JuSIN
Jurnal Sistem Informasi

Website: ojs.itb-ad.ac.id/index.php/JUSIN
E-ISSN: 2797-8516

ANALISIS PERBANDINGAN *FRAMEWORK NODE.JS NODE.JS* DAN SPRINGBOOT MENGGUNAKAN METODE *GRAPHQL*

Affu Dina Ilma Barkah[(1)], I Nyoman Yudi Anggara Wijaya[2] , Nengah Widya Utami[3]
[1]*Universitas Primakara, Denpasar*
[2]*Universitas Primakara, Denpasar*
[3]*Universitas Primakara, Denpasar*

Abstract

As the demand for fast, responsive, and scalable web applications grows, API technologies have rapidly evolved. GraphQL offers efficient data management and transfer, with Node.js and Spring Boot being two widely used frameworks for its implementation. This study aims to compare the performance of GraphQL APIs built with these frameworks. Performance testing focused on response time, throughput, error rate, and concurrent user capacity, using dummy data under varying loads via Postman. Results show that under low loads (10 virtual users), both frameworks perform similarly. However, at higher loads (50 virtual users), Spring Boot demonstrates better performance with faster response time (514 ms vs. 1,766 ms), higher throughput (53.64 rps vs. 18.46 rps), and no errors, while Node.js records a 1,79% error rate. In Spike and Peak scenarios, Spring Boot consistently outperforms Node.js. These findings provide valuable insights for developers in selecting the most suitable framework based on application needs.

Kata Kunci: *GraphQL, Node.js, Springboot, Performance testing, API.*

Informasi Artikel:
Dikirim : 17 September 2025
Ditelaah: 22 September 2025
Diterima: 30 September 2025
Publikasi: 23 Desember 2025

Juli – Desember 2025, Vol 6 (2) : hlm 91-104

(*) Korespondensi: affudina663@gmail.com *(Affu Dina Ilma Barkah)*

PENDAHULUAN

Bab Seiring perkembangan teknologi, tuntutan terhadap aplikasi yang cepat, responsif, dan dapat diakses oleh banyak pengguna secara bersamaan terus meningkat (Darmi dkk., 2024). Untuk memenuhi kebutuhan ini, pengembang aplikasi web memerlukan cara yang efisien dalam mengelola dan mentransfer data antara *server* dan klien. Selama bertahun-tahun, arsitektur REST (*Representational State Transfer*) menjadi pilihan utama untuk membangun API (*Application Programming Interface*). Namun, dengan semakin kompleksnya kebutuhan aplikasi modern, muncul berbagai masalah dalam penggunaan REST, seperti over-fetching, di mana klien menerima data lebih banyak dari yang dibutuhkan, dan under-fetching, di mana klien harus melakukan beberapa permintaan untuk mendapatkan data yang diperlukan (Hanif dkk., 2022).

Hingga akhirnya pada tahun 2015, Facebook memperkenalkan arsitektur API baru yang disebut *GraphQL* (Brito, 2020). *GraphQL* hadir sebagai solusi alternatif untuk mengatasi berbagai masalah yang dihadapi dalam pemrosesan data. Dengan *GraphQL*, klien dapat secara eksplisit menentukan data yang mereka butuhkan, sehingga mengurangi jumlah data yang ditransfer dan meningkatkan efisiensi dalam permintaan dan pengambilan data (Mulana dkk., 2022). Selain itu, *GraphQL* juga mendukung pengambilan data secara terstruktur, yang memungkinkan developer untuk mengoptimalkan struktur data yang kompleks dengan lebih mudah.

Dalam pengembangan API, ketersediaan berbagai bahasa pemrograman dan *framework* memberikan fleksibilitas yang tinggi dalam mengimplementasikan layanan yang optimal. Memilih *framework* yang tepat untuk pengembangan *GraphQL* API sangat penting, karena keputusan ini akan memengaruhi kinerja *server*, terutama dalam hal waktu respons (Hunter, 2020). Oleh karena itu, pemilihan *framework* yang sesuai menjadi kunci agar *server GraphQL* API mampu menangani permintaan dari klien dengan efisien dan handal.

Salah satu *framework* yang sangat populer dan banyak digunakan dalam pengembangan *Node.js* adalah *Node.js*. Berdasarkan survei Stack Overflow Developer tahun 2024, *Node.js* menempati posisi pertama dalam kategori *server Node.js* yang paling banyak digunakan, dengan persentase 40,8% dari total 48,503 responden (Sharma, 2023). *Node.js*, yang berbasis JavaScript, terkenal karena kemampuannya dalam menangani operasi I/O secara asinkron, membuatnya ideal untuk aplikasi yang memerlukan kecepatan dan skalabilitas tinggi (Shudhuashar, 2023).

Gambar 1. Survey Stack Overflow 2023

Namun, di samping popularitas *Node.js*, ada *framework* lain yang tidak kalah menarik untuk digunakan, yaitu *Spring Boot*. Meskipun hanya memperoleh 12,7% dari total 48.503 responden dalam survei yang sama, *Spring Boot* tetap menjadi pilihan utama bagi banyak pengembang aplikasi enterprise sebeagai contoh startup besar yang masih menggunakan Springboot adalah BliBli (Sharma, 2023). *Framework* ini berbasis Java dan menawarkan berbagai fitur unggulan seperti konfigurasi otomatis, dukungan penuh untuk ekosistem Java, serta kemampuan untuk menangani aplikasi yang kompleks dengan stabilitas tinggi.

Penelitian tentang perbandingan implementasi *GraphQL* menggunakan *Node.js* dan *Spring Boot* ini sangat menarik untuk diteliti, karena masing-masing memiliki karakteristik berbeda dalam hal performa dan cara kerja. *Node.js*, dengan arsitektur berbasis event-driven, menawarkan efisiensi dalam penggunaan sumber daya serta responsivitas tinggi (Choma dkk., 2023). Di sisi lain, *Spring Boot* menggunakan pendekatan berbasis thread dan dukungan penuh dari ekosistem Java, menjadikannya stabil dan andal untuk aplikasi enterprise (Sharma, 2023). Dalam penelitian ini, performa kedua *framework* diukur menggunakan metode performance testing. Beberapa indikator kunci digunakan untuk mengevaluasi kinerja, seperti *Response time, Throughput, Error rate, dan concurrent user* (Dwinur, 2024).

Selain itu, penelitian ini juga menggunakan load testing sebagai metode pengujian untuk mensimulasikan beban real-case pada sistem. Load testing berfokus pada pengukuran kemampuan sistem dalam menangani jumlah permintaan yang tinggi secara bersamaan. Teknik ini penting untuk mengevaluasi batas maksimal kapasitas *server* dan memastikan bahwa sistem tetap dapat berfungsi dengan baik di bawah beban yang tinggi (Hasnain, 2020).

Beberapa penelitian sebelumnya telah membahas perbandingan *GraphQL* dengan REST API pada berbagai *framework* (Hanif dkk., 2022; Brito, 2020), serta analisis performa *framework Node.js* seperti Express, Django, dan *Spring Boot* (Choma dkk., 2023). Namun, sebagian besar penelitian tersebut hanya menekankan pada perbandingan arsitektur API atau benchmarking *framework* tanpa fokus mendalam pada implementasi *GraphQL*. Beberapa studi lain memang meneliti performa *Node.js*, tetapi terbatas pada lingkungan *server* atau basis data tertentu (Dwinur, 2024; Putu dkk., 2023).

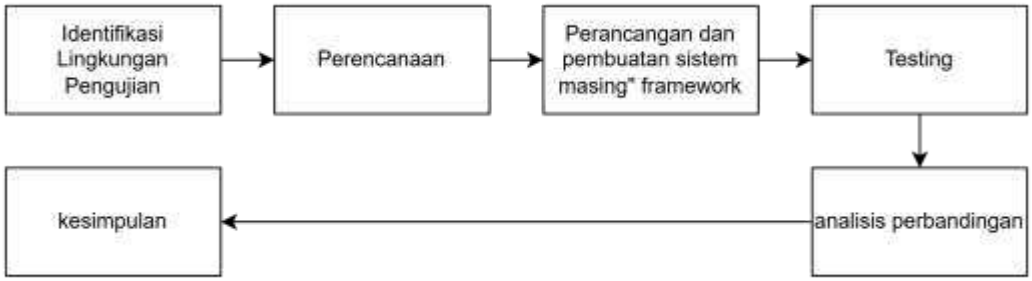
Berbeda dengan penelitian-penelitian tersebut, penelitian ini menawarkan kebaruan (*novelty*) dengan secara khusus membandingkan performa *Node.js* dan *Spring Boot* dalam implementasi *GraphQL* API melalui pengujian beban (*load testing*) menggunakan Postman. Fokus pada indikator *response time*, *throughput*, *error rate*, dan kapasitas pengguna konkuren memberikan sudut pandang baru yang belum banyak diulas dalam penelitian sebelumnya. Dengan demikian, penelitian ini diharapkan dapat menjadi referensi komprehensif bagi pengembang dalam menentukan *framework Node.js* yang paling sesuai dengan kebutuhan performa aplikasi berbasis *GraphQL*

METODE

Penelitian ini menggunakan pendekatan metode *performance testing* untuk mengevaluasi kinerja *framework Node.js* dan *Spring Boot* dalam membangun API berbasis

GraphQL. Jenis pengujian kinerja yang digunakan adalah *load testing*, yang bertujuan untuk mengukur kemampuan sistem dalam menangani beban kerja tertentu secara simultan (Barus dkk., 2021).

Pengujian Proses eksperimen melibatkan pengujian pada masing-masing implementasi dengan berbagai variasi jumlah data dan entitas *Employee*. Hasil pengujian kemudian akan dianalisis untuk menentukan *framework* yang memiliki performa lebih baik berdasarkan indikator yang telah ditentukan. Data hasil eksperimen akan disajikan dalam bentuk grafik untuk mempermudah interpretasi dan perbandingan (Dwinur, 2024)



Gambar 2. Alur Pengujian

1. Identifikasi Lingkungan Pengujian

Adapun Lingkungan pengujian dalam penelitian ini disiapkan untuk memastikan proses eksperimen terhadap *framework Node.js* dan *Spring Boot* berjalan secara konsisten dan terkontrol. Lingkungan tersebut terdiri dari perangkat keras, perangkat lunak, database, serta layanan *cloud server* yang digunakan sebagai media implementasi dan pengujian API *GraphQL*.

Tabel 1. Instrumen Penelitian

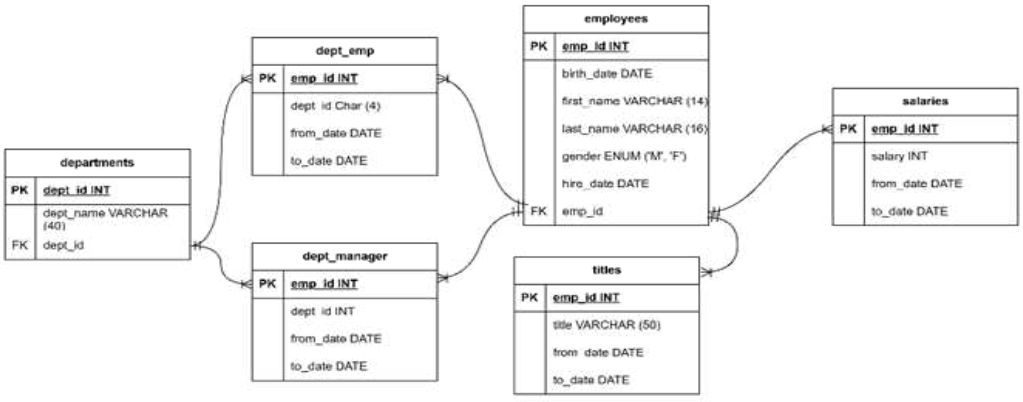
No	Nama	Spesifikasi	Deskripsi
1	Laptop Acer Nitro AN515-57	Intel® CoreTM i5-11400H with RAM 8GB SSD 512GB (PCIe NVMe) and OS Windows 10	*Hardware* yang digunakan untuk melakukan proses pengembangan aplikasi
2	Visual Studio Code	Versi 1.81.1	*Software* yang digunakan untuk pengembangan aplikasi *Node.js*
3	Intellij Idea	Versi 2023.1	*Software* yang digunakan untuk pengembangan aplikasi Java
4	Java	Versi 18.0.1.1	Bahasa pemrograman untuk pengembangan aplikasi menggunakna *framework* springboot

5	Springboot	Versi 3.0.1	*Framework* Java untuk mengembangkan aplikasi berbasis web
6	Apache Maven	Versi 3.8.6	*Software* untuk manage project Java
7	*Node.js*	Versi 22.11.0	Runtime Javascript untuk pengembangan aplikasi *Node.js*
8	*GraphQL*	Versi 22	*Query language* untuk pengelolaan data API
9	*PostgreSQL*	Versi 17	*Database* data
10	Heroku	Dyno Type 512 MB RAM, Shared CPU, *non autoscaling*	*Cloud Server*

2. Perencanaan

Pada tahap perencanaan ini, dibuatlah perancangan basis data *Employee* yang menjadi inti dari sistem yang akan dikembangkan. Perancangan dilakukan dengan menggunakan *Entity Relationship Diagram* (ERD) seperti yang ditampilkan pada gambar di atas, yang mencakup entitas utama seperti *Employees, departments, dept_emp, dept_manager, salaries, dan titles.*

Relasi antar entitas tersebut dirancang untuk merepresentasikan struktur organisasi dan riwayat pekerjaan karyawan secara lengkap, mulai dari penempatan departemen, status manajerial, posisi jabatan, hingga riwayat gaji.Relasi antar entitas tersebut dirancang untuk merepresentasikan struktur organisasi dan riwayat pekerjaan karyawan secara lengkap, mulai dari penempatan departemen, status manajerial, posisi jabatan, hingga riwayat gaji.



Gambar 3. ERD *Employee*

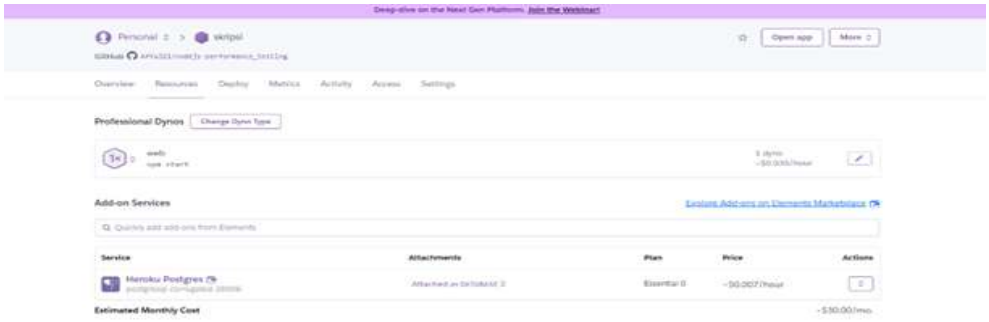
Pada Gambar 3 menunjukkan *Entity Relationship Diagram* (ERD) yang menggambarkan relasi antar entitas dalam sistem kepegawaian. Entitas utama terdiri dari *Employees*, *departments, titles, salaries, dept_emp, dan dept_manager.* Setiap karyawan

(*Employees*) dapat memiliki beberapa jabatan (*titles*), riwayat gaji (*salaries*), serta dapat ditempatkan di berbagai departemen melalui entitas *dept_emp*. Manajer departemen direpresentasikan melalui dept_manager. Relasi ini memungkinkan sistem menyimpan data historis karyawan secara terstruktur dan terintegrasi.

3. Implementasi Perancangan

Pada tahap ini dilakukan implementasi dari perancangan ERD yang telah dibuat dengan cara mengembangkan API *GraphQL* menggunakan *Spring Boot* dan *Node.js*. API tersebut kemudian diuji untuk membandingkan performa kedua teknologi dalam mendukung fitur sistem CRUD (*Create, Read, Update, Delete*) pada data *Employee*.

Selanjutnya, hasil implementasi dari perancangan *framework* tersebut di-*setup* pada sebuah *server* untuk dilakukan pengujian di sisi *Node.js*. Untuk mendukung koneksi database pada *server Node.js*, Heroku menyediakan add-on *PostgreSQL* secara langsung, sehingga memudahkan integrasi antara aplikasi dan database di lingkungan produksi. Bisa terlihat pada gambar 4 adalah konfigurasi dari heroku *server*



Gambar 4. *add on* Heroku

4. Pengujian

Pada Pada tahap ini, masing-masing *framework* yaitu *Spring Boot* dan *Node.js* diuji menggunakan API *GraphQL* yang telah dikembangkan dengan fungsionalitas yang setara. Pengujian dilakukan melalui endpoint POST untuk menjalankan operasi *Create, Read, Update*, dan *Delete* (CRUD) pada data karyawan yang tersimpan di basis data.

Pengujian performa dilakukan menggunakan pendekatan *load testing* dengan berbagai skenario beban untuk mensimulasikan kondisi penggunaan nyata. Adapun skenario yang digunakan meliputi:

a. *Initial Load*, yaitu jumlah pengguna awal yang langsung mengirim permintaan ke *server* pada awal pengujian.

b. *Base Load,* yaitu jumlah pengguna yang bertahan secara konstan selama proses pengujian.

c. *Ramp-Up,* yaitu peningkatan jumlah pengguna secara bertahap untuk mengamati adaptasi sistem terhadap lonjakan beban.

d. *Spike*, yaitu peningkatan beban secara tiba-tiba dan drastis dalam waktu singkat untuk menguji ketahanan sistem.

e. *Peak Load*, yaitu simulasi beban maksimum yang dipertahankan dalam durasi tertentu untuk menguji stabilitas sistem di kondisi ekstrem.

Pengujian dilakukan menggunakan aplikasi Postman dengan metode load testing untuk mensimulasikan jumlah pengguna yang mengakses API secara bersamaan (*virtual users*). Durasi pengujian ditetapkan selama 5 menit untuk setiap skenario dengan pengaturan sebagai berikut:

Tabel 2. Skema Pengujian

Jenis Testing	*Virtual User (VU)*	Durasi	*Initial Load*	*Base Load*
Ramp Up	10	5 menit	3	-
Ramp Up	50	5 menit	15	-
Spike	10	5 menit	-	1
Spike	50	5 menit	-	5
Peak	10	5 menit	-	2
Peak	50	5 menit	-	10

5. Analisis Perbandingan

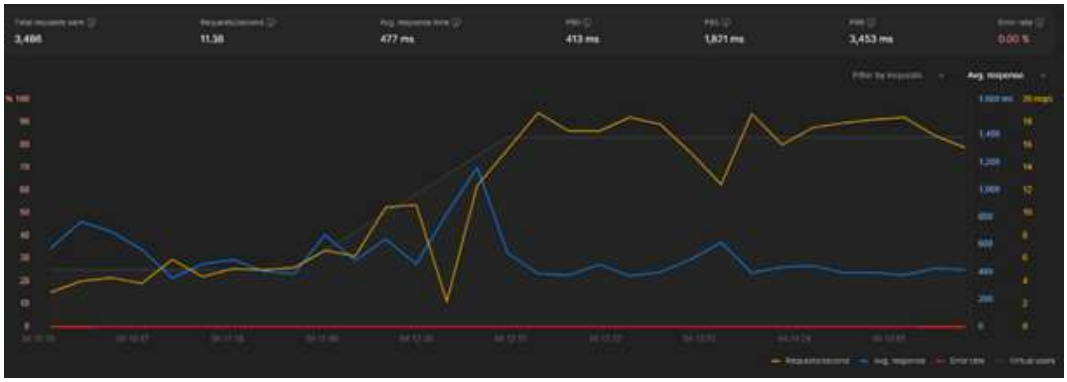
Setelah pengujian dilakukan, hasilnya akan berupa output dalam bentuk grafik yang dihasilkan melalui *software* Postman. Grafik tersebut akan dianalisis untuk membandingkan performa antara *Node.js* dan *Spring Boot* dalam implementasi API *GraphQL*, guna menentukan teknologi yang memberikan kinerja terbaik.

6. Kesimpulan

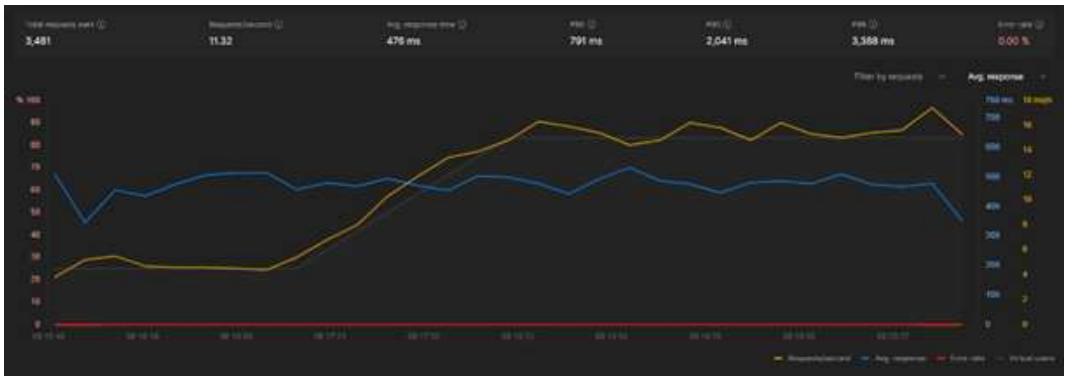
Hasil analisis performa yang dilakukan diharapkan dapat memberikan wawasan yang jelas mengenai keunggulan dan kelemahan masing-masing *framework* berdasarkan indikator seperti *response time, throughput, dan error rate.* Dengan demikian, penelitian ini diharapkan dapat menjadi referensi yang bermanfaat bagi pengembang dalam memilih teknologi yang paling sesuai dengan kebutuhan sistem.

HASIL DAN PEMBAHASAN

Adapun Setelah seluruh test plan yang telah dirancang sebelumnya dijalankan, maka diperoleh hasil pengujian untuk masing-masing *framework* yang digunakan dalam penelitian ini. Hasil pengujian tersebut menjadi dasar untuk melakukan analisis perbandingan kinerja antar *framework*. Berikut adalah hasil pengujian dari masing-masing *framework*, Berikut adalah hasil pengujian dari masing-masing *framework*:



Gambar 5. *Ramp Up* 10 VU *Node.js*

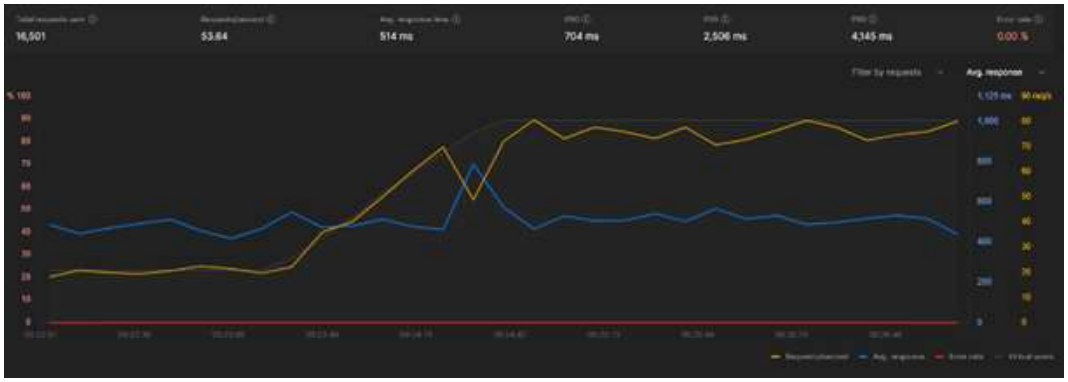


Gambar 6. *Ramp Up* 10 *VU* Springboot

Pengujian *Ramp Up* 10 *VU* tersebut menghasilkan:

a. *Response times: Node.js* (477 ms), Springboot (476 ms)
b. *Throughput*: *Node.js* (11.38 rps), Springboot (11.32 rps)
c. *Error rate*: *Node.js* (0%), Springboot (0%)'

Gambar 7. *Ramp Up* 50 VU *Node.js*



Gambar 8. *Ramp Up* 50 VU Springboot

Pengujian *Ramp Up* 50 *VU* tersebut menghasilkan:

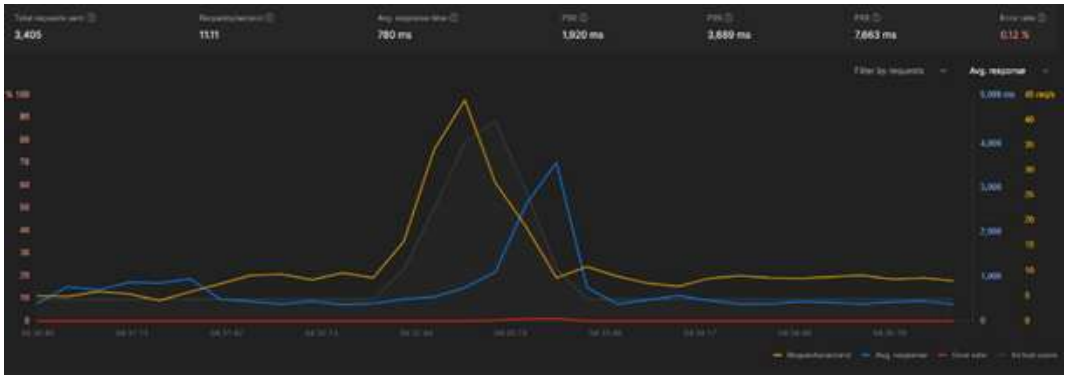
a. *Response times: Node.js* (1.766 ms), *Springboot* (514 ms)
b. *Throughput*: *Node.js* (18.46 rps), *Springboot* (53.64 rps)
c. *Error rate*: *Node.js* (1,79%), *Springboot* (0%)'

Gambar 9. *Spike* 10 *VU Node.js*

Gambar 10. *Spike* 10 *VU* Springboot

Pengujian *Spike* 10 *VU* tersebut menghasilkan:

a. Response times: Node.js (447 ms), Springboot (551 ms)
b. Throughput: Node.js (3.20 rps), Springboot (2.76 rps)
c. Error rate: Node.js (0%), Springboot (0%)



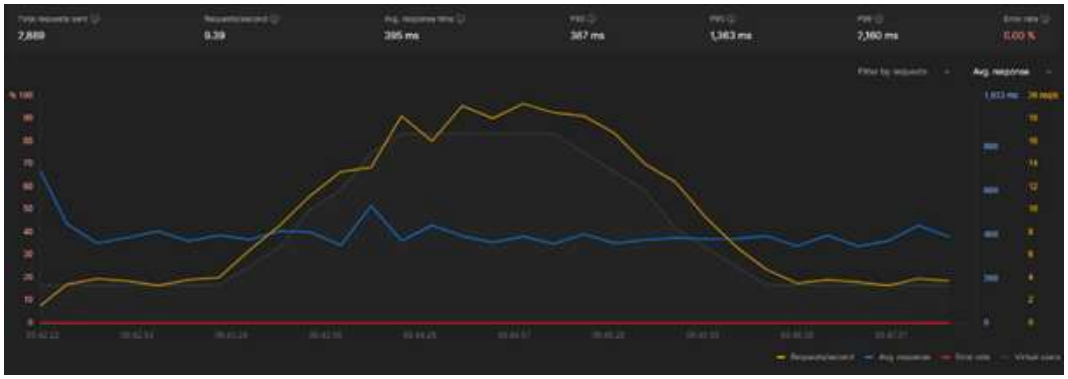
Gambar 11. *Spike* 50 VU *Node.js*

Gambar 12. *Spike* 50 VU Springboot

Pengujian *Spike* 50 *VU* tersebut menghasilkan:

a. *Response times: Node.js (780 ms), Springboot (404 ms)*
b. *Throughput: Node.js (11.11 rps), Springboot (17.22 rps)*
c. *Error rate: Node.js (0,12%), Springboot (0%)*

Gambar 13. *Peak* 10 *VU Node.js*

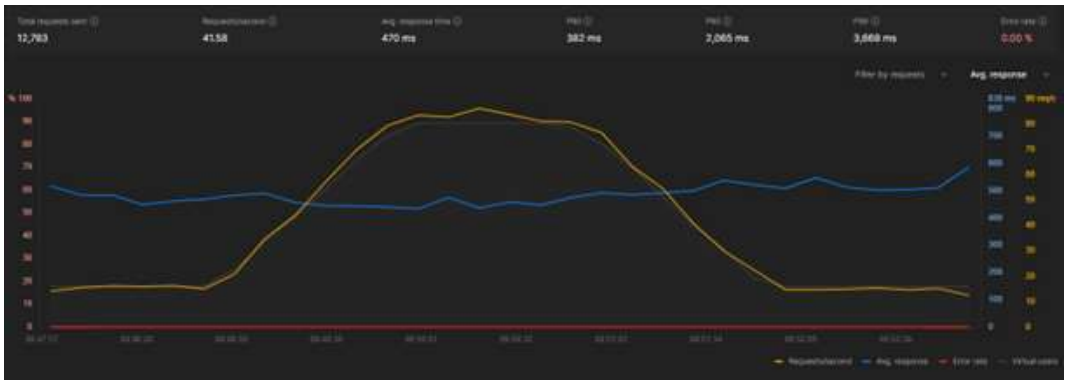


Gambar 14. *Peak* 10 *VU* Springboot

Pengujian *Peak* 50 *VU* tersebut menghasilkan:

a. *Response times: Node.js (477 ms), Spring Boot(395 ms)*
b. *Throughput: Node.js (11.11 rps), Spring Boot(17.22 rps)*
c. *Error rate: Node.js (0%), Spring Boot (0%)*

Gambar 15. *Peak* 50 VU *Node.js*



Gambar 16. *Peak* 50 VU *Spring Boot*

Pengujian *Peak* 50 *VU* tersebut menghasilkan:

a. *Response times: Node.js (1.725 ms), Springboot (470 ms)*
b. *Throughput: Node.js (14.27 rps), Springboot (41.58 rps)*
c. *Error rate: Node.js (0,78%), Springboot (0%)*

Note:
Merah = *Error rate*
Kuning = *Request/second*
Biru = *Average Response*

Tabel 3. Analisis Perbandingan

No	Jenis Testing	*Virtual User*	*Framework*	*Avg. Respone time (ms)*	*Throughput (req/s)*	*Error*
1	*Ramp Up*	10	*Node.js*	477	11.38	-
			Spring Boot	476	11.32	-
2	*Ramp Up*	50	*Node.js*	1.766	18.46	1.79
			Spring Boot	514	53.64	-
3	*Spike*	10	*Node.js*	477	3.20	-
			Spring Boot	551	2.76	-
4	*Spike*	50	*Node.js*	780	11.11	0.12
			Spring Boot	404	17.22	-
5	*Peak*	10	*Node.js*	477	8.15	-
			Spring Boot	395	9.39	-
6	*Peak*	50	*Node.js*	1.725	14.27	0.78
			Spring Boot	470	41.58	-

Berdasarkan hasil pengujian yang ditampilkan pada Tabel 4.2, performa antara *framework Node.js* dan *Spring Boot* menunjukkan perbedaan yang cukup jelas, terutama saat diuji dengan beban tinggi. Pada pengujian *Ramp Up* 10 VU, keduanya menunjukkan performa yang hampir sama, baik dari sisi *response time* maupun *throughput*, dan tidak ditemukan *error*. Namun, ketika jumlah *virtual user* meningkat menjadi 50, *Spring Boot* unggul signifikan, dengan *response time* lebih cepat (514 ms vs 1.766 ms), *throughput* lebih tinggi (53.64 rps vs 18.46 rps), serta tidak mengalami *error*, sedangkan *Node.js* mencatat *error rate* sebesar 1,79%.

Pada skenario *Spike* 10 VU, *Node.js* sedikit lebih baik dari segi *response time* dan *throughput*, tetapi keunggulan tersebut tidak bertahan saat spike meningkat ke 50 VU, di mana *Spring Boot* kembali lebih unggul dengan *response time* yang lebih rendah dan *throughput* yang lebih tinggi. Pada kedua *framework*, *error rate* tetap 0% di skenario ini., Sedangkan pada pengujian Peak, *Spring Boot* secara konsisten menunjukkan performa lebih baik, terutama saat diuji dengan 50 VU. *Spring Boot* mampu mempertahankan *response time* di 470 ms dengan throughput mencapai 41.58 rps, jauh di atas *Node.js* yang mencatat *response time* 1.725 ms dan throughput 14.27 rps.

Dari keseluruhan pengujian, dapat disimpulkan bahwa *Spring Boot* lebih unggul dari sisi *response* time dan throughput, khususnya saat menangani beban yang lebih besar, sedangkan *Node.js* menunjukkan kestabilan dari sisi error rate meskipun mengalami penurunan performa saat beban meningkat. Hal ini menunjukkan bahwa pemilihan *framework* sebaiknya

disesuaikan dengan kebutuhan sistem, apakah lebih memprioritaskan efisiensi performa atau kestabilan sistem.

KESIMPULAN

Berdasarkan hasil penelitian yang telah dilakukan dapat diambil kesimpulan sebagai berikut.

1. Keduanya mampu mengimplementasikan *GraphQL* API dengan baik, menjalankan fungsionalitas CRUD menggunakan basis data *PostgreSQL*, dan dapat di-*deploy ke cloud* menggunakan Heroku. Skema dan *endpoint* yang digunakan telah disamakan agar pengujian berlangsung secara objektif.
2. *Spring Boot* menunjukkan performa yang lebih unggul dari segi *response time* dan *throughput*, terutama pada skenario dengan beban tinggi *(Ramp Up, Spike,* dan *Peak* dengan 50 *virtual user*). Selain itu, *Spring Boot* mencatatkan hasil pengujian tanpa error pada semua skenario.
3. *Node.js* mulai mengalami penurunan performa dan kestabilan saat beban tinggi, ditandai dengan *response* time yang lebih lambat, throughput yang lebih rendah, dan munculnya error rate sebesar 1,79% pada skenario Ramp Up 50 VU. Namun, *Node.js* tetap dapat menjadi pilihan untuk sistem ringan atau real-time yang memprioritaskan efisiensi sumber daya.

DAFTAR PUSTAKA

Anwar, M. D., & Kautsar, I. A. (2024). **Arsitektur Perangkat Lunak Berbasis Layanan Mikro pada Sistem Manajemen Informasi Kantin**. *Physical Sciences, Life Science and Engineering*, *1*(2), 13. https://doi.org/10.47134/pslse.v1i2.196

Barus, A. C., Harungguan, J., & Manulu, E. (2021). **PENGUJIAN API WEBSITE UNTUK PERBAIKAN PERFORMANSI APLIKASI DITENUN**. *Journal of Applied Technology and Informatics*.

Brito, G., & Valente, M. T. (2020). **REST vs *GraphQL***: A controlled experiment. *Proceedings - IEEE 17th International Conference on Software Architecture, ICSA 2020*, 81–91. https://doi.org/10.1109/ICSA47634.2020.00016

Choma, D., Chwaleba, K., & Dzieńkowski, M. (2023). **THE EFFICIENCY AND RELIABILITY OF *NODE.JS* TECHNOLOGIES: EXPRESS, DJANGO, AND *SPRING BOOT***. *Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Srodowiska*, *13*(4), 73–78. https://doi.org/10.35784/iapgos.4279

Darmi, Y., Pinandita, K., & Muhammadiyah Bengkulu, U. (2024). **IMPLEMENTASI PERBANDINGAN METODE *GRAPHQL* DAN REST API PADA TEKNOLOGI NODEJS COMPARATIVE IMPLEMENTATION OF *GRAPHQL* AND REST API METHODS IN NODEJS TECHNOLOGY**. *Journal of Information Technology and Computer Science (INTECOMS)*, *7*(1).

Dwinur Andrianto, L., & Fatrianto Suyatno, D. (2024). ***Analisis Performa Load Testing Antara Mysql Dan Nosql Mongodb Pada RestAPI Nodejs Menggunakan Postman*.**

Hanif, F., Ahmad, I., Darwis, D., Lazuardi Putra, I., & Fauzan Ramadhani, M. (2022). **ANALISA PERBANDINGAN METODE *GRAPHQL* API DAN REST API DENGAN MENGGUNAKAN ASP.NET CORE WEB API *FRAMEWORK*.** Dalam *Jl. ZA. Pagar Alam* (Vol. 3, Nomor 2).

Hermanto, M., Amir, T., Siregar, H., & Ginting, S. (2024). ***FULLSTACK PROGRAMMING: MEMBANGUN APPLICATION PROGRAMMING INTERFACE (API) DENGAN LARAVEL*** (M. H. Tinambunan, A. H. Siregar, & S. Ginting, Ed.).

Hunter, T. (2020). ***Distributed Systems with Node.js*.** O'Reilly Media, Inc.

mdn. (2022). ***What is JavaScript?*** https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

Muhammad Hasnain, & Fermi Pasha. (2020). ***An Efficient Performance Testing of Web Services*.**

Mulana, L., Prihandani, K., Rizal, A., Singaperbanga, U., & Abstract, K. (2022). **Analisis Perbandingan Kinerja *Framework* Codeigniter Dengan Express.Js Pada *Server* RESTful Api.** *Jurnal Ilmiah Wahana Pendidikan*, *8*(16), 316–326. https://doi.org/10.5281/zenodo.7067707

Mythily, M., Samson Arun Raj, A., & Thanakumar Joseph, I. (2022a). **An Analysis of the Significance of *Spring Boot* in The Market.** ***2022 International Conference on Inventive Computation Technologies*** *(ICICT)*, 1277–1281. https://doi.org/10.1109/ICICT54344.2022.9850910

Node. (2024). ***Node.js, "about" Node.js*.**

Ogbu Uzoma. (2023). ***A Brief Hitory of Node.js***. https://medium.com/@ogbuuzoma413/brief-history-of-nodejs-de0cac0af448

Postman. (2024). ***Simulate user traffic to test your API performance*.** https://learning.postman.com/docs/collections/performance-testing/testing-api-performance/

Putu, I., Eka Pratama, A., Made, I., & Raharja, S. (2023). ***Node.js Performance Benchmarking and Analysis at Virtualbox, Docker, and Podman Environment Using Node-Bench Method***. www.joiv.org/index.php/joiv

Santhy Toamain, A., Pusat, B., & Maluku, S. P. (2021). **Rancang Aplikasi Chatbot Sebagai Virtual Asisten Dalam Pelayanan Pengguna Data Di Badan Pusat Statistik Provinsi Maluku**. *Jurnal Teknologi Informasi*, *7*. http://ejournal.urindo.ac.id/index.php/TI

Sharma, S. (2023). ***Modern API Development with Spring 6 and Spring Boot 3*.** Packt Publishing.

Shudhuashar, M. (2023). **Sistem Pendataan Berat Badan Kambing Dengan Sensor Load Cell dan RFID Menggunakan Teknologi *Cloud* Computing Berbasis IOT**. *Journal Computer Islamic*.