IMPLEMENTASI GRAPHQL UNTUK MENGATASI *UNDER-FETCHING* PADA PENGEMBANGAN SISTEM INFORMASI PELACAKAN ALUMNI POLITEKNIK NEGERI MALANG

Fany Ervansyah[1], Putra Prima Arhandi[2], ST., M.Kom.
[1]program studi teknik informatika, Politeknik Negeri Malang
email: [1]fanyervansyah.9c@gmail.com, [2]putraprima@gmail.com

Abstract - Data distribution using REST API has several problems. Under-fetching is one of those problems, where the frontend side have to do more than 1 request to get the data they needed. This problem can be solved by implementing GraphQL. The main point of this research is to compare the performance REST API and GraphQL when under-fetching occured. The case study of the research is the development of alumni tracking information system. GraphQL showed a good performance on case where the data is complex and quite large, also when there are many users accessing the data at the same time. On the other hand, REST API showed good performance if the data is simple and not really big. But, if there are many users trying to access data at the same time, REST API can handle less users than GraphQL.

Intisari - Pendistribusian data dengan menggunakan REST API memiliki beberapa kekurangan. Salahsatunya adalah masalah *under-fetching*, dimana bagian *frontend* harus melakukan lebih dari 1 kali *request*. Masalah ini dapat diselesaikan dengan menerapkan GraphQL. Tujuan dari penelitian kali ini adalah untuk membandingkan performa antara sistem informasi dengan REST API dan sistem informasi dengan GraphQL ketika *under-fetching* terjadi. Studi kasus pada penelitian ini adalah pada pengembangan sistem informasi pelacakan alumni Politeknik Negeri Malang. GraphQL menunjukkan performa yang baik pada jumlah data yang besar dan kompleks, serta ketika terdapat banyak pengguna yang mengakses data dalam waktu yang bersamaan. Sedangkan untuk data yang sederhana dan sistem informasi yang tidak memiliki banyak pengguna yang akan mengakses data secara bersamaan, maka REST API masih lebih unggul.

Kata Kunci : *Sistem Informasi, GraphQL, under-fetching*

I. PENDAHULUAN

Distribusi data merupakan hal yang sangat penting dalam pengembangan sebuah sistem informasi. Dalam pendistribusian data, beberapa *website* menerapkan metode yang berbeda-beda. Mulai dari menggabungkan antara bagian yang bertugas menampilkan informasi ke pengguna (*Frontend*) dan bagian yang mengatur bagaimana data - data diolah (*Backend*) hingga menyediakan layanan distribusi data seperti REST API, untuk sistem dengan struktur *frontend* dan *backend* yang terpisah.

Pada tahun 2017, metode yang paling banyak digunakan dalam pendistribusian data adalah dengan menggunakan metode REST API (Motroc, 2017). Namun, dalam metode REST API, terdapat suatu masalah yang disebut *under-fetching*, dimana bagian *frontend* perlu untuk melakukan permintaan data lebih dari 1 kali ke bagian *backend* (Porcello, Banks, 2018). Hal itu akan meningkatkan latensi, yang membuat pengakses website harus menunggu lebih lama sebelum data dikirim pada bagian *frontend* dengan sempurna. Selain itu, kompleksitas program juga semakin bertambah karena bagian *frontend* harus menambahkan 1 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.

Pada tahun 2015, secara publik, Facebook meluncurkan sebuah *query language* yang menjadi metode baru dalam mengatur pendistribusian data. Nama *query language* tersebut adalah GraphQL. Salah satu masalah yang dapat diatasi oleh GraphQL adalah masalah *under-fetching*.

Oleh karena itu, kali ini penulis mencoba mengimplementasikan GraphQL untuk mengatasi *under-fetching* pada Pengembangan Sistem Informasi Pelacakan Alumni Politeknik Negeri

Malang yang akan penulis kembangkan. Dengan adanya GraphQL, diharapkan dapat meningkatkan performa situs web Sistem Informasi Pelacakan Alumni Politeknik Negeri Malang.

II. PENELITIAN TERKAIT

Sejauh ini, penulis menemukan 5 jurnal yang membahas mengenai graphQL. Jurnal - jurnal tersebut adalah:

a. "*An Initial Analysis of Facebook's GraphQL Language*" bertujuan untuk memahami bahasa Graph milik GraphQL dan menunjukkan bahwa bahasa tersebut memiliki kompleksitas yang rendah (Hartig, Pérez, 2017).
b. "*API Design in Distributed Systems: A Comparison between GraphQL and REST*" bertujuan untuk membandingkan REST API dan GraphQL (Eizinger, 2017).
c. "*Improving the OEEU's data-driven technological ecosystem's interoperability with GraphQL*" yaitu menerapkan GraphQL pada Observatory of Employment and Employability, sebuah grup riset untuk lulusan universitas di Spanyol. Hasil dari penerapan GraphQL menunjukkan peningkatan performa, *flexibility*, dan *maintainability* (Vazquez, Cruz, García, 2017).
d. "*Performance of frameworks for declarative data fetching: An evaluation of Falcor and Relay+GraphQL*" membahas tentang perbandingan performa pengambilan data menggunakan Falcor, dan Relay+GraphQL. Falcor merupakan produk buatan Netflix yang juga dapat membantu pengambilan data dari *backend* ke *frontend*. Sedangkan Relay adalah sebuah *framework* untuk mempermudah pengambilan data dari sisi *client* (Cederlund, 2016).
e. "*Using GraphQL for Content Delivery in Kentico Cloud*" bertujuan untuk meriset GraphQL sebagai alternatif untuk mengantar konten pada Kentico Cloud, selain menggunakan API yang sudah disediakan oleh Kentico Cloud, yaitu sebuah CMS online yang menyediakan konten sebagai *service*. Kentico menyediakan REST API untuk melakukan pengambilan data, namun di sini, dicoba diterapkan GraphQL (Čechák, 2017).

III. METODE PENELITIAN

A. Metode Pengambilan Data

Pengambilan data dilakukan dengan cara *scraping* pada situs Linkedin dan input secara manual melalui sistem informasi yang akan dibuat.

Pada pengambilan data di Linkedin, data yang diambil adalah data mahasiswa yang sudah mendaftarkan Politeknik Negeri Malang sebagai riwayat pendidikan mereka. Mekanisme pengambilan datanya adalah dengan menggunakan bot untuk melakukan login terotomatisasi, kemudian masuk ke halaman Politeknik Negeri Malang, lalu membuka halaman detail setiap *card* dari mahasiswa yang muncul. Dari halaman detail yang sudah dibuka, akan diambil data-data yang diperlukan.

Pada pengambilan data dari input manual, pengambilan data dilakukan dengan mengisikan data alumni melalui *form* yang sudah disediakan di sistem. *Form* tersebut memiliki *field* diantaranya berupa nama, tahun masuk dan lulus dari Politeknik Negeri Malang, jurusan yang diambil, pekerjaan saat ini, jabatan/posisi yang dipegang pada pekerjaan saat ini, dan email yang bisa dihubungi.

B. Metode Pengujian

Pengujian pertama akan dilakukan dengan cara memuat suatu halaman yang sama dan memiliki masalah *under-fetching*, pada Sistem Informasi yang menggunakan GraphQL maupun yang menggunakan REST sebagai metode pendistribusian datanya. Kemudian, halaman akan di-*refresh* sebanyak 20 kali. Pada masing-masing sesi *refresh,* akan ada beberapa data yang dicatat pada masing-masing sistem informasi. Diantaranya adalah jumlah *request* ke *backend*, waktu muat *request* hingga selesai, dan waktu total memuat halaman. Dari data-data tersebut, akan dirata-rata pada masing-masing sistem informasi dan akan dibandingkan. Sehingga, dapat diketahui kelebihan/kekurangan dari graphQL dan REST jika digunakan dalam pendistribusian data.

Uji coba kedua adalah dengan menguji kemampuan *backend* dari masing-masing metode pendistribusian data dalam menangani jumlah *user* yang melakukan permintaan data secara bersamaan. Jumlah *user* yang disimulasikan sejumlah 50 *users* dan batas durasi yang diberikan adalah 1 menit. Sehingga, 50 *users* akan melakukan *request* secara bersamaan dan selama 1 menit, akan dicatat berapa *user* yang mendapatkan respon dari *backend*.

GraphQL dikatakan berhasil menyelesaikan masalah *under-fetching* jika sistem yang dikembangkan dengan mengimplementasikan GraphQL dapat melakukan pengaksesan *endpoint*

lebih sedikit daripada sistem dengan metode REST API dalam memenuhi kebutuhan datanya atau jika server dengan GraphQL dapat merespon sejumlah *user* yang lebih banyak dibandingkan dengan REST API.

IV. HASIL DAN PEMBAHASAN

A. Hasil Uji Muat Halaman Dashboard

Pada halaman *dashboard* di sistem informasi dengan REST, terjadi 5 kali *request* dengan data yang sederhana, yaitu berupa total dari beberapa kategori data, dan ditampilkan sekaligus.

Hasil akhir uji muat halaman *dashboard* menunjukkan bahwa GraphQL berhasil mengatasi masalah *under-fetching* dengan hanya memerlukan 1 *endpoint* saja untuk memenuhi semua kebutuhan datanya. Sedangkan REST membutuhkan 5 kali *request* ke *backend* untuk memenuhi kebutuhan datanya. Namun, kecepatan muat halaman pada sistem informasi dengan REST masih lebih cepat dibandingkan sistem informasi dengan GraphQL. Selisih waktu dari masing-masing rata-rata kecepatan muat halaman adalah 12,7 milisekon dengan sistem informasi yang REST lebih unggul.

Dari hasil tersebut, terlihat bahwa GraphQL dapat mengatasi masalah *under-fetching*. Namun, untuk kasus data yang sederhana, kasus *under-fetching* tidak memberikan dampak yang mempengaruhi performa. Sehingga, REST API masih lebih cepat dibandingkan GraphQL dalam pemuatan data untuk ditampilkan pada halaman yang dimaksud.

B. Hasil Uji Muat Halaman List Alumni

Pada halaman *list* alumni, terdapat masalah *under-fetching* pada sistem informasi dengan REST. Dimana, sistem harus melakukan 2 kali *request* ke *backend* untuk memenuhi kebutuhan yang berupa data beberapa alumni dengan sumber yang berbeda, pengaturan halaman, total halaman dan total data yang didapat sekaligus. Hal ini menunjukkan bahwa pada halaman *list* alumni, terdapat data yang lebih kompleks dan besar dibanding pada halaman *dashboard*, namun dengan kasus *under-fetching* tidak sebesar pada halaman *dashboard* karena hanya terdapat 2 kali *request* ke *backend*.

Hasil uji coba muat halaman list alumni menunjukkan bahwa GraphQL berhasil mengatasi masalah *under-fetching* dengan hanya melakukan 1 kali *request* ke *backend* untuk memenuhi kebutuhan datanya. Sedangkan sistem informasi dengan REST membutuhkan 2 kali *request* ke bagian *backend* untuk memenuhi kebutuhan data yang akan ditampilkan. Perbandingan rata-rata kecepatan muat halaman pada masing-masing sistem menunjukkan bahwa GraphQL lebih cepat dibandingkan dengan REST API. Selisih dari rata-rata kecepatan muat halaman dari masing-masing sistem informasi adalah 453,5 milisekon dengan dengan GraphQL yang lebih unggul.

Dari hasil tersebut, dapat dilihat bahwa, untuk data yang besar dan lebih kompleks dibandingkan dengan halaman *dashboard*, kasus *under-fetching* mulai memberikan dampak pada performa suatu halaman web dan GraphQL mampu mengatasi masalah tersebut sekaligus meningkatkan performa suatu halaman web. Sehingga, waktu muat halaman web pada sistem informasi dengan GraphQL dapat lebih singkat dibandingkan dengan waktu muat halaman web dengan REST API.

C. Hasil Uji Muat Halaman Edit Alumni

Pada halaman ini, terdapat kasus *under-fetching* dimana sistem informasi dengan REST melakukan 2 kali *request* dengan data yang tidak terlalu besar dibandingkan dengan halaman *list* alumni, yaitu meminta *detail* dari data alumni yang akan diubah dan meminta *list* jurusan yang sudah terdaftar pada sistem. Artinya, pada halaman *edit* alumni ini terdapat kasus *under-fetching* yang sama dengan halamn *list* alumni, yaitu 2 kali *request*, namun datanya tidak sebesar data pada halaman *list* alumni.

Hasil uji muat halaman *edit alumni* menunjukkan bahwa GraphQL dapat mengatasi masalah *under-fetching* dengan hanya membutuhkan 1 kali *request* saja ke *backend* untuk memenuhi data yang diperlukan. Sedangkan sistem informasi dengan REST API membutuhkan 2 kali *request* untuk memenuhi kebutuhan data yang akan ditampilkan. Selisih rata-rata waktu muat halaman pada sistem informasi dengan REST API dan sistem informasi dengan GraphQL adalah 40,85 milisekon dengan REST API lebih unggul dibandingkan GraphQL.

Dari data tersebut, dapat dilihat bahwa untuk kasus data yang lebih kecil dibandingkan pada halaman *list* alumni dengan jumlah *request* yang sama, yaitu 2 kali *requests*, kasus under-fetching tidak memberikan pengaruh yang besar pada performa halaman web, namun lebih memberikan pengaruh pada efisiensi penggunaan kuota data internet. Sehingga, meskipun kasus *under-fetching* dapat diatasi, kecepatan muat halaman pada sistem

informasi dengan GraphQL masih sedikit lebih lambat dibandingkan dengan sistem informasi dengan REST API.

D. Hasil Load Testing Endpoint Halaman Dashboard

Hasil *load testing endpoint* halaman *dashboard* pada sistem informasi dengan REST dan GraphQL terdiri dari beberapa kategori dengan perlakuan yang sama pada masing-masing sistem. Dilakukan permintaan ke *backend* oleh 50 *user*. Ketika 1 *user* telah menyelesaikan *request* data yang dibutuhkan dan mendapatkan respon dari *backend*, *user* akan segera melakukan request kembali ke *backend*. Perlakuan ini akan terulang secara terus-menerus selama 1 menit.

Hasil dari uji coba pada *backend* dengan REST API menunjukkan bahwa terjadi 151 iterasi permintaan dari 50 *user* dengan jumlah *request* sebanyak 755 dan semua permintaan berhasil mendapatkan respon sukses dari *backend*. Rata-rata waktu yang dibutuhkan untuk masing-masing iterasi adalah 16 detik dengan waktu maksimal 21 detik dan waktu minimal 4 detik.

Hasil dari *load testing backend* dengan GraphQL menunjukkan bahwa terjadi 156 iterasi pengguna dengan jumlah *request* sebanyak 156 kali dan semua iterasi mendapatkan respon sukses dari *backend*. Rata-rata waktu yang dibutuhkan untuk setiap iterasi adalah 16 detik dengan waktu maksimal 20 detik dan waktu minimal 2 detik.

Dari segi eifisiensi penggunakan kuota internet, jumlah data yang diterima pada sistem informasi dengan REST API adalah 1012434 bytes dan jumlah data yang dikirimkan adalah 361524 bytes. Pada sistem informasi dengan GraphQL, data yang diterima adalah 222666 bytes dan data jumlah data yang dikirmkan adalah 108636 bytes.

Dari percobaan tersebut, dapat dilihat bahwa GraphQL lebih mampu dalam memberikan respon ke *user* dengan jumlah respon sebanyak 156 kali. Jumlah ini lebih banyak 5 iterasi dibandingkan dengan respon yang berhasil diberikan oleh backend dengan REST API. Untuk kecepatan respon, *backend* dengan GraphQL mapun *backend* dengan REST API memiliki rata-rata kecepatan yang sama. Dari segi efisiensi kuota data internet, dapat dilihat bahwa GraphQL mengirimkan dan menerima data lebih efisien dari REST API. Hal ini terjadi karena pada masing-masing *request* di sistem informasi dengan REST, terjadi validasi token dan token yang sama dikirimkan berkali-kali ke *backend*. Karena mengakses banyak endpoint, maka sistem informasi dengan REST menerima berbagai data dengan tambahan *header* data pada masing-masing respon. Sedangkan pada GraphQL, karena hanya terjadi 1 kali request, maka token hanya dikirmkan sekali dan juga mendapatkan keseluruhan data dalam 1 respon.

E. Hasil Load Testing Endpoint Halaman List Alumni

Hasil *load testing endpoint* halaman *list* alumni pada sistem informasi dengan REST dan GraphQL terdiri dari beberapa kategori dengan perlakuan yang sama pada uji coba sebelumnya. Dilakukan permintaan ke *backend* oleh 50 *user*. Ketika 1 *user* telah menyelesaikan *request* data yang dibutuhkan dan mendapatkan respon dari *backend*, *user* akan segera melakukan request kembali ke *backend*. Perlakuan ini akan terulang selama 1 menit.

Hasil dari percobaan pada backend dengan REST menunjukkan bahwa terjadi 291 iterasi permintaan dari 50 user dengan jumlah request sebanyak 602 request, dengan 582 request berhasil mendapatkan respon sukses dari backend, sedangkan sisanya masih belum selesai diproses. Rata-rata durasi waktu yang dibutuhkan untuk masing-masing iterasi adalah 9,5 detik dengan waktu maksimal 13,7 detik dan waktu tersingkat 7 detik.

Pada *backend* dengan GraphQL, terjadi 309 iterasi permintaan dari 50 *user* dengan jumlah *request* 309 kali serta keseluruhannya mendapat respon sukses dari *backend*. Rata-rata durasi waktu yang diperlukan untuk masing-masing iterasi adalah 8,9 detik dengan waktu maksimal 17,8 detik dan waktu minimal 1,7 detik.

Dari segi efisiensi kuota internet, jumlah data yang diterima pada sistem informasi dengan REST adalah 5220098 bytes dan data yang dikirimkan adalah 232650 bytes. Sedangkan pada sistem informasi dengan GraphQL, data yang diterima adalah 4263453 bytes dan data yang dikirimkan adalah 321321 bytes.

Dari hasil di atas, dapat dilihat bahwa GraphQL mampu menangani jumlah *user* yang lebih banyak dibanding dengan REST API di mana GraphQL mampu merespon hingga iterasi ke 309, sedangkan REST mampu merespon *request* hingga iterasi ke-291. untuk durasi respon, GraphQL dan REST API memiliki selisih 0,6 detik dengan GraphQL yang lebih unggul. Dari segi efisiensi kuota data, dapat dilihat bahwa untuk data yang diterima, GraphQL lebih hemat

sumber daya. Namun, pada bagian data yang dikirmkan, REST API lebih unggul. Hal ini terjadi karena GraphQL mengirimkan data berupa *query* pada *backend*, sedangkan REST hanya mengirimkan *endpoint* apa yang dituju. Sehingga, REST menggunakan kuota data lebih sedikit dibandingkan dengan GraphQL. Namun, karena GraphQL mengirimkan *query* ke *backend*, maka GraphQL dapat menentukan data apasaja yang dibutuhkan oleh *frontend*. Sehingga, tidak terjadi kelebihan data dari jumlah data yang diperlukan. Hal inilah yang membuat GraphQL dapat menghemat kuota lebih baik dari REST ketika menerima data.

F. Hasil Load Testing Endpoint Halaman Edit Alumni

Load testing pada *endpoint* halaman *edit* alumni dilakukan dengan menerapkan perlakuan yang sama pada *load testing endpoint* halaman *list* alumni dan halaman *dashboard*, dimana terdapat 50 *user* yang akan melakukan *request* ke bagian *backend*. Setiap 1 *user* selesai mendapatkan respon dari *backend, user* tersebut akan segera melakukan *request* kembali. Hal ini akan terus berulang selama 1 menit.

Hasil yang diberikan oleh *backend* dengan REST menunjukkan bahwa terjadi 410 iterasi dengan jumlah *request* sebanyak 821 dan sebanyak 820 *request* berhasil direspon dengan sukses oleh *backend*. Sedangkan sisanya belum selesai direspon. Rata-rata kecepatan tiap iterasi adalah 6,8 detik dengan waktu terlama 8,7 detik dan waktu tercepat 1,8 detik.

Sedangkan hasil yang diberikan oleh *backend* dengan GraphQL menunjukkan bahwa terjadi 415 iterasi dengan jumlah *request* 415 kali dan keseluruhannya mendapatkan respon sukses dari *backend*. Rata-rata durasi tiap iterasi adalah 6,8 detik dengan durasi terlama 8,5 detik dan durasi tersingkat adalah 0,5 detik.

Dari segi efisiensi kuota internet, sistem informasi dengan REST API menerima data sebesar 1697150 bytes dan mengirimkan data sebesar 316560 bytes. Pada sistem informasi dengan GraphQL, data yang diterima sebesar 956845 bytes dan data yang dikirimkan adalah 344355 bytes. Dari data tersebut, dapat dilihat bahwa GraphQL mampu menghemat penggunaan data yang diterima hampir 2 kali dari REST API.

Dari hasil tersebut, dapat dilihat bahwa GraphQL dapat melakukan lebih banyak iterasi dibandingkan REST API yang artinya GraphQL dapat memanajemen *request* dari *user* dengan baik ketika terdapat banyak *user* yang mengakses data secara bersamaan. Untuk rata-rata kecepatan, kedua sistem *backend* memiliki rata-rata kecepatan yang sama. Dari segi efisiensi penggunaan kuota internet, GraphQL cukup unggul dibandingkan REST API.

V. KESIMPULAN

Dari hasil *load test* sistem informasi yang sudah dibuat dengan GraphQL dan sistem informasi dengan REST API, dapat diambil kesimpulan bahwa, pada proses pemuatan halaman dengan data yang besar dan beragam, kasus *under-fetching* memberikan dampak pada durasi pemuatan halaman dan GraphQL mampu menangani pemuatan halaman lebih cepat. Pada kasus pemuatan halaman dengan data yang sederhana seperti pada halaman *dashboard* dan *edit* alumni, masalah *under-fetching* tidak memberikan dampak pada kecepatan muat halaman dan REST API masih lebih unggul.

Pada *load test endpoint* masing-masing *backend*, dapat diambil kesimpulan bahwa GraphQL mampu melayani lebih banyak *user* dibandingkan dengan REST API. Ini dibuktikan dengan lebih banyaknya iterasi yang terjadi pada hasil *load test backend* GraphQL dibandingkan dengan jumlah iterasi yang terjadi pada hasil *load test backend* dengan REST API. Kemudian, dari segi efisiensi penggunakan kuota internet, *under-fetching* memberikan masalah berupa pengiriman data pada beberapa respon. Sehingga, selain *frontend* harus mengirimkan token yang sama untuk divalidasi ke *backend* pada masing-masing *endpoint*, *frontend* juga akan mendapatkan data dari berbagai respon dengan berbagai *header*. Hal ini menyebabkan pembuangan kuota internet untuk mengirimkan dan mendapatkan data yang tidak perlu. GraphQL dapat mengatasi hal ini dengan menggunakan 1 *endpoint* dan mengirimkan *query* yang berisikan kebutuhan data apa saja yang kita inginkan untuk digunakan di *frontend*.

Dengan begitu, dapat diambil kesimpulan bahwa, untuk suatu halaman web yang sederhana dan dengan jumlah *user* yang tidak terlalu banyak dalam satu waktu akses, REST API masih dapat diandalkan sebagai metode distribusi data. Namun, ketika aplikasi atau *website* tersebut sudah mulai berkembang menjadi lebih kompleks dengan data yang cukup banyak dan dengan jumlah pengguna yang besar dalam satu waktu akses, serta berpeluang besar untuk terjadi *under-fetching*, GraphQL mulai diperlukan sebagai solusi dalam

mengatasi *under-fetching* dan mengoptimalkan kemampuan *server* dalam memberikan respon secara cepat kepada pengguna serta menghemat penggunaan kuota data internet oleh *user*. Sehingga, pengguna akan lebih nyaman dan merasa betah berada di halaman web yang dibuat. Karena pengguna merasa lebih nyaman, maka tentu hal ini akan dapat meningkatkan reputasi dari halaman web yang dikembangkan.

DAFTAR PUSTAKA

Banks, A., & Porcello, E. (2017). Learning React: functional web development with React and Redux. Sebastopol, CA: O'Reilly.

Brown, E. (2014). Web Development with Node and Express. Sebastopol, CA: O'Reilly & Associates.

Čechák, D. (2017). Using GraphQL for Content Delivery in Kentico Cloud.

Cederlund, M. (2016). Performance of frameworks for declarative data fetching An evaluation of Falcor and Relay GraphQL.

Chodorow, K., & Dirolf, M. (2015). MongoDB: the definitive guide. Sebastopol, CA: O'Reilly.

Eizinger, T. (2017). Api Design in Distributed Systems: A Comparison between GraphQL and Rest.

Hartig, O., & Pérez, J. (2017). An Initial Analysis of Facebook's GraphQL Language.

Kniberg, H., Skarin, M., Poppendieck, M., & Anderson, D. (2010). Kanban and Scrum: making the most of both. United States: InfoQ.

Porcello, E., & Banks, A. (2018). Learning GraphQL: declarative data fetching for modern web apps. Cambridge: OReilly.

Motroc, G. (2017). The State of API Integration: SOAP vs. REST ... - JAXenter. Diakses pada 2 Desember 2019, dari https://jaxenter.com/state-of-api-integration-report-136342.html

Vazquez, A., Cruz, J., & García, F. J. (2017). Improving the Oeeu's data-driven technological ecosystem's interoperability with GraphQL.