p-ISSN : 2406-7733
e-ISSN : 2597-9922

ANALISIS PERBANDINGAN PERFORMA APLIKASI *WEB* DENGAN MENGGUNAKAN GRAPHQL DAN REST API PADA SKENARIO BEBAN TINGGI (*HIGH LOAD*) DENGAN STUDI KASUS APLIKASI BERITA DAN MEDIA

M. Ichsan
Teknik Informatika, Saintek, Universitas Harapan Medan
Jl. HM. Joni No.70 C, Kota Medan
E-mail: 013ichsanm@gmail.com

Abstrak - Semakin berkembang pesatnya sebuah teknologi dan banyaknya manusia yang mengakses internet maka website juga semakin banyak dikunjungi dan dipergunakan oleh manusia. Penggunaan yang banyak ini membuat meningkatnya kompleksitas permintaan dan tuntutan agar *website* tersebut tetap dalam performa yang tinggi. Semakin Berkembang pesatnya sebuah teknologi dan banyaknya manusia yang mengakses internet maka *website* juga semakin banyak dikunjungi dan dipergunakan oleh manusia. Penggunaan yang banyak ini membuat meningkatnya kompleksitas permintaan dan tuntutan agar website tersebut tetap dalam performa yang tinggi. Selama dua dekade terakhir, *Representational State Transfer* (REST) telah dianggap sebagai arsitektur standar untuk merancang dan mengimplementasikan *Application Programming Interfaces* (APIs) sebagai aplikasi *backend* atau layanan *web* sisi server. Prosedur penelitian dilakukan untuk memberikan penjelasan berupa analisis komparatif langkah demi langkah kinerja aplikasi web menggunakan graphql dan rest api dalam skenario beban tinggi. antara arsitektur GraphQL dan REST, GraphQL memiliki kemampuan membaca lebih cepat dan efisien ketimbang REST pada aplikasi *website* Berita & Media. Dapat dilihat dari pengujian yang telah dilakukan dan diterapkan pada *parameter response time, troughput, CPU load* dan *memory utilization* yang menerima hasil response yang sangat cepat dibandingkan REST.

Kata Kunci: Arsitektur, Graphql, Rest, *Website*

I. PENDAHULUAN

Penyebaran berita pada masa dulu dan sekarang sangatlah berbeda dari kemajuan teknologinya. Begitu pula cara dalam mengakses ikut berubah dari yang sebelumnya kita mengakses berita melalui media cetak (koran, tabloid, televisi, radio) berubah menjadi serba online yang bisa diakses dimana saja dan kapan saja. Media *online* sebagai alat atau sarana yang digunakan dalam mengakses suatu informasi atau berita dengan perangkat yang terhubung dalam jaringan internet (Bangun et al., 2019).

Tidak bisa dipungkiri bahwa penggunaan media baru internet terus bergerak naik, dan belum sampai pada titik kulminasi. Oleh karena itu, menurunnya jumlah pendengar dan penonton televisi, pembaca koran di belahan dunia ini karena banyak munculnya media digital. Waktu luang yang biasanya dipergunakan seseorang untuk mendengarkan radio menonton televisi atau membaca koran sekarang mereka lebih tertarik ber-online di dunia *cyber* (Harliantara, 2019).

Penggunaan portal *website* media dan berita saat ini merupakan cara yang efektif dalam mengakses berita yang selalu *update* setiap saat, sehingga itu membuat *website* tersebut harus mempunyai ketahanan terhadap banyaknya pengguna dalam mengakses *website* tersebut.

Semakin berkembang pesatnya sebuah teknologi dan banyaknya manusia yang mengakses internet maka website juga semakin banyak dikunjungi dan dipergunakan oleh manusia. Penggunaan yang banyak ini membuat meningkatnya kompleksitas permintaan dan tuntutan agar *website* tersebut tetap dalam performa yang tinggi dan jangan sampai mempengaruhi kecepatan dalam mengakses *website* tersebut. Dalam pembuatan *website* pada jaman sekarang API sangat berperan penting di dalamnya sebagai jembatan untuk berkomunikasi antar klien (pengguna) dan *server*.

REST API merupakan gaya arsitektual perangkat lunak yang didalamnya mendefinisikan aturan-aturan untuk membuat *web service* dan bersifat *stateless*. *Stateless* merupakan ketidak adaan sebuah *state* di dalam sebuah aplikasi dimana *server* tidak menyimpan *state* terhadap kegiatan yang dilakukan oleh *client* dan infomasi yang diberikan harus detail beserta informasi otentikasi. Penggunaan API sendiri bertujuan agar bisa digunakan untuk berbagi data dengan aplikasi lain dengan mudah. Dengan konfigurasi yang mudah dipahami dan bersifat opensoure REST API sering digunakan untuk lebih memudahkan dalam mengambil data (Umami & Ningrum, 2021).

Selama dua dekade terakhir, *Representational State Transfer* (REST) telah dianggap sebagai arsitektur standar untuk merancang dan

p-ISSN : 2406-7733
e-ISSN : 2597-9922

mengimplementasikan *Application Programming Interface*s (APIs) sebagai aplikasi *backend* atau layanan *web* sisi *server*. Namun, itu diyakini bahwa penggunaan praktis REST di dunia API akan segera berubah karena potensi munculnya GraphQL (Lawi et al., 2021).

Walaupun REST API banyak diminati dan digunakan oleh banyak *user* bukan berarti REST API tidak memiliki kekurangan, terdapat masalah pada REST API dimana *frontend* melakukan permintaan data yang berlebihan dan bahkan tidak dibutuhkan untuk ditampilkan ke *website*. Hal itu akan meningkatkan *latensi*, yang membuat pengakses halaman *web* harus menunggu lebih lama sebelum data dikirim pada bagian *frontend* dengan sempurna. Selain itu, kompleksitas program juga semakin bertambah karena bagian *frontend* harus menambahkan satu permintaan data lagi ke bagian *backend*. Salah satu cara mengatasi *under-fetching* pada REST API adalah dengan membuat *endpoint* baru yang melakukan pengambilan data sesuai dengan apa yang diminta oleh *frontend*. Namun, jika terdapat banyak data yang mirip dan memiliki *endpoint* masing-masing, maka bentuk kode pada bagian *backend* akan menjadi kompleks dan kurang baik diakibatkan oleh adanya kode yang memiliki fungsi mirip namun ditulis lebih dari 1 kali (Firdausi et al., 2021).

II. TINJAUAN PUSTAKA

Underfetching menjadi duduk perkara pada suatu sistem info karena mempunyai dampak pada penurunan performa suatu situs web. Penurunan performa tersebut terjadi dikarenakan adanya proses `meminta ulang` suatu data pada *backend* pada e*ndpoint* yang berbeda. Hal ini mengakibatkan adanya tambahan kegiatan di sistem. Secara teori, hal itulah yang mengakibatkan performa sistem menjadi terhambat.

Contohnya, kalau di dalam sistem adalah laman yang melakukan *request* ke-2 *endpoint* yang berbeda, sebagai contoh */product/id* ini untuk mendapatkan data barang dan *product/id/user* ini untuk mengambil data barang yang dipunyai oleh *user*, maka akan ada 2 aktivitas yang harus dilakukan oleh sistem, yaitu meminta data pada enpoint *product/id* dan juga meminta pada *product/id/user.* (Firdausi et al., 2021)

Masalah tersebut bisa diatasi menggunakan cara dengan menggabungkan antara dua data yang tidak selaras tadi menjadi 1 *endpoint* saja. Akan tetapi hal itu juga memiliki kekurangan yang dimana jika sistem hanya menggunakan data diri pengguna saja, sistem akan juga menerima data *product* penggunakan yang tidak diperlukan. Sebagai akibatnya terjadilah pengambilan data yang berlebihan.

Masalah tersebut dapat dipecahkan di dalam GraphQL. Sistem di dalam GraphQL dapat membantu menentukan data apa saja yang diperlukan berdasarkan *query* yang tersedia di dalam *backend.* Sedangkan di dalam *backend* harus menentukan data apa saja yang dapat diambil oleh bagian *frontend*, bagian *frontend* juga harus menentukan data apa saja yang dapat diambil sesuai yang disediakan oleh *backend*. Misalnya terdapat data pengguna yg terdiri dari *name*, *email*, *role*, serta *password*. Bagian *backend* menyediakan *query* buat mendapatkan data pengguna yang berupa *name*, *email*, dan *role*. Maka, bagian *frontend* bisa mendapatkan data dari *query* tersebut, entah hanya mendapatkan *name* saja, *name* serta *email* saja, dan lain sebagainya. namun, bagian *frontend* tidak akan bisa mendapatkan *password* sebab *query* tidak disediakan oleh *backend*.

GraphQL pula bisa mendapatkan data asal beberapa *query* sekaligus. Misalkan ada *query user* serta item, maka bagian *frontend* dapat melakukan pengambilan data *user* dan data item dalam 1 kali request. Hal inilah yg menjadikan GraphQL mampu buat menyelesaikan persoalan *under-fetching*.

1. Delay

Delay adalah parameter QoS menunjukan total waktu yang dibutuhkan paket dalam menempuh jarak dari source ke tujuan. Hal-hal yang dapat mempengaruhi *delay* yaitu perangkat keras, jarak dan *congestion*. Standar *delay* berdasarkan versi TIPHON.

Tabel 1. *Delay standart*

Kategori *Delay*	Besar *Delay*	Indeks
Sangat Bagus	< 150 ms	4
Bagus	150 ms-300 ms	3
Sedang	300 ms-450 ms	2
Tidak Bagus	>450	1

$$Rata - Rata\ Delay = \frac{Jumlah\ Delay}{Jumlah\ Paket\ Diterima}$$

2. *Jitter*

Jitter ataupun variasi *delay* berkaitan dengan *latency*, yang menerangkan banyaknya variasi *delay* dalam transmisi data pada jaringan. *Delay* antrian dalam *router* serta *switch* menghasilkan *jitter*. Hal tersebut dihasilkan oleh variasi variasi waktu mengolah data, panjang antrian beserta waktu

p-ISSN : 2406-7733
e-ISSN : 2597-9922

pengimpunan ulang terhadap paket pada akhir perjalanan *jitter*. Kategori penurunan kinerja jaringan menurut nilai *peak jitter* sebagai berikut:

Tabel 2. *Jitter standart*

Kategori *Jitter*	Besar *Jitter*	Indeks
Sangat Bagus	0 ms	4
Bagus	0 ms – 75 ms	3
Sedang	75 ms – 125 ms	2
Tidak Bagus	125 ms – 225 ms	1

3. HTTP

HTTP adalah protokol untuk *hypertext*. *Server* HTTP umumnya digunakan untuk melayani dokumen *hypertext*, karena HTTP adalah protokol dengan *overhead* yang sangat rendah, sehingga pada kenyataan navigasi informasi dapat ditambahkan langsung ke dalam dokumen dan dengan demikian protokolnya sendiri tidak harus mendukung navigasi secara penuh seperti halnya protokol FTP dan Gopher lakukan (Aryani et al., 2018).

Terdapat *method* yang digunakan dalam HTTP yang disediakan *server* sebagai sarana untuk melakukan *request* terhadap *server*. Berikut adalah *method request* tersebut:

Tabel 3. *Method* HTTP

Method	Keterangan
GET	Digunakan untuk mengambil data
POST	Digunakan untuk mengirimkan entitas ke *server* yang ditentukan, sering menyebabkan perubahan keadaan atau efek samping
PUT	Digunakan untuk menggantikan data yang ada dengan data yang dikirimkan
DELETE	Digunakan untuk menghapus data yang ada

III. METODE PENELITIAN

Penulis dalam penelitian ini menggunakan metode komparatif atau perbandingan untuk menganalisis data. Metode ini bertujuan untuk membandingkan kinerja penggunaan GraphQL dan REST dalam situasi beban tinggi. Adapun tahapan tahapan dari metode ini adalah sebagai berikut:

1. Perancangan uji kasus
   Pada tahap ini, skenario pengujian dirancang untuk menggambarkan kondisi beban tinggi pada aplikasi Portal Berita dan Media. Situasi ini dapat melibatkan permintaan *user* seperti pengambilan artikel, gambar, video, komentar, dan fitur lainnya.
2. Implementasi GraphQL dan REST
   Setelah perancangan uji kasus selesai, kita lanjut ke tahap implementasi GraphQL dan REST. Tahap ini bertujuan untuk membuat *endpoint-endpoint* yang dibuat sesuai dengan permintaan aplikasi tersebut.
3. Pengujian Performa
   Pengujian kinerja aplikasi yang telah dibangun dengan bantuan alat pendukung seperti Apache Jmeter dan alat penguji lainnya dilakukan setelah implementasi GraphQL dan REST. Pengujian ini akan mengukur metrik performa seperti waktu respons, *throughput*, dan penggunaan sumber daya. Ini akan mensimulasikan situasi saat *endpoint* di*hit* secara bersamaan atau dianggap memiliki *load* tinggi.
4. Pengumpulan Data
   Untuk mengetahui seberapa baik kedua metode berfungsi dalam situasi beban tinggi, data dari tes pengujian akan dikumpulkan dan dianalisis.
5. Kesimpulan dan Rekomendasi
   Dari pengumpulan data tersebut akan menyimpulkan hasil perbandingan performa dan menyarankan kapan dan bagaimana menggunakan GraphQL atau REST API dalam kasus aplikasi berita dan media yang memiliki beban tinggi.

1. Tahap Pengembangan GraphQL

Tahap pertama pengembangan arsitektur GraphQL adalah GraphQL *Server*. Implementasi *server* GraphQL mewujudkan 2 pendekatan modal, yaitu skema terlebih dahulu dan kode terlebih dahulu. Pada penelitian ini pendekatan diagram *first* dipilih karena bentuk diagram sudah terdefinisi dengan jelas dan mudah dipahami untuk pengembangan selanjutnya.

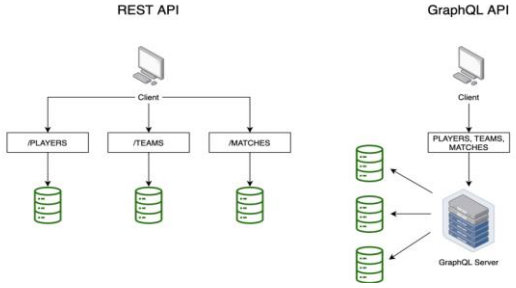
2. Tahap Pengembangan REST API

Untuk melakukan pengembangan REST di dalam aplikasi menggunakan bahasa pemograman express js ada beberapa *package* yang harus diinstal kedalamnya seperti, *sequalize* untuk menjadi ORM, mysql2 *package* untuk menghubungkan ke dalam *database* nya dan juga *package* pelengkap seperti cors,jsonwebtoken,bcryp dan lain sebagainya.

p-ISSN : 2406-7733
e-ISSN : 2597-9922

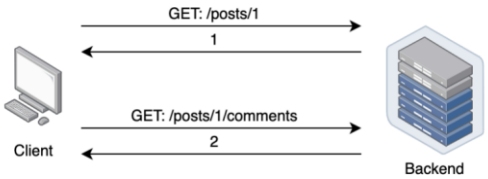
3. Cara Kerja RES & GraphQL

Arsitektur ReST membantu mengidentifikasi sumber daya yang tersedia pada sistem dan menggunakan metode HTTP seperti POST, GET, DELETE, dan UPDATE untuk mengakses dan memanipulasi sumber daya ini. Arsitektur ReST memiliki beberapa titik akhir untuk mengakses sumber daya untuk setiap permintaan ke sistem. Di sisi lain, arsitektur GraphQL menggunakan bahasa kueri untuk mengidentifikasi kebutuhan data spesifik pengguna. Arsitektur GraphQL memiliki beberapa endpoint dengan 2 sistem terpisah yaitu GraphQL *Server* dan GraphQL *Client* untuk mengambil *resource* yang ada.



Gambar 1. Perbandingan cara kerja rest dan graphql

Dalam kasus pertama, penulis akan mewujudkan peran sebuah perusahaan rintisan kecil yang menawarkan aplikasi media sosial media sosial. Perusahaan ini ingin menguji kueri untuk salah satu tampilan mereka dengan GraphQL untuk mengidentifikasi apakah mereka dapat mengharapkan peningkatan kecepatan dengan beralih ke teknologi baru ini. Tampilan yang bersangkutan menampilkan postingan dari pengguna tertentu dan komentarnya. Dengan REST API saat ini, mereka perlu menjalankan dua kueri dari front-end untuk dapat mengambil semua data.

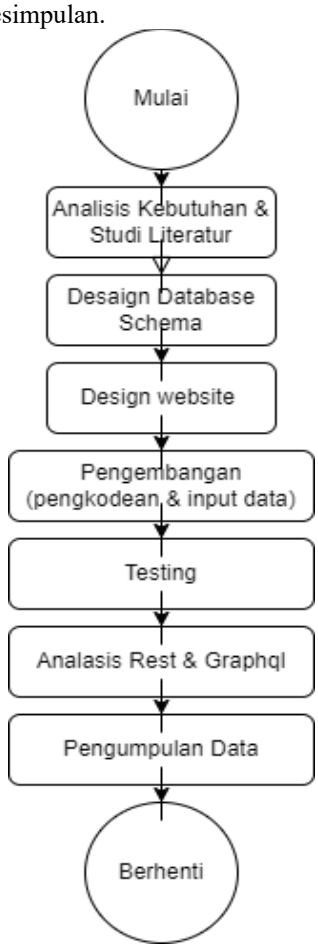


Gambar 2. Perbedaan Pengambilan data

4. Alur Penelitian

Pada alur penelitian ini penulis memulai nya dengan mengalisis terlebih dahulu peneltian terkait perbedaan graphql dan rest API, setelah mendapatkan berbagai macam referensi dari berbagai macam sumber berulah penulis mulai merancang dan membuat alur skema *database* untuk digunakan nantinya di dalam program *website* berita dan media. Setelah *design schema database* barulah penulis membuat desain kasar mengenai *website* portal berita dan media menggunakan *software* mockup dan melanjutkannya dengan pengkodean dan penginputan data yang cukup banyak.

Setelah pengkodean selesai dan aplikasi sudah siap barulah penulis melakukan uji testing antara Graphql dan rest api untuk melihat dari kedua sisi baik itu kecepatan, efisiensi dan lain sebagainya. Barulah penulis mendapatkan hasil dari analisis tersebut yang dikumpulkan hingga menjadi data dan sebagai kesimpulan.



Gambar 3. Alur Penelitian

1. Analisis kebutuhan
   Alur pertama dalam merancang analisis dalam penelitian ini adalah menganalisis kebutuhan dari *software*, *hardware* hingga kebutuhan literatur seperti penelitian terkait atau penelitian terdahulu.
2. *Design Schema Database*
   Design schema database dirancang untuk memenuhi kebutuhan dalam perancangan sistem *backend* yang dimana sistem tersebut menggunakan bahasa pemrograman express.js dan menggunakan *database* MySql sehingga perancangan *database* sangat penting.
3. *Design Website*
4. Setelah perancangan *database* masuk ke *design website* atau mockup dari sistem *frontend* yang akan dibangun untuk membuat *user interface* menjadi lebih menarik. Untuk

p-ISSN : 2406-7733
e-ISSN : 2597-9922

sistem *front end* penulis membuatnya dengan Vue.js.

5. Pengembangan (Pengkodean & Input data)
   Perancangan dari sisi *frontend* dan *backend* sudah selesai maka waktunya mulai pengkodean dimana di tahap ini penulis mulai membangun aplikasi tersebut dan mulai melakukan pengembangan terhadap sistem yang akan dibangun, untuk melengkapi pengkodean tersebut penulis juga melakukan input data yang lumayan banyak.
6. *Testing*
   Testing dilakukan agar sistem yang telah melalui tahap pengkodean selesai agar setelah sistem dibuat tidak adanya *error* kembali dan siap digunakan oleh *user*.
7. Analisis antara REST & GraphQL
   Masuk ke tahap analisis diantara kedua metode tersebut yaitu REST dan GraphQL. Tahap ini penulis melakukan analasis diantara kedua metode tersebut.
8. Pengumpulan data
   Setelah analisis selasai masuk ke dalam pengumpulan data yang diambil dari serangkaian analisis yang sudah dijalankan oleh penulis untuk digunakan pada tahap kesimpulan.

IV. HASIL DAN PEMBAHASAN

1. Tahap Pengujian sistem

Dalam pengujian, sistem bertujuan untuk menguji arsitektur REST API dan GraphQL API untuk mengetahui performa masing-masing web API. Pemeriksaan akan dilakukan berdasarkan empat skenario: GET, POST, PUT, DELETE. Penjelasan untuk setiap situasi adalah sebagai berikut:

1. Skenario Pengujian GET: Pengujian ini dilakukan untuk menerima data dari layanan web yang dibuat dan mengembalikannya ke klien berdasarkan data yang diperlukan untuk keluaran yang diharapkan
2. Skenario Pengujian POST: Pengujian ini dilakukan untuk mengirimkan data ke *server* sesuai dengan permintaan yang diidentifikasi, kemudian *server* akan menyimpan sumber data di layanan web yang dibuat dan klien akan menerima respons jika data berhasil disimpan atau gagal.
3. Skenario Pengujian PUT: Pengujian ini dilakukan dengan mengirimkan data ke *server* sesuai dengan *query* yang telah ditentukan, kemudian *server* akan mengupdate data sesuai dengan sumber data di web *server* yang telah dibuat dan klien akan menerima menerima tanggapan yang menunjukkan apakah data berhasil dimodifikasi atau tidak.
4. Skenario Pengujian DELETE: Pengujian ini dilakukan untuk menghapus sumber data yang ada pada web *server* dengan mengirimkan *primary key* (id) kemudian *client* mendapat respon yang menunjukkan apakah data berhasil dihapus atau gagal.

Tabel 4. Konfigurasi Aplikasi Apache JMeter

Number of Thread	Banyaknya *user* yang akan melakukan permintaan
Loop Count	Pengulangan yang akan dilakukan per *user*
Ramp-Up Period	Total waktu untuk melakukan jumlah dari *Number of Threads*, jadi setiap *user* yang melakukan permintaan akan membutuhkan per satu detik.
HTTP Cookie Manager	Menghapus *cookies* setiap iterasi dilakukan
HTTP Cache Manager	Menghapus setiap iterasi dilakukan dengan maksimal yang dijumlah

Tabel 5. Konfigurasi Aplikasi Apache JMeter

Constant Timer	Sebagai jeda pada *thread* yang ditentukan pada jumlah dalam bentuk *milliseconds*
HTTP Header Manager	Menambahkan HTTP header application/json pada setiap request
CSV Data Set Config	Menyisipkan data dummy yang diperlukan sebagai data pengujian

2. Response Time

Waktu yang dihasilkan menunjukan arsitektur GraphQL lebih cepat dalam menanggapi request data daripada arsitektur REST Api dengan nilai rata-rata 120.535ms sedangkan arsitektur REST Api memiliki nilai rata-rata 265.055ms.

Tabel 6. Hasil *Response Time Method Get* JMeter

No	Arsitekturr	GraphQL	JMeter
1	*Request 1*	114.48	203.60
2	*Request 2*	175.14	260.36
3	*Request 3*	185.95	292.57
4	*Request 4*	192.52	303.69
5	*Request 5*	21.26	28.28
6	*Request 6*	22.98	28.79
7	*Request 7*	22.55	28.20
8	*Request 8*	21.08	27.15

p-ISSN : 2406-7733
e-ISSN : 2597-9922

No	Arsitekturr	GraphQL	JMeter
9	*Request 9*	79.52	113.31
10	*Request 10*	145.71	219.67

Dari sisi *response time*s bisa melihat bahwa GraphQL memiliki *response time* yang lebih cepat ketimbang REST dengan rata rata yang cukup jauh berbeda. Namun, membandingkan waktu respon antara GraphQL dan REST tidak selalu mudah. Kecepatan waktu respon dipengaruhi oleh banyak faktor berbeda, termasuk kompleksitas kueri, jumlah data yang diminta, serta penerapan *server* dan *database*.

Tabel 7. Hasil *Response Time Method Get JMeter*

No	Arsitekturr	GraphQL	JMeter
1	*Request 1*	114.48	203.60
2	*Request 2*	175.14	260.36
3	*Request 3*	185.95	292.57
4	*Request 4*	192.52	303.69
5	*Request 5*	21.26	28.28
6	*Request 6*	22.98	28.79
7	*Request 7*	22.55	28.20
8	*Request 8*	21.08	27.15
9	*Request 9*	79.52	113.31
10	*Request 10*	145.71	219.67

3. *Throughput*

Throughput yang dihasilkan melalui pengujian pada Jmeter menghasilkan *throughput* pada hasil nya GraphQL menghasilkan lebih banyak request yang dihasilkan perdetik ketimbang REST walaupun memang tidak terlalu jauh perbandingannya.

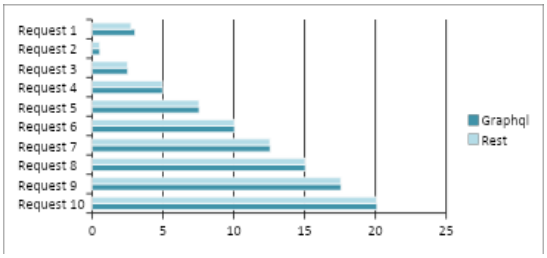
Tabel 8. Hasil *Throughput Method Get JMeter*

No	Arsitekturr	GraphQL	JMeter
1	*Request 1*	80.5	96
2	*Request 2*	75.5	198
3	*Request 3*	250.7	379
4	*Request 4*	500.8	695
5	*Request 5*	877.4	955
6	*Request 6*	1785.83	1888
7	*Request 7*	1101.89	1227
8	*Request 8*	135.01	1509
9	*Request 9*	2191.18	2164
10	*Request 10*	2276.96	2400

Hasil penelitian ini menunjukan bahwa kinerja dari GraphQL lebih baik dibandingkan dengan REST api dalam segi *Response Times* namun dari segi *throughput* REST API tidak terlalu berbeda jauh dengan GraphQl. Dari segi penggunaan dan kesederhanaan dalam pengambilan data maka API adalah jawabannya namun jika kita ingin membuat aplikasi lebih cepat maka GraphQL adalah jawabannya.

REST API menunjukan kestabilitasan meskipun beban nya tinggi namun dibandingkan dengan Graphql pengambilan data yang dilakukan REST API mengalami masalah data yang berlebihan atau terkadang data tersebut tidak dibutuhkan oleh *client*. Selama penulis bekerja memang lebih mudah dan nyama ketika menggunakan metode REST API dibandingkaan GraphQL namun bukan berarti seluruh projek atau aplikasi yang akan dibuat menggunakan REST sebagai *developer* yang baik maka harus bisa menentukan mana yang cocok buat aplikasi yang akan dibangun.

Nilai perbandingan *throughput* antara arsitektur GraphQL dan REST didapatkan bahwa arsitektur Graph lebih banyak memproses transaksi per detik. GraphQL memberi pengembang *fleksibilitas* untuk menentukan struktur dan kedalaman data yang diinginkan dalam kueri. Hal ini berbeda dengan REST, di mana titik akhir tertentu menentukan data apa yang disediakan. *Fleksibilitas* ini dapat mengurangi jumlah permintaan dan respons yang diperlukan, sehingga meningkatkan efisiensi komunikasi antara klien dan *server*.



Gambar 4. *Grafik Request Througput*

Dari hasil pengujian dari CPU Load dan *Memory Utilization* mendapatkan hasil sebagai berikut:

Tabel 8. Hasil CPU *Load* dan *Memory*

No	**Arsitektur**	**GraphQL**		**Rest**	
		CPU Load	*Memory utilization*	*CPU Load*	*Memory utilization*
1	*Request 1*	60.4808	78.9065	61.3213	81.1635
2	*Request 2*	55.9465	76.1889	57.7026	77.6132
3	*Request 3*	56.3663	80.5276	59.6478	80.7463
4	*Request 4*	57.4518	71.5985	58.5182	73.8492
5	*Request 5*	68.8399	59.5707	54.8871	81.1202
6	*Request 6*	51.9634	60.2584	54.8533	90.4845

p-ISSN : 2406-7733
e-ISSN : 2597-9922

No	Arsitektur	GraphQL		Rest	
		CPU Load	*Memory utilization*	*CPU Load*	*Memory utilization*
7	*Request 7*	52.9916	61.7984	34.4672	75.2712
8	*Request 8*	51.9658	58.8059	34.8299	75.4152
9	*Request 9*	58.3600	64.5043	16.8338	74.0986
10	*Request 10*	62.5527	76.4899	26.9870	73.5333

Dari semua hasil tes tersebut dapat disimpulkan bahwa dari sisi CPU Load dan *Memory Utilization* bahwa GraphQL lebih sedikit unggul ketimbang arsitektur REST untuk melihat detail dari semua uji test yang sudah dilakukan bisa dilihat sebagai berikut:

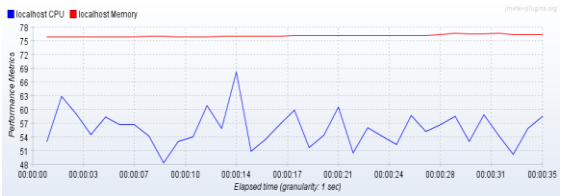
1. *Request* 1

Hasil dari request pertama terdapat hasil percobaan menunjukan hasil dari GraphQL dan menunjukan hasil dari REST. Dari hasil tes ini didapatkan rata-rata GraphQL lebih unggul ketimbang REST.

Gambar 5. Grafik *Request* ke 1

2. *Request* 2

Pada Gambar 6 menunjukkan hasil *request* ke 2 pada GraphQL dan pada Gambar 7 menunjukan hasil dari request ke 2 REST. Dari kedua hasil tersebut menunjukan bahwa GraphQL lebih sedikit menggunakan CPU Load dan Memory ketimbang REST



Gambar 6. Grafik *Request* ke 2

Hasil penelitian merupakan tahapan dimana sistem siap dioperasikan pada tahap yang sebenarnya, sehingga akan diketahui apakah sistem yang telah dibuat benar-benar sesuai dengan yang direncanakan. Penelitian ini mengungkapkan beberapa wawasan penting terkait perbandingan kinerja aplikasi *web* menggunakan GraphQL dan REST API dalam skenario beban tinggi, dengan fokus pada aplikasi berita dan media. Hasil penelitian menyoroti manfaat masing-masing teknologi dalam menghadapi tantangan beban tinggi dan memberikan wawasan penting bagi pengembang aplikasi *web* dan pemangku kepentingan di industri informasi dan komunikasi.

Dalam kondisi beban tinggi, GraphQL menunjukkan *fleksibilitas* dan efisiensi yang mengesankan dalam menangani berbagai permintaan data. Kemampuan untuk menghindari eksploitasi yang berlebihan dan kurang memberikan solusi efektif untuk memenuhi kebutuhan pelanggan. Namun REST API juga memiliki keunggulan dalam beberapa aspek, terutama dalam hal kesederhanaan dan kinerja.

Hasil penelitian ini menunjukan bahwa kinerja dari GraphQL lebih baik dibandingkan dengan REST api dalam segi *Response Times* namun dari segi *throughput* REST API tidak terlalu berbeda jauh dengan GraphQl. Dari segi penggunaan dan kesederhanaan dalam pengambilan data maka API adalah jawabannya namun jika kita ingin membuat aplikasi lebih cepat maka GraphQL adalah jawabannya.

REST API menunjukan kestabilitasan meskipun beban nya tinggi namun dibandingkan dengan Graphql pengambilan data yang dilakukan REST API mengalami masalah data yang berlebihan atau terkadang data tersebut tidak dibutuhkan oleh *client*. Selama penulis bekerja memang lebih mudah dan nyaman ketika menggunakan metode REST API dibandingkaan GraphQL namun bukan berarti seluruh projek atau aplikasi yang akan kita buat menggunakan REST sebagai *developer* yang baik maka kita harus bisa menentukan mana yang cocok buat aplikasi yang akan dibangun.

V. KESIMPULAN DAN SARAN

Kesimpulan

Berdasarkan dari hasil analisis, maka dapat disimpulkan sebagai berikut:

1. Bahwa antara arsitektur GraphQL dan REST, GraphQL memiliki kemampuan membaca lebih cepat dan efisien ketimbang REST pada aplikasi website Berita & Media. Dapat dilihat dari pengujian yang telah dilakukan dan diterapkan pada parameter *response time, troughput, CPU load* dan *memory utilization* yang menerima hasil *response* yang sangat cepat dibandingkan REST.
2. GraphQL memiliki kelebihan dalam hal pengelolaan permintaan yang kompleks dan fleksibilitas dalam mengambil data yang

p-ISSN : 2406-7733
e-ISSN : 2597-9922

dibutuhkan oleh klien. Di sisi lain, REST API menunjukkan kestabilan dalam menghadapi beban tinggi, tetapi dapat memiliki masalah dengan *over-fetching* atau *under-fetching* data.
3. GraphQL mungkin lebih cocok untuk aplikasi yang membutuhkan fleksibilitas dalam mengambil data, sementara REST API dapat lebih sesuai untuk kasus di mana kestabilan dan prediktibilitas sangat diutamakan.

Saran

Berdasarkan penelitian yang sudah dilakukan, beberapa saran yang dapat dijadikan sebagai acuan untuk peneliti selanjutnya sebagai penyempurnaan, sebagai berikut:

1. Peneliti hanya fokus pada kecepatan dan efisiensi antara kedua arsitektur diharapkan peneliti selanjutkan menambahkan parameter yang mungkin dapat menambah referensi untuk memilih antara kedua arsitektur tersebut
2. Peneliti belum menambahkan keaamanan kedalam kedua arsitektur tersebut diharapkan kepada peneliti selanjutnya dapat menambahkan parameter penelitian terhadap keamanan pengiriman data atau authentikasi.

DAFTAR PUSTAKA

Agnihotri, J., & Phalnikar, R. (2018). Development of performance testing suite using apache JMeter. *Advances in Intelligent Systems and Computing*, *673*, 317–326. https://doi.org/10.1007/978-981-10-7245-1_32

Andaru, A. (2018). *Fakultas Komputer Andry Andaru Section Class Content Pengertian Database Secara Umum*.

Aryani, D., Setiadi, A., & Alfiah, F. (2018). Aplikasi Web Pengiriman Dan Penerimaan Sms Dengan Gammu Sms Engine Berbasis Php. *CCIT Journal*, *8*(3), 174–190. https://doi.org/10.33050/ccit.v8i3.340

Bangun, E. P., A Koagouw, F. V. I., & Kalangi, J. S. (2019). Analisis Isi Unsur Kelengkapan Berita Pada Media Online Manadopostonline.com. *Acta Diurna Komunikasi*, *1*(3), 4–13. https://ejournal.unsrat.ac.id/index.php/actadiurnakomunikasi/article/view/25560

Dwi Nurul Huda, Aggry Saputra, & Yulinda. (2020). Perancangan Aplikasi It Help Desk Menggunakan Platform Node.Js Pada Mittasys. *Jurnal Bangkit Indonesia*, *9*(1), 137–143. https://doi.org/10.52771/bangkitindonesia.v9i1.144

Firdausi, A. T., Hormansyah, D. S., Ervansyah, F., Informasi, J. T., & Malang, P. N. (2021). *Implementasi GraphQL Untuk Mengatasi Under- Fetching Pada Pengembangan Sistem Informasi Pelacakan Alumni Politeknik Negeri Malang*. 73–80.

Fitriani, S., Sholahuddin, M. R., & Setiarini, S. D. (2022). Rancang Bangun REST API Aplikasi Sistem Informasi Gardu Distribusi berbasis Android dan Web. *Journal of Information System Research (JOSH)*, *4*(1), 219–226. https://doi.org/10.47065/josh.v4i1.2362

Hanif, F., Ahmad, I., Darwis, D., Putra, I. L., & Ramadhani, M. F. (2023). Analisa Perbandingan Metode Graphql Api Dan Rest Api Dengan Menggunakan Asp.Net Core Web Api Framework. *TELEFORTECH : Journal of Telematics and Information Technology*, *3*(2), 33–37. https://ejurnal.teknokrat.ac.id/index.php/telefortech/article/view/2511

Harliantara. (2019). *Website pada Industri Penyiaran Radio di Indonesia: Live Streaming dan Podcasting English Title: Website Function on Indonesian Radio Broadcasting Industry: Live Streaming and Podcasting*. *3*(1), 82–100. https://doi.org/10.25139/jsk.3i1.983

Hasanuddin, Asgar, H., & Hartono, B. (2022). Rancang Bangun Rest Api Aplikasi Weshare Sebagai Upaya Mempermudah Pelayanan Donasi Kemanusiaan. *Jurnal Informatika Teknologi Dan Sains*, *4*(1), 8–14. https://doi.org/10.51401/jinteks.v4i1.1474

Kurniawan, Y. K., Oslan, Y., & Kristanto, H. (2013). Implementasi Rest - Api Untuk Portal Akademik UKDW Berbasis Android. *Jurnal EKSIS*, *6*, 29–40.

Lawi, A., Panggabean, B. L. E., & Yoshida, T. (2021). Evaluating graphql and rest api services performance in a massive and intensive accessible information system. *Computers*, *10*(11). https://doi.org/10.3390/computers10110138

Masdiyasa, I. G. S., Budiwitjaksono, G. S., M, H. A., Sampurno, I. A. W., & Mandenni, N. M. I. M. (2020). Graph-QL Responsibility Analysis at Integrated Competency Certification Test System Base on Web Service. *Lontar Komputer : Jurnal Ilmiah Teknologi Informasi*, *11*(2), 114. https://doi.org/10.24843/lkjiti.2020.v11.i02.p05

Neumann, A., Laranjeiro, N., & Bernardino, J. (2021). An Analysis of Public REST Web Service APIs. *IEEE Transactions on Services Computing*, *14*(4), 957–970. https://doi.org/10.1109/TSC.2018.2847344

Oggier, C. (2020). *How fast GraphQL is compared to REST APIs (Bachelor thesis)*. 44. https://www.theseus.fi/bitstream/handle/1002

p-ISSN : 2406-7733
e-ISSN : 2597-9922

4/340318/Thesis_Camille_Oggier.pdf?sequence=2

Putra, I. B. A. E. M., Adnyana, M. S. I. D., & Jasa, L. (2021). Analisis *Quality of Service* Pada Jaringan Komputer. *Majalah Ilmiah Teknologi Elektro*, *20*(1), 95. https://doi.org/10.24843/mite.2021.v20i01.p11

Ridlo, I. A. (2017). Pedoman Pembuatan Flowchart. *Academia.Edu*, 27. academia.edu/34767055/Pedoman_Pembuatan_Flowchart

Rosaly, R., & Prasetyo, A. (2019). Pengertian Flowchart Beserta Fungsi dan Simbol-simbol Flowchart yang Paling Umum Digunakan. *Https://Www.Nesabamedia.Com*, *2*, 2. https://www.nesabamedia.com/pengertian-flowchart/https://www.nesabamedia.com/pengertian-flowchart/

Russell, N. C., Schaub, F., McDonald, A., & Sierra-Rocafort, W. (2019). APIs and Your Privacy. *SSRN Electronic Journal*, *January*. https://doi.org/10.2139/ssrn.3328825

Sahi, A. (2020). *TEMATIK - Jurnal Teknologi Informasi Dan Komunikasi Vol. 7, No. 1 Juni 2020*. *7*(1), 120–129.

Soufitri, F. (2019). Perancangan Data Flow Diagram Untuk Sistem Informasi Sekolah (Studi Kasus Pada Smp Plus Terpadu). *Ready Star*, *2*(1), 240–246.

Umami, Z., & Ningrum, N. K. (2021). Pengujian Implementasi Rest Api Pada Website Sistem Pencarian Informasi Produk Fashion Di Shopee. *Jurnal SITECH : Sistem Informasi Dan Teknologi*, *3*(2), 69–82. https://doi.org/10.24176/sitech.v3i2.5671

Wardhana, W. G., Arwani, I., & Rahayudi, B. (2020). Implementasi Teknologi Restful Web Service Dalam Pengembangan Sistem Informasi Perekaman Prestasi Mahasiswa Berbasis Website (Studi Kasus: Fakultas Teknologi Pertanian Universitas Brawijaya). *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer; Vol 4 No 2 (2020)*, *4*(2), 680–689. https://j-ptiik.ub.ac.id/index.php/j-ptiik/article/view/7024%0Ahttp://j-ptiik.ub.ac.id