

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

SKRIPSI

Digunakan Sebagai Syarat Maju Ujian Diploma IV
Politeknik Negeri Malang

Oleh:

FANY ERVANSYAH. NIM. 1641720080



**PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNOLOGI INFORMASI
POLITEKNIK NEGERI MALANG
JUNI 2020**

HALAMAN PENGESAHAN

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

Disusun oleh:

FANY ERVANSYAH. NIM. 1641720080

Laporan Akhir ini telah diuji pada tanggal 30 Juni 2020

Disetujui oleh:

- | | | | |
|------------------|---|---|-------|
| 1. Penguji I | : | <u>Ridwan Rismanto, S.ST., M.Kom.</u>
NIP. 198603182012121001 | |
| 2. Penguji II | : | <u>Rosa Andrie Asmara, ST., MT., Dr. Eng.</u>
NIP. | |
| 3. Pembimbing I | : | <u>Putra Prima Arhandi, S.T., M.Kom.</u>
NIP. 198611032014041001 | |
| 4. Pembimbing II | : | <u>Dhebys Suryani H, S.Kom., MT.</u>
NIP. 198311092014042001 | |

Mengetahui,

Ketua Jurusan
Teknologi Informasi

Ketua Program Studi
Teknik Informatika

Rudy Ariyanto, S.T., M.Cs.
NIP. 19711110 199903 1 002

Imam Fahrur Rozi, S.T., M.T.
NIP. 19840610 200812 1 004

PERNYATAAN

Dengan ini saya menyatakan bahwa pada Skripsi ini tidak terdapat karya, baik seluruh maupun sebagian, yang sudah pernah diajukan untuk memperoleh gelar akademik di Perguruan Tinggi manapun, dan sepanjang pengetahuan saya juga tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini serta disebutkan dalam daftar sitasi/pustaka.

Malang, 7 Juli 2020

Fany Ervansyah

ABSTRAK

Ervansyah, Fany. “Implementasi Graphql Untuk Mengatasi Under-Fetching Pada Pengembangan Sistem Informasi Pelacakan Alumni Politeknik Negeri Malang”. **Pembimbing: (1) Putra Prima Arhandi, S.T., M.Kom., (2) Dhebys Suryani H, S.Kom., MT.**

Skripsi, Program Studi Teknik Informatika, Jurusan Teknologi Informasi, Politeknik Negeri Malang, 2020.

Pendistribusian data pada sistem informasi memiliki berbagai macam cara yang digunakan. Diantaranya adalah dengan menggunakan metode REST API. Namun, terdapat beberapa kekurangan yang menjadi masalah pada REST API. Salahsatunya adalah masalah *under-fetching*, yaitu masalah dimana bagian *frontend* harus melakukan lebih dari 1 kali *request* untuk memenuhi kebutuhan data yang diperlukan. Masalah ini dapat diselesaikan dengan menerapkan GraphQL sebagai metode pendistribusian data. Tujuan dari penelitian kali ini adalah untuk membandingkan performa antara sistem informasi dengan REST API dan sistem informasi dengan GraphQL. Studi kasus pada penelitian ini adalah pada pengembangan sistem informasi pelacakan alumni Politeknik Negeri Malang.

Hasil penelitian ini berupa perbandingan performa antara sistem informasi dengan metode pendistribusian data REST API dan sistem informasi dengan metode pendistribusian data GraphQL. 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

ABSTRACT

Ervansyah, Fany. “*Graphql Implementation To Solve Under-Fetching In Development Of State Polytechnic of Malang Alumni Searcher Information System*”. **Counseling Lecturer:** (1) **Putra Prima Arhandi, S.T., M.Kom.,** (2) **Dhebys Suryani H, S.Kom., MT.**

Thesis, Informatics Management Study Program, Department of Information Technology, State Polytechnic of Malang, 2020.

Data distribution of information system has many ways to implement. One of those ways is using REST API as data distribution method. But, there are several problems the REST API has. 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 as data distribution method to the system. The main point of this research is to compare the performance of information system using REST API as its data distribution method and information system using GraphQL as its data distribution method. The study case of the research is in the development of alumni tracking information system.

The result of this research is a comparison between both information system that have been developed earlier. 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.

Keywords: *Information System, GraphQL, Under-fetching*

KATA PENGANTAR

Puji Syukur atas kehadiran Allah SWT/Tuhan YME karena atas segala rahmat dan hidayah-Nya, penulis dapat menyelesaikan skripsi dengan judul “IMPLEMENTASI GRAPHQL UNTUK MENGATASI UNDER-FETCHING PADA PENGEMBANGAN SISTEM INFORMASI PELACAKAN ALUMNI POLITEKNIK NEGERI MALANG”. Skripsi ini penulis susun sebagai persyaratan untuk menyelesaikan studi program Diploma IV Program Studi Teknik Informatika, Jurusan Teknologi Informasi, Politeknik Negeri Malang.

Penulis menyadari bahwasannya tanpa adanya dukungan dan kerja sama dari berbagai pihak, kegiatan laporan akhir ini tidak akan dapat berjalan baik. Untuk itu, penulis ingin menyampaikan rasa terima kasih kepada:

1. Bapak Rudy Ariyanto, ST., M.Cs., selaku ketua jurusan Teknologi Informasi.
2. Bapak Imam Fahrur Rozi, ST., MT., selaku ketua program studi Manajemen Informatika.
3. Putra Prima Arhandi, S.T., M.Kom., selaku pembimbing 1.
4. Dhebys Suryani H, S.Kom., MT., selaku pembimbing 2.
5. Dan seluruh pihak yang telah membantu lancarnya pembuatan Laporan Akhir dari awal hingga akhir yang tidak dapat penulis sebutkan satu persatu.

Penulis menyadari bahwa dalam penyusunan laporan akhir ini, masih banyak terdapat kekurangan dan kelemahan yang dimiliki penulis baik itu sistematika penulisan maupun penggunaan bahasa. Untuk itu penulis mengharapkan saran dan kritik dari berbagai pihak yang bersifat membangun demi penyempurnaan laporan ini. Semoga laporan ini berguna bagi pembaca secara umum dan penulis secara khusus. Akhir kata, penulis ucapkan banyak terima kasih.

Malang, 2020

Penulis

DAFTAR ISI

	Halaman
SAMPUL DEPAN.....	Error! Bookmark not defined.
HALAMAN PENGESAHAN.....	ii
PERNYATAAN.....	iii
ABSTRAK.....	iv
ABSTRACT.....	v
KATA PENGANTAR.....	vi
DAFTAR ISI.....	vii
DAFTAR GAMBAR.....	ix
DAFTAR TABEL.....	x
DAFTAR LAMPIRAN.....	xi
BAB I. PENDAHULUAN.....	1
1.1 Latar Belakang.....	1
1.2 Rumusan Masalah.....	2
1.3 Tujuan.....	2
1.4 Batasan Masalah.....	2
1.5 Sistematika Penulisan.....	3
BAB II. LANDASAN TEORI.....	4
2.1 Teori.....	4
2.2 Penelitian Terdahulu.....	5
BAB III. METODOLOGI PENELITIAN.....	7
3.1 Metode Pengambilan Data.....	7
3.2 Metode Pengembangan Sistem.....	7
3.3 Fase pengembangan sistem.....	8
3.4 Proses Pengujian.....	8
BAB IV. ANALISIS DAN PERANCANGAN.....	10
4.1 Kebutuhan Fungsional.....	10
4.1.1 Administrator.....	10
4.1.2 Pengguna.....	10
4.2 Kebutuhan Non-Fungsional.....	10
4.3 Rancangan Sistem.....	11
4.3.1 Arsitektur Sistem.....	11
4.3.2 Use Case.....	12
4.3.3 Struktur Data.....	13
4.3.4 Epic.....	15
4.3.5 User Story.....	16

BAB V. IMPLEMENTASI DAN PENGUJIAN.....	19
5.1 Implementasi.....	19
5.1.1 Pembuatan Scraper Linkedin.....	19
5.1.2 Pengunggahan Database ke Cloud.....	21
5.1.3 Pengembangan Backend REST.....	22
5.1.4 Pengembangan Backend GraphQL.....	25
5.1.5 Pengembangan Frontend REST.....	28
5.1.6 Pengembangan Frontend GraphQL.....	30
5.1.7 Tampilan Halaman.....	34
5.2 Pengujian.....	40
5.2.1 Uji Muat Halaman Dashboard.....	41
5.2.2 Uji Muat Halaman List Alumni.....	43
5.2.3 Uji Muat Halaman Ubah Data Alumni.....	45
5.2.4 Load Test Endpoint Untuk Halaman Dashboard.....	48
5.2.5 Load Test Endpoint Untuk Halaman List Alumni.....	48
5.2.6 Load Test Endpoint Untuk Halaman Edit Alumni.....	48
BAB VI. HASIL DAN PEMBAHASAN.....	49
6.1 Hasil Uji Muat Halaman Dashboard.....	49
6.2 Hasil Uji Muat Halaman List Alumni.....	49
6.3 Hasil Uji Muat Halaman Edit Alumni.....	50
6.4 Hasil Load Testing Endpoint Halaman Dashboard.....	51
6.5 Hasil Load Testing Endpoint Halaman List Alumni.....	52
6.6 Hasil Load Testing Endpoint Halaman Edit Alumni.....	53
BAB VII. KESIMPULAN DAN SARAN.....	55
7.1 Kesimpulan.....	55
7.2 Saran.....	56
DAFTAR PUSTAKA.....	57

DAFTAR GAMBAR

	Halaman
Gambar 4.1 Arsitektur Sistem.....	11
Gambar 4.2 Use Case Pengguna.....	12
Gambar 4.3 Use Case Administrator.....	13
Gambar 5.1 Struktur proyek backend REST.....	22
Gambar 5.2 potongan kode Router untuk /major.....	24
Gambar 5.3 Sub-router dari /major.....	24
Gambar 5.4 Controller untuk endpoint major.....	24
Gambar 5.5 Struktur Proyek backend dengan GraphQL.....	25
Gambar 5.6 Query untuk mengambil data detail jurusan.....	27
Gambar 5.7 Menyediakan query majorDetail.....	27
Gambar 5.8 type dari major.....	27
Gambar 5.9 Struktur proyek frontend REST API.....	28
Gambar 5.10 Playground GraphQL.....	32
Gambar 5.11 Dokumentasi GraphQL.....	33
Gambar 5.12 Schema GraphQL.....	33
Gambar 5.13 Halaman login dengan isi.....	35
Gambar 5.14 Halaman login.....	35
Gambar 5.15 Halaman Dashboard.....	36
Gambar 5.16 Halaman tambah data alumni.....	36
Gambar 5.17 Halaman list alumni.....	36
Gambar 5.18 Halaman edit data alumni.....	37
Gambar 5.19 Halaman list jurusan.....	37
Gambar 5.20 Halaman tambah jurusan.....	37
Gambar 5.21 Halaman ubah data jurusan.....	38
Gambar 5.22 Halaman list pengguna.....	38
Gambar 5.23 Halaman tambah data pengguna.....	38
Gambar 5.24 Halaman ubah data pengguna.....	39
Gambar 5.25 Halaman reset password pengguna.....	39
Gambar 5.26 Halaman list alumni oleh non-administrator.....	39
Gambar 5.27 Halaman list jurusan oleh non-administrator.....	40

DAFTAR TABEL

	Halaman
Tabel 4.1 Desain Collection alumni_inputs.....	14
Tabel 4.2 Desain Collection alumni_linkedins.....	14
Tabel 4.3 Desain Collection major.....	14
Tabel 4.4 Desain Collection user.....	15
Tabel 5.1 Hasil uji coba muat halaman dasboard REST API.....	42
Tabel 5.3 Hasil uji coba muat halaman list alumni REST API.....	44
Tabel 5.4 Hasil uji coba muat halaman list alumni GraphQL.....	45
Tabel 5.5 Hasil uji coba muat halaman ubah data alumni REST API.....	46
Tabel 5.6 Hasil uji coba muat halaman ubah data alumni GraphQL.....	47

DAFTAR LAMPIRAN

Lampiran 1	Kode tsconfig.json
Lampiran 2a	Kode untuk menyediakan resolver dari majorDetail
Lampiran 2b	Logic dari resolver majorDetail
Lampiran 3	Mengambil data jurusan dengan http method GET
Lampiran 4	Contoh Query untuk halaman Dashboard
Lampiran 5	Contoh penggunaan UseQuery
Lampiran 6a	Hasil Load Testing Endpoint Dashboard REST
Lampiran 6b	Hasil Load Testing Endpoint Dashboard GraphQL
Lampiran 7a	Hasil Load Testing Endpoint List Alumni REST
Lampiran 7b	Hasil Load Testing Endpoint List Alumni GraphQL
Lampiran 8a	Hasil Load Testing Endpoint Ubah Data Alumni REST
Lampiran 8b	Hasil Load Testing Endpoint Ubah Data Alumni GraphQL

BAB I. PENDAHULUAN

1.1 Latar Belakang

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 akses 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.

1.2 Rumusan Masalah

Berdasarkan latar belakang diatas, maka rumusan masalah yang dapat diambil adalah sebagai berikut:

- a. Bagaimana cara mengimplementasikan GraphQL pada pengembangan Sistem Informasi Pelacakan Alumni Politeknik Negeri Malang?
- b. Bagaimana GraphQL dapat mengatasi masalah *under-fetching* pada sistem informasi yang dikembangkan?
- c. Bagaimana komparasi performa dari REST API dan GraphQL.

1.3 Tujuan

Tujuan dari pengimplementasian GraphQL pada Sistem Informasi Pelacakan Alumni adalah:

- a. Mengimplementasikan GraphQL pada Sistem Informasi Pelacakan Alumni Politeknik Negeri Malang.
- b. Mengatasi masalah *under-fetching* pada sistem yang dikembangkan.
- c. Mengetahui komparasi performa dari REST API dan GraphQL.

1.4 Batasan Masalah

Agar skripsi penulis yang berjudul “Implementasi GraphQL Untuk Mengatasi Under-fetching pada Pengembangan Sistem Informasi Pelacakan Alumni” dapat berjalan sesuai rencana dan tujuan awal, maka penulis memberikan batasan - batasan masalah sebagai berikut:

- a. Sistem Informasi Pelacakan Alumni menampilkan data alumni hasil Input Manual dan *Scraping* dari LinkedIn.
- b. Implementasi GraphQL berfokus pada bagaimana mengatasi masalah *under-fetching* pada pendistribusian data.

- c. Teknologi yang digunakan adalah React.js untuk bagian *frontend*, Node.js untuk bagian *backend*, Selenium untuk *scraping*, dan Typescript sebagai bahasa pemrograman.

1.5 Sistematika Penulisan

Skripsi ini terdiri dari 7 bab. Pada bab I, penulis memberikan latar belakang yang menjadi alasan mengapa penulis mengangkat masalah *under-fetching* sebagai bahan penelitian. Pada bab II, penulis menjelaskan landasan teori mengapa GraphQL dikatakan mampu mengatasi *under-fetching* yang terjadi pada metode pendistribusian data dengan REST. Pada bab III, berisi metode-metode yang digunakan terkait keperluan yang dibutuhkan untuk mengembangkan sistem informasi pelacakan alumni. Bab IV berisi beberapa rancangan dan desain perangkat lunak sebelum dikembangkan. Bab V berisikan hasil implementasi dari rancangan yang sudah dibuat pada bab IV dan pengujian performa antara sistem informasi dengan REST dan GraphQL. Bab VI membahas tentang hasil komparasi performa sistem informasi dengan REST dan GraphQL. Bab VII berisikan kesimpulan dari hasil penelitian yang sudah dilakukan terkait masalah *under-fetching* dan pengaruhnya pada performa suatu sistem informasi. Bab VII juga berisikan saran terkait penelitian lebih lanjut.

BAB II. LANDASAN TEORI

2.1 Teori

Under-fetching menjadi masalah pada suatu sistem informasi karena memiliki dampak pada penurunan performa suatu situs web. Penurunan performa tersebut terjadi dikarenakan adanya proses `meminta kembali` suatu data pada *backend* di *endpoint* yang berbeda. Hal ini menyebabkan adanya tambahan aktivitas pada sistem. Secara teori, hal itulah yang menyebabkan performa sistem informasi menjadi terhambat.

Sebagai contoh, jika dalam suatu sistem terdapat halaman yang melakukan request ke 2 *endpoint* yang berbeda, misalnya */user/id* untuk mendapatkan data pengguna dan */users/id/items* untuk mendapatkan data barang yang dimiliki pengguna, maka sistem memiliki 2 aktivitas yang harus dilakukan, yaitu meminta data pada *endpoint /users*, sekaligus meminta data pada *endpoint /users/items*.

Masalah ini dapat diatasi dengan cara menggabungkan 2 data yang berbeda tersebut menjadi 1 *endpoint* saja. Namun, hal ini juga memiliki kekurangan dimana jika sistem hanya akan menggunakan data diri pengguna saja, sistem juga akan menerima data item pengguna yang tidak diperlukan. Sehingga, terjadilah pengambilan data yang berlebihan.

Masalah inilah yang diselesaikan oleh GraphQL. GraphQL membantu sistem untuk menentukan data apa yang harus diambil berdasarkan *query* yang tersedia dari *backend*. *Backend* menentukan data apa saja yang boleh diambil, dan bagian *frontend* juga bisa menentukan data apa saja yang bisa diambil sesuai dengan apa yang disediakan oleh *backend*.

Misalnya, terdapat data pengguna yang terdiri dari *name*, *email*, *role*, dan *password*. Bagian *backend* menyediakan *query* untuk mengambil data pengguna yang berupa *name*, *email*, dan *role*. Maka, bagian *frontend* dapat mengambil data dari *query* tersebut, entah mengambil *name* saja, *name* dan *email* saja, dan sebagainya. Namun, bagian *frontend* tidak akan bisa mengambil *password* karena *query* tidak disediakan oleh *backend*.

GraphQL juga dapat mengambil data dari beberapa *query* sekaligus. Misalkan terdapat *query user* dan *item*, maka bagian *frontend* dapat melakukan pengambilan data *user* dan data *item* dalam 1 kali *request*. Hal inilah yang menjadikan GraphQL mampu untuk menyelesaikan masalah *under-fetching*.

2.2 Penelitian Terdahulu

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).

BAB III. METODOLOGI PENELITIAN

3.1 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 mengisi 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.

3.2 Metode Pengembangan Sistem

Metode pengembangan aplikasi yang penulis gunakan adalah metode Agile dengan *framework* Kanban. Alasan penulis memilih metode Kanban karena metode ini termasuk pada metode Agile yang memberikan kebebasan pada pengembang untuk mengembangkan fungsional sistem secara kasar (*iterative*) dan fokus pada beberapa fitur tertentu terlebih dahulu (*incremental*). Selain itu, kanban memiliki WIP (*Work-In-Progress*) yang memberi batasan berapa banyak tugas yang bisa diselesaikan. Sehingga, pengembang dapat fokus pada beberapa tugas terlebih dahulu.

3.3 Fase pengembangan sistem

Fase pengembangan sistem menggunakan metode Agile dengan *framework* Kanban memiliki fase sebagai berikut:

- a. User Story, merupakan suatu kasus berupa cerita ketika pengguna sedang mengakses produk yang kita buat. User story berisi tentang *role* pengguna, kegiatan apa yang dilakukan, dan alasan pengguna melakukan kegiatan tersebut. Contoh dari user story adalah sebagai berikut:

“Sebagai Administrator, saya ingin mencari data alumni.
Sehingga, saya bisa menemukan data alumni yang saya inginkan.”

Dari contoh user story di atas, dapat diketahui bahwa pengguna dengan *role* sebagai administrator memerlukan fitur untuk mencari data alumni. Dari sini, pengembang dapat menentukan bahwa sistem yang akan dikembangkan memerlukan suatu fitur pencarian alumni, yang dapat diakses oleh pengguna dengan role administrator, yang nantinya akan menampilkan data alumni sesuai yang dicari.

- b. Menentukan prioritas user story. Dengan menentukan prioritas dari suatu user story, pengembang dapat mengetahui fitur mana yang harus dikerjakan terlebih dahulu. Sehingga, pengembang dapat mengetahui tujuannya dengan jelas dan pengerjaan produk menjadi lebih terstruktur.
- c. Menentukan WIP (*Work In Progress*). Dengan adanya WIP, pengembang dapat membatasi fitur yang harus dikerjakan dan fokus untuk mengerjakan fitur-fitur yang dipilih terlebih dahulu. Jika salahsatu fitur sudah selesai, maka pengembang dapat memilih fitur lainnya yang ingin dikerjakan.

3.4 Proses Pengujian

Metode pengujian akan dilakukan dengan 2 cara. Cara pertama adalah dengan membandingkan performa akses website sistem informasi pelacakan alumni Politeknik Negeri Malang menggunakan GraphQL, dengan sistem

informasi pelacakan alumni Politeknik Negeri Malang dengan menggunakan metode pendistribusian data REST API.

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.

BAB IV. ANALISIS DAN PERANCANGAN

4.1 Kebutuhan Fungsional

Kebutuhan fungsional dari sistem informasi pencarian alumni dengan graphql dan sistem informasi pencarian alumni REST yang akan penulis kembangkan adalah:

4.1.1 Administrator

- a. Mengelola jurusan, diantaranya melihat daftar jurusan, menambah jurusan baru, mengubah data jurusan, dan menghapus jurusan.
- b. Autentikasi pengguna, diantaranya *login* dan *logout*.
- c. Mengelola data alumni, diantaranya melakukan *scraping* dari LinkedIn, pencarian alumni, mengubah data alumni, menghapus data alumni, melihat daftar alumni, dan melihat detail data dari alumni.
- d. Mengelola pengguna, diantaranya melihat daftar pengguna, mengubah data pengguna, menambah pengguna, dan menghapus pengguna.

4.1.2 Pengguna

- a. Mengakses data jurusan, diantaranya melihat daftar jurusan.
- b. Autentikasi pengguna, diantaranya login, register, dan logout.
- c. Mengakses data alumni, diantaranya melakukan pencarian alumni, melihat daftar alumni, dan melihat detail data dari alumni.

4.2 Kebutuhan Non-Fungsional

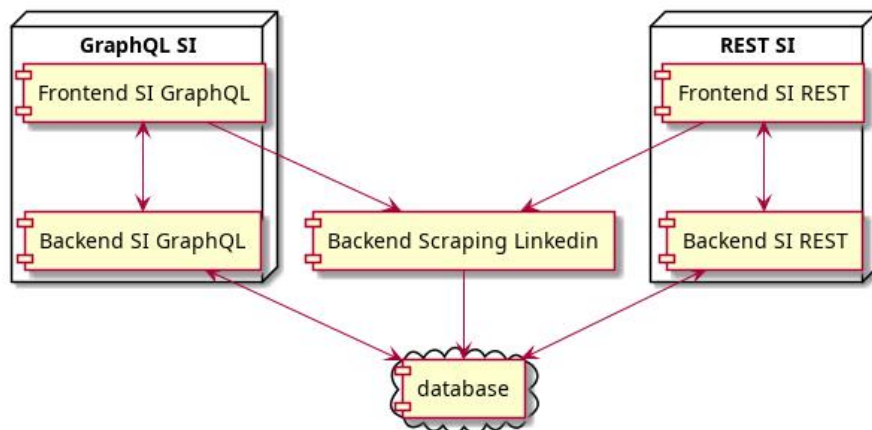
Kebutuhan non-fungsional dari sistem informasi pencarian alumni adalah:

- a. Sistem informasi berjalan pada *web browser*.
- b. *Frontend* dikembangkan dengan menggunakan React.js, baik sistem informasi yang menggunakan GraphQL maupun REST.
- c. *Backend* dikembangkan dengan menggunakan *Node.js*, baik sistem informasi yang menggunakan GraphQL maupun REST.

- d. Bahasa pemrograman yang digunakan untuk mengembangkan kedua sistem informasi adalah Typescript.
- e. *Scraping* data alumni dari LinkedIn dilakukan dengan menggunakan selenium.js.
- f. Database yang digunakan adalah mongodb dan tersimpan pada *cloud service* milik mongodb. Sehingga, dapat diakses oleh backend GraphQL maupun REST.

4.3 Rancangan Sistem

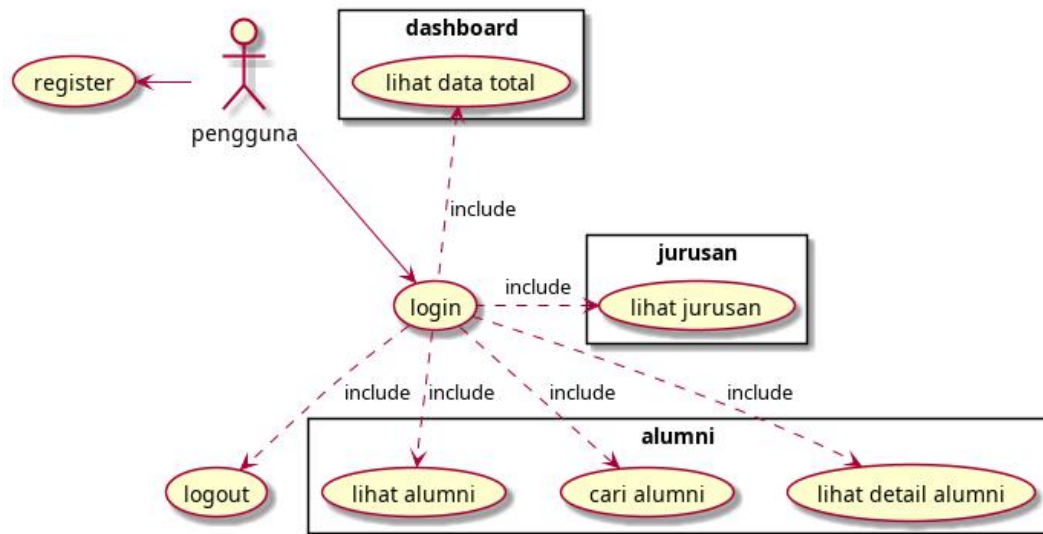
4.3.1 Arsitektur Sistem



Gambar 4.1 Arsitektur Sistem

Pada gambar 4.1, dapat dilihat bahwa akan dibuat 2 sistem informasi dengan metode pendistribusian data yang berbeda. Kedua sistem informasi tersebut menggunakan 1 database yang sama. Sehingga, tidak ada perbedaan data yang terjadi. Kemudian, terdapat *microservice* berupa *scraper* untuk mengambil ulang data alumni dari situs web LinkedIn.

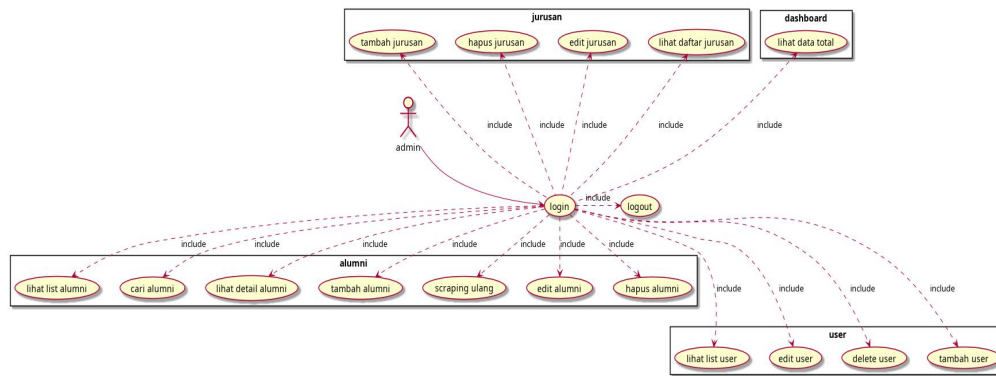
4.3.2 Use Case



Gambar 4.2 Use Case Pengguna

Pada gambar 4.2, pengguna bukan merupakan administrator. Pengguna yang belum memiliki akun atau belum masuk ke sistem, dapat melakukan 2 aksi. Aksi tersebut adalah daftar (*register*) dan masuk (*login*).

Kemudian, setelah berhasil login, pengguna dapat mengakses beberapa aksi kecuali daftar dan masuk. Pada bagian alumni, pengguna non-administrator dapat mengakses data alumni dengan aksi berupa melihat data alumni, mencari data alumni, dan melihat detail dari data alumni. Pada bagian jurusan, pengguna non-administrator dapat mengakses data jurusan dengan aksi berupa melihat data jurusan yang sudah terdaftar pada sistem. Pada bagian dashboard, pengguna non-administrator dapat mengakses data total, seperti data total alumni, total alumni yang sudah bekerja dan total alumni yang belum bekerja, dan sebagainya.



Gambar 4.3 Use Case Administrator

Gambar 4.3 merupakan diagram use case untuk administrator. Administrator tidak dapat mendaftar, namun administrator dapat masuk ke sistem (*login*). Setelah melakukan *login*, administrator dapat mengakses data dari berbagai kategori.

Pada bagian alumni, administrator dapat melakukan akses data alumni dengan aksi berupa melihat daftar alumni, mencari data alumni, melihat detail data dari alumni, menambah data alumni, melakukan *scraping* ulang ke situs web LinkedIn, mengubah data alumni, dan menghapus data alumni.

Pada bagian jurusan, administrator dapat melakukan akses data jurusan dengan aksi berupa melihat daftar jurusan, menambah data jurusan, mengubah data jurusan, dan menghapus data jurusan.

Pada bagian *dashboard*, administrator dapat melihat data total seperti data total alumni yang sudah bekerja, total alumni yang belum bekerja, dan sebagainya.

Pada bagian pengguna, administrator dapat mengakses data pengguna berupa melihat daftar pengguna, mengubah data pengguna seperti mengubah *role* pengguna atau mereset kata sandi pengguna, menghapus akun pengguna, dan menambahkan data pengguna baru.

4.3.3 Struktur Data

Struktur data dari sistem informasi yang akan penulis kembangkan adalah:

- a. *Collection* alumni_inputs, untuk menyimpan data alumni hasil input dari sistem.

Tabel 4.1 Desain Collection alumni_inputs

Column Name	Tipe Data
_id	ID
name	String
entry_year	Number
graduate_year	Number
major	String
work_at	String
work_position	String
email	String
data_source	String

- b. *Collection* alumni_linkedins, untuk menyimpan data alumni hasil *scraping* dari LinkedIn. Meski bentuk datanya hampir sama dengan data dari alumni_inputs, data dari hasil *scraping* LinkedIn sengaja diletakkan pada *collection* berbeda untuk mempermudah pengelolaan data dari sumber yang berbeda seperti melihat data hasil dari input sistem saja, dan sebaliknya.

Tabel 4.2 Desain Collection alumni_linkedins

Column Name	Tipe Data
_id	ID
name	String
entry_year	String
graduate_year	String
major	String
work_at	String
work_position	String
email	String
data_source	String

- c. *Collection* major, berisi nama dari jurusan.

Tabel 4.3 Desain Collection major

Column Name	Tipe Data
_id	ID
name	String

- d. *Collection* user, berisi data dari pengguna. Baik administrator maupun pengguna biasa.

Tabel 4.4 Desain Collection user

Column Name	Tipe Data
_id	ID
name	String
email	String
password	String
level	Number

4.3.4 Epic

Epic adalah suatu *user story* yang memiliki cangkupan cukup luas. Sehingga, memiliki kemungkinan besar untuk dapat di-*breakdown* menjadi beberapa *user story* yang lebih detail.

Epic yang penulis buat pada pengembangan sistem informasi kali ini adalah:

- Administrator: Mengelola Jurusan. Sebagai administrator, saya ingin mengelola jurusan. Sehingga, saya dapat mengelola jurusan yang ada di sistem informasi alumni.
- Administrator: Autentikasi. Sebagai administrator, saya ingin melakukan autentikasi pada sistem. Sehingga, saya dapat mengakses fitur-fitur yang ada di website sesuai dengan wewenang saya.
- Administrator: Mengelola Pengguna. Sebagai administrator, saya ingin mengelola user. Sehingga, saya dapat mengelola data user yang dapat login ke sistem.
- Administrator: Mengelola data alumni. Sebagai administrator, saya ingin dapat mengelola data alumni. Sehingga, saya bisa tahu data alumni yang sudah terdaftar pada sistem.
- User: Autentikasi. Sebagai user, saya ingin melakukan autentikasi pada sistem. Sehingga, saya dapat mengakses fitur-fitur yang ada di website sesuai dengan wewenang saya.
- User: Melacak data alumni. Sebagai user, saya ingin melacak data alumni. Sehingga, saya bisa mengetahui data dari alumni yang saya cari.
- User: Melihat data jurusan. Sebagai user, saya ingin dapat melihat data jurusan. Sehingga, saya tahu jurusan apa saja yang ada pada kampus Politeknik Negeri Malang.

4.3.5 User Story

User Story di sini adalah hasil *breakdown* dari Epic yang sudah dibuat diatas. Karena penulis akan mengembangkan 2 sistem informasi yang sama, maka *user story* untuk kedua sistem informasi adalah sama. Berikut adalah:

- a. Administrator: Melihat list jurusan. Sebagai administrator, saya ingin dapat melihat daftar jurusan yang ada. Sehingga, ketika menambah alumni baru, saya bisa memastikan bahwa jurusan dari alumni tersebut sudah tersedia.
- b. Administrator: Menambah jurusan baru. Sebagai administrator, saya ingin dapat menambahkan jurusan baru. Sehingga, saya dapat menambahkan jurusan yang belum tersedia di sistem.
- c. Administrator: Mengubah data jurusan. Sebagai administrator, saya ingin dapat mengubah data jurusan. Sehingga, apabila ada salah input, maka data bisa diperbarui.
- d. Administrator: Menghapus jurusan. Sebagai administrator, saya ingin dapat menghapus data jurusan yang sudah ada. Sehingga, nama jurusan yang sudah tidak ada, tidak akan muncul ketika memilih jurusan untuk menambahkan alumni.
- e. Administrator: login. Sebagai administrator, saya ingin dapat melakukan login. Sehingga, saya dapat mengakses fitur-fitur administrator pada website.
- f. Administrator: logout. Sebagai administrator, saya ingin dapat melakukan logout. Sehingga, saya dapat mengakhiri sesi saya dari sistem.
- g. Administrator: melihat daftar user. Sebagai administrator, saya ingin dapat melihat daftar user. Sehingga, saya dapat melihat daftar beberapa user sekaligus.
- h. Administrator: mengubah data user. Sebagai administrator, saya ingin dapat memperbarui data user yang sudah ada. Sehingga, saya bisa menambahkan administrator baru dari user yang sudah ada, atau mereset data user yang lupa password.

- i. Administrator: Menghapus user. Sebagai administrator, saya ingin dapat menghapus data user. Sehingga, saya dapat menghapus data user tertentu.
- j. Administrator: melakukan *scraping* ulang dari linkedin. Sebagai administrator, saya ingin melakukan scraping data alumni dari LinkedIn. Sehingga, saya bisa memperbarui data alumni dari LinkedIn.
- k. Administrator: pencarian alumni. Sebagai administrator, saya ingin mencari data alumni. Sehingga, saya bisa menemukan data alumni yang saya inginkan.
- l. Administrator: mengubah data alumni. Sebagai administrator, saya ingin meng-*update* data alumni yang diinputkan manual. Sehingga, saya bisa mengubah data alumni jika ada kesalahan input.
- m. Administrator: menghapus data alumni. Sebagai administrator, saya ingin menghapus data alumni. Sehingga, saya bisa menghapus data alumni yang tidak valid.
- n. Administrator: menambahkan data alumni. Sebagai administrator, saya ingin menambahkan data alumni. Sehingga, saya bisa menambahkan data alumni selain melalui *scraping*.
- o. Administrator: melihat daftar alumni. Sebagai Administrator, saya ingin melihat daftar data alumni. Sehingga, saya bisa melihat beberapa data alumni sekaligus.
- p. Administrator: melihat detail alumni. Sebagai Administrator, saya ingin melihat detail data alumni. Sehingga, saya bisa melihat detail data setiap alumni.
- q. User: login. Sebagai user, saya ingin dapat melakukan login. Sehingga, saya dapat mengakses fitur-fitur user pada website.
- r. User: register. Sebagai user, saya ingin dapat melakukan register. Sehingga, saya dapat melakukan login pada website.
- s. User: logout. Sebagai User, saya ingin dapat melakukan logout. Sehingga, saya dapat mengakhiri sesi pada website.

- t. User: mencari data alumni. Sebagai User, saya ingin dapat melakukan pencarian alumni. Sehingga, saya dapat mengetahui data dari alumni yang saya inginkan.
- u. User: melihat daftar alumni. Sebagai User, saya ingin dapat melihat daftar alumni. Sehingga, saya dapat melihat beberapa data alumni sekaligus.
- v. User: melihat detail alumni. Sebagai User, saya ingin dapat melihat detail dari suatu alumni. Sehingga, saya dapat melihat detail data dari seorang alumni.
- w. User: melihat daftar jurusan. Sebagai user, saya ingin dapat melihat data jurusan. Sehingga, saya tahu jurusan apa saja yang ada pada kampus Politeknik Negeri Malang.

BAB V. IMPLEMENTASI DAN PENGUJIAN

5.1 Implementasi

5.1.1 Pembuatan Scraper LinkedIn

Scraper dibuat menggunakan Selenium.js dengan backend Node.js. Scraper dibuat pada *backend* tersendiri. Sehingga, dapat diakses oleh sistem informasi dengan GraphQL maupun sistem informasi dengan REST.

Langkah pertama yang penulis lakukan adalah membuat akun linkedin dan menambah koneksi dengan dengan akun linkedin yang sudah memasukkan politeknik negeri malang pada riwayat pendidikan dan memiliki banyak relasi. Kemudian memastikan bahwa tab alumni pada halaman Politeknik Negeri Malang dapat diakses dan memunculkan data alumni. Namun, ketika pertama kali akun dibuat, akun tersebut tidak bisa melihat data akun lain pada alumni Politeknik Negeri Malang. Sehingga, penulis mendiamkan akun tersebut selama 3 hari. Setelah itu, beberapa data alumni muncul pada bagian tab alumni di halaman Politeknik Negeri Malang, namun masih ada beberapa data alumni yang belum muncul (hanya muncul tulisan “LinkedIn Member” tanpa bisa diklik). Setelah memastikan data berhasil dimuat, barulah pengembangan *backend* untuk *scraping* dimulai.

Pada pengembangan *backend* ini, pengembangan dilakukan dengan menggunakan bahasa typescript untuk memberikan batasan pada variabel agar tidak bisa menerima sembarang typedata seperti pada javascript. Dengan begitu, variabel memiliki typedata yang jelas dan tidak menimbulkan variabel yang tidak diketahui tipe datanya. Karena node.js tidak memiliki bawaan untuk menggunakan typescript, maka penulis melakukan pengaturan pada node.js dengan menambahkan modul typescript pada node.js dan menambahkan file tsconfig.json yang menjadi pengaturan untuk modul typescript, bagaimana typescript bertindak untuk mengatur kode yang ditulis. Isi dari tsconfig.json dapat dilihat pada lampiran 1.

Setelah typescript sudah dipasang, pengembangan dilanjutkan dengan memasang modul selenium untuk javascript, modul express, dan modul mongoose sebagai database untuk menyimpan data ke database setelah scraping selesai. Setelah semua modul yang dibutuhkan terpasang, pengembangan scraper untuk linkedin dimulai.

Pengembangan *scraper* dimulai dengan membuka browser chrome webdriver dan melakukan login menggunakan akun yang sudah dibuat. Dimana setelah login, linkedin akan langsung mengarahkan ke halaman alumni dari Politeknik Negeri Malang. Jika login berhasil, *scraper* akan menunggu hingga data alumni sudah dimuat dengan batas waktu tunggu 10 detik. Jika lebih dari itu, maka *scraper* akan berhenti dan menganggap data alumni tidak ditemukan. Setelah data alumni berhasil dimuat, maka *scraper* akan memulai untuk *scroll* halaman untuk memuat semua data alumni yang ada hingga halaman tidak bisa lagi di-*scroll*. Setelah semua data alumni sudah muncul, maka *scraper* mengambil masing-masing url alumni yang sudah ada dan menyimpannya pada suatu *array*. *Array* tersebut nantinya akan di-*looping* dan pada tiap iterasi, akan membuka tab baru dengan alamat sesuai dengan *array*. Setelah membuka halaman alumni yang dimaksud, *scraper* melakukan *scroll down* agar data *experience* dari alumni dapat dibaca dan diambil datanya. Beberapa alumni terkadang tidak memberikan data yang diperlukan. Sehingga, menyebabkan data yang tersimpan adalah berupa *string* kosong (“”). Setelah data alumni berhasil diambil, maka data alumni tersebut disimpan dalam suatu objek dan dimasukkan pada *array* untuk alumni. Proses ini diulangi hingga semua alamat pada *array* berhasil dibuka dan diambil datanya. Setelah selesai pada proses memuat data alumni, sistem akan mengosongkan *collection* dengan nama “alumni_linkedins” dan menyimpan data alumni yang baru. Pengosongan *collection* dilakukan untuk menghindari adanya duplikasi data atau adanya 2 data alumni yang sama.

Setelah berhasil menyimpan data pada *database*, maka *scraper* akan menutup browser dan mengakhiri proses *scraping* linkedin.

5.1.2 Pengunggahan Database ke Cloud

Salahsatu kelebihan dari menggunakan mongodb sebagai penyimpanan data adalah, mongodb menyediakan server cloud gratis untuk setiap penggunanya dengan kapasitas sebesar 500 MB. Dengan adanya kelebihan tersebut, maka penulis memutuskan untuk mengunggah data alumni hasil scraping ke cloud server untuk mencegah terjadinya hal yang tidak diinginkan apabila database menggunakan server lokal.

Untuk mendapatkan server cloud dari mongodb, penulis harus memiliki akun untuk login pada sistem atau dapat login menggunakan akun Google. Setelah login berhasil, penulis diarahkan untuk membuat suatu *cluster* baru. 1 *Cluster* dapat berisi banyak *database*. 1 *database* dapat berisi banyak *collection*.

Setelah pembuatan cluster berhasil, penulis membuat 1 database dengan nama si-alumni. Pada si-alumni, terdapat 4 collections bernama alumni_inputs, alumni_linkedins, users, dan majors.

users merupakan collection yang berisikan data-data akun pengguna untuk login ke dalam sistem.

Majors berisikan data-data jurusan yang sudah terdaftar di sistem.

alumni_inputs merupakan collection untuk menyimpan data alumni hasil input dari sistem informasi yang akan dibuat nanti.

alumni_linkedins adalah collection untuk menyimpan data alumni hasil dari scraping data alumni Politeknik negeri malang di LinkedIn. Meskipun bentuk datanya bisa dibilang sama dengan collection alumni_inputs, alasan penulis memisahkannya menjadi 2 collections yang berbeda adalah untuk memudahkan penyimpanan data dalam proses *scraping*. Dengan dipisahkannya data hasil scraping dari LinkedIn dan hasil input dari sistem, maka ketika pengguna sistem informasi (administrator) akan melakukan scraping ulang, penyimpanan data akan lebih cepat karena server tidak perlu melakukan *query* untuk menyeleksi data mana yang bersumber dari LinkedIn dan mana yang bukan, kemudian menghapusnya.

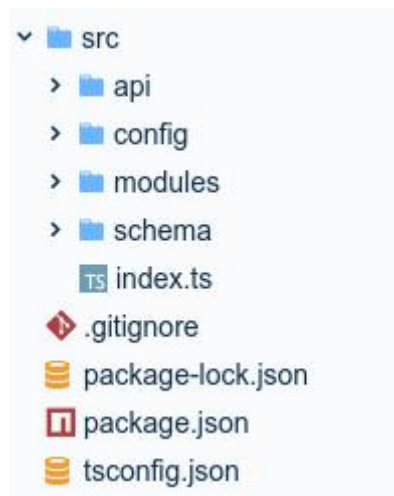
Proses pengunggahan data dilakukan dengan menggunakan aplikasi Compass, yaitu aplikasi GUI untuk mengatur database mongodb.

5.1.3 Pengembangan Backend REST

Pengembangan *backend* untuk REST dilakukan menggunakan Node.js dan bahasa pemrograman Typescript.

Pengambilan data pada REST dilakukan dengan membuat *router* untuk memberi nama *endpoint* yang bisa diakses, *controller* untuk mengatur perintah apa yang akan dikerjakan ketika pengguna mengakses *endpoint* yang dimaksud, dan model untuk mengatur data pada *database* melalui *controller*.

Pada *backend* yang penulis kembangkan, struktur program memiliki bentuk sebagai berikut:



Gambar 5.1 Struktur proyek backend REST

Pada *folder* “api”, *folder* berisikan *subfolder* - *subfolder* yang mewakili masing - masing sekelompok *endpoint*. Sebagai contoh, pada *subfolder* “alumni” berisikan *model*, *controller*, sekaligus *subrouter* untuk mengatur segala aktivitas yang melibatkan *endpoint* /*alumni*. Aktivitas tersebut dapat berupa menampilkan daftar alumni, menambahkan alumni baru, menghapus data alumni, dan sebagainya yang berhubungan dengan pengolahan data-data alumni.

Pada *folder* “config”, *folder* berisikan berkas - berkas yang mengatur segala hal yang berhubungan dengan *backend*. Di pengembangan *backend* kali ini, penulis menerapkan 2 hal di dalam *folder* “config”. Hal tersebut berupa mengatur hubungan *backend* dengan *database* yang berada di *cloud server*, dan juga membuat pengaturan *main router* dimana *main router* nantinya akan mengakses *subrouter* yang berada didalam *subfolder* pada *folder* “api”.

Berlanjut ke *folder* “modules”, modules berisikan fungsi yang dapat berdiri sendiri dan digunakan pada banyak kasus. Dengan begitu, fungsi tersebut dapat menjadi *reusable* dan menghemat banyak baris kode. Pada pengembangan kali ini, penulis membuat 1 fungsi di dalam *folder* tersebut, yaitu fungsi untuk melakukan verifikasi token yang didapat dari *frontend*. Dengan diverifikasinya token terlebih dahulu, *backend* dapat menentukan apakah pengguna saat ini berhak untuk mengakses data atau tidak. Verifikasi token ini diterapkan pada hampir semua endpoint dari *backend* dengan metode REST. Bagian yang tidak menggunakan verifikasi token adalah registrasi dan login.

Terakhir, terdapat *folder* bernama “Schema”. Folder tersebut berisi skema skema dari *collections* MongoDB. Skema merupakan kerangka bagaimana struktur data dari suatu collection. Dari skema tersebut, nantinya akan dibuat *model* yang akan digunakan ketika suatu *endpoint* tertentu diakses.

Untuk lebih memahami bagaimana alur program dengan struktur program yang penulis gunakan, penulis akan memberikan contoh kasus beserta potongan kode. Contoh kasus yang penulis ambil adalah ketika pengguna mengakses endpoint `/major/:id` dengan metode http GET untuk mendapatkan detail data dari jurusan.

Hal pertama yang terjadi ketika pengguna mengakses *endpoint* `/major/:id` adalah, *main router* pada *folder* “config” memeriksa apakah ada *endpoint* dengan nama awalan `/major` dan dengan metode http apapun (GET, POST, PUT, DELETE).

```
...
router.use("/major", verifyToken, majorRouter);
...
```

Gambar 5.2 potongan kode Router untuk /major

Potongan kode tersebut berarti *backend* menyediakan endpoint */major* yang dapat diakses dengan metode http apasaja. Apabila ada yang mengakses *endpoint* tersebut, maka token yang didapat akan divalidasi oleh fungsi *verifyToken* yang berada pada folder “modules”. Jika validasi dinyatakan sukses, maka sistem akan mengarahkan ke *router* lain yang bernama *majorRouter*. *majorRouter* berada di dalam folder “api/major”.

```
...
majorRouter.get("/:id", majorController.showOne);
...
```

Gambar 5.3 Sub-router dari /major

Dalam *majorRouter*, akan dilakukan pengecekan *endpoint* kembali. *Endpoint* manakah yang sesuai dengan yang *frontend* minta, dan metode http apa yang diinginkan. Sesuai contoh kasus, *endpoint* yang diminta adalah */major/:id* dengan metode GET. Karena bagian *backend* telah menyediakan *endpoint* tersebut, maka *backend* akan mengarahkan ke *majorController.showOne* yang juga berada pada folder yang sama, yaitu folder *api/major*.

```
export const majorController = {
  ...
  showOne: async (req: Request, res: Response) => {
    ...
  },
  ...
}
```

Gambar 5.4 Controller untuk endpoint major

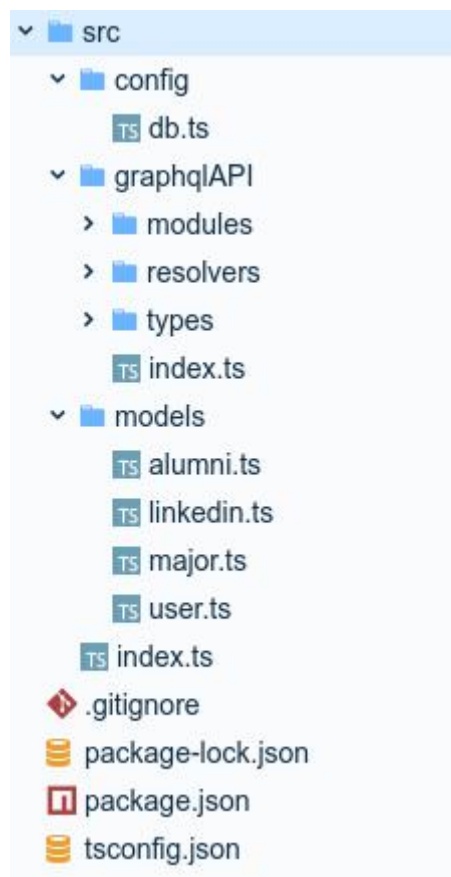
Pada controller *majorController*, terdapat sebuah objek bernama *majorController*. Objek tersebut memiliki properti berupa fungsi yang bernama *showOne*. Di dalamnya terdapat segala aktivitas yang akan dilakukan ketika

pengguna mengakses *endpoint* /major/:id seperti contoh kasus di atas. Aktivitas dalam properti inilah yang akan dilakukan setelah pengguna berhasil mengakses *endpoint* /major/:id.

5.1.4 Pengembangan Backend GraphQL

Pengembangan *backend* untuk GraphQL dilakukan menggunakan Node.js dan bahasa pemrograman Typescript.

Cara kerja GraphQL yang berbeda dengan REST mempengaruhi penulis dalam menentukan bagaimana struktur program yang baik dan nyaman untuk diimplementasikan pada pengembangan *backend* dengan metode GraphQL ini. Oleh karena itu, bentuk struktur program dari backend dengan GraphQL menjadi seperti berikut:



Gambar 5.5 Struktur Proyek backend dengan GraphQL

Sama seperti pada *backend* dengan REST, *folder* “config” berisikan mengenai berkas - berkas yang menjadi pengaturan penting untuk *backend*. Dalam

pengembangan *backend* dengan GraphQL kali ini, penulis menerapkan pengaturan *database* untuk mengakses data pada *database* yang berada di *cloud server* melalui *backend* ini.

Pada folder “graphqlAPI”, terdapat beberapa *subfolder* yang terdiri dari “modules”, “resolvers”, dan “types”. terdapat juga berkas bernama *index.ts*. Berkas ini berfungsi sebagai berkas utama yang merangkum seluruh *type*, *query*, dan *resolver* yang ada pada ketiga *folder* di atas. Hal ini bertujuan untuk memudahkan pembacaan kode dan mencegah baris kode yang terlalu panjang dibandingkan menjadikan seluruh *type*, *query*, dan *resolver* berada dalam 1 berkas yang sama.

Folder “modules” berisi tentang fungsi-fungsi yang dapat digunakan ulang pada lebih dari 1 *query/mutation*. Sama dengan *folder* “modules” yang ada pada REST, tujuan dari memisahkan fungsi yang dapat berdiri sendiri ini adalah mempersingkat penulisan kode dan mencegah penulisan kode yang sama berulang-ulang dengan menjadikannya 1 komponen sendiri yang dapat diakses oleh banyak fungsi lainnya.

Di *folder* “resolver”, *folder* berisi *resolver* dari masing-masing *query* yang ada, dimana *resolver* berisi aktivitas-aktivitas yang akan dilakukan apabila *query* yang diminta oleh *frontend* sesuai dengan yang ada pada *backend*.

Folder “types” berisi tipe-tipe dari data yang dapat diakses. Tipe inilah yang menentukan bentuk data dan tipedata dari suatu *query*.

Terakhir, terdapat *folder* “models” yang berisi model-model dari *collection database* pada *Mongodb*.

Untuk mempermudah dalam memahami alur program dari *backend GraphQL* yang penulis kembangkan, penulis akan memberikan contoh kasus ketika bagian *frontend* memerlukan data berupa detail data dari jurusan seperti contoh kasus yang sama pada pengembangan *backend* dengan REST.

Pertama kali, bagian *frontend* akan mengirimkan *query* ke *backend*. *Query* tersebut akan dicek apakah tersedia atau tidak. Jika *query* tersedia, maka *resolver* yang sesuai dengan *query* yang didapat akan dijalankan.

Misal, *query* yang dikirimkan oleh bagian *frontend* adalah sebagai berikut:

```

query detailMajor {
  majorDetail(id: "5eaa548bb9452718a1ddf311") {
    _id
    name
  }
}

```

Gambar 5.6 Query untuk mengambil data detail jurusan

Query ini berarti *frontend* akan mengambil data *_id* dan *name* dari query *majorDetail* dengan id *5eaa548bb9452718a1ddf311*. query ini akan diperiksa ketersediaannya pada file *index.ts* di folder *graphqlAPI/index.ts*.

```

const typeDefs = gql`
  ...
  type Query {
    ...
    majorDetail(id: String!): major
    ...
  }
  ...
`

```

Gambar 5.7 Menyediakan query *majorDetail*

Pada potongan kode di atas, dapat dilihat bahwa *query* yang disediakan oleh *backend* adalah *query* bernama *majorDetail* yang harus menerima parameter *id* dan akan mengembalikan tipe bernama *major* yang berada pada folder “types”.

```

type major {
  _id: ID
  name: String
}

```

Gambar 5.8 type dari *major*

Potongan kode di atas adalah tipe dari *major*. *Major* memiliki data berupa *_id* dengan tipe data *ID* dan *name* dengan tipe data *String* juga. Sehingga, *query* yang dikirimkan oleh *frontend* kepada *backend* dapat dikatakan valid. Karena, *frontend* meminta data berupa *_id* dan juga *name* dimana *backend* memang menyediakannya.

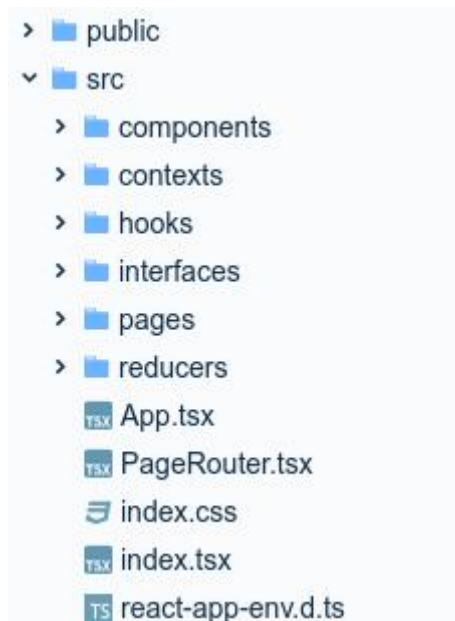
Setelah query dinyatakan valid, maka resolver yang bertanggungjawab atas query tersebut akan dijalankan. Resolver terletak di dalam `index.ts`, kemudian akan mengarahkan pada fungsi yang berada di *file* `majorResolver` pada *folder* “resolver”. Untuk contoh kodenya, dapat dilihat pada lampiran 2.

Resolver akan mengembalikan nilai sesuai dengan apa yang tertulis pada *query*. Dengan begitu, keperluan data pada *frontend* dapat terpenuhi.

5.1.5 Pengembangan Frontend REST

Pengembangan *frontend* untuk bagian REST menggunakan React.js dengan bahasa pemrograman Typescript.

Struktur program yang dibuat untuk *frontend* menggunakan REST adalah sebagai berikut:



Gambar 5.9 Struktur proyek frontend REST API

Folder “components” berisikan komponen-komponen yang dapat berdiri sendiri dan digunakan berulang-ulang pada suatu halaman. Sehingga, jika akan dibuat suatu komponen pada halaman yang memiliki bentuk/sifat yang sama dengan komponen yang sudah dibuat pada halaman lain, pengembang tidak perlu membuat komponen dari awal, melainkan cukup dengan menggunakan komponen yang sudah dibuat. Karena kemungkinan akan cukup banyak komponen yang

penulis buat, maka penulis memasukkan kumpulan komponen ke dalam suatu *folder* tersendiri agar struktur program lebih rapi.

Folder “contexts” dan *folder* “reducers” berhubungan dengan pengaturan *state* pada React. Kedua *folder* tersebut berisi kode yang mengatur bagaimana suatu *state* pada React dapat diakses dari komponen manapun (*state* global). Dengan dipisahkannya antara *contexts* dan *reducers*, pengembang akan dapat lebih mudah menangani manajemen *state* global yang ada pada sistem ataupun mengatasi *error* jika suatu saat ada masalah yang terjadi berkaitan dengan *state* global.

Pada *folder* “hooks”, penulis membuat *hooks* untuk React yang berguna ketika ada suatu aksi yang akan sering dilakukan. Dengan bantuan *hooks*, 2 komponen yang berbeda dapat memiliki aksi yang sama tanpa perlu menulis ulang *logic* dari program.

Selanjutnya, *folder* “interfaces”. *Folder* ini berisi tentang *interface* untuk menentukan tipe data suatu variabel, *array*, ataupun objek. *Interface* merupakan keunikan dan menjadi pembeda antara bahasa pemrograman Javascript dengan bahasa pemrograman Typescript. Dengan adanya *interface*, variabel, *array*, ataupun objek sekalipun dapat diketahui bentuk datanya. Hal ini mencegah terjadinya perubahan data pada variabel yang tidak sesuai dengan tipenya. Sehingga, pelacakan terhadap *error* akan lebih mudah. Jika pada bahasa pemrograman Javascript, pengembang dapat menimpa nilai suatu variabel dengan nilai lain yang berbeda tipe data, seperti *string* ke *number*. Hal ini menyebabkan perubahan data suatu variabel menjadi sulit untuk dilacak dan dikontrol. Sehingga, jika terjadi suatu proses yang panjang terhadap suatu nilai variabel, nilai variabel tersebut dapat berubah dan tidak konsisten. Karena nantinya kemungkinan akan ada beberapa variabel, *array*, dan objek yang memiliki tipe data yang sama, maka *folder* “interface” dibuat untuk menyimpan segala berkas berupa *interface*. Sehingga, komponen manapun dapat menggunakannya dan dapat digunakan berulang-ulang tanpa harus menyetikkan kode yang sama di komponen yang membutuhkan.

Folder “pages” berisi kode untuk halaman-halaman pada halaman web. Setiap halaman akan mewakili 1 *folder*. Pemisahan ini dilakukan untuk

memudahkan manajemen setiap halaman. Dengan dipisahkannya kode menjadi per halaman, maka segala urusan yang ada pada suatu halaman akan diatur dalam 1 *folder*. Sehingga, pengembang dapat mengetahui jika ada yang salah dalam suatu tampilan halaman, *folder* mana yang harus dituju untuk memperbaiki halaman yang dimaksud.

File “PageRouter.tsx” merupakan komponen React yang penulis buat untuk mengatur segala hal tentang *routing* pada sistem. Dengan dipisahkannya *router* dari *App.tsx*, komponen *App.tsx* menjadi lebih rapi dan pengaturan akan *routing* di *frontend* menjadi lebih mudah.

Perbedaan mendasar pada *frontend* dengan REST dan *frontend* dengan GraphQL terletak pada permintaan pengambilan data terhadap *backend*. Pada pengembangan *frontend* dengan REST, *frontend* melakukan permintaan pengambilan data melalui *http request* menggunakan bantuan Axios.js. *Frontend* harus mengakses suatu *endpoint* tertentu, memberikan data yang diperlukan ke *backend*, sehingga *backend* dapat merespon dengan memberikan data yang diperlukan ke *frontend*.

Sebagai contoh, potongan kode yang digunakan untuk mengambil data jurusan dari backend dapat dilihat pada lampiran 3.

Kode di atas merupakan kode untuk melakukan permintaan ke *backend* dengan metode *http GET* untuk mengambil data detail dari jurusan dengan *id* tertentu. Setelah permintaan data dikirimkan ke *backend*, *response* akan diberikan oleh *backend*. Apabila terjadi *error*, maka kode di dalam *catch()* akan dijalankan, jika permintaan berhasil, maka kode di dalam *then()* akan dijalankan, sedangkan kode di dalam *finally()* akan selalu dijalankan setelah *then()* atau *catch()* selesai dijalankan.

5.1.6 Pengembangan Frontend GraphQL

Pembuatan *frontend* pada sistem informasi menggunakan GraphQL dikembangkan menggunakan React.js dan bahasa pemrograman Typescript.

Struktur program yang digunakan pada pengembangan *frontend* dengan GraphQL sama dengan struktur program yang digunakan pada *frontend* dengan

REST. Perbedaan yang mendasar antara *frontend* dengan GraphQL dan *frontend* dengan REST adalah bagaimana pengolahan data terjadi.

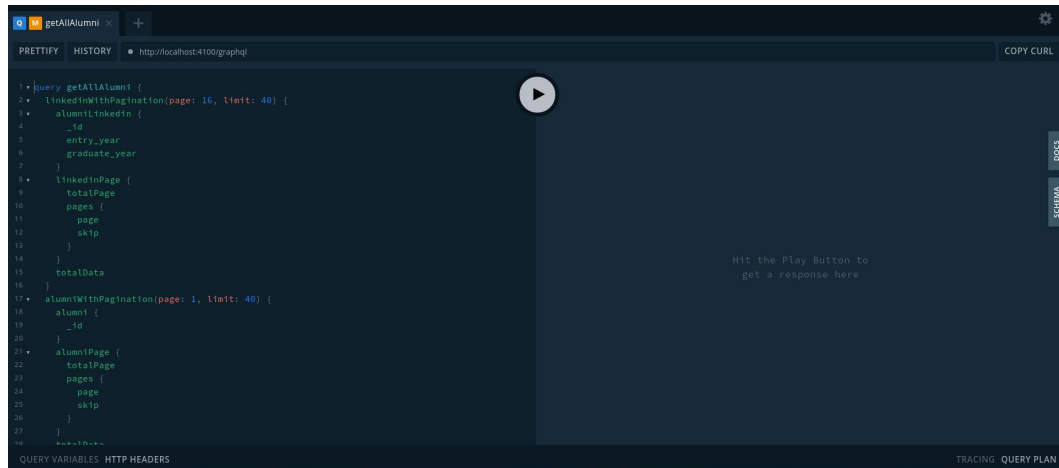
Pada *frontend* dengan GraphQL, pengolahan data terjadi dengan mengirimkan *query* ke bagian *backend*. *Query* yang dikirim akan diolah oleh *backend* sehingga mengembalikan suatu nilai tertentu. Nilai tersebut dapat berupa *error*, ataupun data yang diinginkan.

Dalam pengembangan *frontend* dengan GraphQL, pengembang menggunakan bantuan Apollo.js, yaitu sebuah *library* yang dapat membantu proses pengiriman *query* ke bagian *backend* dan pengaturan bagaimana data nantinya akan diolah.

Pada REST, *frontend* cukup melakukan *http request* ke bagian *backend*, kemudian *frontend* akan mendapatkan data yang diinginkan apabila *request* berhasil. Namun, pada GraphQL, *frontend* harus mengirimkan sebuah *query* yang cukup panjang ke bagian *backend*. Hal ini tentu cukup memakan baris kode jika *query* yang diketikkan cukup panjang. Sehingga, terkadang penulisan *query* dan kode utama perlu dipisah untuk menjaga kerapian kode pada tiap berkas.

Selain masalah di atas, pada REST, terdapat juga masalah dimana bagian *frontend* tidak mengetahui apa *endpoint* yang harus diakses untuk mengambil suatu data tertentu. Sehingga, terkadang perlu dilakukan koordinasi yang baik antara bagian *frontend* dan *backend* dalam menentukan *endpoint* dari suatu data. Pada GraphQL, *backend* dapat menyediakan sebuah *playground* sekaligus dokumentasi akan *query* apa saja yang dapat dilakukan oleh *frontend*. Hal ini akan sangat memudahkan bagian *frontend* dalam menuliskan *query* untuk mengambil data yang diperlukan. Bagian *frontend* cukup mengakses 1 jenis *endpoint* saja, dan mengirimkan berbagai jenis *query* sesuai kebutuhan.

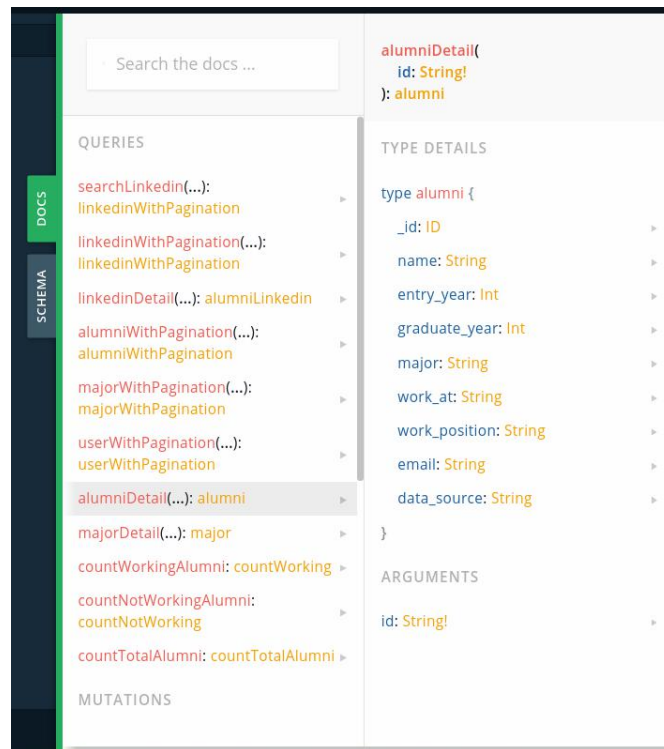
Berikut merupakan tampilan *playground* yang disediakan oleh *backend* sebagai sarana untuk melakukan percobaan *query*:



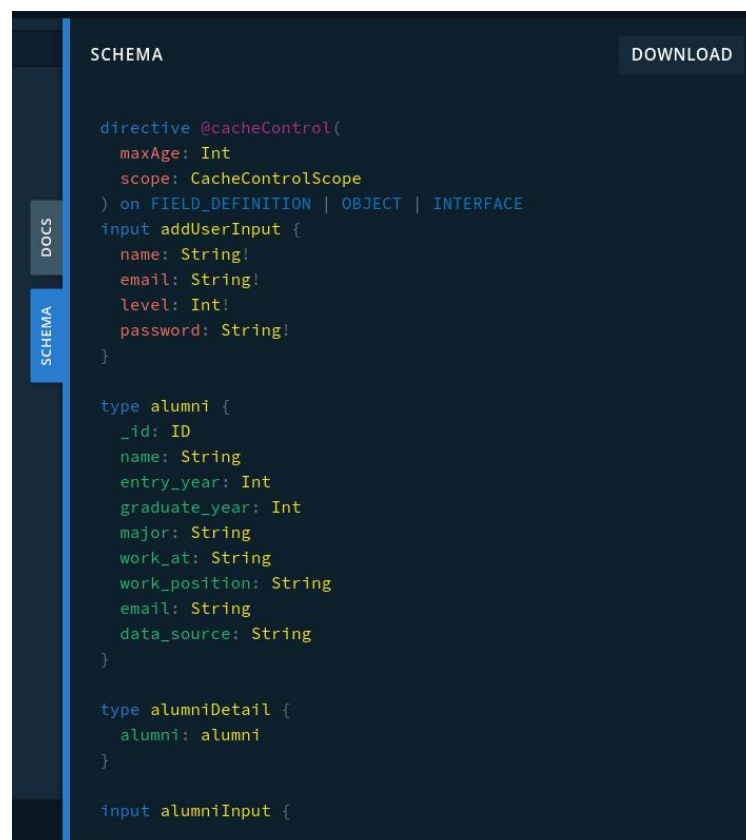
Gambar 5.10 Playground GraphQL

Bagian ruas kiri merupakan tempat untuk mengetikkan *query*. Pengetikan *query* juga dibantu dengan adanya fitur *autoComplete* seperti yang ada pada *text editor* ketika menekan tombol `ctrl + spasi`. Sedangkan bagian kanan merupakan bagian yang menampilkan hasil dari suatu *query* ketika dijalankan.

Pada *playground*, juga terdapat tombol *docs* dan *schema*. *Docs* berfungsi sebagai dokumentasi otomatis, dimana bagian *frontend* dapat melihat dokumentasi tersebut untuk mengetahui *query* dan *mutation* apa saja yang tersedia. *Schema* berfungsi sebagai dokumentasi yang mencatat tipe apa saja yang sudah dibuat oleh bagian *backend*. Semua dokumentasi tersebut dibuat secara otomatis oleh GraphQL. Sehingga, *backend* tidak perlu membuat dokumentasi untuk setiap *query* yang ada pada *backend* secara manual.



Gambar 5.11 Dokumentasi GraphQL



Gambar 5.12 Schema GraphQL

Setelah bagian *frontend* melakukan percobaan *query* di *playground* GraphQL dan berhasil, *query* tersebut dapat disalin ke dalam *text editor* untuk digunakan. Pada Apollo.js, *query* harus dibungkus terlebih dahulu menggunakan *gql``* dimana *query* harus diletakkan diantara tanda *backtick(``)*. contoh dari penulisan *query* pada *frontend* dengan GraphQL dapat dilihat pada lampiran 4. Dengan diberikannya tanda *gql``* pada awal *string*, maka *string* akan dikenali sebagai *query* atau *mutation* dari graphQL.

Setelah dikenali sebagai *query*/mutation, maka *query* sudah dapat digunakan. Pada Apollo.js, penggunaan *query* dapat dilakukan dengan 2 cara, yaitu menggunakan *useQuery* atau *useLazyQuery*. *useQuery* akan melakukan *http request* langsung ketika halaman pertamakali dimuat. Sedangkan *useLazyQuery* akan melakukan *http request* ketika suatu kondisi tertentu terpenuhi.

Untuk *mutation*, terdapat suatu fungsi pada Apollo.js yang bernama *useMutation*. Penggunaannya mirip dengan *useLazyQuery*, namun fungsi ini khusus digunakan untuk *mutation* pada GraphQL.

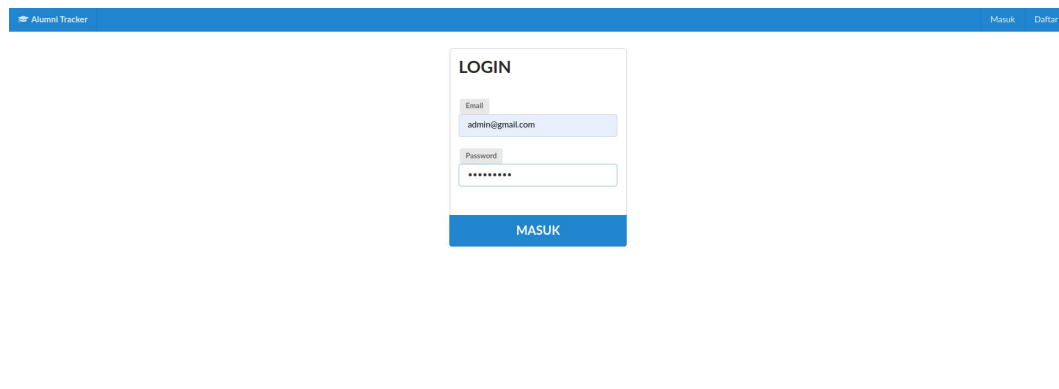
Sebagai contoh, penggunaan *useQuery* untuk menjalankan *query* Q_GET_ALL_TOTAL pada lampiran 4, dapat dilihat pada lampiran 5.

useQuery akan menjalankan *query* tersebut. Kemudian, apabila *query* berhasil dijalankan, *onCompleted* akan dijalankan. Apabila terjadi *error*, maka *onError* akan dijalankan. Pada saat proses pengambilan data, pengembang dapat memanfaatkan variabel *loading* yang didapat dari Apollo.js ketika menggunakan *useQuery* untuk menentukan apa yang harus ditampilkan atau dilakukan ketika data masing dalam proses pengambilan. Sehingga, hal ini dapat menghindari terjadinya *error* karena data yang akan ditampilkan masih belum selesai diproses seutuhnya. *Context* pada *useQuery* dapat digunakan sebagai headers untuk mengirimkan *token* pada *backend* untuk keperluan validasi dan otorisasi pengguna.

5.1.7 Tampilan Halaman

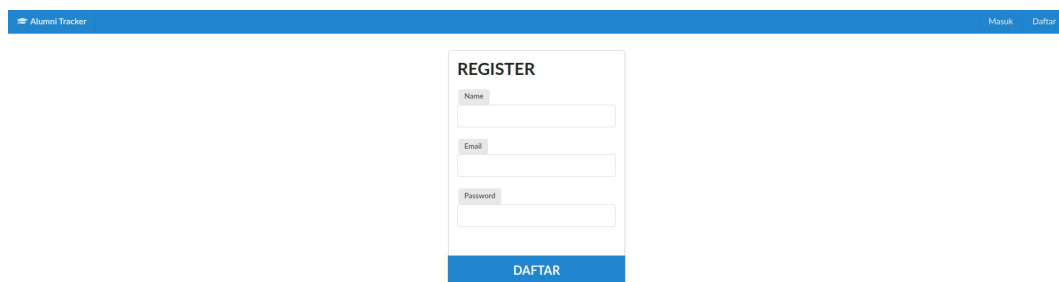
Karena kedua sistem informasi adalah sistem informasi pencarian alumni, maka tampilan dari kedua sistem informasi, baik yang menggunakan GraphQL

maupun yang menggunakan REST, adalah sama. Berikut tampilan dari sistem informasi yang dikembangkan:



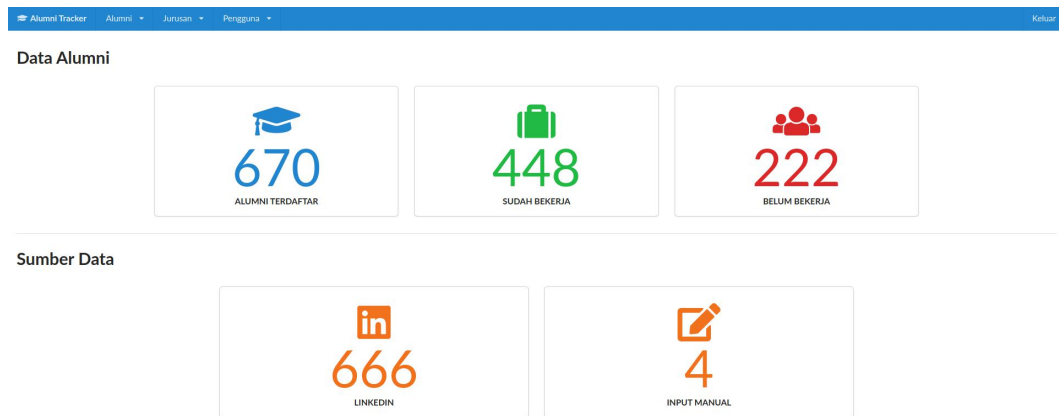
The screenshot shows the LOGIN page of the Alumni Tracker system. At the top, there is a blue header bar with the text "Alumni Tracker" on the left and "Masuk" and "Daftar" on the right. The main content area is white and contains a central login form. The form has a title "LOGIN" at the top. Below the title, there are two input fields: "Email" with the value "admin@gmail.com" and "Password" with a masked value "*****". At the bottom of the form is a blue button labeled "MASUK".

Gambar 5.13 Halaman login dengan isi



The screenshot shows the REGISTER page of the Alumni Tracker system. At the top, there is a blue header bar with the text "Alumni Tracker" on the left and "Masuk" and "Daftar" on the right. The main content area is white and contains a central registration form. The form has a title "REGISTER" at the top. Below the title, there are three input fields: "Name", "Email", and "Password". At the bottom of the form is a blue button labeled "DAFTAR".

Gambar 5.14 Halaman login



Gambar 5.15 Halaman Dashboard

The form includes the following fields:

- Nama
- Tahun Mulai Kuliah (1986)
- Tahun Lulus (2020)
- Jurusan (Pilih Jurusan)
- Nama Tempat Kerja/Perusahaan
- Jabatan
- Email

Buttons: [Tambah Alumni](#), [Batal](#)

Gambar 5.16 Halaman tambah data alumni

Data Input Manual

Jumlah alumni: 4 | Jumlah halaman: 1

Nama	Jabatan	Aksi
Fany Ervansyah Freelance Junior Frontend Engineer fanyervansyah.9c@gmail.com		Edit Delete
Johla Johny Hustling Senior Web Designer johlahjohny@gmail.com		Edit Delete
Tutik Sulistawati Back to the Future Senior Web Designer susilo@gmail.com		Edit Delete
GraphQL TRY 2 Back to the Future Security susilo2@gmail.com		Edit Delete

1

Data LinkedIn

Jumlah alumni: 666 | Jumlah halaman: 17

Scripting Masing

Nama	Jabatan
Aditya Eka Pradana Putra Pabrik Teknologi Frontend Web Developer	
Nadhifa Tiara PT IMAJIKU CIPITA MEDIA Internship Web Developer	
Hafidh Sajid Workshop Riset Informatika Crew	
Gusti Ananda Workshop Riset Informatika Crew	

Gambar 5.17 Halaman list alumni

Edit data Alumni

Nama
Johnny

Tahun Mulai Kuliah
2015

Tahun Lulus
2019

Jurusan
D-IV Akuntansi Manajemen

Nama Tempat Kerja/Perusahaan
Halomalang

Jabatan
Senior Web Designer

Email
johnjohny@gmail.com

Simpan Data Alumni Batal

Gambar 5.18 Halaman edit data alumni

Jurusan Terdaftar

Jumlah Jurusan: 22 Jumlah Halaman: 2

Nama Jurusan	EDIT	HAPUS
D-IV Teknik Informatika	EDIT	HAPUS
D-III Manajemen Informatika	EDIT	HAPUS
D-IV Akuntansi Manajemen	EDIT	HAPUS
D-IV Akuntansi Manajemen (Kelas Internasional)	EDIT	HAPUS
D-IV Bahasa Inggris	EDIT	HAPUS
D-IV Jaringan Telekomunikasi Digital	EDIT	HAPUS
D-IV Keuangan	EDIT	HAPUS
D-IV Manajemen Pemasaran	EDIT	HAPUS
D-IV Manajemen Pemasaran (Kelas Internasional)	EDIT	HAPUS
D-IV Manajemen Rekayasa Kinesioteknik	EDIT	HAPUS

Gambar 5.19 Halaman list jurusan

Tambah Jurusan

Gelar ☐ D-III ☐ D-IV

Nama Jurusan

Tambah Jurusan Batal

Gambar 5.20 Halaman tambah jurusan

Alumni Tracker Alumni Jurusan Pengguna Keluar

Edit Jurusan

Gelar ☐ D-III ☒ D-IV

Nama Jurusan

Teknik Informatika

Edit Jurusan Batal

Gambar 5.21 Halaman ubah data jurusan

Alumni Tracker Alumni Jurusan Pengguna Keluar

Daftar Akun Pengguna

Jumlah pengguna: 4 Jumlah halaman: 1

Nama	Email	Role			
admin	admin@gmail.com	admin	EDIT	RESET PASSWORD	HAPUS
user2	user2@gmail.com	user	EDIT	RESET PASSWORD	HAPUS
GraphQL TRY	graphql@gmail.com	admin	EDIT	RESET PASSWORD	HAPUS
Reg No Level	ofcoursethisis@gmail.com	admin	EDIT	RESET PASSWORD	HAPUS

1

Gambar 5.22 Halaman list pengguna

Alumni Tracker Alumni Jurusan Pengguna Keluar

Tambah Pengguna

Nama

Email

Password

Role

admin

Tambah Pengguna

Gambar 5.23 Halaman tambah data pengguna

Alumni Tracker Alumni Jurusan Pengguna Keluar

Ubah Data Pengguna

Nama
admin

Email
admin@gmail.com

Role
admin

Ubah Data Pengguna

Gambar 5.24 Halaman ubah data pengguna

RESET PASSWORD

anda akan mengubah password anda disini

Password

Password

UBAH PASSWORD

Gambar 5.25 Halaman reset password pengguna

Alumni Tracker Alumni Jurusan Pengguna Keluar

cari alumni di sini...

Cari

Data Input Manual

Jumlah alumni: 4 Jumlah halaman: 1

Fany Ervansyah
Freelance
Junior Frontend Engineer
fanyervansyah9c@gmail.com

Johla Johny
Hustlepreneur
Senior Web Designer
johlahjohny@gmail.com

Tutik Sulistawati
Back to Kindergarten
Senior Web Designer
susiilo@gmail.com

GraphQL TRY 2
Back to the Future
Security
susiilo2@gmail.com

Data LinkedIn

Jumlah alumni: 666 Jumlah halaman: 17

Aditya Eka Pradana Putra
Pirava Technology
Frontend Web Developer

Nadhifa Tiara
PT. MAJU RU. CITRA MEDIA Internship
Web Developer

Hafidh Sajid
Workshop Riset Informatika
Crew

Gusti Ananda
Workshop Riset Informatika
Crew

Gambar 5.26 Halaman list alumni oleh non-administrator

Nama Jurusan
D-IV Teknik Informatika
D-III Manajemen Informatika
D-IV Akuntansi Manajemen
D-IV Akuntansi Manajemen (Kelas Internasional)
D-IV Bahasa Inggris
D-IV Jaringan Telekomunikasi Digital
D-IV Keuangan
D-IV Manajemen Pemasaran
D-IV Manajemen Pemasaran (Kelas Internasional)
D-IV Manajemen Rekayasa Konstruksi
D-IV Manajemen Rekayasa Konstruksi (Kelas Internasional)
D-IV Sistem Kelistrikan
D-IV Sistem Kelistrikan (Kelas Internasional)

Gambar 5.27 Halaman list jurusan oleh non-administrator

5.2 Pengujian

Sesuai dengan apa yang dituliskan pada bab 3 subbab 3.4, pengujian akan dengan cara me-*refresh* halaman dan membandingkan masing-masing kecepatan pengambilan data dari tiap sistem, baik GraphQL maupun REST. Setelah halaman berhasil di-*refresh*, pada masing-masing *refresh* akan dicatat beberapa data yaitu:

- Waktu muat halaman,
- Jumlah *request* yang terjadi ke *backend* dalam memenuhi data yang akan ditampilkan,

Terdapat 3 kasus *under-fetching* pada sistem informasi pencarian alumni yang dibuat. Masalah tersebut terdapat pada:

- Halaman *dashboard*, dimana halaman tersebut menampilkan data berupa total alumni yang sudah bekerja, total alumni yang belum bekerja, total alumni keseluruhan, total data alumni yang berasal dari hasil *scraping* situs LinkedIn, dan total data alumni yang berasal dari *input* melalui sistem.
- Halaman *list* alumni, dimana halaman tersebut menunjukkan data total alumni yang sudah didapat, total halaman, data-data alumni dari hasil *scraping* situs LinkedIn, dan data alumni dari hasil *input* melalui sistem.
- Halaman ubah data alumni, dimana halaman tersebut harus melakukan *request* ke *backend* untuk mengisi *form* berupa data detail dari alumni, dan mengisi daftar jurusan yang tersedia pada suatu *dropdown*.

5.2.1 Uji Muat Halaman Dashboard

Pada halaman *dashboard*, kasus *under-fetching* terjadi ketika halaman harus melakukan beberapa permintaan GET ke bagian *backend* untuk memenuhi data berupa total alumni yang sudah bekerja, total alumni yang belum bekerja, total keseluruhan alumni, total data alumni yang didapat dari hasil *scraping* situs LinkedIn, dan total data alumni yang didapat dari hasil *input* manual melalui sistem. Pemuatan halaman memunculkan masalah *under-fetching* karena halaman tersebut harus melakukan request sebanyak 5 kali.

Hasil percobaan kecepatan pengambilan data pada sistem informasi dengan REST dan GraphQL adalah sebagai berikut:

Tabel 5.1 Hasil uji coba muat halaman dashboard REST API

Refresh ke-	Waktu Muat Halaman (ms)
1	987
2	589
3	548
4	587
5	604
6	608
7	584
8	592
9	591
10	583
11	664
12	598
13	631
14	593
15	592
16	639
17	619
18	664
19	569
20	618
Rata-Rata	623

Tabel 5.2 Hasil uji coba muat halaman dashboard GraphQL

Refresh ke-	Waktu Muat Halaman (ms)
1	761
2	652
3	635
4	603
5	591
6	654
7	697
8	634
9	620
10	598
11	612
12	613
13	602
14	690
15	595
16	597
17	627
18	619
19	660
20	654
Rata-Rata	635,7

5.2.2 Uji Muat Halaman List Alumni

Halaman *list* alumni menampilkan maksimal 80 data alumni, yaitu 40 data alumni yang berasal dari LinkedIn dan 40 data alumni yang berasal dari *input* melalui sistem. Pada halaman ini, terjadi *under-fetching* dimana halaman harus melakukan beberapa permintaan untuk mendapatkan data alumni yang bersumber dari LinkedIn dan data alumni yang bersumber dari *input* manual. Sehingga, terjadi 2 kali *request* pada sistem informasi dengan REST.

Hasil percobaan pemuatan halaman *list* alumni pada sistem informasi dengan menggunakan REST adalah sebagai berikut:

Tabel 5.3 Hasil uji coba muat halaman list alumni REST API

Refresh ke-	Waktu Muat Halaman (s)
1	1,37
2	1,38
3	1,47
4	1,37
5	1,40
6	1,44
7	1,50
8	1,42
9	1,45
10	1,37
11	1,36
12	1,39
13	1,37
14	1,38
15	1,40
16	1,39
17	1,39
18	1,41
19	1,34
20	1,36
Rata-Rata	1,398

Pada sistem informasi GraphQL, hanya terjadi 1 kali *request*. Hasil percobaan pemuatan halaman *list* alumni pada sistem informasi dengan menggunakan GraphQL adalah sebagai berikut:

Tabel 5.4 Hasil uji coba muat halaman list alumni GraphQL

Refresh ke-	Waktu Muat Halaman (ms)
1	954
2	892
3	913
4	942
5	921
6	926
7	905
8	960
9	932
10	938
11	944
12	975
13	992
14	946
15	912
16	967
17	962
18	979
19	989
20	941
Rata-Rata	944,5

5.2.3 Uji Muat Halaman Ubah Data Alumni

Pada halaman ubah data alumni, bagian *frontend* harus melakukan permintaan data kepada *backend* untuk memuat detail dari data alumni yang akan diubah, kemudian memuat daftar jurusan yang sudah terdaftar pada sistem. Sehingga, proses pengubahan data alumni dapat berjalan dengan baik dan data jurusan di halaman ubah data alumni selalu sinkron dengan daftar jurusan yang tersedia pada halaman *list* jurusan. Terjadi 2 kali *request* pada halaman ubah data

alumni dengan metode REST. Sedangkan pada sistem informasi dengan GraphQL, hanya terjadi 1 *request*.

Hasil percobaan pemuatan halaman ubah data alumni pada sistem informasi menggunakan REST adalah sebagai berikut:

Tabel 5.5 Hasil uji coba muat halaman ubah data alumni REST API

Refresh ke-	Waktu Muat Halaman (ms)
1	798
2	736
3	757
4	746
5	725
6	698
7	723
8	743
9	718
10	731
11	733
12	722
13	746
14	740
15	669
16	768
17	745
18	736
19	763
20	780
Rata-Rata	738,85

Hasil percobaan pemuatan halaman ubah data alumni pada sistem informasi dengan menggunakan GraphQL adalah sebagai berikut:

Tabel 5.6 Hasil uji coba muat halaman ubah data alumni GraphQL

Refresh ke-	Jumlah Request	Waktu Muat Halaman (ms)
1	1	944
2	1	800
3	1	735
4	1	758
5	1	790
6	1	722
7	1	843
8	1	772
9	1	765
10	1	767
11	1	777
12	1	732
13	1	740
14	1	782
15	1	734
16	1	780
17	1	720
18	1	744
19	1	910
20	1	779
Rata-Rata	1	779,7

5.2.4 Load Test Endpoint Untuk Halaman Dashboard

Load Test pada *endpoint* data yang ditampilkan pada halaman *dashboard* dilakukan untuk menguji coba kemampuan *backend* dengan REST dan GraphQL dalam menangani jumlah user yang cukup banyak dan terjadi permintaan secara bersamaan.

Hasil dari *load test endpoint* untuk halaman *dashboard* pada *backend* dengan REST API dapat dilihat pada lampiran 6a. Sedangkan hasil dari *load test endpoint* dari halaman *dashboard* pada *backend* dengan GraphQL dapat dilihat pada lampiran 6b.

5.2.5 Load Test Endpoint Untuk Halaman List Alumni

Load testing dilakukan pada *endpoint* yang diperlukan untuk menampilkan data di halaman list alumni. *Pengujian* dilakukan untuk menguji kemampuan *backend* dengan GraphQL dan *backend* dengan REST dalam menangani kebutuhan pengguna pada halaman *list* alumni ketika terjadi permintaan data oleh beberapa *user* sekaligus dalam waktu bersamaan.

Hasil dari *load testing endpoint* halaman *list* alumni pada *backend* dengan REST dapat dilihat pada lampiran 7a. Sedangkan hasil *load testing endpoint* halaman list alumni pada *backend* dengan GraphQL dapat dilihat pada lampiran 7b.

5.2.6 Load Test Endpoint Untuk Halaman Edit Alumni

Load test dilakukan pada *endpoint* yang diperlukan untuk memenuhi kebutuhan halaman *edit* alumni dan menguji kemampuan *backend* ketika terdapat beberapa user yang melakukan perubahan data alumni secara bersama-sama.

Hasil dari *load testing endpoint* halaman *edit* alumni pada *backend* dengan REST pada lampiran 8a. Sedangkan hasil dari *load testing endpoint* halaman *edit* alumni pada *backend* dengan GraphQL dapat dilihat pada lampiran 8b.

BAB VI. HASIL DAN PEMBAHASAN

6.1 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.

6.2 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 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.

6.3 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 halaman *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.

6.4 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 efisiensi penggunaan 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 dikirimkan 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 maupun *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 dikirimkan sekali dan juga mendapatkan keseluruhan data dalam 1 respon.

6.5 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 dikirimkan, 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 apa saja 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.

6.6 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.

BAB VII. KESIMPULAN DAN SARAN

7.1 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 penggunaan 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.

7.2 Saran

Kedua sistem informasi pelacakan alumni yang penulis buat, dikembangkan dengan bentuk GraphQL dan REST API yang sederhana dan membahas kekurangan REST API yaitu *under-fetching*. Pengembangan *website* dilakukan oleh penulis sesuai dengan apa yang penulis pahami mengenai bagaimana cara mengembangkan suatu halaman web dengan GraphQL dan REST API, baik di bagian *frontend* maupun *backend*. Penulis tidak membahas lebih jauh mengenai performa kedua metode pendistribusian data tersebut apabila kedua sistem informasi dikembangkan dan dilakukan penerapan metode-metode peningkatan performa yang lebih dalam seperti *caching*, *batching*, *momization* dan sebagainya. Sehingga, untuk peneliti yang ingin melanjutkan penelitian ini, dapat membahas tentang bagaimana performa GraphQL dan REST API yang sudah diterapkan dengan teknik optimasi tertentu.

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: O'Reilly.
- 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.

Lampiran 1. Kode tsconfig.json

```
{  
  "compilerOptions": {  
    "module": "commonjs",  
    "esModuleInterop": true,  
    "target": "es6",  
    "noImplicitAny": true,  
    "moduleResolution": "node",  
    "sourceMap": true,  
    "outDir": "dist",  
    "baseUrl": "src"  
  }  
}
```

Lampiran 2a. Kode untuk menyediakan resolver dari majorDetail

```
const resolvers = {  
  Query: {  
    ...  
    majorDetail: majorResolver.majorDetail,  
    ...  
  }  
}
```

Lampiran 2b. logic dari resolver majorDetail

```
export const majorResolver = {  
  ...,  
  majorDetail: async (  
    parent: any,  
    args: { id: string },  
    context: { token: string },  
    info: any  
  ) => {  
    verifyToken(context.token);  
    try {  
      const result = majorModel.findById(args.id);  
      return result;  
    } catch (error) {  
      throw new ApolloError(error);  
    }  
  },  
  ...  
}
```

Lampiran 3. mengambil data jurusan dengan http method GET

```
axios
  .get(`http://localhost:4000/major/${location.state}`, {
    headers: {
      authorization: `bearer ${token}`,
    },
  })
  .then((res) => {
    const degree = res.data.name.split(" ")[0];
    const name = res.data.name.replace(`${degree} `, "");
    setMajor({ degree, name });
  })
  .catch((error) => alert(error))
  .finally(() => setLoadingToFalse());
```


Lampiran 4. Contoh Query untuk halaman Dashboard

```
const Q_GET_ALL_TOTAL = gql`
  query counter {
    countWorkingAlumni {
      total
    }
    countNotWorkingAlumni {
      total
    }

    linkedinWithPagination {
      totalData
    }
    alumniWithPagination {
      totalData
    }
    countTotalAlumni {
      total
    }
  }
`;
```

Lampiran 5. Contoh penggunaan UseQuery

```
const { loading } = useQuery(Q_GET_ALL_TOTAL, {  
  context: {  
    ...  
  },  
  onCompleted: (data) => {  
    ...  
  },  
  onError: (error) => {  
    ...  
  }  
});
```

Lampiran 6a. Hasil Load Testing Endpoint Dashboard REST

```
{
  "metrics": {
    "checks": {
      "fails": 0,
      "passes": 755,
      "value": 0
    },
    "data_received": {
      "count": 1012434,
      "rate": 16873.865510100128
    },
    "data_sent": {
      "count": 361524,
      "rate": 6025.387684207997
    },
    "http_req_blocked": {
      "avg": 524.515193913907,
      "max": 2403.825321,
      "med": 0.008514,
      "min": 0.00207,
      "p(90)": 1479.5165854,
      "p(95)": 2214.2747550999998
    },
    "http_req_connecting": {
      "avg": 98.6629763668874,
      "max": 366.43499,
      "med": 0,
      "min": 0,
      "p(90)": 310.28249460000006,
      "p(95)": 328.97447789999998
    },
    "http_req_duration": {
      "avg": 14596.891970517876,
      "max": 20423.86639,
      "med": 15469.065901,
      "min": 2086.858578,
      "p(90)": 18796.854333400002,
      "p(95)": 18869.1891671
    }
  },
```

Lampiran 6a. Hasil Load Testing Endpoint Dashboard REST

```
"http_req_receiving": {
  "avg": 0.0961212331125827,
  "max": 0.319473,
  "med": 0.088005,
  "min": 0.042537,
  "p(90)": 0.1250028,
  "p(95)": 0.15804379999999999
},
"http_req_sending": {
  "avg": 0.04781197615894041,
  "max": 0.397023,
  "med": 0.033602,
  "min": 0.009182,
  "p(90)": 0.087433000000000002,
  "p(95)": 0.10320819999999996
},
"http_req_tls_handshaking": {
  "avg": 356.1563462013245,
  "max": 1821.782907,
  "med": 0,
  "min": 0,
  "p(90)": 941.4120884,
  "p(95)": 1661.3292841999998
},
"http_req_waiting": {
  "avg": 14596.748037308585,
  "max": 20423.641377,
  "med": 15468.973553,
  "min": 2086.564367,
  "p(90)": 18796.7273238,
  "p(95)": 18868.9883082
},
"http_reqs": {
  "count": 755,
  "rate": 12.583307613262294
},
"iteration_duration": {
  "avg": 16809.271156331128,
  "max": 21908.05061,
```

Lampiran 6a. Hasil Load Testing Endpoint Dashboard REST

```
        "med": 18749.826547,
        "min": 4153.023874,
        "p(90)": 20476.24521,
        "p(95)": 21237.632892499998
    },
    "iterations": {
        "count": 151,
        "rate": 2.5166615226524587
    },
    "vus": {
        "max": 50,
        "min": 50,
        "value": 50
    },
    "vus_max": {
        "max": 50,
        "min": 50,
        "value": 50
    }
},
"root_group": {
    "name": "",
    "path": "",
    "id": "d41d8cd98f00b204e9800998ecf8427e",
    "groups": {},
    "checks": {
        "is response CountAll 200?": {
            "name": "is response CountAll 200?",
            "path": "::is response CountAll 200?",
            "id": "3a9f2e0f9e07b4051996e41333547c29",
            "passes": 151,
            "fails": 0
        },
        "is response countAlumni 200?": {
            "name": "is response countAlumni 200?",
            "path": "::is response countAlumni 200?",
            "id": "98a43388412b337559970dd48af53005",
            "passes": 151,
            "fails": 0
        }
    }
}
```

Lampiran 6a. Hasil Load Testing Endpoint Dashboard REST

```
    },
    "is response countLinkedin 200?": {
      "name": "is response countLinkedin 200?",
      "path": "::-is response countLinkedin 200?",
      "id": "0f1dedd9429eb6e10c9c1afcaf8cca71",
      "passes": 151,
      "fails": 0
    },
    "is response countNotWorking 200?": {
      "name": "is response countNotWorking 200?",
      "path": "::-is response countNotWorking 200?",
      "id": "9d4f6aa20c8ca3975d268d193fb49e76",
      "passes": 151,
      "fails": 0
    },
    "is response countWorking 200?": {
      "name": "is response countWorking 200?",
      "path": "::-is response countWorking 200?",
      "id": "8d5694147ea605da4b53e755ac9a02f7",
      "passes": 151,
      "fails": 0
    }
  }
}
```

Lampiran 6b. Hasil Load Testing Endpoint Dashboard GraphQL

```
{
  "metrics": {
    "checks": {
      "fails": 0,
      "passes": 156,
      "value": 0
    },
    "data_received": {
      "count": 222666,
      "rate": 3711.0920815191225
    },
    "data_sent": {
      "count": 108636,
      "rate": 1810.59613667067
    },
    "http_req_blocked": {
      "avg": 379.724684224359,
      "max": 1186.631018,
      "med": 0.005503,
      "min": 0.004066,
      "p(90)": 1184.97150300000002,
      "p(95)": 1186.00177925
    },
    "http_req_connecting": {
      "avg": 98.28030036538462,
      "max": 308.20743,
      "med": 0,
      "min": 0,
      "p(90)": 307.156774000000004,
      "p(95)": 307.5146535
    },
    "http_req_duration": {
      "avg": 15848.431024980773,
      "max": 19757.150932,
      "med": 18634.4202715,
      "min": 736.518271,
      "p(90)": 18720.3935255,
      "p(95)": 18752.69569175
    }
  },
```

Lampiran 6b. Hasil Load Testing Endpoint Dashboard GraphQL

```
"http_req_receiving": {
  "avg": 0.0977950641025641,
  "max": 0.257663,
  "med": 0.08924850000000001,
  "min": 0.070405,
  "p(90)": 0.1192735,
  "p(95)": 0.14836675
},
"http_req_sending": {
  "avg": 0.06380787820512818,
  "max": 0.461452,
  "med": 0.038101,
  "min": 0.027399,
  "p(90)": 0.12848,
  "p(95)": 0.164321
},
"http_req_tls_handshaking": {
  "avg": 229.8881454615385,
  "max": 719.72126,
  "med": 0,
  "min": 0,
  "p(90)": 717.5712149999999,
  "p(95)": 717.7931925
},
"http_req_waiting": {
  "avg": 15848.269422038458,
  "max": 19757.033895,
  "med": 18634.3038445,
  "min": 736.291182,
  "p(90)": 18720.2488765,
  "p(95)": 18752.573789
},
"http_reqs": {
  "count": 156,
  "rate": 2.599994452305171
},
"iteration_duration": {
  "avg": 16228.468737589748,
  "max": 20232.163229,
```


Lampiran 6b. Hasil Load Testing Endpoint Dashboard GraphQL

```
        "med": 18635.44067,  
        "min": 1918.941204,  
        "p(90)": 18731.583906,  
        "p(95)": 18781.492193  
    },  
    "iterations": {  
        "count": 156,  
        "rate": 2.599994452305171  
    },  
    "vus": {  
        "max": 50,  
        "min": 50,  
        "value": 50  
    },  
    "vus_max": {  
        "max": 50,  
        "min": 50,  
        "value": 50  
    }  
},  
"root_group": {  
    "name": "",  
    "path": "",  
    "id": "d41d8cd98f00b204e9800998ecf8427e",  
    "groups": {},  
    "checks": {  
        "is response 200?": {  
            "name": "is response 200?",  
            "path": "::            "id": "affbb1946d73175d0677234760f1ae72",  
            "passes": 156,  
            "fails": 0  
        }  
    }  
}
```

Lampiran 7a. Hasil Load Testing Endpoint List Alumni REST

```
{
  "metrics": {
    "checks": {
      "fails": 0,
      "passes": 582,
      "value": 0
    },
    "data_received": {
      "count": 5220098,
      "rate": 87001.44505930618
    },
    "data_sent": {
      "count": 232650,
      "rate": 3877.4916089789085
    },
    "http_req_blocked": {
      "avg": 180.6292264933552,
      "max": 1220.161734,
      "med": 0.006096,
      "min": 0.003296,
      "p(90)": 1050.2395715,
      "p(95)": 1117.9599224999997
    },
    "http_req_connecting": {
      "avg": 45.334806450166134,
      "max": 310.367718,
      "med": 0,
      "min": 0,
      "p(90)": 266.3993891,
      "p(95)": 278.48137629999997
    },
    "http_req_duration": {
      "avg": 7003.127434935213,
      "max": 13702.845559,
      "med": 7173.0885165,
      "min": 516.441919,
      "p(90)": 9641.568163200001,
      "p(95)": 9828.749418349998
    }
  },
}
```

Lampiran 7a. Hasil Load Testing Endpoint List Alumni REST

```
"http_req_receiving": {
  "avg": 0.12001296013289038,
  "max": 0.354257,
  "med": 0.10905200000000001,
  "min": 0.068546,
  "p(90)": 0.1666779,
  "p(95)": 0.19825669999999999
},
"http_req_sending": {
  "avg": 0.03796306810631234,
  "max": 0.233299,
  "med": 0.0297345,
  "min": 0.014673,
  "p(90)": 0.0655665,
  "p(95)": 0.08614384999999998
},
"http_req_tls_handshaking": {
  "avg": 104.30461701993353,
  "max": 728.550347,
  "med": 0,
  "min": 0,
  "p(90)": 599.9748027,
  "p(95)": 646.63030355
},
"http_req_waiting": {
  "avg": 7002.969458906978,
  "max": 13702.729968,
  "med": 7172.9563495,
  "min": 516.116549,
  "p(90)": 9641.422218099999,
  "p(95)": 9828.569615199998
},
"http_reqs": {
  "count": 602,
  "rate": 10.033311620912542
},
"iteration_duration": {
  "avg": 9588.549022243995,
  "max": 13703.195309,
```

Lampiran 7a. Hasil Load Testing Endpoint List Alumni REST

```
        "med": 9533.11716,
        "min": 7068.292621,
        "p(90)": 10420.463843,
        "p(95)": 10922.623660000001
    },
    "iterations": {
        "count": 291,
        "rate": 4.849989504461046
    },
    "vus": {
        "max": 50,
        "min": 50,
        "value": 50
    },
    "vus_max": {
        "max": 50,
        "min": 50,
        "value": 50
    }
},
"root_group": {
    "name": "",
    "path": "",
    "id": "d41d8cd98f00b204e9800998ecf8427e",
    "groups": {},
    "checks": {
        "is alumni status 200": {
            "name": "is alumni status 200",
            "path": ">::is alumni status 200",
            "id": "5c17f51d4a8b7e94fb6e7746aa4ca517",
            "passes": 291,
            "fails": 0
        },
        "is linkedin alumni status 200": {
            "name": "is linkedin alumni status 200",
            "path": ">::is linkedin alumni status 200",
            "id": "ea47acac2a341d9a20952af3620d7a35",
            "passes": 291,
            "fails": 0
        }
    }
}
```

Lampiran 7a. Hasil Load Testing Endpoint List Alumni REST

	}
	}
}	
}	

Lampiran 7b. Hasil Load Testing Endpoint List Alumni GraphQL

```
{
  "metrics": {
    "checks": {
      "fails": 0,
      "passes": 309,
      "value": 0
    },
    "data_received": {
      "count": 4263453,
      "rate": 71057.40206204177
    },
    "data_sent": {
      "count": 321321,
      "rate": 5355.33885045228
    },
    "http_req_blocked": {
      "avg": 166.92572722977334,
      "max": 1078.215494,
      "med": 0.005318,
      "min": 0.00144,
      "p(90)": 1023.0680851999999,
      "p(95)": 1042.8629054
    },
    "http_req_connecting": {
      "avg": 43.62394172815533,
      "max": 289.817236,
      "med": 0,
      "min": 0,
      "p(90)": 267.97394560000004,
      "p(95)": 273.7156016
    },
    "http_req_duration": {
      "avg": 8776.949641126212,
      "max": 17811.121521,
      "med": 9138.930165,
      "min": 767.589421,
      "p(90)": 11723.5521044,
      "p(95)": 12682.284150399997
    }
  },
```

Lampiran 7b. Hasil Load Testing Endpoint List Alumni GraphQL

```
"http_req_receiving": {
  "avg": 11.537834071197418,
  "max": 272.255398,
  "med": 0.113933,
  "min": 0.038357,
  "p(90)": 0.2141824,
  "p(95)": 0.5604187999999995
},
"http_req_sending": {
  "avg": 0.048121152103559876,
  "max": 0.365541,
  "med": 0.037346,
  "min": 0.011019,
  "p(90)": 0.083283799999999998,
  "p(95)": 0.1087832
},
"http_req_tls_handshaking": {
  "avg": 104.13607062459548,
  "max": 675.886399,
  "med": 0,
  "min": 0,
  "p(90)": 637.62820619999999,
  "p(95)": 650.268478
},
"http_req_waiting": {
  "avg": 8765.363685902916,
  "max": 17810.980335,
  "med": 9138.816155,
  "min": 549.05967,
  "p(90)": 11723.383713,
  "p(95)": 12682.139395399996
},
"http_reqs": {
  "count": 309,
  "rate": 5.149989277979823
},
"iteration_duration": {
  "avg": 8944.193019854365,
  "max": 17811.436713,
```

Lampiran 7b. Hasil Load Testing Endpoint List Alumni GraphQL

```
        "med": 9179.606966,  
        "min": 1766.052313,  
        "p(90)": 11723.9100534,  
        "p(95)": 12682.604004399996  
    },  
    "iterations": {  
        "count": 309,  
        "rate": 5.149989277979823  
    },  
    "vus": {  
        "max": 50,  
        "min": 50,  
        "value": 50  
    },  
    "vus_max": {  
        "max": 50,  
        "min": 50,  
        "value": 50  
    }  
},  
"root_group": {  
    "name": "",  
    "path": "",  
    "id": "d41d8cd98f00b204e9800998ecf8427e",  
    "groups": {},  
    "checks": {  
        "is status 200": {  
            "name": "is status 200",  
            "path": "::            "id": "548d37ca5f33793206f7832e7cea54fb",  
            "passes": 309,  
            "fails": 0  
        }  
    }  
}
```


Lampiran 8a. Hasil Load Testing Endpoint Ubah Data Alumni REST

```
{
  "metrics": {
    "checks": {
      "fails": 0,
      "passes": 820,
      "value": 0
    },
    "data_received": {
      "count": 1697150,
      "rate": 28285.76981103721
    },
    "data_sent": {
      "count": 316560,
      "rate": 5275.988151537542
    },
    "http_req_blocked": {
      "avg": 139.52971779171727,
      "max": 1234.482858,
      "med": 0.006201,
      "min": 0.002673,
      "p(90)": 1105.055119,
      "p(95)": 1151.971284
    },
    "http_req_connecting": {
      "avg": 33.711557747868454,
      "max": 310.525933,
      "med": 0,
      "min": 0,
      "p(90)": 264.745933,
      "p(95)": 279.775418
    },
    "http_req_duration": {
      "avg": 6555.193409225339,
      "max": 8236.149603,
      "med": 7034.473293,
      "min": 529.331429,
      "p(90)": 7169.250133,
      "p(95)": 7232.880054
    }
  },
```

Lampiran 8a. Hasil Load Testing Endpoint Ubah Data Alumni REST

```
"http_req_receiving": {
  "avg": 0.10307387697929346,
  "max": 0.338932,
  "med": 0.095754,
  "min": 0.047924,
  "p(90)": 0.133554,
  "p(95)": 0.151979
},
"http_req_sending": {
  "avg": 0.03669027283800243,
  "max": 0.321436,
  "med": 0.030519,
  "min": 0.015521,
  "p(90)": 0.062168,
  "p(95)": 0.07326
},
"http_req_tls_handshaking": {
  "avg": 86.08346671132763,
  "max": 760.432623,
  "med": 0,
  "min": 0,
  "p(90)": 682.542011,
  "p(95)": 707.647942
},
"http_req_waiting": {
  "avg": 6555.05364507551,
  "max": 8236.027914,
  "med": 7034.349052,
  "min": 529.140906,
  "p(90)": 7169.109158,
  "p(95)": 7232.718735
},
"http_reqs": {
  "count": 821,
  "rate": 13.683302604284565
},
"iteration_duration": {
  "avg": 6883.005466770734,
  "max": 8753.161906,
```

Lampiran 8a. Hasil Load Testing Endpoint Ubah Data Alumni REST

```
        "med": 7081.644627,
        "min": 1893.146179,
        "p(90)": 7251.2458408,
        "p(95)": 7523.0033920999995
    },
    "iterations": {
        "count": 410,
        "rate": 6.833317987523351
    },
    "vus": {
        "max": 50,
        "min": 50,
        "value": 50
    },
    "vus_max": {
        "max": 50,
        "min": 50,
        "value": 50
    }
},
"root_group": {
    "name": "",
    "path": "",
    "id": "d41d8cd98f00b204e9800998ecf8427e",
    "groups": {},
    "checks": {
        "is Response get Detail Alumni 200?": {
            "name": "is Response get Detail Alumni 200?",
            "path": ">::is Response get Detail Alumni 200?",
            "id": "324272cd21515444029fee8907beb7b0",
            "passes": 411,
            "fails": 0
        },
        "is Response get majors 200?": {
            "name": "is Response get majors 200?",
            "path": ">::is Response get majors 200?",
            "id": "cbc113b71981eaec429ec789a08fc476",
            "passes": 411,
            "fails": 0
        }
    }
}
```

Lampiran 8a. Hasil Load Testing Endpoint Ubah Data Alumni REST

```
}  
  }  
    }  
  }
```

Lampiran 8b. Hasil Load Testing Endpoint Ubah Data Alumni GraphQL

```
{
  "metrics": {
    "checks": {
      "fails": 0,
      "passes": 415,
      "value": 0
    },
    "data_received": {
      "count": 956845,
      "rate": 15947.382815688743
    },
    "data_sent": {
      "count": 344355,
      "rate": 5739.237817511193
    },
    "http_req_blocked": {
      "avg": 133.65053000481916,
      "max": 1114.461031,
      "med": 0.005433,
      "min": 0.001923,
      "p(90)": 1108.2088978,
      "p(95)": 1109.1586235
    },
    "http_req_connecting": {
      "avg": 32.178403000000001,
      "max": 289.276804,
      "med": 0,
      "min": 0,
      "p(90)": 256.0248014,
      "p(95)": 269.2211841
    },
    "http_req_duration": {
      "avg": 6667.052146277104,
      "max": 7991.372117,
      "med": 7021.131145,
      "min": 531.533335,
      "p(90)": 7116.0807116,
      "p(95)": 7143.9020616
    }
  },
```

Lampiran 8b. Hasil Load Testing Endpoint Ubah Data Alumni GraphQL

```
"http_req_receiving": {
  "avg": 0.09692807710843371,
  "max": 0.315045,
  "med": 0.086795,
  "min": 0.035106,
  "p(90)": 0.11889320000000005,
  "p(95)": 0.172349099999999987
},
"http_req_sending": {
  "avg": 0.045884759036144585,
  "max": 0.710866,
  "med": 0.037115,
  "min": 0.015281,
  "p(90)": 0.067985400000000004,
  "p(95)": 0.095652899999999996
},
"http_req_tls_handshaking": {
  "avg": 78.95953020963853,
  "max": 674.062594,
  "med": 0,
  "min": 0,
  "p(90)": 643.305285,
  "p(95)": 656.0163696999999
},
"http_req_waiting": {
  "avg": 6666.909333440964,
  "max": 7991.262702,
  "med": 7021.007551,
  "min": 531.256735,
  "p(90)": 7115.9526412,
  "p(95)": 7143.6205584
},
"http_reqs": {
  "count": 415,
  "rate": 6.916651984920053
},
"iteration_duration": {
  "avg": 6801.014609754218,
  "max": 8540.850248,
```

Lampiran 8b. Hasil Load Testing Endpoint Ubah Data Alumni GraphQL

```
        "med": 7024.381618,  
        "min": 1639.337438,  
        "p(90)": 7127.9792452,  
        "p(95)": 7163.8951902  
    },  
    "iterations": {  
        "count": 415,  
        "rate": 6.916651984920053  
    },  
    "vus": {  
        "max": 50,  
        "min": 50,  
        "value": 50  
    },  
    "vus_max": {  
        "max": 50,  
        "min": 50,  
        "value": 50  
    }  
},  
"root_group": {  
    "name": "",  
    "path": "",  
    "id": "d41d8cd98f00b204e9800998ecf8427e",  
    "groups": {},  
    "checks": {  
        "is response status 200": {  
            "name": "is response status 200",  
            "path": ">::is response status 200",  
            "id": "04a1a203447895e588df93161ced45e7",  
            "passes": 415,  
            "fails": 0  
        }  
    }  
}
```