurement of the execution time of GraphQL mutation.

Each group was tested using varying loads in order to determine how load affects API performance and stability. It was decided to use the following parameters:

- sending 1000 requests in one second,
- sending 2000 requests in one second,
- sending 4000 requests in one second.

GraphQL allow to define exact expected shape of response. GraphQL DSL is very similar to JSON language, which is widely used for communication between web applications. While sending only one request to the server, it's possible to fetch data from multiple related entities. GraphQL query used for experiment for complex test set is shown in listing 1. Query is used to obtain data of flight entities related to selected client. There are appended other fields which presence may be reasonable in real application. Each of nested fields is associated with table from database (see chapter 5.3).

Listing 1: Example of complex GraphQL query

```
query GET_FLIGHT_DATA($client_id: ID!) {
  client(id: $client_id) {
    id
    firstname
    lastname
    email
    tickets {
      id
      flight {
        id
        airportFrom {
          code
        }
        airportTo {
          code
        }
      }
    }
  }
}
```

4.3. Test environment

Experiment was performed on computer with Windows 11 installed. Tests were conducted using docker environment. Each application was built as docker image and launched alongside separate database instance in docker compose. Hardware parameters, relevant to the test, are presented in table 1. Table 2 contains versions of framework and tools used for experiment

Table 1: Hardware parameters of test platform

Component	Parameters
CPU	Intel Core i5-12400F
RAM	16GB DDR4
Operating system	Windows 11 24H2

Table 2: Versions of software used for the experiment

Software	Version
Spring Boot	v3.4.0
NestJS	v11.0.10
Docker	v27.4.0

4.4. Database

In order to test created applications, a database was created, the schema of which is shown in Figure 1. The database contains data related to flights. Data for the experiment was generated using script which generated set of random records for each table. It was decided to populate database with of approximately 200 000 records of artificially generated data. Table 3 contains specific number of rows in every table.

Table 3: Row count of each table

Entity	Rows
Client	100000
Ticket	40000
Transaction	40000
Flight	2000
Airport	1000

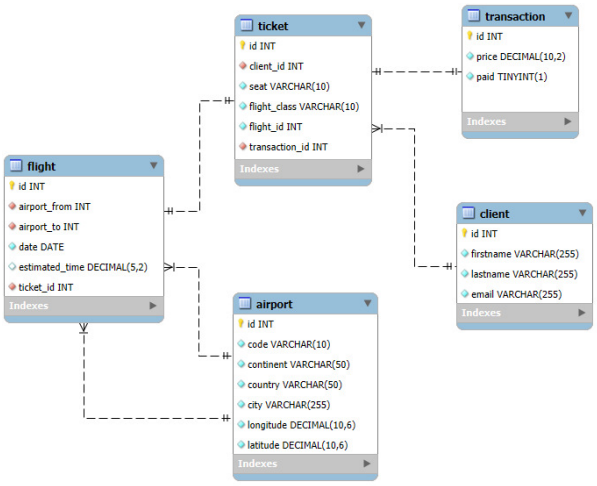

Figure 1: Schema of database used for the experiment.

5. Results

In each test case, there were samples that could not be considered a reasonable response time. These were marked as errors and filtered out from the dataset. For each sample, they accounted for approximately 10%. During load testing it is frequently observed that small number of requests require an order of magnitude more time to complete than average. This behaviour typically results from temporary contention for shared resources, including CPU time or memory. Additionally, under heavy load threads or event loop may become blocked by factors such as time-consuming database queries or internal garbage collection.

The average execution time for handling request was used as a comparison criterion. On charts this value is marked with X symbol. Provided tables contain more detailed data about each test, in particular number of requests sent (samples), average response time and standard deviation. Results shown in following subchapters relate GraphQL operations, which may find analogy in the REST standard. Simple query is associated with GET request in REST. Complex GraphQL query also matches GET operation, but usually requires multiple requests. In this paper GraphQL mutation is equivalent to POST request in REST. Following the charts, there are tables containing statistical values of the data.

5.1. Simple query

Simple GraphQL query is used to fetch data from one table. Results of the first test suite is presented in Figure 2. Horizontal axis represents number of requests sent in test and a vertical axis – response time. NestJS finished with average time 4-7ms which is slightly better than Spring Boot result: 7-9ms. Table 4 presents statistical data, specifically average and standard deviation for each test case.

Table 4: Results of simple query

Framework	Samples	Average [ms]	Std. dev.
NestJS	1000	4.58	0.64
NestJS	2000	5.93	2.31
NestJS	4000	7.84	2.46
Spring Boot	1000	7.13	1.62
Spring Boot	2000	7.61	2.78
Spring Boot	4000	9.40	2.93

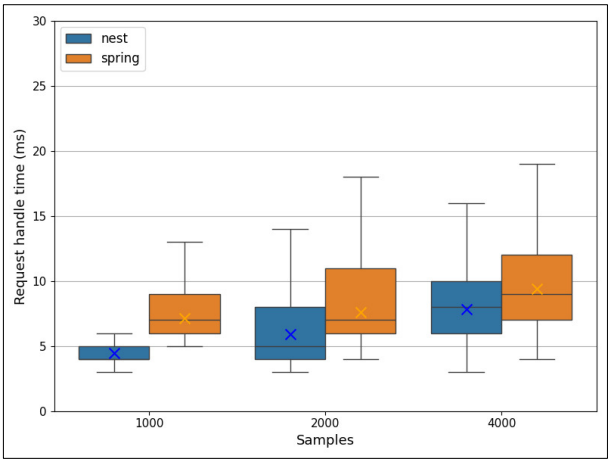

Figure 2: Comparison of results for simple GraphQL request.

5.2. Complex query

Complex GraphQL query is used to resolve relations between entities and obtain data from several database tables. Figure 3 shows result of this test suite, which again show that app written using NestJS on average required less time to complete task. Average request handle time ranged from 7ms to 11ms for NestJS and 9ms to 13ms for Spring Boot. Due to more complex task, the average query handling time increased relative to the previous test. However, in this test suite differences between results are lower. Table 5 presents statistical data calculated for this test suite.

Table 5: Results of complex query

Framework	Samples	Average [ms]	Std. dev.
NestJS	1000	7.17	1.14
NestJS	2000	8.74	2.61
NestJS	4000	11.36	2.88
Spring Boot	1000	9.29	2.66
Spring Boot	2000	10.69	3.93
Spring Boot	4000	12.80	3.63

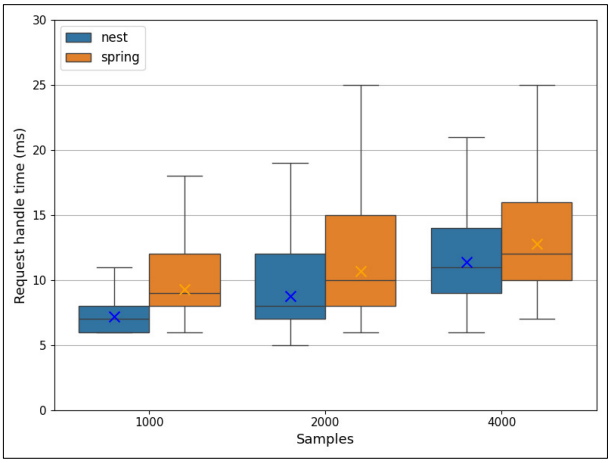

Figure 3: Comparison of results for complex GraphQL requests.

5.3. Mutation

In this test suite GraphQL mutation is used as an equivalent of POST method in REST. Comparison shown in Figure 4 shows different tendency than other test sets. Although in the previous test suites Spring Boot required more time to complete task, here results show the opposite. Contrary to hypothesis H3, Spring Boot performs data insertion faster, in all test cases compared to NestJS. Table 6 presents statistical data for this test.

Table 6: Results of mutation

Framework	Samples	Average [ms]	Std. dev.
NestJS	1000	8.18	1.24
NestJS	2000	10.07	2.68
NestJS	4000	12.73	2.72
Spring Boot	1000	4.20	0.87
Spring Boot	2000	5.46	2.38
Spring Boot	4000	7.87	2.99

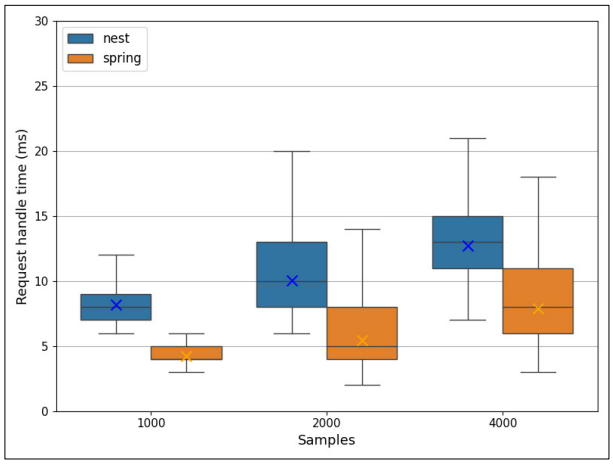

Figure 4: Comparison of results for GraphQL mutations.

6. Results discussion

Results presented in figure 2 confirm that framework NestJS has lower average time of handling simple request, compared to Spring Boot (H1). Figure 3 shows results supporting statement: GraphQL API created using NestJS handles nested queries faster than corresponding API implemented using Spring Boot (H2). However, results from figure 4 contradict hypothesis: GraphQL mutations are performed faster in NestJS application that in its Spring Boot counterpart (H3).

7. Conclusions

The experiment was conducted in line with designed test environment. Applications were developed in accordance with the latest standards provided in framework documentations. The use of docker images facilitated reproducibility of test environment. Moreover, placing JMeter in a virtual machine enabled appropriate allocation of host resources among all component of the experiment. Such setup allowed for reliable repetition of the experiment. Described test suite was conducted 10 times and results did not differ significantly from those presented.

Hypotheses were formulated based on the observed tendency that the Node.JS API server can run faster that its Java alternative. Nevertheless, the results of the experiment proved that only some of the postulates were confirmed. Hypotheses H1 and H2, which posited that GraphQL queries are processed faster in Nest.JS compared to Spring Boot, were confirmed. Conversely, statement that GraphQL mutation is handled faster in NestJS (H3) proved to be false. The rest of the results follow expected behaviour of application. As the load increases, so does the response time.

The conducted study provides valuable insights and opens field for further research in the performance of web applications. While GraphQL provides numerous benefits, it was not chosen as the leading protocol layer by other researchers, which motivated conducting such study. During verification of each hypothesis, the results clearly pointed to eighter confirmation of rejection. The partial support for the hypotheses invites a discussion regarding the underlying causes of such results, along with the classification of operations that favour a particular tool. Furthermore, future research may be extended to include wider spectrum of frameworks, databases, and implementation approaches.

References

[1] S. L. Vadlamani, B. Emdon, J. Arts, O. Baysal, Can GraphQL Replace REST? A Study of Their Efficiency and Viability, In IEEE/ACM 8th International Workshop on Software Engineering Research and Industrial Practice (2021) 10-17, https://doi.org/10.1109/SER-IP52554.2021.00009.

[2] G. Brito, T. Mombach, M. T. Valente, Migrating to GraphQL: A Practical Assessment, In IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER) (2019) 140-150, https://doi.org/10.1109/SANER.2019.8667986.

[3] R. Ala-Laurinaho, J. Mattila, J. Autiosalo, J. Hietala, H. Laaki, K. Tammi, Comparison of REST and GraphQL Interfaces for OPC UA, Computers 11(5) (2022) 1-17, https://doi.org/10.3390/computers11050065.

[4] M. Niswar, R. A. 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.

[5] S. Amareen, O. S. Dector, A. Dado, A. Bosu, GraphQL Adoption and Challenges: Community-Driven Insights from StackOverflow Discussions (2024) arXiv:2408.08363, https://doi.org/10.48550/arXiv.2408.08363.

[6] G. Brito, M. T. Valente, REST vs GraphQL: A Controlled Experiment, In IEEE International Conference on Software Architecture (ICSA) (2020) 81-91, https://doi.org/10.1109/ICSA47634.2020.00016.

[7] H. K. Dhalla, A Performance Comparison of RESTful Applications Implemented in Spring Boot Java and MS.NET Core, Journal of Physics: Conference Series 1933(1) (2021) 1-7, https://doi.org/10.1088/1742-6596/1933/1/012041.

[8] D. Choma, K. Chwaleba, M. Dzieńkowski, The efficiency and reliability of backend technologies: express, Django, and spring boot, Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska 13(4) (2023) 73-78, https://doi.org/10.35784/iapgos.4279.

[9] M. Mythily, A. S. A. Raj, I. T. Joseph, An Analysis of the Significance of Spring Boot in The Market, In International Conference on Inventive Computation Technologies (ICICT) (2022) 1277-1281, https://doi.org/10.1109/ICICT54344.2022.9850910.

[10] M. Kaluža, M. Kalanj, B. Vukelić, A comparison of back-end frameworks for web application development, Zbornik Veleučilišta u Rijeci 7(1) (2019) 317-332, https://doi.org/10.31784/zvr.7.1.10.

[11] M. Golec, M. Plechawska-Wójcik, Comparative analysis of frameworks using TypeScript to build server applications, Journal of Computer Sciences Institute 23 (2022) 128-134, https://doi.org/10.35784/jcsi.2910.

[12] I. Buljic, E. Kadusic, T. Cvijanovic, N. Hadzajlic, N. Zivic, Comparative Performance Analysis of Leading Backend Frameworks for Developers, In IEEE 24th International Symposium INFOTEH-JAHORINA (INFOTEH) (2025) 1-5, https://doi.org/10.1109/INFOTEH64129.2025.10959250.

[13] O. Novac, D. Ghiurău, M. Novac, C. Gordan, M. Oproescu, G. Bujdoso, Comparison of Node.Js and Spring Boot in Web Development, In IEEE 15th International Conference on Electronics, Computers and Artificial Intelligence (ECAI) (2023) 1-7, https://doi.org/10.1109/ECAI58194.2023.10194025.