TeknoIS: Jurnal Ilmiah Teknologi Informasi dan Sains
Volume 16 Number 1, January 2026 Page. 26-36
Journal Homepage: https://teknois.unbin.ac.id/index.php/JBS/
Received: November 2025; Accepted: December 2025; Published: January 2026

Analisis Perbandingan Kinerja *REST API* dan *GraphQL API* menggunakan JavaScript dalam Pengambilan dan Pengolahan Data untuk Aplikasi Web Modern

Moch. Miftachul Huda[1*], Kemal Farouq Mauladi[2], M. Hasan Wahyudi[3]

[1,2,3]Fakultas Sains dan Teknologi/Teknik Informatika/Universitas Islam Lamongan
Email[1]: miftachulhd5@gmail.com
Email[2]: kemalfarouq@unisla.ac.id
Email[3]: hasanwahyudi@unisla.ac.id
*) *Corresponding Author*

ABSTRACT

Modern web applications are interactive and dynamic, requiring real-time data management. API architecture plays a key role in communication between the frontend and backend, with two popular approaches. REST API and GraphQL API. This study compares their performance using JavaScript (BunJS runtime) and PostgreSQL. Tests include GET, POST, PUT, and DELETE operations with Load Test, Spike Test, and Stress Test methods using K6. Metrics measured include average response time, requests per second (RPS), error rate, success rate, and responses under 2000 ms. Results show REST API excels in large-scale data retrieval (GET) with simple queries, while GraphQL is more stable and faster for data mutation operations (POST, PUT, DELETE). GraphQL also offers greater flexibility for specific data retrieval but experiences performance drops under heavy loads. The choice depends on applications needs. REST suits massive data retrieval, while GraphQL is better for complex data manipulation.

Keywords: *REST API, GraphQL API, JavaScript, K6*

ABSTRAK

Aplikasi web masa kini tidak hanya bersifat interaktif dan dinamis, memerlukan manajemen data real-time. Arsitektur API menjadi kunci komunikasi antara frontend dan backend, dengan dua pendekatan populer REST API dan GraphQL API. Penelitian ini membandingkan kinerja keduanya menggunakan JavaScript (runtime BunJS) dan PostgreSQL. Pengujian mencakup operasi GET, POST, PUT, DELETE dengan metode Load Test, Spike Test, dan Stress Test menggunakan K6. Metrik yang diukur meliputi average response time, request per second (RPS), error rate, persentase keberhasilan, dan respon <2000 ms. Hasil menunjukkan REST API unggul dalam pengambilan data besar (GET) dengan query sederhana, sedangkan GraphQL lebih stabil dan cepat untuk operasi mutase data (POST, PUT, DELETE). GraphQL juga lebih fleksibel untuk pengambilan data spesifik, namun performanya menurun saat beban meningkat. Pemilihan keduanya tergantung pada kebutuhan. REST cocok untuk data massif, GraphQL untuk manipulasi data kompleks.

Kata Kunci: REST API, GraphQL API, JavaScript, K6

A. PENDAHULUAN

Perkembangan teknologi web modern telah mendorong transformasi aplikasi dari *platform* informasi statis atau bisa disebut web konvensional menjadi sistem yang dinamis dan interaktif. Aplikasi web saat ini, seperti media sosial, *dashboard* analitik, dan layanan berbasis *real-time*, membutuhkan sistem komunikasi data yang cepat, efisien, dan mampu menangani permintaan dalam jumlah besar [1]. *API* merupakan sekumpulan protokol dan definisi untuk membuat data dan fungsionalitas aplikasi atai sistem agar tersedia bagi pengembang pihak ketiga eksternal, ataupun dalam organisasi internal sendiri [2]. Banyak aplikasi web mengalami kendala seperti lambatnya respon saat memuat data besar, permintaan data tidak efisien, serta tingginya beban server karena

permintaan yang kompleks. Untuk mendukung kebutuhan atas permasalahan tersebut, arsitektur *Application Programming Interface* (*API*) memainkan peran penting dalam menjembatani pertukaran data antara sisi klien (*frontend*) dan server (*backend*) [3]. Masalah-masalah tersebut menimbulkan kebimbangan dalam pemilihan arsitektur *API* yang paling tepat dan efisien.

Dua pendekatan *API* yang paling umum digunakan adalah *REST* (*Representational State Transfer*) dan *GraphQL*. *REST API* memiliki keunggulan dalam kesederhanaan dan struktur *endpoint* yang eksplisit, tetapi kerap mengalami kendala seperti *over-fetching* (pengambilan data berlebih) atau *under-fetching* (pengambilan data kurang) karena keterbatasan fleksibilitas. Sebaliknya, *GraphQL API* dikembangkan oleh *Facebook* pada tahun 2015, memungkinkan klien menentukan secara spesifik data apa yang dibutuhkan dalam satu *query*, sehingga meningkatkan efisiensi komunikasi data [4]. Bisa dikatakan *GraphQL* hadir dengan tujuan memberikan optimasi yang lebih daripada *REST API.*

Menguji *REST API* merupakan sebuah tantangan karena ukuran ruang pencarian yang besar untuk dijelajahi, banyaknya operasi, potensi perintah, ketergantungan antar parameter, dan batasan nilai parameter masukan yang terkait [5]. *GraphQL* bekerja dengan model *schema*, yang mendefinisikan tipe data dan hubungan antar data, serta menyediakan efisiensi tinggi dalam pengambilan data untuk aplikasi modern [6].

Beberapa penelitian terdahulu telah mengkaji perbandingan performa antara *REST* dan *GraphQL*. Dari penelitian Permana dengan judul "Analisis Perbandingan Metode *GraphQL* Dan Metode *REST API* Pada Teknologi *NodeJS*" menilai bahwa *GraphQL* lebih unggul dalam fleksibilitas dan efisiensi permintaan data dinamis, sementara *REST* lebih stabil untuk desain *URL* dan struktur data tetap [7].

Penelitian Arman Lawi dan temannya dengan judul "*Evaluating GraphQL and REST API Services Performance in a Massive and Intensive Accessible Information System*" menilai bahwa dari segi kecepatan *REST* lebih unggul sebesar 51% dalam respon waktu dan 37% dalam *throughput* [8].

Kemudian penelitian oleh Derry Dwi Aditya Hendarto yang berjudul "Analisis Perbandingan Arsitektur *REST* dan *GraphQL* untuk Aplikasi Pengenalan Emosi pada Pembelajaran Daring Sinkronis" menyimpulkan bahwa performa *backend* dengan arsitektur *REST* lebih unggul. *Response time REST* lebih efisien sebesar 4,42%, serta *throughput REST* lebih efisien sebesar 9,31% [9].

Terbaru, dari penelitian Muntahanah, Yulia Darmi, dan Keken Pinandita dengan judul "Implementasi Perbandingan Metode *GraphQL* Dan *REST API* Pada Teknologi *NodeJS*" menegaskan bahwa *GraphQL* mampu meminimalkan data yang tidak dibutuhkan dengan fleksibilitas struktur *query*, meskipun waktu responnya lebih panjang dibandingkan *REST* [10].

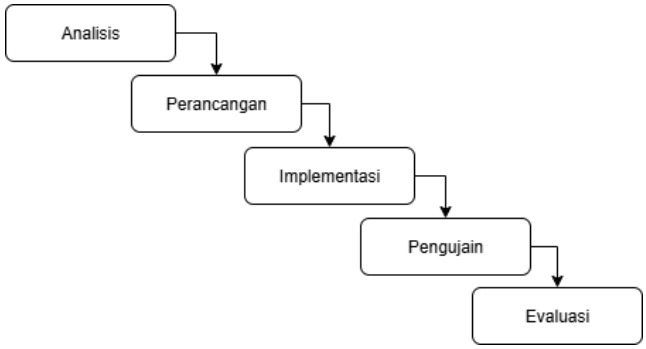
Walaupun telah banyak studi yang membandingkan kedua arsitektur ini, sebagian besar menggunakan pendekatan pengujian yang berbeda-beda, baik dari sisi *runtime*, jenis dataset, maupun beban pengujiannya. Oleh karena itu, penelitian ini hadir untuk memberikan kontribusi melalui pendekatan pengujian yang konsisten dan terstandar menggunakan *runtime JavaScript* (*BunJS*), basis data *PostgreSQL*, serta pengujian performa menggunakan *K6*. Fokus utama adalah membandingkan kinerja *REST API* dan *GraphQL API* dalam menangani operasi *GET, POST, PUT,* dan *DELETE*, menggunakan skenario *load test, spike test*, dan *stress test*.

Penelitian ini diharapkan dapat memberikan gambaran objektif dan praktis bagi pengembang, akademisi, maupun industri dalam menentukan pilihan arsitektur *API* yang paling sesuai berdasarkan konteks kebutuhan, kompleksitas data, dan skala penggunaan aplikasi web modern.

B. METODE

TeknoIS: Jurnal Ilmiah Teknologi Informasi dan Sains
Volume 16 Number 1, January 2026 Page. 26-36
Journal Homepage: https://teknois.unbin.ac.id/index.php/JBS/
Received: November 2025; Accepted: December 2025; Published: January 2026

Penelitian ini dilakukan dengan pendekatan eksperimental untuk membandingkan performa *REST API* dan *GraphQL API* dalam pengambilan serta manipulasi pada aplikasi web. Proses pengembangan sistem dalam penelitian ini mengacu pada model rekayasa perangkat lunak *waterfall*, yang terdiri dari tahapan berurutan yaitu analisis kebutuhan, perancangan sistem, implementasi, pengujian, dan evaluasi. Model ini dipilih karena kesederhanaannya dan kemampuannya untuk mendukung proses eksperimen yang terstruktur dan terukur.



Gambar 1. *Waterfall*

API dikembangkan menggunakan bahasa pemrograman *JavaScript* dengan *runtime BunJS*, serta basis data *PostgreSQL* sebagai sistem manajemen data. Kedua arsitektur *API* (*REST* dan *GraphQL*) dirancang dengan struktur *endpoint*, skema data, dan fungsi yang identik agar perbandingan performa dapat dilakukan secara adil dan objektif.

Dataset yang digunakan diambil dari *website Kaggle* yang berisi data produk kosmetik dengan total data 11338 data [11], yang akan digunakan dalam pengujian sebanyak 10000 data. Terdapat 16 kolom atribut yaitu *id, product_name, website, country, category, subcategory, title_href, price, brand, ingredients, form, type, color, size, rating*, dan *noofratings*. Dataset ini dirancang untuk merepresentasikan volume data pada aplikasi web berskala menengah, serta memungkinkan pengujian yang mencerminkan kondisi beban nyata.

Pengujian performa dilakukan dengan alat *K6* yang merupakan sebuah open-source load test yang membuat *load test* menjadi lebih mudah. Dengan menggunakan *K6*, dapat menguji keandalan dan kinerja sistem, dan mengetahui regresi dan masalah kinerja lebih awal. *K6* akan membantu membangun aplikasi yang tangguh dan berperforma tinggi yang dapat diskalakan [12]. *K6* mendukung skrip *load testing* berbasis *JavaScript*. Tiga skenario pengujian digunakan untuk mengukur ketahanan dan efisiensi sistem:

1. *Load test* untuk mengamati performa saat permintaan meningkat secara bertahap,
2. *Spike test* untuk mengevaluasi respon terhadap lonjakan permintaan mendadak.
3. *Stress test* untuk mengetahui batas maksimum sistem sebelum terjadi penurunan performa.

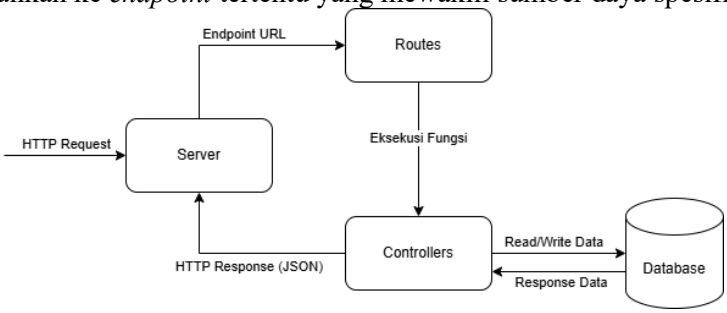
Pengujian dilakukan terhadap empat operasi utama *API*, yaitu *GET, POST, PUT, DELETE*, dengan lima matrik evaluasi: rata-rata waktu respon (*average response time*), permintaan per detik (*requests per second/RPS*), tingkat kesalahan (*error rate*), persentase keberhasilan (*checked passed*), serta persentase respon yang diterima di bawah 2000 ms. Seluruh pengujian dijalankan dalam lingkungan lokal dengan konfigurasi perangkat keras yang konsisten dan setiap skenario diuji sebanyak tiga kali untuk memperoleh nilai rata-rata yang lebih stabil dan representatif.

1. DESAIN ARSITEKTUR *REST API*

Gambar 2 memperlihatkan rancangan arsitektur *REST API* yang diimplementasikan pada penelitian ini, menekankan sifat *stateless* dan pemisahan tanggung jawab (*routes-controllers*-akses data) sehingga setiap sumber daya terekspos melalui *endpoint* spesifik dan pertukaran data dilakukan melalui *HTTP* dengan *payload JSON*.

TeknoIS: Jurnal Ilmiah Teknologi Informasi dan Sains
Volume 16 Number 1, January 2026 Page. 26-36
Journal Homepage: https://teknois.unbin.ac.id/index.php/JBS/
Received: November 2025; Accepted: December 2025; Published: January 2026

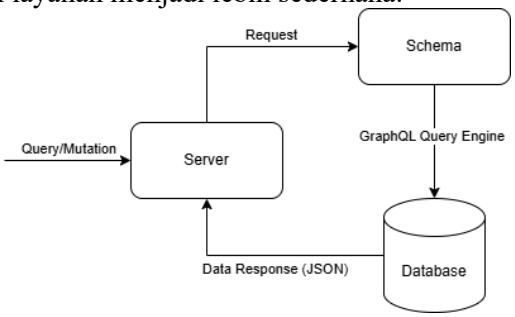
Klien mengirimkan permintaan ke server melalui metode *HTTP request*, yang diarahkan ke *routes* untuk menentukan *endpoint* yang sesuai, kemudian diproses pada *controllers*, yang berkomunikasi dengan *database* untuk membaca atau menulis data. Setelah diproses, server akan mengembalikan *HTTP response* berupa data dalam format *JSON* kepada klien. *REST API* memiliki struktur *endpoint* yang terpisah untuk setiap operasi dan entitas data. Setiap permintaan diarahkan ke *endpoint* tertentu yang mewakili sumber daya spesifik.



Gambar 2. Alur Sistem *REST API*

2. DESAIN ARSITEKTUR *GRAPHQL API*

Gambar 3 memperlihatkan struktur *GraphQL* berbasis *schema* pada satu *endpoint.* Pendekatan deklaratif ini memisahkan kebutuhan data klien dari perakitan di server, sehingga agregasi lintas sumber dan evolusi layanan menjadi lebih sederhana.



Gambar 3. Alur Sistem *GraphQL API*

Klien mengirimkan *query* atau *mutation* ke server, memvalidasi permintaan tersebut menggunakan *schema* yang telah didefinisikan. Diteruskan menggunakan *GraphQL Query Engine* untuk mengakses ke *database* yang kemudian menghasilkan *response* berupa data *JSON* dan dikirimkan ke klien. Alur ini memastikan pengambilan data yang efisien, hanya memberikan data yang diperlukan sesuai dengan permintaan saja.

Dengan menyertakan kedua model arsitektur, perbandingan tidak hanya dilakukan dari sisi performa, tetapi juga dari struktur desain teknisnya. Hal ini penting sebagai dasar justifikasi dalam memilih pendekatan *API* yang sesuai dengan konteks dan kebutuhan sistem.

C. HASIL DAN PEMBAHASAN

1. HASIL

Pengujian dilakukan terhadap empat operasi utama pada *REST API* dan *GraphQL API* dengan skenario *load test, spike test*, dan *stress test* menggunakan alat *K6* serta menggunakan beberapa variasi *Virtual Users* (*VU*). Berikut ini hasil performa dari pengujian *REST API.*

Tabel 1. *Load Test REST API* 10000 Data

Metrik	30 VU	40 VU	50 VU
Average Response Time	6,91 s	13,92 s	11,39 s
95th *Percentile* (P95)	7,96 s	22,66 s	11,4 s
Request Per Second (RPS)	3,61 rps	2,53 rps	3,82 rps

TeknoIS: Jurnal Ilmiah Teknologi Informasi dan Sains
Volume 16 Number 1, January 2026 Page. 26-36
Journal Homepage: https://teknois.unbin.ac.id/index.php/JBS/
Received: November 2025; Accepted: December 2025; Published: January 2026

Metrik	30 VU	40 VU	50 VU
Total Request	300	261	446
Error Rate	0%	10,34%	95,29%
Checks Lolos	50,50% (303 dari 600)	45,21% (236 dari 522)	2,35% (21 dari 892)
Response <2000ms	1% (3 dari 300)	0% (2 dari 261)	0% (0 dari 446)

Performa sistem hanya stabil pada 30 VU, di mana semua metrik utama masih dalam batas wajar. Pada 40 VU dan 50 VU, terjadi degradasi tajam, dengan peningkatan *error rate*, penurunan keberhasilan validasi, dan tidak ada *request* yang memenuhi waktu respon < 2000 ms.

Tabel 2. *Spike Test REST API*

Metrik	1k Data	5k Data	10k Data
Average Response Time	1,31 s	10,9 s	10,9 s
95th *Percentile* (P95)	1,83 s	23,45 s	22,87 s
Request Per Second (RPS)	26,26 rps	5,75 rps	4,88 rps
Total Request	1734	377	372
Error Rate	0%	51,98%	80,64%
Checks Lolos	100% (3468 dari 3468)	32,62% (246 dari 754)	15,18% (113 dari 744)
Response <2000ms	100% (1734 dari 1734)	17% (65 dari 377)	11% (41 dari 331)

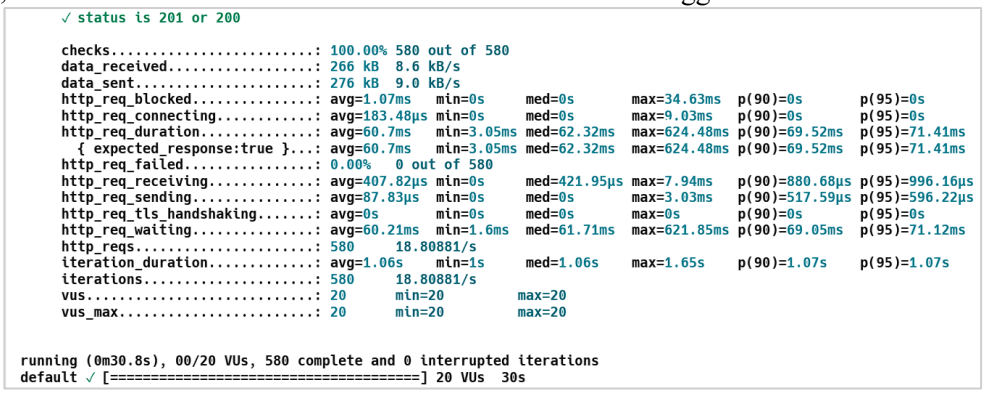
Berdasarkan tabel 2 pengujian *spike test* dengan 1000, 5000, dan 10000 data, terlihat bahwa peningkatan jumlah data secara signifikan menurunkan performa sistem. Pada 1000 data, sistem masih dapat merespon cukup baik meskipun hanya 32,62% permintaan yang berhasil, namun waktu respon mulai melebihi ambang batas. Pada 5000 data, performa semakin menurun dengan tingkat kegagalan mencapai 51,98% dan waktu respon rata-rata meningkat menjadi 10,9 detik. Sementara itu, pada 10000 data, sistem mengalami penurunan drastis dengan tingkat kegagalan mencapai 80,64% dan hanya 15,18% permintaan yang berhasil serta waktu respon rata-rata tetap tinggi di 10,9 detik. Hal ini menunjukkan bahwa sistem tidak mampu menangani lonjakan beban besar secara efektif.

Tabel 3. *Stress Test REST API*

Metrik	1k Data	5k Data	10k Data
Average Response Time	593,77 ms	5,77 s	8,91 s
95th *Percentile* (P95)	1,76 s	20,41 s	21,58 s
Request Per Second (RPS)	25,22 rps	6,46 rps	4,18 rps
Total Request	3794	976	667
Error Rate	0%	3,68%	54,27%
Checks Lolos	100% (7588 dari 7588)	68,59% (1339 dari 1952)	32,38% (432 dari 1334)

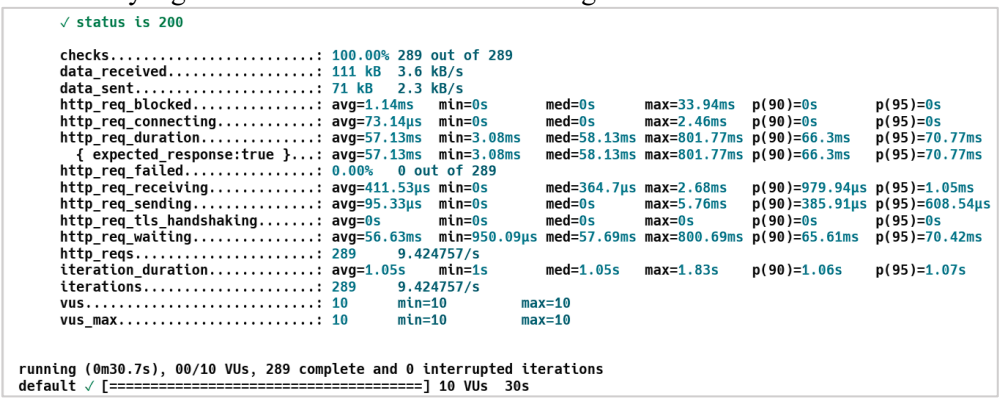
Metrik	1k Data	5k Data	10k Data
Response <2000ms	100% (3794 dari 3794)	40% (399 dari 976)	19% (127 dari 667)

Pengujian *stress test* menunjukkan bahwa semakin tinggi jumlah data (1000, 5000, hingga 10000) performa sistem menurun. Waktu respon rata-rata meningkat dari 5,77 detik menjadi 9,46 detik, dan persentase respon di bawah 3000 ms turun drastis dari 40% menjadi hanya 19%, menandakan sistem mulai kewalahan di bawah beban tinggi.



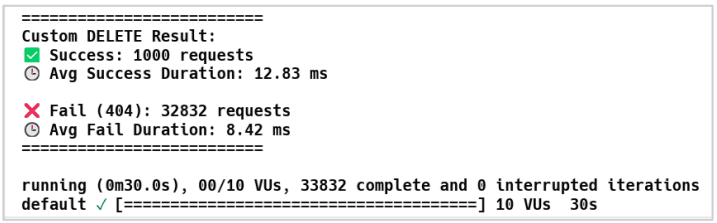
Gambar 4. *POST Test REST API*

Dari hasil gambar 4 didapatkan semua permintaan berhasil diproses, rata-rata waktu respon berada pada angka 60,7 ms yang menunjukkan performa cepat dan efisien dalam kondisi beban yang rendah. Waktu respon maksimum hanya mencapai 624,48 ms dan P(95) tetap berada di bawah 150 ms yang menandakan stabilitas sistem sangat baik.



Gambar 5. *PUT Test REST API*

Hasil pengujian gambar 5 dengan *Virtual Users* (VU) selama 30 detik menunjukkan bahwa sistem memiliki performa yang baik dan stabil untuk menangani permintaan *update* data. Seluruh permintaan berhasil diproses tanpa adanya *error*, dan waktu respon rata-rata sangat cepat.



Gambar 6. *DELETE Test REST API*

Pada gambar 6 menghasilkan 33832 *request*, namun hanya 1000 yang berhasil, sementara 32832 *request* gagal dengan status 404 karena data sudah tidak tersedia untuk dihapus.

TeknoIS: Jurnal Ilmiah Teknologi Informasi dan Sains
Volume 16 Number 1, January 2026 Page. 26-36
Journal Homepage: https://teknois.unbin.ac.id/index.php/JBS/
Received: November 2025; Accepted: December 2025; Published: January 2026

Meskipun *error rate* tinggi (97%), rata-rata *response time* sangat cepat (12,83 ms untuk sukses dan 8,42 ms untuk gagal), menunjukkan performa sistem sangat baik. Kegagalan terjadi bukan karena *error* sistem, melainkan karena data sudah dihapus oleh *request* lain. Kesimpulannya, sistem mampu menangani *request* dengan cepat, namun perlu penyesuaian jumlah data atau logika agar tiap *DELETE* diarahkan ke data unik.
Kemudian terdapat hasil performa dari pengujian pada *GraphQL API* berikut ini.

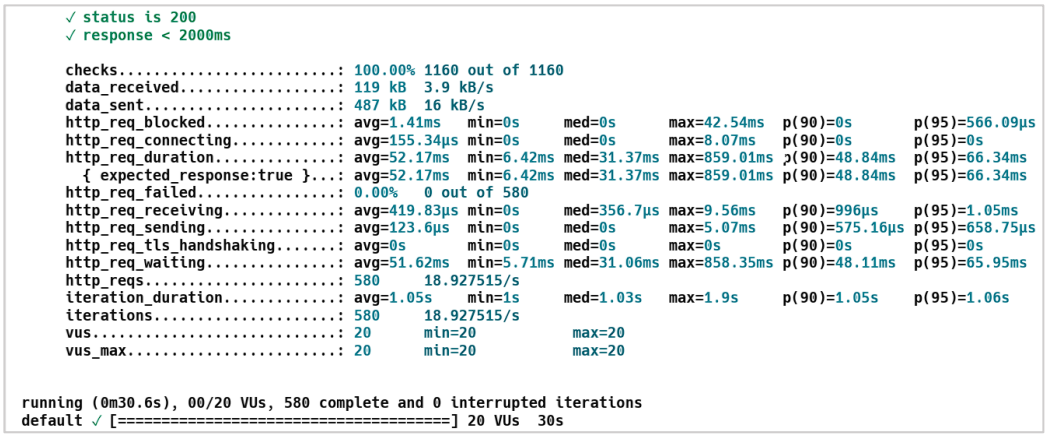
Tabel 4. *Load Test GraphQL API* 10000 Data

Metrik	**30 VU**	**40 VU**	**50 VU**
Average Response Time	13,54 s	18,06 s	28,15 s
95th *Percentile* (P95)	16,5 s	21,39 s	33,98 s
Request Per Second (RPS)	1,90 rps	1,92 rps	1,52 rps
Total Request	171	207	204
Error Rate	0%	0%	0%
Checks Lolos	50% (171 dari 342)	50% (207 dari 414)	50% (204 dari 408)
Response <2000ms	0% (0 dari 171)	0% (0 dari 207)	0% (0 dari 204)

Tabel 4 menjelaskan bahwa sistem mampu menjaga stabilitas dengan seluruh permintaan berhasil dan tanpa *error*. Namun, waktu respon meningkat signifikan seiring bertambahnya VU. Rata-rata respon naik dari 13 detik (30 VU) menjadi 18 detik (40 VU), dan 28 detik (50 VU), dan P95 waktu respon mendekati 34 detik pada beban tertinggi. Ini menunjukkan sistem stabil tapi belum optimal untuk menangani beban tinggi secara cepat.

Tabel 5. *Spike Test GraphQL API*

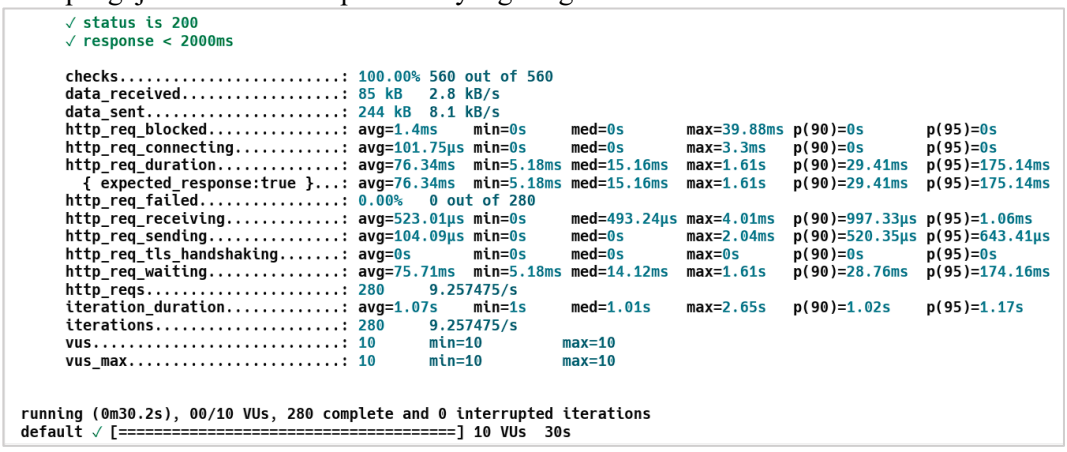
Metrik	**1k Data**	**5k Data**	**10k Data**
Average Response Time	349,52 ms	16,79 s	23,71 s
95th *Percentile* (P95)	621,48 ms	27,41 s	44,43 s
Request Per Second (RPS)	45,3 rps	3,85 rps	2,66 rps
Total Request	2954	294	215
Error Rate	0%	0%	0%
Checks Lolos	100% (5908 dari 5908)	56,46% (332 dari 588)	53,95% (232 dari 430)
Response <2000ms	100% (2954 dari 2954)	12% (38 dari 294)	7% (17 dari 215)



Gambar 7. *POST Test GraphQL API*

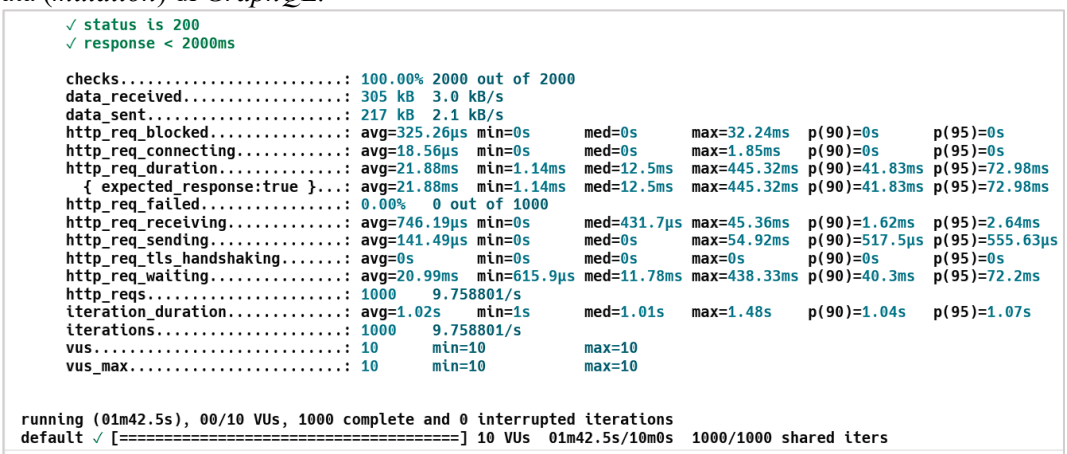
Tabel 5 hasilnya menunjukkan bahwa sistem hanya optimal pada data dengan skala kecil yaitu 1000 data. Kinerja sistem turun drastis seiring meningkatnya jumlah data (*avg time* meningkat 10 hingga 18 kali).

Pada gambar 7 dengan rata-rata respon 52,17 ms yang sangat ideal untuk *POST request*. Tidak ada respon yang melebihi waktu kurang dari 2 detik, tidak ada *error* dan *check* lolos semua losos. Untuk maksimal *response time* mencapai 859 ms. Jadi sistem menunjukkan bahwa dalam pengujian ini memiliki performa yang sangat baik dan stabil.



Gambar 8. *PUT Test GraphQL API*

Dari hasil gambar 8 semua permintaan tidak ada yang melebihi 2000 ms. Tidak ada permintaan yang gagal, hampir semua *checks* lolos dengan akurasi 99,48%. Jadi hasilnya menunjukkan bahwa performanya baik dan efisien serta cocok dijadikan sebagai *baseline* kinerja *update* data (*mutation*) di *GraphQL*.



TeknoIS: Jurnal Ilmiah Teknologi Informasi dan Sains
Volume 16 Number 1, January 2026 Page. 26-36
Journal Homepage: https://teknois.unbin.ac.id/index.php/JBS/
Received: November 2025; Accepted: December 2025; Published: January 2026

Gambar 9. *DELETE Test GraphQL API*

Pada gambar 9 waktu respon rata-rata 21,88 ms dengan maksimum waktu respon hingga 445,32 ms menunjukkan performa yang baik, kemungkinan karena beban server atau GC (*Garbage Collection*). Total *checks* berhasil 100% tidak ada kegagalan permintaan jadi cocok untuk digunakan dalam skala besar karena latensi yang rendah dan stabil.

2. PEMBAHASAN

Berangkat dari pengujian berbasis *K6* pada tumpukan *JavaScript* yang seragam, bagian ini mengkaji kekuatan hingga kelemahan masing-masing arsitektur *API* secara kontekstual, mengurai faktor penyebab perbedaan kinerja, dan merumuskan rekomendasi penerapan di situasi nyata. Tabel 6 merangkum rata-rata respon pada seluruh hasil pengujian yang telah dilakukan dengan berbagai *request method*.

Tabel 6. Analisis *Overall Avg Response Time*

No.	Jenis Operasi	REST API (ms)	GraphQL (ms)
1	GET	4137,07	7323,43
2	POST	92,08	50,8
3	PUT	88,26	5,75
4	DELETE	Gagal (98% error)	31,57

Operasi *GET*

Pada operasi *GET, REST API* menunjukkan waktu respon rata-rata sebesar 4137,07 ms, sementara *GraphQL API* jauh lebih tinggi yaitu 7323,43 ms. Hasil ini menunjukkan bahwa *REST API* lebih efisien dalam pengambilan data dalam jumlah besar. Hal ini dapat dijelaskan oleh arsitektur *REST* yang menggunakan *endpoint* khusus untuk setiap sumber daya, sehingga proses eksekusi *query* lebih langsung dan tidak membutuhkan parsing struktur *query* seperti pada *GraphQL*.

Waktu respon *GraphQL* yang lebih tinggi dalam operasi *GET* disebabkan oleh kompleksitas pemrosesan *query* dan *resolver* pada sisi server. Meskipun *GraphQL* menawarkan fleksibilitas dalam menentukan struktur data yang diinginkan, keunggulan ini menjadi beban ketika jumlah permintaan data sangat besar dan terstruktur kompleks.

Operasi *POST* dan *PUT*

GraphQL API menunjukkan performa lebih unggul secara signifikan pada operasi *POST* dan *PUT*. Waktu respon *GraphQL* untuk *POST* hanya 50,8 ms dibandingkan *REST* yang mencapai 92,08 ms. Untuk *PUT*, selisihnya bahkan lebih drastis, di mana *GraphQL* mencatat 5,75 ms, sedangkan *REST* 88,26 ms.

Keunggulan *GraphQL* dalam operasi manipulasi data ini dapat dikaitkan dengan desain arsitekturnya yang hanya menggunakan satu *endpoint* dan memproses mutasi secara langsung menggunakan *schema resolver*. Hal ini memungkinkan efisiensi tinggi dalam penanganan data kecil yang bersifat spesifik atau parsial. *REST* di sisi lain, mengandalkan pemanggilan latensi, terutama jika harus melewati *middleware* atau otorisasi berlapis.

Operasi *DELETE*

Pada operasi *DELETE, REST API* gagal mencapai eksekusi stabil, dengan tingkat kegagalan mencapai 98% dan waktu respon tidak dapat diukur secara konsisten. Sementara itu, *GraphQL API* berhasil menjalankan operasi *DELETE* dengan waktu respon 31,57 ms, menunjukkan ketangguhan dan efisiensi arsitektur *GraphQL* dalam menangani permintaan penghapusan data, khususnya dalam konteks mutasi yang terintegrasi dalam *query payload*.

Kegagalan *REST* dalam operasi *DELETE* disebabkan oleh *overload* atau ketidaksesuaian konfigurasi pada *endpoint DELETE* saat dilakukan pengujian simultan, terutama di bawah beban tinggi. Hal ini memperkuat temuan bahwa *REST API* cenderung sensitif terhadap beban berat pada operasi mutasi dibanding *GraphQL*

D. KESIMPULAN

Hasil penelitian menunjukkan bahwa *REST API* lebih unggul dalam operasi *GET*, terutama ketika menangani permintaan data dalam jumlah besar dengan *query* data yang sederhana, karena arsitekturnya yang sederhana dan langsung. Sebaliknya, *GraphQL API* menunjukkan kinerja yang lebih baik dalam operasi mutasi data (*POST, PUT, DELETE*) dengan waktu respon yang lebih rendah dan stabilitas yang lebih tinggi, terutama dalam kondisi beban tinggi atau lonjakan trafik. Meskipun begitu, *GraphQL* cenderung mengalami penurunan performa pada *query* yang kompleks dan permintaan data berskala besar.

Dengan demikian, pemilihan antara *REST API* dan *GraphQL API* sebaiknya disesuaikan dengan kebutuhan spesifik aplikasi. *REST* lebih sesuai untuk sistem yang memerlukan pengambilan data masif dan sederhana, sedangkan *GraphQL* lebih ideal untuk sistem yang menuntut fleksibilitas tinggi dan interaksi data yang kompleks.

Penelitian ini memberikan kontribusi praktis dalam membantu pengembang sistem menentukan pendekatan *API* yang paling efisien berdasarkan skenario penggunaan. Untuk pengembangan penelitian selanjutnya, disarankan agar dilakukan pengujian lebih lanjut dengan mempertimbangkan aspek lain seperti penggunaan sumber daya server, keamanan, dan integrasi otentikasi *multi-layer*.

E. DAFTAR PUSTAKA

[1] M. Asqia, M. Afif, T. Wahyudi, A. R. Adriansyah, and K. Panji, "Development of a Web-Based Correspondence Information System to Enhance Administrative Services in Higher Education," *Indones. J. Comput. Sci.*, vol. 3460, 2023.

[2] W. Barker, N. Maisel, C. E. Strawley, G. K. Israelit, J. Adler-Milstein, and B. Rosner, "A national survey of digital health company experiences with electronic health record application programming interfaces," *J. Am. Med. Informatics Assoc.*, vol. 31, no. 4, pp. 866–874, 2024, doi: 10.1093/jamia/ocae006.

[3] A. Asari *et al.*, *Pengembangan Website*. Malang: Media Nusa Creative, 2023. [Online]. Available: https://books.google.co.id/books?hl=en&lr=&id=akLBEAAAQBAJ&oi=fnd&pg=PP1&dq=apa+itu+web+modern&ots=LqcqIzi6iv&sig=iegX2Z9Qv5NzzVu75g4wej8QfBA&redir_esc=y#v=onepage&q=apa itu web modern&f=false

[4] K. I. Eka Putra, "GraphQL vs REST API: Apa bedanya?," 2019, *Dicoding*. [Online]. Available: https://www.dicoding.com/blog/graphql-api-vs-rest-api-apa-bedanya/

[5] M. Kim, Q. Xin, S. Sinha, and A. Orso, *Automated Test Generation for REST APIs: No Time to Rest yet*. ISSTA, 2022.

[6] S. Buna, *GraphQL in action*. New York: Simon and Schuster, 2021. [Online]. Available: https://books.google.co.id/books?hl=en&lr=&id=1jszEAAAQBAJ&oi=fnd&pg=PA1&dq=graphql&ots=fynEU_PxYD&sig=Fca7V-DECqe8YbkSZT1BUxTy1xA&redir_esc=y#v=onepage&q=graphql&f=false

[7] A. Permana, "ANALISIS PERBANDINGAN METODE GRAPHQL DAN METODE REST API PADA TEKNOLOGI NODEJS," 2020. [Online]. Available: https://eprints.utdi.ac.id/8998/

[8] A. Lawi, B. L. Panggabean, and T. Yoshida, *Evaluating GraphQL and REST API Services*

Performance in a Massive and Intensive Accessible Information System. MDPI, 2021.

[9] D. A. Hendarto, "ANALISIS PERBANDINGAN ARSITEKTUR REST DAN GRAPHQL UNTUK APLIKASI PENGENALAN EMOSI PADA PEMBELAJARAN DARING SINKRONIS," in *ANALISIS PERBANDINGAN ARSITEKTUR REST DAN GRAPHQL UNTUK APLIKASI PENGENALAN EMOSI PADA PEMBELAJARAN DARING SINKRONIS*, 2023. [Online]. Available: https://repository.upi.edu/87468/

[10] D. Muntahanah, Y., and K. Pinandita, "IMPLEMENTASI PERBANDINGAN METODE GRAPHQL DAN REST API PADA TEKNOLOGI NODEJS," *J. Inf. Technol. Comput. Sci.*, vol. INTECOMS, pp. 25–34, 2024.

[11] A. Juryala, D. Ganji, and P. Tirumalareddy, "E-commerce Cosmetic Products," 2023. [Online]. Available: https://www.kaggle.com/datasets/devi5723/e-commerce-cosmetics-dataset

[12] F. R. Anindita, "Tutorial K6 API Test," 2023. [Online]. Available: https://fadhilara.medium.com/tutorial-k6-api-test-2bda78f2275