Jurnal Teknik Informatika (JUTIF)
Vol. 5, No. 2, April 2024, pp. 447-453

DOI: https://doi.org/10.52436/1.jutif.2024.5.2.1725
p-ISSN: 2723-3863
e-ISSN: 2723-3871

REDUCING UNDER-FETCHING AND OVER-FETCHING IN REST API WITH GRAPHQL FOR WEB-BASED SOFTWARE DEVELOPMENT

Rizki Nuzul Muzaki [*1], Abu Salam[*2]

[1,2]Study Program in Informatics Engineering, Faculty of Computer Science,
Universitas Dian Nuswantoro, Semarang, Indonesia
Email: [1]111202012762@mhs.dinus.ac.id, [2]abu.salam@dsn.dinus.ac.id

(Article received: January 12, 2024; Revision: February 6, 2024; Published: April 04, 2024)

Abstract

Rest API is the most popular architectural style in website-based software development. However, Rest API has under-fetching and over-fetching problems. Under-fetching is a situation when the client has to make requests to several endpoints, while over-fetching is a situation when the client receives more data than needed. There is an alternative technology to Rest API, namely GraphQL. GraphQL has the potential to solve both under-fetching and over-fetching problems. This research aims to analyze how quickly GraphQL responds in overcoming under-fetching and over-fetching problems and conducting condition analysis to determine when it is best to use GraphQL. In this research, tests were conducted to answer these problems by applying each of the five test scenarios for under-fetching and over-fetching problems. Test results show that GraphQL can provide response speeds of 36.84% to 93.04% superior to Rest API. In the case of under-fetching, it is best to choose GraphQL when there is a need to call more than four endpoints. Meanwhile, for over-fetching problems, using the Rest API provides adequate response speed. However, if a more optimal response speed is needed, using GraphQL could be an alternative.

Keywords: *GraphQL, Over-fetching, Rest API, Under-fetching*

REDUKSI UNDER-FETCHING DAN OVER-FETCHING PADA REST API DENGAN GRAPHQL DALAM PENGEMBANGAN PERANGKAT LUNAK BERBASIS WEB

Abstrak

Rest API merupakan gaya arsitektur paling populer dalam pengembangan perangkat lunak berbasis *website*. Namun, Rest API memiliki masalah *under-fetching* dan *over-fetching*. *Under-fetching* merupakan keadaan ketika klien harus melakukan permintaan kepada beberapa *endpoint*, sedangkan *over-fetching* merupakan keadaan ketika klien menerima data yang lebih banyak daripada yang dibutuhkan. Terdapat teknologi alternatif untuk Rest API, yaitu GraphQL. GraphQL memiliki potensi untuk mengatasi masalah *under-fetching* dan *over-fetching*. Penelitian ini bertujuan untuk melakukan analisis seberapa cepat respon GraphQL dalam mengatasi masalah *under-fetching* dan *over-fetching*, serta melakukan analisis kondisi untuk menentukan kapan sebaiknya penggunaan GraphQL. Pada penelitian ini dilakukan pengujian untuk menjawab permasalahan tersebut dengan menerapkan masing-masing lima skenario pengujian untuk masalah *under-fetching* dan *over-fetching*. Hasil pengujian menunjukkan bahwa GraphQL mampu memberikan kecepatan respon 36.84% hingga 93.04% lebih unggul dibandingkan Rest API. Pada masalah *under-fetching*, sebaiknya memilih GraphQL ketika terdapat kebutuhan pemanggilan lebih dari 4 *endpoint*. Sedangkan pada masalah *over-fetching*, penggunaan Rest API memberikan kecepatan respon yang memadai. Namun, apabila diperlukan kecepatan respon yang lebih optimal, penggunaan GraphQL dapat menjadi pilihan.

Kata kunci: *GraphQL, Over-fetching, Rest API, Under-fetching*

1. PENDAHULUAN

Rest API adalah gaya arsitektur paling populer dalam pengembangan perangkat lunak berbasis web [1]. Rest API menggunakan protokol HTTP untuk komunikasi data pada perangkat lunak berbasis web [2]. Rest API telah menjadi salah satu teknologi paling populer dan paling banyak digunakan dalam pengembangan perangkat lunak [3], [4]. Meskipun populer, Rest API memiliki beberapa kekurangan termasuk *under-fetching* dan *over-fetching* [1], [2], [5], [6], [7]. *Under-fetching* adalah keadaan ketika klien harus melakukan permintaan kepada beberapa *endpoint* untuk mendapatkan data yang dibutuhkan,

sedangkan *over-fetching* adalah keadaan ketika klien menerima data yang lebih banyak daripada yang dibutuhkan [1].

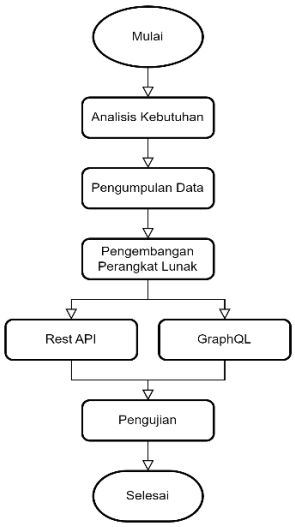
Seiringnya perkembangan teknologi, pada tahun 2015, Facebook merancang teknologi alternatif Rest API, yaitu GraphQL [8]. Sejak saat itu, GraphQL telah menarik perhatian para pengembang [9]. GraphQL memiliki fleksibilitas dalam mengekstraksi data [10], serta memiliki kemampuan memberikan data secara tepat sesuai dengan kebutuhan [11]. Tidak hanya itu, GraphQL dapat mengurangi beban transfer data jika dibandingkan dengan Rest API [12]. Fleksibilitas GraphQL memiliki potensi untuk mengatasi masalah *under-fetching* serta *over-fetching* dalam Rest API [5].

Terdapat beberapa penelitian yang telah mengkaji terkait perbandingan antara GraphQL dan Rest API, hasil penelitian menunjukkan bahwa penggunaan GraphQL menghasilkan performa yang lebih unggul [13], [14], [15], serta mempermudah upaya pengembang dalam mengembangkan perangkat lunak berbasis web [16]. Meskipun demikian, masih terdapat kekosongan informasi terkait efektivitas kecepatan respon dalam mengatasi *under-fetching* dan *over-fetching*, serta analisis kondisi untuk menentukan kapan sebaiknya memilih GraphQL dibandingkan Rest API.

Berdasarkan uraian permasalahan tersebut, penelitian ini memiliki fokus untuk menjawab seberapa efektif kecepatan respon GraphQL dan melakukan analisis kondisi untuk menentukan penggunaan GraphQL. Hasil penelitian ini diharapkan mampu memberikan wawasan bagi para praktisi di bidang teknologi dalam menentukan gaya arsitektur pada pengembangan perangkat lunak berbasis web.

2. METODE PENELITIAN

Pada bagian ini penulis akan menjelaskan alur penelitian, dimulai dengan analisis kebutuhan perangkat lunak dan perangkat keras, kemudian pengumpulan *dataset*, kemudian dilakukan pengembangan perangkat lunak dengan gaya arsitektur Rest API dan GraphQL dan terakhir terdapat tahapan pengujian pada perangkat lunak yang telah dirancang, seperti yang ditunjukkan pada Gambar 1 di bawah ini.



Gambar 1. Hasil Pengujian *Under-fetching* Skenario Pertama

2.1. Analisis Kebutuhan

Pada tahap ini, memiliki tujuan untuk memperoleh spesifikasi yang diperlukan [17]. Pada Tabel 1 terdapat rincian daftar perangkat lunak yang dibutuhkan, sedangkan pada Tabel 2 terdapat rincian spesifikasi perangkat keras yang digunakan dalam penelitian ini.

Tabel 1. Daftar Perangkat Lunak

Nama	Keterangan
Apache JMeter	Aplikasi yang dibutuhkan untuk menjalankan pengujian
MongoDB	Database yang dibutuhkan untuk menyimpan dataset
Visual Studio Code	Dibutuhkan untuk membuat kode program perangkat lunak

Tabel 2. Spesifikasi Perangkat Keras

Nama	Keterangan
Processor	AMD Ryzen 5 3500U
RAM	8 GB
SSD	512 GB
Graphic	Radeon Vega Mobile Gfx
Operating System	Windows 11

2.2. Metode Pengumpulan Data

Pada pengumpulan data, penelitian ini menggunakan *dummy dataset* yang disimpan pada database MongoDB. Awal mulanya hanya terdapat *dummy dataset* 1 tabel lalu dilakukan duplikasi menjadi 6 tabel untuk menyesuaikan kebutuhan skenario pengujian. Setiap tabel memiliki 5000 jumlah data dengan tipe data *long-text.*

2.3. Tahap Pengembangan Perangkat Lunak

Pada pengumpulan data, dilakukan pengembangan perangkat lunak berbasis web dengan dua gaya arsitektur berbeda yakni Rest API dan GraphQL. Kedua perangkat lunak tersebut dibangun dengan menggunakan *library* Express JS dan database MongoDB. Setiap perangkat lunak

memiliki fitur yang identik untuk mendapatkan data sesuai dengan skenario pengujian.

2.4. Tahap Pengujian

Pengujian dilaksanakan untuk memperoleh data performa kecepatan respon dari setiap perangkat lunak. Kecepatan respon didapatkan dengan satuan milidetik dan bisa juga disebut dengan latensi. Semakin rendah kecepatan respon, semakin cepat performa kecepatan respon perangkat lunak tersebut. Pengujian akan dilakukan masing-masing 5 skenario dalam menangani *under-fetching* serta *over-fetching*. Pada Tabel 3 dan Tabel 4 terdapat rincian lengkap mengenai skenario pengujian.

Tabel 3. Skenario Pengujian *Under-fetching*

Nama	Detail
Skenario Pertama	Pengujian dilakukan dengan memanggil 2 *endpoint* berbeda untuk mendapatkan data yang dibutuhkan
Skenario Kedua	Pengujian dilakukan dengan memanggil 3 *endpoint* berbeda untuk mendapatkan data yang dibutuhkan
Skenario Ketiga	Pengujian dilakukan dengan memanggil 4 *endpoint* berbeda untuk mendapatkan data yang dibutuhkan
Skenario Keempat	Pengujian dilakukan dengan memanggil 5 *endpoint* berbeda untuk mendapatkan data yang dibutuhkan
Skenario Kelima	Pengujian dilakukan dengan memanggil 6 *endpoint* berbeda untuk mendapatkan data yang dibutuhkan

Tabel 4. Skenario Pengujian *Over-fetching*

Nama	Detail
Skenario Pertama	Pengujian dilakukan dengan memanggil 1 *endpoint* untuk mendapatkan 2 atribut data yang dibutuhkan sedangkan *server* memberikan dua puluh atribut data
Skenario Kedua	Pengujian dilakukan dengan memanggil 1 *endpoint* untuk mendapatkan 4 atribut data yang dibutuhkan sedangkan *server* memberikan dua puluh atribut data
Skenario Ketiga	Pengujian dilakukan dengan memanggil 1 *endpoint* untuk mendapatkan 8 atribut data yang dibutuhkan sedangkan *server* memberikan dua puluh atribut data
Skenario Keempat	Pengujian dilakukan dengan memanggil 1 *endpoint* untuk mendapatkan 12 atribut data yang dibutuhkan sedangkan *server* memberikan dua puluh atribut data
Skenario Kelima	Pengujian dilakukan dengan memanggil 1 *endpoint* untuk mendapatkan 16 atribut data yang dibutuhkan sedangkan *server* memberikan dua puluh atribut data

Untuk mendapatkan hasil pengujian yang maksimal, dilakukan sejumlah 100 iterasi pada masing-masing skenario pengujian. Adapun rincian konfigurasi pengujian pada perangkat lunak Apache JMeter yang dapat dilihat pada Tabel 5.

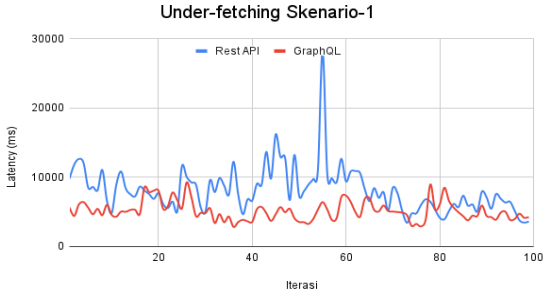
Tabel 5. Konfigurasi Apache JMeter

Nama	Nilai	Keterangan
Number of Thread	1	Jumlah pengguna *virtual* yang akan disimulasikan selama pengujian
Ramp-up Period	50	Waktu yang diperlukan untuk meningkatkan jumlah pengguna hingga nilai Number of Thread
Loop Count	100	Jumlah iterasi setiap pengguna selama pengujian

3. HASIL DAN PEMBAHASAN

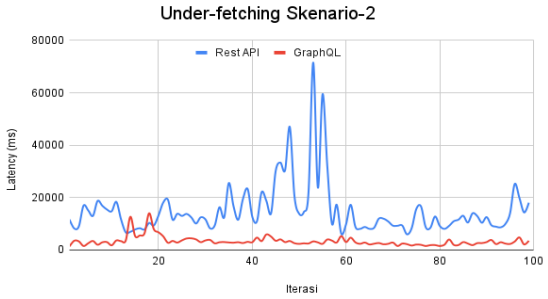
3.1. Hasil Pengujian *Under-fetching*

Berikut adalah hasil pengujian *under-fetching* pada perangkat lunak dengan gaya Rest API serta GraphQL, disajikan dalam bentuk grafik yang dapat dilihat pada Gambar 2-7.



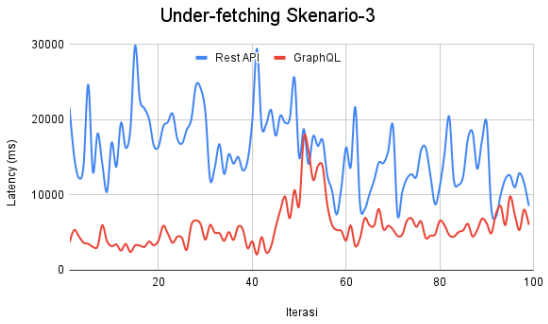
Gambar 2. Hasil Pengujian *Under-fetching* Skenario Pertama

Pada Gambar 2 terdapat hasil pengujian skenario-1, terlihat bahwa GraphQL memiliki latensi yang lebih rendah dibandingkan dengan Rest API. Meskipun perbedaannya tidak terlalu signifikan, terdapat anomali ketika iterasi ke-58, terlihat bahwa latensi Rest API meningkat secara drastis.



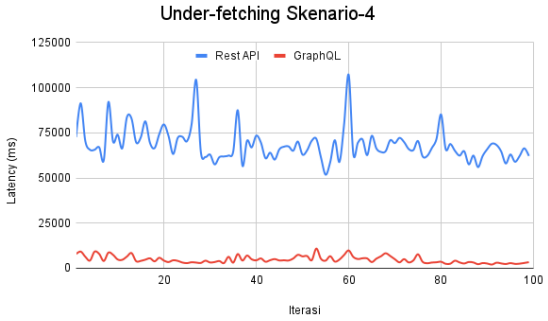
Gambar 3. Hasil Pengujian *Under-fetching* Skenario Kedua

Pada Gambar 3 terdapat hasil pengujian skenario-2, terlihat bahwa GraphQL menunjukkan latensi yang lebih rendah dibandingkan dengan Rest API. Namun sama seperti hasil skenario-1, pada skenario-2 terdapat anomali ketika iterasi 50 – 60, pada rentang tersebut terlihat bahwa latensi Rest API meningkat secara drastis namun iterasi setelahnya kembali ke angka normal.



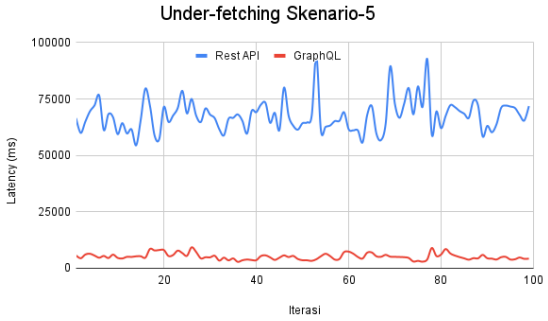
Gambar 4. Hasil Pengujian *Under-fetching* Skenario Ketiga

Hasil pengujian skenario-3 terlihat jelas bahwa terdapat perbedaan latensi yang cukup signifikan ketika iterasi awal yang dapat dilihat pada Gambar 4. Namun pada iterasi ke-50 latensi GraphQL meningkat secara drastis, setelahnya kembali normal dan meningkat seiringnya iterasi. Rest API memiliki latensi yang cukup tinggi dari awal iterasi hingga akhir.



Gambar 5. Hasil Pengujian *Under-fetching* Skenario Keempat

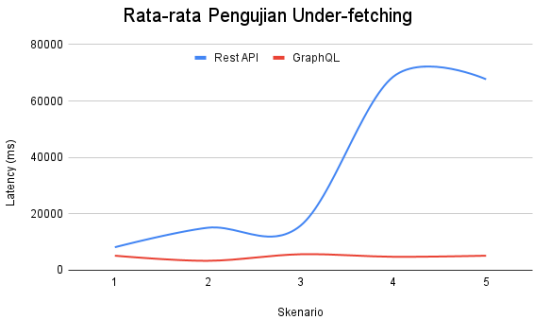
Pada hasil pengujian skenario-4 terlihat bahwa latensi GraphQL dengan Rest API cukup berbeda. GraphQL memiliki latensi yang lebih rendah dan cukup konsisten pada pengujian skenario-4. Rest API memiliki latensi yang lebih tinggi dan terdapat peningkatan secara drastis pada iterasi 30 dan 60.



Gambar 6. Hasil Pengujian *Under-fetching* Skenario Kelima

Pada hasil pengujian terakhir, terlihat bahwa GraphQL dengan Rest API memiliki perbedaan latensi yang sangat jauh. GraphQL memiliki latensi yang cukup konsisten. Sedangkan Rest API terdapat peningkatan latensi secara drastis pada itersi 55, 70, dan 78.

Berdasarkan hasil pengujian *under-fetching* tersebut, didapatkan grafik rata-rata latensi pengujian *under-fetching* yang dapat dilihat pada gambar berikut.



Gambar 7. Hasil Rata-rata Pengujian *Under-fetching*

Hasil setiap skenario pengujian *under-fetching*, Rest API memiliki peningkatan latensi seiring bertambahnya *endpoint* yang dipanggil. Sedangkan GraphQL memiliki latensi yang cukup stabil. Rincian data latensi yang diperoleh dapat dilihat pada Tabel 6 dan Tabel 7.

Tabel 6. Latensi Rest API Pengujian *Under-fetching*

Pengujian	**Latensi (milidetik)**		
	Minimum	**Maksimum**	**Rata-rata**
Skenario Pertama	3429	28319	8165,6
Skenario Kedua	5985	7152	15072,7
Skenario Ketiga	7176	29861	15797,21
Skenario Keempat	51828	107145	68618,17
Skenario Kelima	54463	94096	67696,1

Tabel 7. Latensi GraphQL Pengujian *Under-fetching*

Pengujian	**Latensi (milidetik)**		
	Minimum	**Maksimum**	**Rata-rata**
Skenario Pertama	2835	9270	5156,26
Skenario Kedua	1373	14035	3351,33
Skenario Ketiga	2094	17663	5666,52
Skenario Keempat	2038	10900	4771,75
Skenario Kelima	2835	9270	5156,26

Pada Tabel 6 dan Tabel 7 terlihat bahwa Rest API memiliki rata-rata latensi yang terus meningkat seiringnya skenario pengujian. Sedangkan GraphQL memiliki rata-rata yang cukup konsisten seiringnya skenario pengujian. Dari hasil pengujian tersebut, dapat diolah dan dilakukan analisis terkait perbandingan latensi Rest API dan GraphQL pada pengujian *under-fetching* yang dapat dilihat pada Tabel 8.

Tabel 8. Perbandingan Latensi Rest API dan GraphQL Dalam Pengujian *Under-fetching*

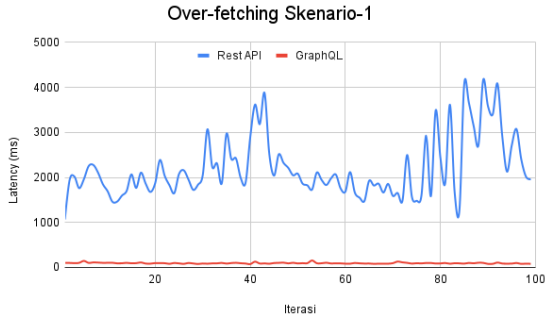
Pengujian	**Perbandingan Rata-rata**		
	Rest API (milidetik)	**GraphQL (milidetik)**	**Perbandingan %**
Skenario Pertama	8165,56	5156,26	36,84
Skenario Kedua	15072,7	3351,33	77,71

Skenario Ketiga	15797,21	5666,52	64,16
Skenario Keempat	68618,17	4771,75	93,04
Skenario Kelima	67696,1	5156,26	92,39

Pada Tabel 8 terdapat hasil analisis yang menunjukan bahwa GraphQL memberikan latensi yang lebih cepat dalam mengatasi masalah *under-fetching*. Pada skenario pertama Rest API memiliki perbandingan yang paling sedikit dan terus meningkat seiringnya skenario pengujian. Terdapat perbandingan yang sangat tinggi pada skenario keempat dan skenario kelima, GraphQL mampu meningkatkan kecepatan performa latensi hampir 100% dibandingkan dengan Rest API.

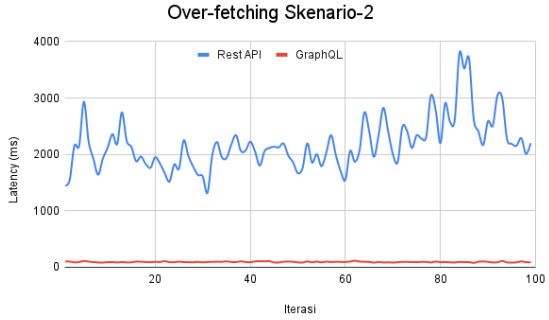
3.1. Hasil Pengujian *Over-fetching*

Berikut adalah hasil pengujian over-fetching pada aplikasi Rest API dan GraphQL disajikan dalam bentuk grafik dapat dilihat pada Gambar 8-13.



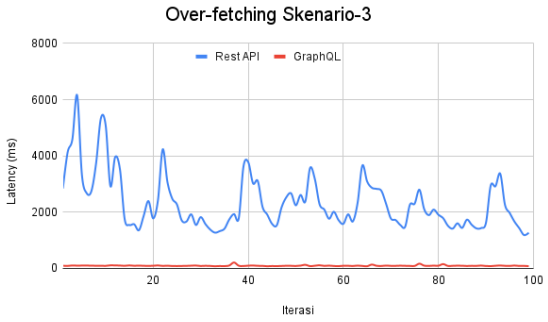
Gambar 8. Hasil Pengujian *Over-fetching* Skenario Pertama

Hasil pengujian skenario pertama terlihat bahwa GraphQL memiliki latensi yang sangat cepat dibandingkan dengan Rest API. GraphQL memiliki latensi yang konsisten. Sedangkan Rest API memiliki latensi yang beragam, sempat memiliki penurunan pada iterasi ke-50 namun meningkat lagi pada iterasi ke-80.



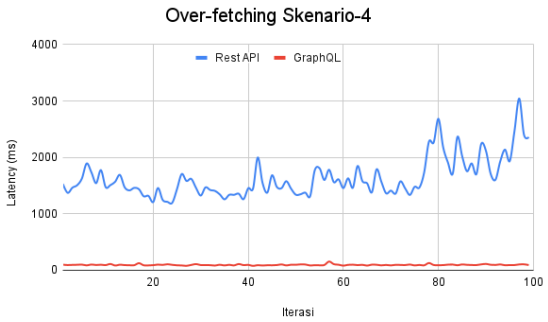
Gambar 9. Hasil Pengujian *Over-fetching* Skenario Kedua

Hasil pengujian skenario kedua, GraphQL memiliki latensi yang cukup cepat dan konsisten dibandingkan Rest API. Pada Rest API memiliki latensi yang semakin tinggi seiring bertambahnya iterasi pengujian.



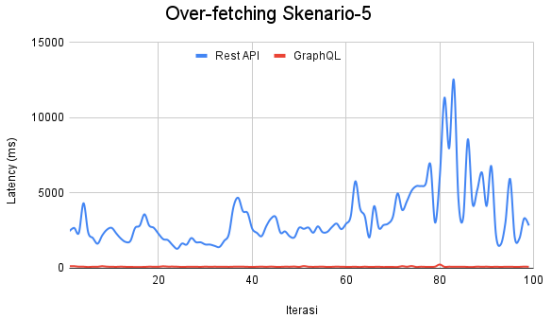
Gambar 10. Hasil Pengujian *Over-fetching* Skenario Ketiga

Pada hasil pengujian skenario ketiga, GraphQL juga mampu memiliki latensi yang cukup cepat dan konsisten. Latensi Rest API pada pengujian ini memiliki penurunan setiap 20 iterasi.



Gambar 11. Hasil Pengujian *Over-fetching* Skenario Keempat

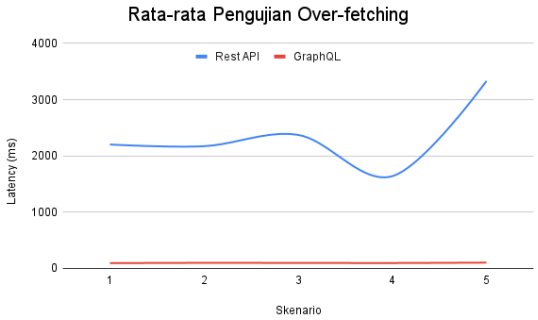
Hasil pengujian skenario keempat juga terlihat bahwa GraphQL memiliki latensi yang cukup cepat dibandingkan dengan Rest API. Rest API memiliki latensi yang semakin tinggi seiring bertambahnya iterasi pengujian.



Gambar 12. Hasil Pengujian *Over-fetching* Skenario Kelima

Hasil pengujian skenario kelima menunjukan bahwa GraphQL memiliki latensi yang rendah dan konsisten. Rest API pada awal iterasi memiliki latensi yang cukup rendah namun meningkat secara drastis pada iterasi ke-80.

Berdasarkan hasil tersebut, didapatkan grafik rata-rata pengujian *over-fetching* yang dapat dilihat pada gambar berikut.



Gambar 13. Hasil Rata-rata Pengujian *Over-fetching*

Hasil setiap skenario pengujian *over-fetching*, Rest API sempat memiliki penurunan latensi pada skenario-4 namun tetap memiliki selisih yang sangat jauh dibandingkan dengan GraphQL. GraphQL memiliki latensi yang cukup stabil. Rincian data latensi yang diperoleh dapat dilihat pada Tabel 9 dan Tabel 10.

Tabel 9. Latensi Rest API Pengujian *Over-fetching*

Pengujian	Latensi (milidetik)		
	Minimum	Maksimum	Rata-rata
Skenario Pertama	1066	4168	2206,49
Skenario Kedua	1319	3795	2172,85
Skenario Ketiga	1186	6155	2372,32
Skenario Keempat	1195	3043	1640,09
Skenario Kelima	1292	12506	3331,43

Tabel 10. Latensi GraphQL Pengujian *Over-fetching*

Pengujian	Latensi (milidetik)		
	Minimum	Maksimum	Rata-rata
Skenario Pertama	74	706	100,58
Skenario Kedua	78	650	99,34
Skenario Ketiga	78	218	98,02
Skenario Keempat	78	228	96,46
Skenario Kelima	83	250	103,53

Pada Tabel 9 dan Tabel 10 terlihat bahwa Rest API dan GraphQL memiliki rata-rata yang cukup konsisten seiringnya skenario pengujian. Selanjutnya, dari data tersebut dapat diolah dan dilakukan analisis terkait perbandingan latensi pada pengujian *over-fetching* yang dapat dilihat pada Tabel 11.

Tabel 11. Perbandingan Latensi Rest API dan GraphQL Dalam Pengujian *Over-fetching*

Pengujian	Perbandingan Rata-rata		
	Rest API (milidetik)	GraphQL (milidetik)	Perbandingan %
Skenario Pertama	2206,49	100,58	95,43
Skenario Kedua	2172,85	99,34	95,43
Skenario Ketiga	2372,32	98,02	95,88
Skenario Keempat	1640,09	96,46	94,12
Skenario Kelima	3331,43	103,53	96,89

Pada Tabel 11 terdapat hasil analisis yang menunjukan bahwa GraphQL memberikan latensi yang lebih cepat dalam mengatasi masalah *over-fetching*. Keduanya memiliki rata-rata yang cukup konsisten sehingga memiliki perbandingan yang stabil. Meski GraphQL memiliki latensi yang sangat rendah, Rest API masih tetap memadai dengan rentang rata-rata antara 2000 ms hingga 3000 ms.

4. DISKUSI

Penelitian ini telah menjawab dari rumusan masalah yang telah disampaikan sebelumnya. Berdasarkan hasil pengujian secara objektif, GraphQL telah memberikan hasil yang positif dalam menangani masalah *under-fetching* dan *over-fetcing*. Serta penelitian ini juga dapat menjadi acuan bagi pengembang untuk menentukan gaya arsitektur saat pengembangan perangkat lunak berbasis web.

Pada penelitian terdahulu, seperti pada penelitian [13], dilakukan pengujian dengan menggunakan ASP. Net Core sebagai framework backend. Pada penelitan tersebut, GraphQL didapatkan data kecepatan respon rata-rata 58,9 ms, sedangkan Rest API mendapatkan data kecepatan respon rata-rata 4167,13 ms. Hal ini menunjukkan bahwa GraphQL memberikan kecepatan respon yang lebih unggul. Begitu juga pada penelitian [14], dengan studi kasus sistem informasi pelacakan alumni Politeknik Negeri Malang didapatkan hasil kecepatan respon rata-rata GraphQL yakni 944,5 ms serta Rest API memiliki kecepatan respon rata-rata 1.398 ms.

Selanjutnya pada penelitian [15], didapatkan hasil bahwa GraphQL mampu mengurangi ukuran dokumen JSON sebesar 94% hingga 99%. Berbeda dengan penelitian [13], [14], [15], pada penelitian [16], dilakukan penelitian dari segi pengalaman pengembang dalam mengembangkan perangkat lunak dengan gaya Rest API dan GraphQL. Pada penelitian tersebut, didapatkan hasil bahwa kueri dengan banyak parameter lebih sulit diimplementasikan pada Rest API dibandingkan GraphQL.

Berdasarkan hasil dan tujuan, penelitian ini memiliki perbedaan dengan penelitian terdahulu. Penelitian ini telah berkontribusi mengisi kekosongan informasi pada penelitian sebelumnya, yakni terkait efektivitas kecepatan respon serta analisis kondisi dalam penggunaan GraphQL.

5. KESIMPULAN

Berdasarkan hasil dan analisis yang telah dilakukan, dapat disimpulkan bahwa GraphQL efektif dalam menangani masalah *under-fetching* dan *over-fetching*. Ditemukan bahwa pada kasus *under-fetching*, perbandingan kinerja bervariasi antara 36.84% hingga 93.04%. Sebaliknya, pada masalah *over-fetching*, perbandingan berkisar antara 94.12% hingga 96.89%. Pada permasalahan *under-*

fetching, penggunaan GraphQL disarankan apabila terdapat kebutuhan pemanggilan lebih dari 4 *endpoint*. Sementara itu, dalam penanganan *over-fetching*, penggunaan Rest API memberikan kecepatan respon yang memadai. Namun, apabila diperlukan kecepetan yang lebih optimal, penggunaan GraphQL dapat menjadi pilihan yang lebih efisien.

DAFTAR PUSTAKA

[1] P. Roksela, M. Konieczny, and S. Zielinski, "Evaluating execution strategies of GraphQL queries," in *2020 43rd International Conference on Telecommunications and Signal Processing (TSP)*, Milan, Italy: IEEE, Jul. 2020, pp. 640–644. doi: 10.1109/TSP49548.2020.9163501.

[2] A. Lawi, B. L. E. Panggabean, and T. Yoshida, "Evaluating GraphQL and REST API Services Performance in a Massive and Intensive Accessible Information System," *Computers*, vol. 10, no. 11, p. 138, Oct. 2021, doi: 10.3390/computers10110138.

[3] I. Kurniawan, Humaira, and F. Rozi, "REST API Menggunakan NodeJS pada Aplikasi Transaksi Jasa Elektronik Berbasis Android," *jitsi*, vol. 1, no. 4, pp. 127–132, Dec. 2020, doi: 10.30630/jitsi.1.4.18.

[4] A. Neumann, N. Laranjeiro, and J. Bernardino, "An Analysis of Public REST Web Service APIs," *IEEE Trans. Serv. Comput.*, vol. 14, no. 4, pp. 957–970, Jul. 2021, doi: 10.1109/TSC.2018.2847344.

[5] S. K. Mukhiya, F. Rabbi, V. K. I Pun, A. Rutle, and Y. Lamo, "A GraphQL approach to Healthcare Information Exchange with HL7 FHIR," *Procedia Computer Science*, vol. 160, pp. 338–345, 2019, doi: 10.1016/j.procs.2019.11.082.

[6] A. Quiña-Mera, P. Fernandez, J. M. García, and A. Ruiz-Cortés, "GraphQL: A Systematic Mapping Study," *ACM Comput. Surv.*, vol. 55, no. 10, pp. 1–35, Oct. 2023, doi: 10.1145/3561818.

[7] M. Vogel, S. Weber, and C. Zirpins, "Experiences on Migrating RESTful Web Services to GraphQL," in *Service-Oriented Computing – ICSOC 2017 Workshops*, vol. 10797, L. Braubach, J. M. Murillo, N. Kaviani, M. Lama, L. Burgueño, N. Moha, and M. Oriol, Eds., in Lecture Notes in Computer Science, vol. 10797. , Cham: Springer International Publishing, 2018, pp. 283–295. doi: 10.1007/978-3-319-91764-1_23.

[8] Facebook, "GraphQL." Accessed: Dec. 12, 2023. [Online]. Available: https://spec.graphql.org/October2021/

[9] R. Taelman, M. V. Sande, and R. Verborgh, "GraphQL-LD: Linked Data Querying with GraphQL," *Proceedings of the 17th International Semantic Web Conference: Posters and Demos*, 2018.

[10] D. H. Tinambunan, A. Baehaqi, R. P. Avrianto, and R. E. Indrajit, "MICROGEN IMPLEMENTATION FOR BUILDING ONLINE LEARNING MANAGEMENT SYSTEM WITH MICROSERVICES AND GRAPHQL GENERATOR APPROACH," *Jurnal Teknik Informatika (Jutif)*, vol. 4, no. 4, pp. 967–976, Aug. 2023.

[11] O. Hartig and J. Pérez, "Semantics and Complexity of GraphQL," in *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*, Lyon, France: ACM Press, 2018, pp. 1155–1164. doi: 10.1145/3178876.3186014.

[12] M. Bryant, "GraphQL for archival metadata: An overview of the EHRI GraphQL API," in *2017 IEEE International Conference on Big Data (Big Data)*, Boston, MA: IEEE, Dec. 2017, pp. 2225–2230. doi: 10.1109/BigData.2017.8258173.

[13] F. Hanif, I. Ahmad, and D. Darwis, "ANALISA PERBANDINGAN METODE GRAPHQL API DAN REST API DENGAN MENGGUNAKAN ASP.NET CORE WEB API FRAMEWORK," vol. 3, no. 2, 2022.

[14] A. T. Firdausi, D. S. Hormansyah, and F. Ervansyah, "IMPLEMENTASI GRAPHQL UNTUK MENGATASI UNDER-FETCHING PADA PENGEMBANGAN SISTEM INFORMASI PELACAKAN ALUMNI POLITEKNIK NEGERI MALANG," vol. 7, 2021.

[15] G. Brito, T. Mombach, and M. T. Valente, "Migrating to GraphQL: A Practical Assessment," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Hangzhou, China: IEEE, Feb. 2019, pp. 140–150. doi: 10.1109/SANER.2019.8667986.

[16] G. Brito and M. T. Valente, "REST vs GraphQL: A Controlled Experiment," in *2020 IEEE International Conference on Software Architecture (ICSA)*, Salvador, Brazil: IEEE, Mar. 2020, pp. 81–91. doi: 10.1109/ICSA47634.2020.00016.

[17] K. S. Alim, N. A. Ekowati, R. Y. Kisworini, and L. Riyandari, "DESIGN AND DEVELOPMENT OF WEB-BASED APPLICATION CANGKINGAN USING SCRUM METHOD," *J. Tek. Inform. (JUTIF)*, vol. 4, no. 4, pp. 953–965, Aug. 2023, doi: 10.52436/1.jutif.2023.4.4.1311.