

BAB III

ARTIKEL KARYA ILMIAH

JIKO (JURNAL INFORMATIKA DAN KOMPUTER)

Februari 2024, Volume: 8, No. 1 | Pages 15–21 doi:

10.26798/jiko.v8i1.xxx

e-ISSN : 2477-3964 – p-ISSN : 2477-4413



ARTICLE

Analisis Perbandingan GraphQL dan REST API pada Aplikasi Menu Restoran dengan Node.js

Comparative Analysis of GraphQL and REST API in Node.js-Based Restaurant Menu Applications

Agung Prasetyo,¹ Danny Kriestanto²

^{1,2}Informatika, Fakultas Teknologi Informasi, Universitas Teknologi Digital Indonesia, Yogyakarta, Indonesia

*Penulis Korespondensi: agung.prasetyo@students.udi.ac.id

(Disubmit 22-11-24; Diterima 23-3-24; Dipublikasikan online pada 30-3-24)

Abstrak

Penelitian ini bertujuan untuk menganalisis perbandingan performa antara GraphQL dan REST API pada aplikasi menu restoran berbasis Node.js, dengan fokus pada aspek waktu respons, penggunaan bandwidth, dan fleksibilitas. Masalah yang diangkat adalah menentukan solusi API yang optimal untuk aplikasi yang membutuhkan pengelolaan data secara efisien dan cepat. Pengujian dilakukan di lingkungan cloud menggunakan layanan gratis untuk menggambarkan kondisi nyata. Pendekatan penelitian dilakukan dengan pengujian performa menggunakan K6, alat yang digunakan untuk mensimulasikan beban permintaan pada server. Parameter yang diukur meliputi jumlah total permintaan, rata-rata waktu respons, volume data yang diterima dan dikirim, serta stabilitas server di bawah beban tinggi. Hasil analisis menunjukkan bahwa waktu respons GraphQL dan REST API tidak berbeda secara signifikan. Namun, GraphQL memiliki keunggulan dalam efisiensi bandwidth, karena hanya mengirim data yang diminta oleh klien, sedangkan REST API cenderung kurang fleksibel dan menghasilkan pengiriman data berlebih yang tidak selalu diperlukan klien. Hasil penelitian ini menunjukkan bahwa GraphQL unggul dibandingkan REST API dalam hal efisiensi data, kestabilan performa, dan fleksibilitas pengambilan data. GraphQL lebih hemat bandwidth dan memberikan kontrol lebih besar kepada klien dalam memilih data yang dibutuhkan, menjadikannya pilihan terbaik untuk aplikasi dengan kebutuhan data dinamis dan skalabilitas tinggi. Namun, REST API tetap efektif untuk aplikasi dengan arsitektur sederhana yang tidak memerlukan kustomisasi data kompleks.

Kata kunci: Perbandingan; GraphQL; REST API; Node.js; K6

Abstract

This study aims to analyze the comparative performance of GraphQL and REST API on a Node.js-based restaurant menu application, focusing on aspects of response time, bandwidth usage, and flexibility. The problem raised is to determine the optimal API solution for applications that require efficient and fast data management. Testing is carried out in a cloud environment using free services to describe the actual conditions. The research approach is carried out by testing performance using K6, a tool used to simulate the request load on the server. The parameters measured include the total number of requests, average response time, volume of data received and sent, and server stability under high load. The results of the analysis show that the response time of GraphQL and REST API is not significantly different. However, GraphQL has an advantage in bandwidth efficiency, because it only sends data requested by the client, while REST API tends to be less flexible and results in sending excessive data that is not always needed by the client. The results of this study indicate that GraphQL is superior to REST API in terms of data efficiency, performance stability, and flexibility of data retrieval. GraphQL is more bandwidth efficient and gives clients more control in choosing the data they need, making it the best choice for applications with dynamic data needs and high scalability. However, REST APIs remain effective for applications with simple architectures that do not require complex data customization.

KeyWords: Comparison; GraphQL; REST API; Node.js; K6

This is an Open Access article - copyright on authors, distributed under the terms of the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY SA) (<http://creativecommons.org/licenses/by-sa/4.0/>)

How to Cite: P. Agung *et al.*, "Analisis Perbandingan GraphQL dan REST API pada Aplikasi Menu Restoran dengan Node.js", *JIKO (JURNAL INFORMATIKA DAN KOMPUTER)*, Volume: 8, No.1, Pages 15–21, Februari 2024, doi: 10.26798/jiko.v8i1.xxx.

1. Pendahuluan

Perkembangan teknologi yang cepat telah menghadirkan berbagai metode baru untuk mengelola dan bertukar data antara *backend* dan *frontend*, terutama melalui *Application Programming Interface* (API). Salah satu metode yang paling umum digunakan adalah *Representational State Transfer* (REST), yang telah menjadi tulang punggung aplikasi web modern[1]. Namun, seiring dengan meningkatnya kebutuhan untuk pengelolaan data yang lebih efisien, GraphQL muncul sebagai alternatif yang lebih fleksibel, memungkinkan *client* untuk menentukan dengan tepat data apa yang ingin diambil dari *server*[2].

REST API sering menghadapi masalah *over-fetching* dan *under-fetching*. *Over-fetching* terjadi ketika *client* menerima lebih banyak data daripada yang dibutuhkan, sedangkan *under-fetching* terjadi ketika *client* tidak mendapatkan data yang cukup, sehingga memerlukan permintaan tambahan[3]. Masalah ini bisa mempengaruhi performa aplikasi yang mengelola data kompleks, seperti memperlambat waktu respons dan meningkatkan penggunaan bandwidth. GraphQL hadir sebagai solusi dengan memungkinkan *client* meminta data yang lebih spesifik sesuai kebutuhan[4].

Sampai saat ini, telah dilakukan beberapa penelitian yang membahas perbandingan antara REST API dan GraphQL API. Pada penelitian sebelumnya, rata-rata REST API masih unggul dari segi performa, namun untuk fleksibilitas permintaan data GraphQL bisa menjadi alternatif saat ini. Pada penelitian yang dilakukan penulis, penulis menggunakan lingkungan cloud dengan memanfaatkan layanan gratis. Merujuk pada penelitian sebelumnya sudah ada yang pernah mengulas topik ini menggunakan teknologi Node.js Namun, kelemahannya adalah penelitian tersebut hanya menganalisis HTTP Request dalam lingkungan lokal.

Oleh karena itu berdasarkan fakta dan permasalahan diatas, penulis bertujuan untuk menganalisis terhadap kinerja REST API dan GraphQL yang diharapkan dapat membantu untuk menentukan arsitektur API yang terbaik dalam membangun sebuah menggunakan teknologi Node.js pada lingkungan *cloud*. *Response time*, *bandwidth usage*, dan fleksibilitas menjadi tolok ukur dalam penelitian ini[5]. Semakin cepat waktu respon dalam memproses permintaan data dan mengembalikannya ke *client*, semakin cepat pula informasi yang tersampaikan kepada pengguna. Hal ini berkontribusi langsung terhadap kepuasan pengguna web service, di mana waktu tunggu yang lebih rendah akan meningkatkan pengalaman pengguna secara keseluruhan.

2. Metode

Penelitian ini dilakukan melalui beberapa tahapan yang bertujuan untuk membandingkan implementasi GraphQL dan REST API pada aplikasi menu restoran menggunakan *Node.js*. Setiap tahapan dirancang secara sistematis untuk memastikan keluaran penelitian sesuai dengan harapan dan memberikan gambaran yang jelas mengenai performa kedua metode API dalam konteks yang diujikan. Tahapan penelitian ini meliputi:

2.1 Studi Literatur

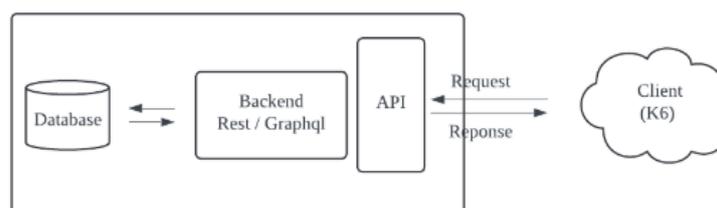
Pada tahap awal, dilakukan studi literatur untuk memahami konsep dasar REST API dan GraphQL, serta kelebihan dan kekurangan masing-masing metode. Selain itu, juga dilakukan kajian terhadap teknologi *Node.js* yang akan digunakan sebagai *platform* pengembangan aplikasi. Informasi dari literatur ini akan menjadi dasar dalam mendesain sistem.

2.2 Perancangan Sistem

Tahap ini melibatkan perancangan aplikasi menu restoran yang akan diimplementasikan menggunakan dua metode API, yaitu REST API dan GraphQL. Desain sistem mencakup arsitektur aplikasi, struktur database, serta *endpoint* API yang digunakan untuk mengakses data menu pada aplikasi menu restoran.

2.2.1 Arsitektur Aplikasi

Arsitektur aplikasi terdiri dari server (Node.js) dan database, tanpa adanya antarmuka pengguna khusus. Pengujian dilakukan secara langsung menggunakan alat khusus untuk setiap metode API:

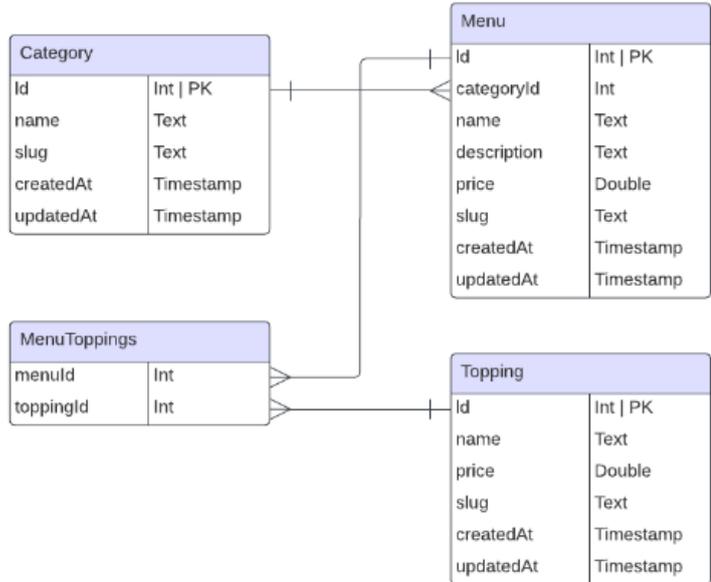


Gambar 1. Arsitektur Aplikasi

Server dibangun menggunakan Node.js dengan Hapi sebagai framework untuk REST API dan Graphi sebagai library untuk GraphQL[6]. Kedua metode API ini terhubung ke database PostgreSQL untuk mengambil data terkait menu. Aplikasi dideploy menggunakan layanan gratis dari *railway.com* dengan spesifikasi 2 vCPU, 512 MB RAM, dan 1 GB SSD Storage. *Server* akan memproses permintaan yang dikirim melalui Postman, GraphQL Playground, atau tools K6 akan merespons sesuai dengan skema API yang digunakan.

2.2.2 Perancangan Basis Data

Struktur database yang digunakan untuk aplikasi ini dirancang dengan basis relasional. Terdapat empat tabel utama yang akan digunakan dalam pengelolaan data aplikasi, yaitu:



Gambar 2. Rancangan Database

2.2.3 Endpoint API

Terdapat beberapa endpoint yang dirancang untuk mengelola data menu. Setiap metode API (*REST dan GraphQL*) memiliki cara pengelolaan endpoint yang berbeda.

Tabel 1. Endpoint REST API

Method	Endpoint	Keterangan
POST	/menu	Menambahkan menu baru
GET	/menu/:id	Mengambil menu berdasarkan id
PUT	/menu/:id	Mempengaruhi menu tertentu berdasarkan id
DELETE	/menu/:id	Menghapus data menu tertentu

Berbeda dengan REST, GraphQL hanya menggunakan satu *endpoint* dengan metode HTTP POST, yaitu */graphql*, yang digunakan untuk menangani seluruh operasi data, baik itu pengambilan, penambahan, pembaruan, maupun penghapusan data. Dalam GraphQL, struktur *query* dan *mutation* yang dibutuhkan untuk mengelola data, seperti kategori, menu, dan topping, didefinisikan secara eksplisit di dalam *schema* GraphQL, sehingga memungkinkan klien untuk menentukan data yang diinginkan dengan lebih fleksibel dan efisien. Sebagai contoh, *query* untuk mengambil data menu pada GraphQL dapat ditulis seperti berikut:

Tabel 2. Operasi GraphQL API

Operasi	Function	Remarks
mutation	{ addMenu (name: "Bakso", price: 5000, categoryId: 1) { id, name, price } }	Menambahkan menu baru
query	{ menu (id: 1) { id, name, price } }	Mengambil menu berdasarkan id
mutation	{ updateMenu (id: 1, name: "Bakso Solo", price: 6000, categoryId: 1) { id, name, price } }	Mempengaruhi menu tertentu berdasarkan id
mutation	{ deleteMenu (id: 1) { id } }	Menghapus data menu

2.2.4 Kebutuhan Pengujian

Pengujian dilakukan untuk mengukur performa kedua metode API. Pengujian meliputi beberapa aspek, seperti:

Table 3. Parameter pengujian

Aspek Pengujian	Keterangan
Response Time	Waktu yang dibutuhkan untuk mengembalikan data ke <i>client</i>
Bandwidth Usage	Jumlah total data yang diterima dan dikirim
flexibility	Kemampuan sistem dalam adaptasi terhadap berbagai kebutuhan dan perubahan yang terjadi dalam aplikasi

Parameter response time mengukur waktu yang dibutuhkan sistem untuk memproses dan mengembalikan data ke klien. Pengujian ini dilakukan untuk mengevaluasi kecepatan dalam merespons permintaan data, yang merupakan faktor penting dalam pengalaman pengguna. Semakin cepat *response time*, semakin baik kinerja aplikasi, dan semakin cepat informasi dapat disampaikan kepada pengguna. Hasil pengukuran yang optimal adalah ketika sistem mampu memberikan respons dalam waktu singkat, sedangkan hasil yang kurang baik adalah ketika waktu respons terlalu lama, yang dapat mengurangi kepuasan pengguna[7].

Parameter *bandwidth usage* mengukur efisiensi penggunaan *bandwidth* jaringan dalam mentransfer data antara server dan klien. Pengujian ini bertujuan untuk mengetahui seberapa banyak sumber daya jaringan yang dibutuhkan untuk mengirimkan data. Hasil yang baik adalah ketika metode pengambilan data menggunakan *bandwidth* yang efisien tanpa mengorbankan kualitas data, sedangkan hasil yang kurang baik adalah ketika penggunaan *bandwidth* terlalu besar, yang dapat memperlambat proses komunikasi dan meningkatkan biaya operasional[8].

Parameter fleksibilitas mengacu pada kemampuan klien untuk menyesuaikan data yang dibutuhkan serta operasi CRUD (*Create, Read, Update, Delete*). Fleksibilitas ini dibagi menjadi dua aspek utama: pertama, fleksibilitas dalam kustomisasi kebutuhan data, yang mengukur sejauh mana klien dapat menyesuaikan data yang ditampilkan sesuai dengan kebutuhannya, di mana hasil yang baik adalah hanya menampilkan data yang relevan, dan buruk jika data yang tidak diperlukan ikut ditampilkan[9]. Kedua, fleksibilitas dalam kustomisasi operasi CRUD, yang mengukur kemudahan klien dalam mengoperasikan seluruh operasi CRUD menggunakan sedikit URL; hasil yang baik adalah hanya membutuhkan satu URL untuk semua operasi, sementara hasil yang kurang baik adalah jika membutuhkan banyak URL untuk setiap operasi CRUD.

Pengujian ini menggunakan K6, sebuah *tool open-source* untuk *load testing* yang mampu mengukur performa API secara *real-time*[10]. Dengan K6, berbagai skenario pengujian seperti waktu respon dan konsumsi *bandwidth*[11].

2.2.5 Prosedur dan Pengumpulan Data

Proses pengukuran parameter dilakukan sebagai berikut:

1. Response Time

- a. Siapkan *script* K6 untuk REST API dan GraphQL seperti pada Koding 1.

```
import http from 'k6/http';
import { check, sleep } from 'k6';

export let options = {
 vus: 10, // Simulasi 10 pengguna
iterations: 200,
};

export default function () {
  // REST API
  const restRes = http.get(
' https://api-restaurant-menu.up.railway.app/api/menu/1');
  check(restRes, { 'status is 200': (r) => r.status === 200});

  // GraphQL
  let query = `
    query {
      menu(id: 1) {
        id
        name
      }
    }
  `;
  let headers = { 'Content-Type': 'application/json' };
  let graphqlRes = http.post(' https://api-restaurant-
menu.up.railway.app/graphql', JSON.stringify({ query }), { headers });
  check(graphqlRes, { 'Menu status 200': (r) => r.status === 200 });
}
```

Koding 1. *Script* pengujian GraphQL dan REST API

- b. Jalankan tes menggunakan K6 untuk mencatat metrik *http_req_duration*, yaitu durasi total setiap request.
 - c. Analisis laporan K6, termasuk nilai rata-rata, minimum, median, dan maksimum dari *http_req_duration*.
- ## 2. Bandwidth Usage
- a. Gunakan *script* K6 yang sama seperti pada pengukuran response time.
 - b. Jalankan tes menggunakan K6 untuk mencatat metrik *total_bandwidth_rest* dan *total_bandwidth_graphql*.
 - c. Evaluasi hasil ini untuk memahami efisiensi penggunaan *bandwidth* oleh masing-masing metode API.
- ## 3. Flexibility
- a. Flexibility pada kebutuhan data diuji dengan mengirim permintaan ke REST API atau GraphQL untuk mendapatkan data tertentu menggunakan Postman.
 - b. Fleksibilitas Kustomisasi Operasi CRUD (*Create, Read, Update, Delete*):
 Pertama, fleksibilitas dalam menyesuaikan kebutuhan data melalui input query dan data menu seperti name, price, categoryId. Kedua, kemudahan bagi klien untuk menjalankan seluruh operasi CRUD menggunakan sedikit URL, baik melalui REST API maupun GraphQL.

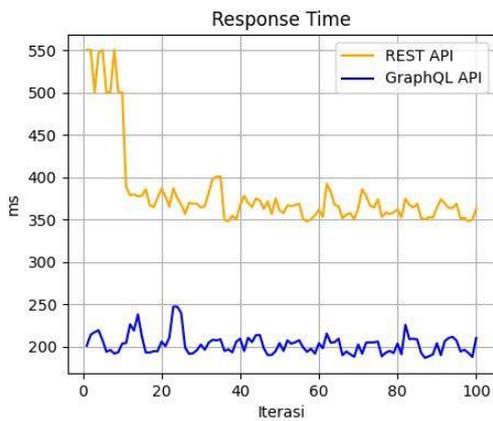
Pengujian dilakukan pada beberapa *endpoint* REST API dan *query* GraphQL yang akan memanipulasi data menu pada aplikasi restoran. Untuk REST API, pengambilan data dilakukan dengan beberapa permintaan, sedangkan untuk GraphQL, hanya satu permintaan diperlukan untuk mengambil semua data yang relevan[12]. Data hasil pengujian akan dikumpulkan dan dianalisis untuk melihat perbedaan metode REST API dan GraphQL.

3. Hasil dan Pembahasan

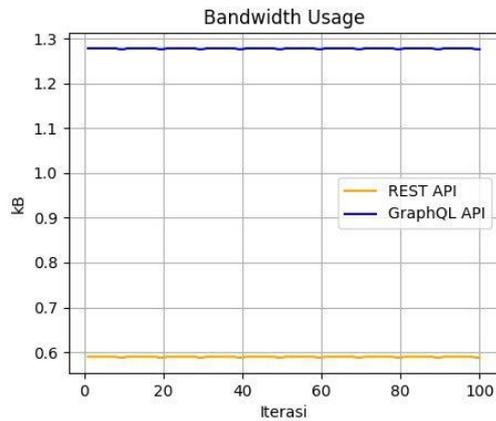
3.1 Parameter Response Time dan Bandwidth Usage

Pada bagian ini menyajikan hasil pengukuran dan analisis performa terhadap parameter response time dan *bandwidth usage* yang telah diimplementasikan. Pengujian dilakukan menggunakan K6 untuk membandingkan performa masing-masing metode. Iterasi 100 dipilih untuk memastikan data *response time* dan *bandwidth usage* stabil dan representatif. Jumlah ini cukup sesuai karena memenuhi konsistensi data, memadai terutama untuk menghindari outlier atau fluktuasi data yang terjadi dalam pengujian tunggal. Iterasi yang terlalu banyak bisa memakan waktu dan sumber daya lebih besar, sehingga iterasi tersebut menjadi kompromi antara akurasi data dan efisiensi proses pengujian[13]. Tolok ukur yang digunakan adalah kecepatan respons, dengan metode yang lebih cepat dianggap lebih baik. Selain itu, *bandwidth usage* dianalisis untuk mengukur efisiensi data yang dikirim. *Bandwidth usage* yang rendah menunjukkan efisiensi yang baik, sementara penggunaan yang tinggi dapat memperlambat aplikasi dan meningkatkan beban jaringan.

3.1.1 Hasil Pengujian Skenario GET



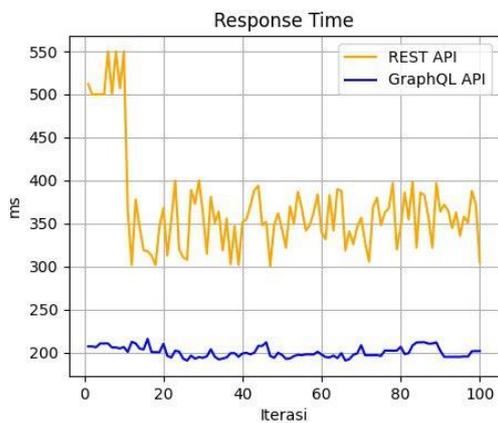
Gambar 3. Skenario GET Response Time



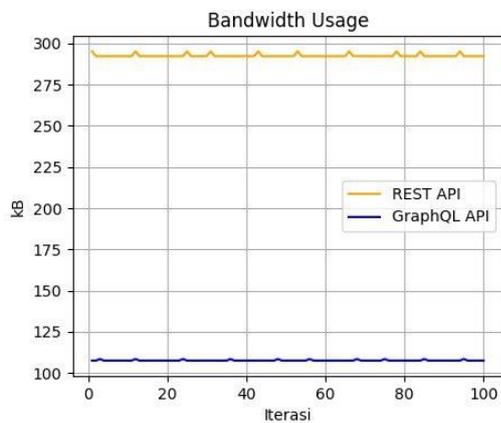
Gambar 4. Skenario GET Bandwidth Usage

Merujuk pada Gambar 3-4, pengujian *response time* REST API menunjukkan waktu respons yang lebih tinggi dan fluktuatif di awal sebelum stabil, sedangkan GraphQL lebih konsisten dan cepat, menunjukkan efisiensi GraphQL dalam memproses permintaan GET. Namun, pada penggunaan *bandwidth*, GraphQL menggunakan *bandwidth* sedikit tinggi daripada REST API.

3.1.2 Hasil Pengujian Skenario POST



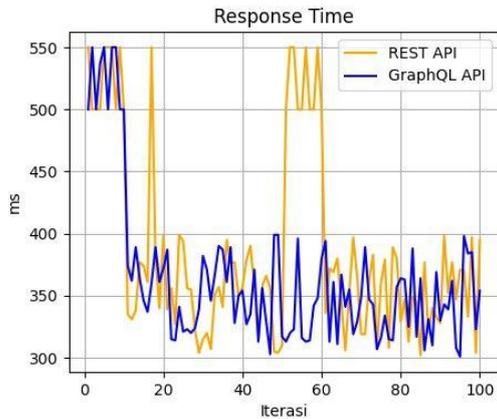
Gambar 5. Skenario POST Response Time



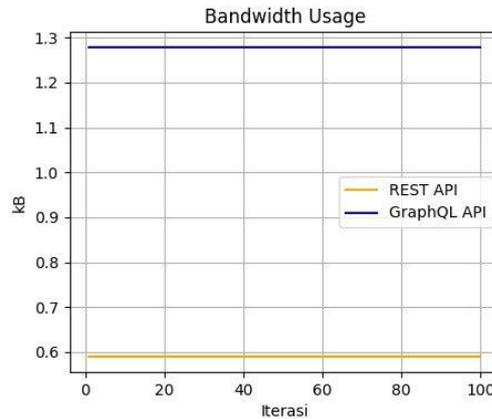
Gambar 6. Skenario POST Bandwidth Usage

Merujuk pada Gambar 5-6, pengujian POST GraphQL memiliki waktu respons lebih rendah dan konsisten dibandingkan REST API yang fluktuatif dan lebih lambat, sementara pada *bandwidth usage*, REST API menggunakan *bandwidth* lebih tinggi dibandingkan GraphQL yang lebih efisien.

3.1.3 Hasil Pengujian Skenario PUT



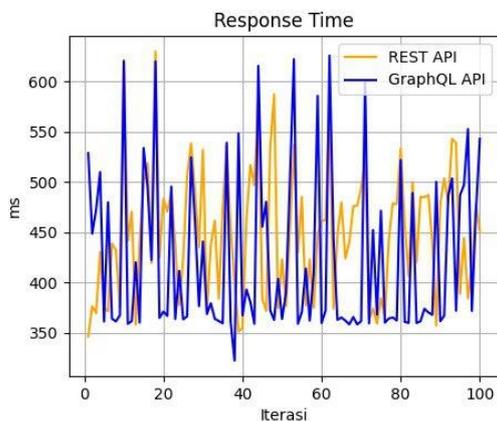
Gambar 7. Skenario PUT Response Time



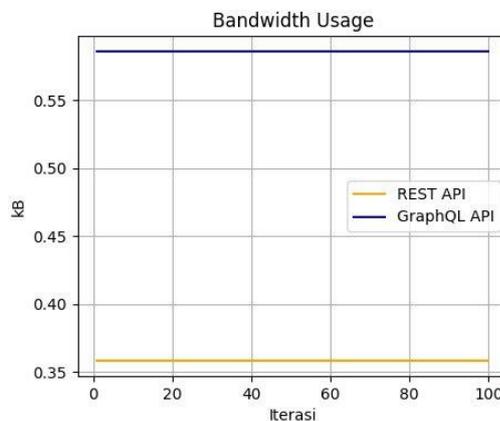
Gambar 8. Skenario PUT Bandwidth Usage

Merujuk pada Gambar 7-8, pengujian PUT waktu respons REST API dan GraphQL fluktuatif tetapi cenderung setara, sementara *bandwidth usage* GraphQL lebih tinggi dibandingkan REST API, menunjukkan REST API lebih hemat dalam penggunaan *bandwidth* untuk operasi ini.

3.1.4 Hasil Pengujian Skenario DELETE



Gambar 9. Skenario DELETE Response Time



Gambar 10. Skenario DELETE Bandwidth Usage

Merujuk pada Gambar 9-10 hasil temuan ini menunjukkan bahwa GraphQL dan REST API sama-sama unggul dalam skalabilitas untuk permintaan yang tinggi, namun GraphQL lebih efisien dalam penggunaan *bandwidth* karena penghindaran *over-fetching* [14][15]. Pada skenario pengujian GET, PUT, dan DELETE hasil pengujian dari kedua arsitektur memang tidak terlalu signifikan namun GraphQL masih menandingi dari segi performa REST API.

GraphQL menunjukkan efektivitas yang tinggi dalam mengurangi waktu respons, seperti yang terlihat dari rata-rata waktu respons yang cepat dan distribusinya yang stabil. Dengan kemampuan memfasilitasi permintaan yang lebih spesifik, GraphQL hanya mengirimkan data yang dibutuhkan oleh klien, berbeda dengan REST API yang sering mengirimkan data berlebih dalam satu respons. Pendekatan ini membuat GraphQL ideal untuk aplikasi yang membutuhkan performa tinggi, terutama dalam lingkungan dengan kebutuhan data yang kompleks namun spesifik. Selain itu, penggunaan *bandwidth* pada GraphQL terbukti lebih efisien, konsisten dengan karakteristiknya yang menghindari pengiriman data yang tidak diperlukan, sehingga mengurangi *overhead* komunikasi [16]. Meskipun REST API menunjukkan kemampuan yang baik dalam skala besar, GraphQL memiliki keunggulan signifikan dalam hal penggunaan *bandwidth* [17].

3.2 Parameter Flexibility

3.2.1 Hasil Pengujian Fleksibilitas Kustomisasi Kebutuhan Data

Hasil *query* dianalisis berdasarkan fleksibilitas metode dalam menyesuaikan data sesuai kebutuhan klien melalui operasi read di Postman. Metode dinilai baik jika mampu menyajikan data relevan sesuai permintaan, sedangkan metode yang menampilkan data berlebihan atau tidak diperlukan dianggap kurang baik. Hasil pengukuran dan pembahasannya adalah sebagai berikut:

a. Implementasi Fleksibilitas Klien REST API

Tabel 3. Fleksibilitas Klien REST API

Kebutuhan	Operasi	Hasil Operasi
Menampilkan data: 1. id 2. name	{	{ "id": 7, "name": "Bakso Sony", "price": 10000, "createdAt": "2025-01-05T11:20:29.376Z", "categoryId": 1 }
Menampilkan data: 1. id 2. name 3. price	{	{ "id": 7, "name": "Bakso Sony", "price": 10000, "createdAt": "2025-01-05T11:20:29.376Z", "categoryId": 1 }

Tabel 3 menunjukkan bahwa REST API tidak mendukung kustomisasi data spesifik. Misalnya, klien hanya membutuhkan id dan name, REST API tetap menampilkan seluruh data tanpa penyaringan sesuai kebutuhan.

b. Implementasi Dengan Parameter Fleksibilitas Klien GraphQL

Tabel 4. Fleksibilitas Klien REST API

Kebutuhan	Operasi	Hasil Operasi
Menampilkan data: 1. id 2. name	query menu { { menu (id: 7) { id name }} }	{ "id": 7, "name": "Bakso Sony", }
Menampilkan data: 1. id 2. name 3. price	query menu { { menu (id: 7) { id name price }} }	{ "id": 7, "name": "Bakso Sony", "price": 10000 }

GraphQL mendukung kustomisasi data sesuai kebutuhan klien, hanya menampilkan informasi yang diminta, seperti id, name, atau price. Implementasi pada Tabel 4 menunjukkan GraphQL lebih fleksibel dibanding REST API, karena mampu menyajikan data secara efisien tanpa menyertakan informasi yang tidak relevan.

3.2.2 Hasil Pengujian Fleksibilitas Kustomisasi Kebutuhan Data

Bagian ini menyajikan hasil perbandingan yang telah diimplementasikan, dianalisis, dan dicatat berdasarkan parameter fleksibilitas dalam melakukan kustomisasi operasi CRUD oleh klien.

Table 5. Jumlah URL pada Operasi CRUD

Metode	URL
REST API	POST https://api-restaurant-menu.up.railway.app/api/menu
	GET https://api-restaurant-menu.up.railway.app/api/menu/:id
	PUT https://api-restaurant-menu.up.railway.app/api/menu/:id
	DELETE https://api-restaurant-menu.up.railway.app/api/menu/:id
GraphQL	POST https://api-restaurant-menu.up.railway.app/graphql

Berdasarkan Tabel 5, terlihat bahwa metode GraphQL hanya memerlukan satu URL untuk menjalankan seluruh operasi CRUD (*Create, Read, Update, Delete*). Sebaliknya, metode REST API membutuhkan URL yang berbeda untuk setiap operasi CRUD. Dari hasil ini, dapat disimpulkan bahwa GraphQL memiliki fleksibilitas lebih tinggi dalam memungkinkan kustomisasi. Dengan hanya satu URL, klien dapat menjalankan semua operasi CRUD, menjadikan GraphQL sebagai metode yang lebih efisien dan fleksibel. Di sisi lain, REST API tidak memberikan fleksibilitas serupa karena memerlukan URL terpisah untuk setiap operasi CRUD, sehingga kurang mendukung kustomisasi yang efektif.

4. Simpulan

Hasil pengujian yang telah dilakukan berdasarkan skenario yang telah disusun menghasilkan beberapa kesimpulan sebagai berikut:

1. Pada skenario pengujian GET, arsitektur GraphQL API menunjukkan perbedaan performa yang cukup signifikan dibandingkan dengan REST API saat menggunakan lingkungan *cloud*. Selain mampu mengatasi beberapa kelemahan REST API, GraphQL API juga mampu bersaing dari segi performa. Pada skenario pengujian POST, PUT, dan DELETE, meskipun perbedaan performa antara kedua arsitektur tidak terlalu signifikan, GraphQL API tetap menunjukkan performa yang lebih baik dibandingkan REST API.
2. Penggunaan *Bandwidth* menunjukkan bahwa GraphQL lebih efisien dibandingkan REST API. GraphQL hanya mengirim dan menerima data yang dibutuhkan, sementara REST API cenderung mengirimkan data berlebih yang tidak selalu diperlukan oleh klien. Efisiensi *bandwidth* ini menjadikan GraphQL lebih hemat dalam penggunaan data dan lebih cepat dalam kondisi jaringan terbatas.
3. Metode GraphQL sangat fleksibel karena dapat mengizinkan klien untuk kustomisasi terhadap kebutuhan data sehingga nantinya data yang ditampilkan ke klien akan sesuai yang di request atau dibutuhkan klien, sedangkan metode REST API tidak dapat melakukan hal yang dilakukan GraphQL karena REST API tidak dapat mengizinkan klien untuk kustomisasi terhadap kebutuhan data.

Adapun kekurangan yang masih ada dalam penelitian ini, penulis memberikan beberapa rekomendasi untuk penelitian selanjutnya sebagai berikut:

1. Penelitian ini dilakukan dalam lingkungan cloud dengan memanfaatkan layanan gratis. Penelitian selanjutnya disarankan menggunakan layanan *cloud* berbayar dengan spesifikasi yang lebih tinggi. Hal ini bertujuan untuk mengevaluasi apakah peningkatan sumber daya infrastruktur, seperti CPU, RAM, dan bandwidth, memiliki dampak signifikan terhadap performa arsitektur GraphQL API dan REST API.
2. Penelitian selanjutnya diharapkan dapat menambahkan parameter uji lain seperti *latency, throughput* atau standar deviasi untuk mendapatkan analisis yang lebih komprehensif.

Ucapan Terima kasih

Dengan penuh rasa syukur kepada Allah SWT atas rahmat, hidayah, dan karunia-Nya, penelitian ini berhasil diselesaikan dengan baik. Penulis juga menyampaikan terima kasih yang sebesar-besarnya kepada semua pihak yang telah memberikan dukungan, saran, serta bantuan selama proses penelitian ini hingga dapat terselesaikan.