KERJA PRAKTIK - EF234603

Pengembangan dan Implementasi Fitur Baru pada Alat Pengembangan *Backend* Berbasis GraphQL dan Homura

Laboratorium Komputasi Berbasis Jaringan, Departemen Teknik Informatika - ITS
Jl. Teknik Kimia - Gedung Departemen Teknik Informatika,
Kampus Institut Teknologi Sepuluh Nopember Surabaya
Jalan Raya ITS, Sukolilo, Surabaya 60111
Periode: 31 Agustus 2024 - 30 November 2024

Oleh:

Muhammad Rafi Sutrisno 5025211167
Ahda Filza Ghaffaru 5025211144

Pembimbing Departemen
Bagus Jati Santoso, S.Kom., Ph.D.
Pembimbing Lapangan
Moch. Nafkhan Alzamzami, S.T., M.T.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2024

KERJA PRAKTIK - EF234603

Pengembangan dan Implementasi Fitur Baru pada Alat Pengembangan *Backend* Berbasis GraphQL dan Homura

Laboratorium Komputasi Berbasis Jaringan, Departemen Teknik Informatika - ITS Jl. Teknik Kimia - Gedung Departemen Teknik Informatika, Kampus Institut Teknologi Sepuluh Nopember Surabaya Jalan Raya ITS, Sukolilo, Surabaya 60111
Periode: 31 Agustus 2024 - 30 November 2024

Oleh:

Muhammad Rafi Sutrisno 5025211167
Ahda Filza Ghaffaru 5025211144

Pembimbing Departemen
Bagus Jati Santoso, S.Kom., Ph.D.

Pembimbing Lapangan
Moch. Nafkhan Alzamzami, S.T., M.T.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2024

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

DAFTAR KODE SUMBER

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN
KERJA PRAKTIK

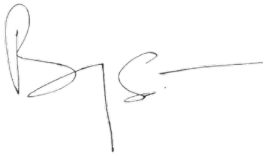
Pengembangan dan Implementasi Fitur Baru pada Alat Pengembangan *Backend* Berbasis GraphQL dan Homura

Oleh:

Muhammad Rafi Sutrisno	5025211167
Ahda Filza Ghaffaru	5025211144

Disetujui oleh Pembimbing Kerja Praktik:

1. Bagus Jati Santoso, S.Kom, Ph.D,
   NIP. 198611252018031001



(Pembimbing Departemen)

2. Moch. Nafkhan Alzamzami, S.T, M.T.
   NIP. 1999202311033

(Pembimbing Lapangan)

[Halaman ini sengaja dikosongkan]

Pengembangan dan Implementasi Fitur Baru pada Alat Pengembangan *Backend* Berbasis GraphQL dan Homura

Nama Mahasiswa	: Muhammad Rafi Sutrisno
NRP	: 5025211167
Nama Mahasiswa	: Ahda Filza Ghaffaru
NRP	: 5025211144
Departemen	: Teknik Informatika FTEIC-ITS
Pembimbing Departemen	: Bagus Jati Santoso, S.Kom., Ph.D.
Pembimbing Lapangan	: Moch. Nafkhan Alzamzami, S.T., M.T.

ABSTRAK

Kerja praktik ini bertujuan untuk mengembangkan dan menambah fitur pada framework GraphQL berbasis TypeScript, yang dikenal dengan nama Homura. Framework ini memungkinkan pengembang untuk membangun API GraphQL secara otomatis berdasarkan struktur database, memudahkan proses pembuatan API dan meningkatkan efisiensi pengembangan. Selama kerja praktik ini, beberapa fitur tambahan diterapkan, termasuk implementasi penyelesaian masalah N+1 query, pengelolaan file upload, serta penambahan fungsi agregasi seperti count, sum, average, min, dan max pada query GraphQL. Selain itu, dilakukan pula implementasi unit testing untuk memastikan bahwa setiap fitur berjalan dengan baik dan sesuai dengan harapan. Evaluasi dilakukan dengan menguji kinerja dan ketepatan fitur-fitur yang ditambahkan, serta membandingkan hasil yang diperoleh dengan ekspektasi yang telah ditetapkan. Berdasarkan pengujian yang telah penulis lakukan, didapatkan bahwa masing-masing fitur yang ditambahkan memenuhi kriteria

pengujian. Hasil dari kerja praktik ini diharapkan dapat memberikan kontribusi dalam pengembangan framework Homura dan meningkatkan pemahaman tentang penerapan GraphQL dan TypeScript dalam pengembangan aplikasi berbasis API.
Kata Kunci : GraphQl, N+1 Problem, Unit Testing, File, Agregasi

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Allah SWT atas penyertaan dan karunia-Nya sehingga penulis dapat menyelesaikan salah satu kewajiban penulis sebagai mahasiswa Departemen Teknik Informatika ITS yaitu Kerja Praktik yang berjudul: Pengembangan dan Implementasi Fitur-Fitur baru pada Sistem Homura.

Penulis menyadari bahwa masih banyak kekurangan baik dalam melaksanakan kerja praktik maupun penyusunan buku laporan kerja praktik ini. Namun penulis berharap buku laporan ini dapat menambah wawasan pembaca dan dapat menjadi sumber referensi.

Melalui buku laporan ini penulis juga ingin menyampaikan rasa terima kasih kepada orang-orang yang telah membantu menyusun laporan kerja praktik baik secara langsung maupun tidak langsung antara lain:

1. Kedua orang tua penulis.
2. Bapak Bagus Jati Santoso, S.Kom., Ph.D. selaku dosen pembimbing kerja praktik.
3. Bapak Moch. Nafkhan Alzamzami, S.T., M.T. selaku pembimbing lapangan selama kerja praktik berlangsung.

Surabaya, 25 Desember 2024
Muhammad Rafi Sutrisno dan Ahda Filza Ghaffaru

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

1.1. Latar Belakang

Dalam pengembangan aplikasi modern, GraphQL semakin populer sebagai alternatif untuk REST API karena fleksibilitasnya dalam memberikan data sesuai kebutuhan klien. GraphQL memungkinkan *developer* untuk mendefinisikan struktur *query* secara eksplisit, sehingga mengurangi jumlah data yang ditransfer melalui jaringan [1]. Salah satu implementasi inovatif dari GraphQL adalah *framework* Homura yang dikembangkan untuk mempermudah pembuatan API berbasis GraphQL secara otomatis. Dengan Homura, pengembang dapat dengan cepat menghasilkan berbagai operasi *query* tanpa harus menulis kode yang kompleks.

Meskipun Homura telah menghadirkan efisiensi yang signifikan dalam pengembangan API, *framework* ini juga menghadapi tantangan yang sering ditemukan dalam GraphQL, seperti *N+1 query problem* pada akses basis data. Masalah ini terjadi ketika satu *query* GraphQL memicu sejumlah besar *query* basis data yang berlebihan, sehingga dapat mempengaruhi performa sistem secara signifikan. Oleh karena itu, optimalisasi *query* melalui solusi seperti *data loaders* atau strategi lainnya menjadi langkah penting untuk meningkatkan kinerja *framework* ini sekaligus mendukung efisiensi dalam pengolahan data.

Selain efisiensi, kebutuhan untuk mendukung tipe data yang lebih kompleks, seperti *file*, juga semakin relevan dengan meningkatnya permintaan untuk aplikasi yang beragam. Penanganan tipe *data file* melalui GraphQL masih membutuhkan pendekatan yang spesifik, seperti penggunaan spesifikasi GraphQL *Upload*, yang memungkinkan *backend* untuk menangani unggahan *file* dengan cara yang standar dan aman. Implementasi fitur ini

dapat memperluas kemampuan *framework* sehingga dapat memenuhi kebutuhan aplikasi yang lebih kompleks.

Di sisi lain, operasi agregasi, seperti menghitung jumlah data (*count*), mencari nilai maksimum atau rata-rata, adalah fitur penting dalam banyak aplikasi berbasis data. Operasi ini memberikan wawasan yang bermanfaat kepada pengembang dan pengguna dalam memahami data secara keseluruhan. Penggunaan fungsi agregasi dalam GraphQL memungkinkan pengembang untuk memanfaatkan data secara efisien tanpa harus memindahkan logika tersebut ke sisi klien, sehingga meningkatkan kinerja dan kesederhanaan aplikasi.

Terakhir, untuk memastikan keandalan dan kualitas *framework* yang dikembangkan, penting untuk mengimplementasikan pengujian unit secara menyeluruh. *Unit testing* memainkan peran penting dalam mendeteksi dan mencegah *bug* sejak tahap awal pengembangan, serta memastikan bahwa setiap fungsi bekerja sesuai spesifikasi. Dalam pengembangan *framework* seperti Homura, *unit testing* menjadi landasan untuk membangun kepercayaan pengembang terhadap stabilitas sistem.

Melalui pengembangan fitur-fitur seperti *solving N+1 problem*, fitur *file upload*, fungsi agregasi, dan pengujian unit, Homura dapat menjadi *framework* GraphQL yang lebih kaya fitur dan andal bagi pengembang *backend*. Langkah ini sejalan dengan tren pengembangan *framework* modern yang mengutamakan efisiensi, fleksibilitas, dan pengalaman pengguna yang optimal.

1.2. Tujuan

Tujuan Kerja Praktik ini adalah untuk menyelesaikan kewajiban kuliah kerja praktik di Institut Teknologi Sepuluh Nopember dengan beban 4 SKS. Selain itu, kerja praktik ini

bertujuan untuk memenuhi kebutuhan penelitian dan pengembangan fitur pada sistem kerangka kerja Homura.

Kerja Praktik ini memiliki beberapa tujuan, yaitu mengembangkan *framework* Homura dengan fitur-fitur baru yang meliputi:

1. Mengimplementasikan solusi untuk *N+1 query problem* dalam GraphQL untuk meningkatkan efisiensi performa *query*.
2. Menambahkan dukungan untuk unggahan *file* dengan memanfaatkan spesifikasi *GraphQL Upload*.
3. Mengembangkan fungsi agregasi, seperti *Count*, *Max*, *Average*, *Min*, dan *Sum*, untuk mendukung analisis data yang efisien.
4. Menyusun dan menjalankan pengujian unit untuk menguji fitur.

1.3. Manfaat

Kerja Praktik ini memberikan manfaat baik dari segi pengembangan *framework* maupun pengembangan kemampuan pribadi. *Framework* Homura yang dikembangkan dalam kerja praktik ini diharapkan dapat memudahkan pengembang *backend* untuk membuat API berbasis GraphQL secara otomatis. Selain itu, bagi pelaksana kerja praktik, pengembangan fitur di Homura juga memberikan kesempatan untuk belajar bagaimana cara mengembangkan suatu kerangka kerja (*framework*) perangkat lunak secara sistematis.

1.4. Rumusan Masalah

Rumusan masalah dari kerja praktik ini adalah sebagai berikut:

1. Bagaimana cara menyelesaikan masalah *N+1* pada sistem Homura?

2. Bagaimana implementasi pengujian unit (*unit testing*) pada sistem Homura?
3. Bagaimana cara mengimplementasikan tipe data *file* pada sistem Homura?
4. Bagaimana cara mengimplementasikan fungsi agregasi pada sistem Homura?

1.5. Lokasi dan Waktu Kerja Praktik

Kerja praktik ini dilaksanakan pada waktu dan tempat sebagai berikut:
Lokasi Waktu : *Work From Home* (di rumah masing-masing)
Waktu : 31 Agustus 2024 – 31 November 2024
Hari Kerja : Senin - Jumat
Jam Kerja : 08.00 WIB - 17.00 WIB

1.6. Metodologi Kerja Praktik

Metodologi dalam pembuatan buku kerja praktik meliputi:

1.6.1. Perumusan Masalah

Kerja praktik ini diawali dengan penjelasan dari dosen pembimbing lapangan mengenai sistem Homura yang telah ada sebelumnya. Penjelasan yang diberikan oleh dosen pembimbing lapangan menghasilkan beberapa catatan terkait gambaran umum kebutuhan dan fitur-fitur yang harus ada dalam sistem Homura. Dalam kerja praktik ini, dibutuhkan beberapa implementasi fitur baru, termasuk penyelesaian masalah *N+1*, implementasi pengujian unit (*unit testing*), penambahan tipe *data file*, dan pengembangan fungsi agregasi.

1.6.2. Studi Literatur

Tahap ini dilakukan untuk memahami secara mendalam konsep dan teknologi yang mendukung

implementasi fitur-fitur baru pada sistem Homura. Studi literatur mencakup kajian mengenai GraphQL, *unit testing*, operasi agregasi, metode untuk mengatasi masalah *N+1*, serta teknologi lain yang berkaitan dengan sistem Homura. Referensi dari penelitian sebelumnya digunakan untuk memastikan implementasi sesuai dengan kebutuhan sistem.

1.6.3. Perancangan Desain Sistem

Pada tahap ini akan dilakukan perancangan sistem. fokus utama tahap ini adalah merancang struktur dan komponen-komponen yang akan membentuk fitur sistem.

1.6.4. Implementasi Sistem

Setelah permasalahan dirumuskan dan kajian literatur selesai dilakukan, implementasi sistem dilakukan. Tahap ini melibatkan penambahan dan pengujian fitur baru pada Homura, termasuk optimasi pengambilan data untuk menyelesaikan masalah *N+1*, penerapan *unit testing*, penyesuaian tipe data untuk mendukung data *file*, dan pengembangan fungsi agregasi untuk memenuhi kebutuhan analisis data.

1.6.5. Pengujian dan Evaluasi

Pada tahap ini, pengujian dilakukan untuk memastikan bahwa fitur-fitur yang dikembangkan pada Homura berfungsi sesuai dengan spesifikasi dan kebutuhan sistem. Selanjutnya, dilakukan evaluasi bersama dosen pembimbing lapangan untuk meninjau hasil implementasi dan menentukan apakah diperlukan penyesuaian atau penambahan fitur agar hasil kerja praktik sesuai dengan tujuan penelitian Homura.

1.6.6. Kesimpulan dan Saran

Pengujian yang dilakukan ini telah memenuhi syarat yang diinginkan, dan berjalan dengan baik dan lancar.

1.7. Sistematika Laporan

1.7.1. Bab I Pendahuluan

Bab ini berisi latar belakang, tujuan, manfaat, rumusan masalah, lokasi dan waktu kerja praktik, metodologi, dan sistematika laporan.

1.7.2. Bab II Profil Perusahaan

Bab ini berisi gambaran umum profil Laboratorium Komputasi Berbasis Jaringan Departemen Teknik Informatika ITS.

1.7.3. Bab III Tinjauan Pustaka

Bab ini berisi dasar teori dari teknologi yang digunakan dalam menyelesaikan proyek kerja praktik.

1.7.4. Bab IV Analisis dan Perancangan Infrastruktur Sistem

Bab ini berisi mengenai tahap analisis sistem fitur dalam menyelesaikan proyek kerja praktik.

1.7.5. Bab V Implementasi Sistem

Bab ini berisi uraian tahap - tahap yang dilakukan untuk proses implementasi fitur.

1.7.6. Bab VI Pengujian dan Evaluasi

Bab ini berisi hasil uji coba dan evaluasi dari fitur yang telah dikembangkan selama pelaksanaan kerja praktik.

1.7.7. Bab VII Kesimpulan dan Saran

Bab ini berisi kesimpulan dan saran yang didapat dari proses pelaksanaan kerja praktik.

[Halaman ini sengaja dikosongkan]

BAB II
PROFIL INSTANSI

2.1. Profil Instansi

Pendidikan tinggi diarahkan untuk mempersiapkan bangsa Indonesia dalam menghadapi era pembangunan industri dan informasi. Untuk itu pemerintah melalui Direktorat Jendral Pendidikan Tinggi pada tahun 1985 menginstruksikan untuk membuka Program Studi S1 baru untuk bidang ilmu teknologi komputer di empat universitas atau institut di mana ITS termasuk di dalamnya. Di ITS, program ini awalnya diberi nama Program Studi Teknik Komputer. Namun sejak tahun 1993, nama Program Studi Teknik Komputer diubah menjadi Jurusan Teknik Komputer. Akhirnya, pada tahun 1996 secara resmi jurusan ini berganti nama menjadi Jurusan Teknik Informatika berdasarkan Surat Keputusan Direktur Jendral Pendidikan Tinggi Nomor 224/DIKTI/Kep/1996, tanggal 11 Juli 1996. Pada saat ini, Departemen Teknik Informatika memperoleh nilai akreditasi A berdasarkan Surat Keputusan Badan Akreditasi Nasional Perguruan Tinggi (BAN-PT) Nomor 003/BANPT/Ak-X/S1/V/2006, tanggal 18 Mei 2006 [2].

2.2. Visi dan Misi Instansi

Visi Departemen Teknik Informatika

Visi Departemen Teknik Informatika adalah menjadi inovator bidang informatika yang unggul di tingkat nasional dengan reputasi internasional, serta berperan aktif dalam upaya memajukan dan mensejahterakan bangsa

Misi Departemen Teknik Informatika

1. Menyelenggarakan proses pembelajaran yang berkualitas, dan memenuhi standar nasional maupun internasional.
2. Melaksanakan penelitian yang inovatif, bermutu, dan bermanfaat.
3. Meningkatkan pemanfaatan teknologi informasi dan komunikasi untuk masyarakat.
4. Menjalin kemitraan dengan berbagai lembaga, baik di dalam maupun di luar negeri.

2.3. Laboratorium Komputasi Berbasis Jaringan

2.3.1. Pendahuluan

Laboratorium Komputasi Berbasis Jaringan berfokus pada Kemampuan lulusan sarjana/magister/doktor dalam membangun infrastruktur jaringan yang aman, kemampuan membangun sistem grid, Kemampuan membangun aplikasi jaringan sesuai Standard dan Kemampuan membangun aplikasi multimedia berbasis jaringan [3].

2.3.2. Mata Kuliah

Yang diampu oleh Rumpun Mata Kuliah KBJ, antara lain:

1. Komputasi Bergerak.
2. Sistem Terdistribusi.
3. Keamanan Informasi dan Jaringan.
4. Jaringan Multimedia.
5. Komputasi Awan.
6. Forensik Digital.

7. Komputasi Pervasif dan Jaringan Sensor.
8. Topik Khusus Komputasi berbasis Jaringan.

2.3.3. Struktur Organisasi Laboratorium Komputasi Berbasis Jaringan

1. Prof. Tohari Ahmad, S.Kom., M.IT., Ph.D. sebagai ketua RMK
2. Bagus Jati Santoso, S.Kom., Ph.D.
3. Hudan Studiawan, S.Kom., M.Kom., Ph.D.
4. Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D.
5. Moch. Nafkhan Alzamzami, S.T., M.T.

2.3.4. Penelitian

1. Identifying the Location of Possible Hidden Data in Spatial Domain Images
2. Deteksi Serangan Siber pada Multivariate Time Series Data Menggunakan Arsitektur Transformer
3. Digitalal Image Forensics: Detecting Secret Information through Image Steganalysis
4. Decompositing Transformer using Autocorrelayion for Long-Term Series Forecasting
5. Analisis Forensik pada Berkas iOS Cookie
6. Deteksi Aktivitas Bot Bergrup Secara Paralel untuk Meningkatkan Keamanan Jaringan Komputer

7. Pengembangan Kerangka Kerja Forensik Drone dengan menggunakan Teknik Analisis Log pada Artefak Data Log Perangkat Drone
8. Meningkatkan Performa Machine Learning dalam Klasifikasi Litologi Sektor Minyak dan Gas (Migas) melalui Teknik Penanganan Missing Values yang Efektif
9. Seleksi dan Pembobotan Fitur Menggunakan Algoritma Gravitational Search pada Customer Churn Prediction
10. Cloud task scheduling menggunakan artificial fish swarm algorithm dan opposition based learning
11. Pembelajaran Interaktif Menggunakan Classpoint Di SD Raden Patah

[Halaman ini sengaja dikosongkan]

BAB III
TINJAUAN PUSTAKA

3.1. Homura

Homura adalah alat pengembangan layanan *backend* berbasis GraphQL yang dirancang untuk menyederhanakan implementasi operasi CRUD (*Create, Read, Update, Delete*), autentikasi, dan sistem otorisasi. Dikembangkan menggunakan *TypeScript*, Homura bertujuan untuk mengurangi jumlah kode yang harus ditulis oleh pengembang saat membangun layanan *backend*. GraphQL dipilih sebagai bahasa *query* karena kemampuannya untuk mendefinisikan model data dan *metadata* melalui *schema*. Dengan memanfaatkan schema ini, Homura dapat menghasilkan API GraphQL secara otomatis berdasarkan definisi model data dan operasi CRUD yang diinginkan.

Homura menggunakan JSON Web Token (JWT) untuk autentikasi. JWT menyediakan konteks subjek yang dapat digunakan dalam metode otorisasi. Sistem otorisasi pada Homura mengadopsi Attribute-Based Access Control (ABAC), yang memungkinkan validasi hingga level atribut untuk memenuhi berbagai kebutuhan sistem. Hasil pengujian menunjukkan bahwa Homura berhasil menciptakan *server* API GraphQL lengkap, termasuk fitur CRUD, autentikasi, dan otorisasi. Selain itu, Homura mampu mengurangi jumlah kode yang perlu ditulis oleh pengembang hingga 82%, menjadikannya alat yang efisien untuk pengembangan layanan *backend* berbasis GraphQL [4].

3.2. GraphQl

GraphQL adalah salah satu bahasa *query* untuk API yang pertama kali dikembangkan oleh Facebook pada tahun 2015 dan dirilis secara publik pada tahun 2016 setelah mereka gunakan selama tiga tahun. GraphQL memberikan pendekatan baru dalam mengirim dan menerima informasi melalui internet, menjadikannya alternatif dari REST API. Sejak tanggal rilisnya, GraphQL telah mendapatkan banyak pengguna dari perusahaan besar seperti Coursera, Github, Neo4J, dan Pinterest [5].

Dalam pengambilan informasi, GraphQL memberikan fleksibilitas yang besar karena klien dapat menyesuaikan penulisan *query*-nya berdasarkan atribut yang dibutuhkan, kemudian *server* hanya akan mengembalikan data tersebut. Selain itu, GraphQL juga menggunakan skema yang mampu untuk mendeskripsikan tipe data dan *query* yang didukung oleh API [4], hal ini dapat memudahkan pengembangan serta pemeliharaan dari API.

3.3. GraphQL *Upload*

GraphQL *Upload* adalah mekanisme untuk menangani pengunggahan *file* melalui API GraphQL. Meskipun GraphQL dirancang untuk menangani data berbentuk *query* dan *mutation*, pengunggahan *file* memerlukan pendekatan khusus karena *file* tidak dapat dikirim langsung sebagai bagian dari JSON *payload*. Untuk mengatasi hal ini, GraphQL Upload memanfaatkan protokol *multipart request*, di mana *file* dikirim bersama data lainnya dalam satu permintaan HTTP. Pendekatan ini memungkinkan pengembang untuk menangani *file* di

dalam *resolvers* GraphQL, memberikan fleksibilitas untuk memproses dan menyimpan *file* sesuai kebutuhan.

Dalam implementasinya, GraphQL *Upload* sering menggunakan library seperti *graphql-upload* untuk menyediakan *middleware* dan *utilities* yang mendukung pengunggahan *file*. Middleware ini menangani *parsing multipart request*, memetakan *file* ke input GraphQL, dan menghasilkan *stream file* yang dapat diakses dalam *resolver*. Dengan GraphQL *Upload*, pengembang dapat dengan mudah mengintegrasikan fitur pengunggahan *file* ke dalam API mereka tanpa memerlukan *endpoint* khusus, sehingga menjaga keseragaman arsitektur API GraphQL.

3.4. Typescript

Typescript adalah bahasa pemrograman yang dikembangkan oleh Microsoft sebagai alternatif dari Javascript, yang membedakan kedua bahasa pemrograman ini adalah kemampuan untuk memberikan tipe data statis pada suatu objek. Pada Typescript, hal tersebut wajib untuk dilakukan sehingga memungkinkan *developer* untuk mengidentifikasi kesalahan lebih awal dalam mengembangkan suatu aplikasi.

Fitur-fitur yang diberikan oleh Typescript terkadang dikaitkan dengan bahasa pemrograman lain yang berorientasi objek, misalkan seperti C atau Java. Menurut [6], hal ini bertujuan untuk memberikan definisi yang jelas sehingga bisa mengurangi ambiguitas dalam dokumentasi dan menjaga konsistensi pada bahasa maupun *compiler*.

3.5. N+1 Problem

N+1 problem adalah istilah yang digunakan untuk menggambarkan masalah performa yang terjadi ketika sebuah sistem melakukan serangkaian *query* secara berulang. *N+1 queries* adalah masalah performa di mana aplikasi melakukan *query database* dalam sebuah *loop*, bukannya melakukan satu *query* yang mengembalikan atau memodifikasi semua informasi sekaligus [7]. Pola ini muncul ketika permintaan awal menghasilkan kebutuhan untuk melakukan sejumlah permintaan tambahan pada data terkait. Akibatnya, jumlah permintaan yang dilakukan menjadi jauh lebih banyak dari yang diperlukan.

Untuk mengatasi masalah *N+1*, salah satu solusi yang sering digunakan adalah *data loader*. Tugas utama dari *data loader* adalah menggantikan beberapa permintaan serupa dengan satu permintaan yang digabungkan [8]. *Data loader* bekerja dengan menggabungkan beberapa permintaan serupa menjadi satu permintaan besar melalui mekanisme *batching*, sehingga mengurangi jumlah permintaan yang dilakukan ke *database*. Dalam konteks GraphQL, penggunaan *data loader* dapat meningkatkan efisiensi pengambilan data dengan menghindari eksekusi *query* berulang untuk setiap elemen data yang diminta.

3.6. Unit Testing

Pengujian unit (*Unit Testing*) adalah *Unit testing* adalah proses pengujian perangkat lunak yang fokus pada pengujian bagian terkecil atau unit dari suatu sistem. Pengujian unit dilakukan untuk memastikan bahwa unit tersebut berfungsi dengan benar sesuai dengan yang diharapkan.

Salah satu contoh metode pengujian unit adalah *Snapshot testing*. *Snapshot testing* pada *backend*

digunakan untuk memverifikasi bahwa *output* dari API atau fungsi *backend* tertentu tetap konsisten. Misalnya, pengujian ini dapat digunakan untuk memastikan bahwa respons dari *endpoint* GraphQL atau REST API tidak berubah setelah pembaruan kode. *Snapshot testing* mencatat respons yang dihasilkan dalam bentuk *snapshot*, yang kemudian digunakan sebagai referensi untuk pengujian berikutnya.

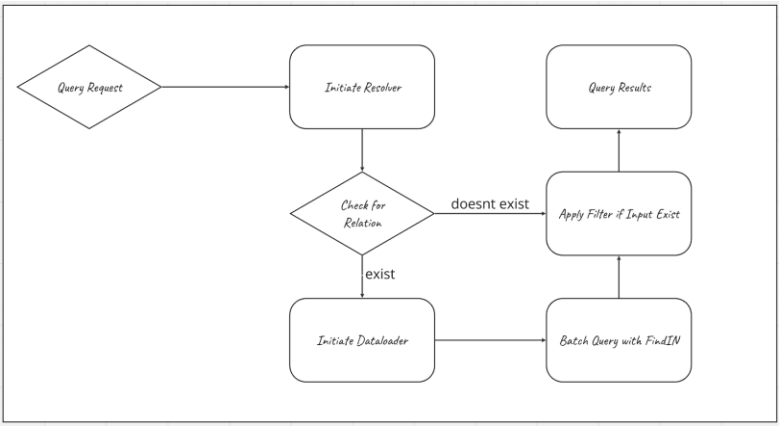
3.7. Agregasi

Agregasi adalah metode *query* untuk mengumpulkan sekumpulan nilai untuk mengembalikan satu nilai tunggal. Hal ini dilakukan dengan bantuan fungsi agregasit, seperti *SUM, COUNT, MIN, MAX* dan *AVG*. *SUM* adalah fungsi agregasi yang digunakan untuk menghitung jumlah total dari nilai-nilai dalam kolom numerik. *COUNT* adalah fungsi agregasi yang digunakan untuk menghitung jumlah baris dalam tabel atau jumlah nilai tidak kosong dalam kolom tertentu. *MIN* adalah fungsi agregasi yang digunakan untuk mendapatkan nilai terendah dari sebuah kolom. *MAX* adalah fungsi agregasi yang digunakan untuk mendapatkan nilai tertinggi dari sebuah kolom. *AVG* adalah fungsi agregasi yang digunakan untuk menghitung rata-rata dari nilai-nilai dalam kolom numerik.

[Halaman ini sengaja dikosongkan]

BAB IV
PERANCANGAN DESAIN SISTEM

4.1. Arsitektur Sistem *Data Loader*



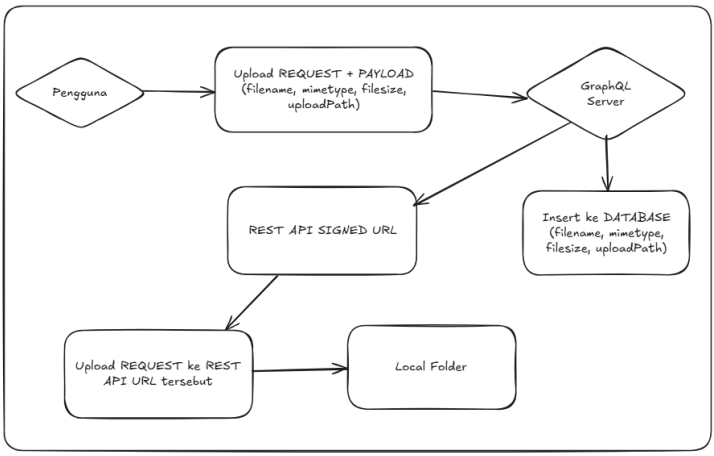
Gambar 4.1. Desain Sistem Data Loader

Gambar 4.1i menggambarkan alur eksekusi *query* menggunakan *DataLoader* untuk mengatasi masalah *N+1 query* saat melakukan relasi data di GraphQL. Proses dimulai dengan permintaan *query* yang memanggil *resolver*. *Resolver* ini memeriksa apakah ada hubungan (*relation*) yang perlu dimuat. Jika relasi tidak ada, proses langsung menghasilkan hasil *query*. Namun, jika relasi ditemukan, *resolver* menginisialisasi *DataLoader*.

DataLoader kemudian digunakan untuk melakukan *batch query* dengan metode *findIn* yang mengelompokkan permintaan berdasarkan kunci tertentu. Proses ini memanfaatkan *filter* yang ada pada *input* untuk

mengoptimalkan pengambilan data secara sekaligus. Setelah *query* dieksekusi, hasilnya diproses sesuai *input* awal dan dikembalikan ke pengguna sebagai hasil akhir *query* GraphQL. Diagram ini menunjukkan efisiensi melalui pengelompokan permintaan yang mencegah pengulangan *query* yang tidak perlu.

4.2. Arsitektur Sistem *Upload File*



Gambar 4.2. Desain Sistem Upload File

Alur kerja sistem dimulai ketika pengguna melakukan permintaan ke GraphQL *server* untuk mengunggah *file*. Gambar 4.1 menggambarkan alur kerja permintaan pengguna saat menggunakan tipe *data file*. Pengguna akan mengirimkan *request* ke GraphQL *server*, yang kemudian akan memproses permintaan tersebut dan menyimpan data *file* berupa *object* JSON dengan isi *filename*, *filesize*, *mimetype*, dan *uploadpath*. Selanjutnya GraphQl *server* akan menghasilkan *signed* URL. Proses

ini dimulai dengan GraphQL *server* memanggil *endpoint /generate-signed-url* pada *server* REST API, dengan mengirimkan parameter berupa nama *file* yang akan diunggah.

REST API *server* kemudian membuat *token* JWT yang menyimpan informasi nama *file,* dengan masa berlaku 15 menit. *Token* ini digunakan untuk membangun *signed* URL yang memungkinkan pengguna mengunggah *file* secara aman. *Signed* URL ini dikembalikan ke GraphQL *server*. GraphQL *server* akan menggunakan *signed* URL tersebut untuk mengunggah *file*.

Setelah GraphQL *server* menerima *signed* URL, *file* dapat diunggah menggunakan URL tersebut. *File* dikirimkan dalam bentuk *buffer*, disertai nama *file* dan jenis MIME-nya, menuju *endpoint* pada REST API *server*. *Endpoint* ini memverifikasi *token* JWT yang diterima untuk memastikan bahwa *file* yang diunggah sesuai dengan nama *file* yang tercantum dalam *token*. Jika validasi berhasil, REST API *server* menyimpan *file* ke sistem berkas lokal menggunakan mekanisme aliran data (*stream*). *File* disalin ke direktori yang telah ditentukan oleh sistem, dan pengguna menerima respons sukses setelah proses unggahan selesai.

[Halaman ini sengaja dikosongkan]

BAB V
IMPLEMENTASI SISTEM

Bab ini membahas tentang implementasi dari sistem yang kami buat. Implementasi ini akan dibagi ke dalam beberapa bagian, yaitu bagian implementasi *data loader*, tipe data *file*, agregasi, dan *unit testing*.

5.1. Implementasi *Data Loader*

Implementasi *Data Loader* dirancang untuk mengatasi permasalahan *N+1 query problem* yang sering terjadi dalam pengambilan data terkait dalam sistem basis data. Masalah ini muncul ketika sebuah *query* utama menghasilkan banyak *query* tambahan untuk setiap elemen hasilnya, sehingga meningkatkan jumlah permintaan ke *database* secara signifikan dan menurunkan kinerja sistem. Dalam pendekatan ini, *Data Loader* digunakan untuk mengelompokkan permintaan data yang serupa dan menggabungkannya menjadi satu permintaan besar, sehingga mengurangi jumlah *query* yang diperlukan.

Metode *getDataloader2* berfungsi untuk mengelola pengambilan data secara *batch* dengan memanfaatkan kunci unik dan opsi *filter* tambahan. Fungsi ini menggunakan struktur *dataloaderMap2* untuk menyimpan atau mengambil kembali *instance AnyDataLoader2* berdasarkan kunci yang dihasilkan dari kombinasi elemen kunci yang diberikan. *AnyDataLoader2* adalah sebuah kelas yang mengimplementasikan fungsionalitas dari *Data Loader*. *AnyDataLoader2* memungkinkan pengambilan data secara efisien dari data source dengan mengelompokkan permintaan serupa, baik dengan atau tanpa filter. Cuplikan kode *getDataloader2* dapat dilihat pada kode sumber 5.1. Adapun kode sumber lengkap *getDataloader2* dan *AnyDataLoader2* dapat dilihat pada Lampiran 1. *DataLoader* akan

mengelompokkan permintaan *query* ke dalam satu *batch* dan, jika diperlukan, menyertakan *filter* tambahan untuk mempersempit hasil. Penggunaan metode seperti *findIn* atau *findMany* memungkinkan pengambilan data dilakukan secara efisien dengan mengurangi redudansi *query*. Hasil *query* kemudian disimpan dalam struktur *map* untuk memastikan data yang diminta dapat diakses kembali dengan cepat tanpa harus melakukan *query* ulang.

Penerapan *DataLoader* ini terintegrasi dalam fungsi *resolver createConnectionToManyResolver*, yang digunakan dalam pengambilan data koneksi GraphQL. Dengan menerapkan mekanisme ini, proses pengambilan data yang sebelumnya membutuhkan banyak *query* individu kini dapat dilakukan hanya dengan satu *query* besar untuk satu *batch* data. Hal ini tidak hanya meningkatkan efisiensi sistem, tetapi juga mengurangi waktu respon aplikasi serta beban kerja pada *database*. Adapun kode sumber untuk fungsi *createConnectionToManyResolver* dapat dilihat pada Lampiran 2.

```
getDataloader2(keys: string[], filter?: FilterManyObject):
AnyDataLoader2 {
    keys.sort();
    const key = this.generateDataloaderKey(keys);

    return this.dataloaderMap2.getValueOrSetFn(
      key,
      () =>
        new AnyDataLoader2(async (values) => {
          const ds = this.forceGetDataSource();

          if (this.dataloaderMap2.contains(key)) {
            this.dataloaderMap2.remove(key);
```

```
          }

          let results: AdvancedMap<unknown>[];

          if (filter) {
            results = await ds.findIn(
              keys,
              values.map((v) => [v]),
              filter,
            );
          } else {
            results = await ds.findIn(
              keys,
              values.map((v) => [v]),
            );
          }
          console.log("results: ", results);

          const resultsMap = new Map<string,
AdvancedMap<unknown>[]>();
          // Rest of the code
  }
```

Kode Sumber 5.1. Implementasi Data Loader

5.2. Implementasi Tipe *Data File*

Pada implementasi tipe *data file* dalam sistem Homura, GraphQLUpload digunakan untuk menangani unggahan *file* dalam proses komunikasi data antara *client* dan *server*. GraphQLUpload adalah tipe *input* khusus di GraphQL yang memungkinkan pengiriman *file* sebagai bagian dari *query* atau *mutation*. Dengan menggunakan tipe ini, *file* yang diunggah dapat diterima dan diproses oleh *server* melalui GraphQL. Oleh karena itu akan ditambahkan tipe data *HFileDef* yang akan dipetakan pada

inputMappers ke *GraphQLUpload*. Kode Sumber M*apping* antara tipe data baru *file* dengan *input* dan *output* dapat dilihat pada Lampiran 3.

Pada implementasi tipe *data file* dalam sistem Homura, terdapat kelas yang mendefinisikan tipe data, yaitu *HFieldDef*. Kelas ini digunakan untuk mendefinisikan berbagai jenis *field* dalam sistem, termasuk tipe data *file*. Untuk menangani tipe *data file*, dibuatlah kelas baru bernama *HFileDef* yang meng-*extend* kelas *HFieldDef*. Kelas *HFileDef* ini bertanggung jawab untuk menangani berbagai aspek *file*, seperti *parsing file* yang diunggah, pengecekan tipe dan ukuran *file*, serta penentuan nama *file* secara acak.

Kelas *HFileDef* menggunakan *Zod* untuk validasi tipe data dan *NanoID* untuk menghasilkan nama *file* yang unik. Kelas ini juga mengimplementasikan metode *parse* yang menerima nilai berupa *buffer* atau *stream* yang kemudian diproses dan disimpan dalam sistem. Setelah *file* diunggah, informasi terkait *file* seperti nama, ukuran, dan tipe mime-nya akan disimpan dalam objek *HFile*. Objek ini nantinya dapat digunakan untuk menyimpan *data file* dalam bentuk *JSON*, dengan informasi lengkap mengenai *file* yang diunggah, termasuk *path* direktori tempat *file* disimpan. Dengan pendekatan ini, tipe *data file* dapat dikelola dengan baik dan dapat diakses secara efisien dalam sistem Homura. Kode Sumber 5.3 menggambarkan cuplikan kode pembuatan *HFileDef* yang meng-*extend* *HFieldDef*. Adapun kode sumber lengkap HFileDef dapat dilihat pada Lampiran 4.

```
@Define(`@homura/file`)
export class HFileDef extends HFieldDef<HFile> {
  public static get DEFAULT_IDENTIFIER() {
    return `File`;
  }
```

```
  private static _instance = new
HFileDef(HFileDef.DEFAULT_IDENTIFIER, {});
  static getInstance(): HFileDef {
    return this._instance;
  }
  constructor(
    public identifier: string,
    public attr: {maxSize?: number; allowedTypes?: string[]} =
{},
  ) {
    super(identifier);
  }

  override async parse(value: unknown): Promise<HFile> {

    // Rest of the code

    return new HFile({
      value: <JSON>(<unknown>{
        filename: filename,
        directorypath: directorypath + "/" + randName + "." +
fileExtension,
        filesize: filesize,
        mimetype: mimetype,
      }),
    });
  }
    // Rest of the code
}
export class HFile extends HValue<JSON> {}
```

Kode Sumber 5.2. HFileDef dan HFile sebagai Tipe data File

Dalam sistem Homura, telah terdapat kode untuk menangani operasi *CRUD (Create, Read, Update, Delete)*. Namun khusus untuk menangani tipe *data file*, diperlukan operasi *CRUD* khusus untuk melakukan pengelolaan *file* yang diunggah. *CRUD* ini sangat penting karena memungkinkan pengguna untuk menghapus atau memperbarui *file* sesuai kebutuhan. Implementasi operasi yang diperlukan untuk mengelola tipe *data file* adalah *deleteOneFile*, *deleteFileMany*, *updateOneFile*, dan *updateFileMany*. Kode sumber 5.4 menggambarkan fungsi *delete file* untuk tipe *data file.* Adapun kode sumber untuk operasi lainnya dapat dilihat pada Lampiran 5.

```
export async function deleteOneFile(ctx: CrudContext, input:
HValue<unknown>) {
  const fileFields: string[] = [];
  ctx.model.attr.modelFieldList.forEach((field: HModelField) =>
{
    const isFile = HPrimitiveField.is(field.attr.field,
HFileDef);
    if (isFile) {
      const attr = field.attr.fieldName;
      //   console.log("ini field: ", attr);
      fileFields.push(attr);
    }
  });

  const value = await ctx.ds.findOne(input.toRaw());
  value?.getValue;

  const valuedata = value?.toRecord();
  //   console.log("ini value:", valuedata);

  if (!valuedata || typeof valuedata !== "object") {
    console.error("Valuedata is undefined or not an object.");
```

```
    return;
  }

  for (const field of fileFields) {
    const fileData = valuedata[field] as {directorypath?:
string} | undefined;
    if (fileData && fileData.directorypath) {
      // Extract the relative path from the URL
      const relativePath = fileData.directorypath.replace(
        "http://localhost:3000/",
        "",
      );
      const fullPath = path.join(process.cwd(), relativePath);

      // Delete the file
      try {
        fs.unlinkSync(fullPath);
        console.log(`Deleted file: ${fullPath}`);
      } catch (error) {
        console.error(`Failed to delete file ${fullPath}:`);
      }
    }
  }
}
```

Kode Sumber 5.3. Delete One untuk tipe data file

Implementasi unggah *file* dalam sistem Homura menggunakan REST API yang memungkinkan pengguna mengirimkan *file* melalui *signed* URL. Proses ini dimulai dengan menghasilkan URL yang aman untuk unggahan *file*, yang kemudian digunakan untuk mengirimkan *file* ke *server*. *Server* akan memverifikasi URL dan *token* yang terkait sebelum menyimpan *file* di direktori lokal, memastikan keamanan dan kontrol akses yang tepat selama proses unggah. Adapun kode sumber untuk fungsi

uploadfile untuk tipe *data file* dapat dilihat pada Lampiran 6.

Implementasi *server* REST API akan digunakan untuk mengelola unggah *file* dalam sistem Homura. *Server* ini menyediakan dua *endpoint* utama yaitu untuk menghasilkan *signed* URL dan untuk menangani unggahan *file* melalui URL tersebut. Adapun Kode sumber untuk *server* REST API dapat dilihat pada Lampiran 7.

5.3. Implementasi Agregasi

Dalam sub bab ini, dijelaskan implementasi fitur agregasi pada kerangka kerja Homura yang mencakup fungsi *average*, *sum*, *max*, *min*, dan *count*. Fitur ini memungkinkan pengembang untuk melakukan operasi agregasi langsung melalui API GraphQL tanpa perlu menulis *query* SQL secara manual. Penambahan fitur ini dilakukan dengan memodifikasi beberapa bagian inti dari kerangka kerja, dimulai dari *file* postgres.ts, yang berfungsi sebagai lapisan koneksi utama dengan basis data PostgreSQL, hingga *file* crud.ts, yang bertanggung jawab untuk mendefinisikan logika CRUD (*Create, Read, Update, Delete*) dalam kerangka kerja.

Pada *file* postgres.ts, penulis merancang suatu fungsi yang bernama *aggregateColumns* dan *countRow*, fungsi ini bertujuan untuk melakukan operasi agregasi, seperti jumlah baris (*count*), rata-rata (*avg*), jumlah (*sum*), nilai maksimum (*max*), atau nilai minimum (*min*), pada kolom numerik dalam sebuah tabel *database*. Kode Sumber 5.4 adalah kutipan fungsi untuk menggambarkan cara kerja fungsi ini.

```
const query = buildFilter(this.table(), filter);
const columnResults: Record<string, string | number> = {};
```

```
    for (const [column, info] of Object.entries(tableInfo)) {
      if (["integer", "float", "decimal",
"real"].includes(info.type)) {
        switch (method) {
          case "avg": {
            await query.avg(`${column} as ${column}`);
            break;
          }
          case "sum": {
            await query.sum(`${column} as ${column}`);
            break;
          }
          case "max": {
            await query.max(`${column} as ${column}`);
            break;
          }
          case "min": {
            await query.min(`${column} as ${column}`);
            break;
          }
          default: {
            throw new Error("invalid method");
          }
        }
      } else {
        columnResults[`${column}`] = "not a numeric column";
      }
    }
```

Kode Sumber 5.4. Kutipan Fungsi aggregateColumns

Setelah operasi agregasi untuk semua kolom relevan ditentukan dan *query* dijalankan, hasil *query* disimpan dalam variabel yang kemudian diproses untuk menghasilkan hasil akhir dalam bentuk *map* (*finalRes*).

Untuk setiap kolom numerik, nilai hasil agregasi ditambahkan ke dalam *map*, dengan penyesuaian seperti pembulatan nilai untuk tipe *integer*. Sementara itu, kolom non-numerik diberi nilai *default* seperti "*not a numeric column*" atau *false* untuk tipe *boolean*. Implementasi fungsi ini secara lengkap bisa dilihat pada Lampiran 8.

Setelah itu, implementasi dilanjutkan di *file* crud.ts, di mana fungsi-fungsi agregasi diterapkan untuk mendukung permintaan GraphQL. Kode Sumber 5.8 adalah kutipan kode dari *file* crud.ts yakni merupakan fungsi *gen_count*, fungsi ini menerima konteks CRUD (*CrudContext*) sebagai parameter, yang berisi informasi tentang model dan sumber data terkait. Di dalamnya, fungsi membuat dan mengembalikan sebuah *instance Action*. Objek *Action* ini mendeskripsikan seluruh proses aksi, mulai dari tipe aksi yang dilakukan (dalam hal ini adalah "*read*"), nama aksi (*actionName*) yang dihasilkan berdasarkan nama model dalam konteks CRUD, hingga pemanggilan fungsi agregasi sebelumnya pada atribut *resolve*. Untuk fungsi agregasi lainnya, implementasi yang dilakukan kurang lebih sama, yang membuatnya berbeda hanyalah pemanggilan fungsi di lapisan *database*-nya, misalkan pada fungsi *gen_average*, akan dipanggil *ctx.ds.aggregateColumns*(*input.toRaw(), "avg")*. Implementasi fungsi CRUD agregasi secara lengkap dapat dilihat pada Lampiran 9.

```
const gen_count = (ctx: CrudContext) => {
  return new Action({
    type: "read",
    identifier: `countRow`,
    actionName: `countRow${ctx.model.attr.modelName}`,
```

```
    input: {
      field: createFilterObjectField(ctx, true),
      pipelineKey: null,
    },
    output: {
      field: countField,
      union: new HNullField(),
      pipelineKey: pipelines.ON_RESULT,
    },
    resolve: async (parent, req, input) => {
      const value = await ctx.ds.countRow(input.toRaw());

      const output = value ? {total:
Number(value.toRecord().total)} : null;
      return output;
    },
  });
};
```

Kode Sumber 5.5. Kutipan Fungsi gen_count

5.4. Implementasi *Unit Testing*

Implementasi *unit testing* dilakukan pada fitur lain yang akan diimplementasi seperti Tipe *data file*, agregasi, dan *data loade*r untuk permasalahan *N+1*. Pada sistem ini akan digunakan *unit testing* menggunakan *snapshot unit testing*.

5.4.1. *Unit Testing* Tipe *Data File*

Implementasi *unit testing* pada tipe *data file* dilakukan dengan cara memvalidasi berbagai operasi terkait dengan pengelolaan *file* pada sebuah sistem berbasis GraphQL. Pengujian dilakukan untuk memastikan fungsi seperti pembuatan (*create*), penemuan (*read*), pembaruan (*update*), dan

penghapusan *file* (*delete)* dapat bekerja sesuai dengan yang diharapkan.

Pada bagian pertama *file*, dilakukan persiapan awal berupa inisialisasi *server* dengan skema *file* (*filesSchema*) dan pembuatan data *user* awal menggunakan operasi *mutation*.

Kode Sumber 5.6 menunjukkan pengujian pembuatan *file* memastikan bahwa sistem dapat berhasil membuat *file* dengan menggunakan operasi *createOneUserFile*. *File* yang akan diunggah disimulasikan melalui penggunaan modul *FormData*, yang memungkinkan penanganan *file* dalam permintaan HTTP. Hasil dari operasi pembuatan *file* ini kemudian diverifikasi melalui *snapshot* dan *query* GraphQL yang memeriksa keberadaan *file* yang telah dibuat.

Pengujian untuk *read* suatu *file* dimulai dengan membuat file menggunakan operasi *mutation* bernama *createOneUserFile*. *File* yang diunggah disimulasikan menggunakan modul *FormData* dengan dua *file* yang dilampirkan. Setelah *file* berhasil diunggah, dilakukan *query* menggunakan operasi *findOneUserFile* untuk mengambil informasi terkait *file* yang telah dibuat.

Pengujian untuk *update* suatu *file* dimulai dengan cara yang sama seperti pengujian sebelumnya, *file* baru disiapkan dan dikirim melalui *FormData*, tetapi kali ini menggunakan operasi *updateOneUserFile*. Setelah pembaruan selesai, hasil respons operasi dan keberadaan *file* yang telah diperbarui diverifikasi dengan *snapshot* yang dihasilkan.

Terakhir, pengujian untuk menghapus *file* dilakukan dengan cara menjalankan operasi *mutation* untuk menghapus *file* berdasarkan ID tertentu yakni *deleteOneUserFile*. Setelah operasi penghapusan,

dilakukan query untuk memastikan *file* yang dimaksud benar-benar telah dihapus dari sistem.

Detail dari pengujian *read*, *update*, dan *delete* bisa dilihat di bagian Lampiran 10.

```
it("should create file", async () => {
    const filePath = process.cwd() + "/tests/file_test/a.jpg";

    // Create FormData
    const formData = new FormData();
    formData.append(
      "operations",
      JSON.stringify({
        query: `
              mutation ($input: CreateUserFileInput!) {
                createOneUserFile(input: $input) {
                  success
                }
              }
            `,
        variables: {
          input: {
            id: "2",
            userId: "a",
            file: null,
            file2: null,
          },
        },
      }),
    );
    formData.append(
      "map",
      JSON.stringify({
        "0": ["variables.input.file"],
        "1": ["variables.input.file2"],
      }),
```

```
    );
    formData.append("0", createReadStream(filePath), "a.jpg");
    formData.append("1", createReadStream(filePath), "a.jpg");

    // Send request
    const response = await fetch("http://localhost:3000/graphql",
{
      method: "POST",
      body: formData,
    });
    const result = await response.json();

    // Assert
    expect(result).toMatchSnapshot();

    expect(
      await t.server.executeOperation({
        query: gql`
          query ($input: UserFileWhereInput!) {
            findOneUserFile(input: $input) {
              id
              userId
              file {
                filename
                filesize
                mimetype
              }
              file2 {
                filename
                filesize
                mimetype
              }
            }
          }
        `,
        variables: {
```

```
        input: {
          id: {
            _eq: "2",
          },
        },
      },
    }),
  ).toMatchSnapshot();
});
```

Kode Sumber 5.6. Kutipan testing pembuatan file

Snapshot yang dihasilkan dan digunakan untuk menyimpan *output testing file* yang diharapkan ditampilkan secara detail pada Lampiran 11.

5.4.2. *Unit Testing* Agregasi

Pengujian agregasi dilakukan dengan cara memvalidasi fungsi-fungsi agregasi dalam sistem, seperti menghitung jumlah baris (*count*), rata-rata (*average*), nilai maksimum (*max*), nilai minimum (*min*), dan total (*sum*) dari data numerik di tabel *ModelA*. Implementasi agregasi secara detail dapat dilihat pada Lampiran 12. Sebelum pengujian, data uji dimasukkan ke dalam tabel dengan berbagai nilai pada kolom numerik untuk memastikan hasil agregasi dapat diverifikasi secara akurat. Setiap pengujian menggunakan operasi GraphQL dengan *query* yang sesuai untuk memanggil fungsi agregasi dan membandingkan hasilnya dengan *snapshot* yang telah disimpan sebelumnya.

Hasil pengujian bertujuan memastikan bahwa setiap fungsi agregasi menghasilkan data yang benar sesuai dengan data *input*. Pengujian meliputi semua kolom tabel, termasuk kolom yang tidak numerik,

untuk mengonfirmasi bahwa fungsi hanya beroperasi pada kolom yang relevan. Ini juga memverifikasi integritas dan akurasi implementasi agregasi dalam sistem. Kode Sumber 5.8 menunjukkan contoh implementasi *unit testing* untuk menghitung jumlah baris (*count*) pada *ModelA.*

```
it(`should find the count row of table`, async () => {
   expect(
     await t.server.executeOperation({
       query: gql`
         query ($input: ModelAWhereInput!) {
           countRowModelA(input: $input) {
             total
           }
         }
       `,
       variables: {
         input: {},
       },
     }),
   ).toMatchSnapshot();
 });
```

Kode Sumber 5.8. Test File untuk Fitur Count Row

Snapshot yang digunakan untuk menyimpan *output testing agregasi* yang diharapkan telah ditampilkan secara detail pada Lampiran 13.

[Halaman ini sengaja dikosongkan]

BAB VI
PENGUJIAN DAN EVALUASI

Bab ini menjelaskan tahap uji coba terhadap implementasi fitur-fitur baru pada sistem Homura.

6.1. Tujuan Pengujian

Tujuan pengujian ini adalah untuk menguji sejauh mana sistem dapat memenuhi beberapa kriteria, di antaranya:

1. Mengidentifikasi kemampuan sistem dalam menangani permasalahan *N+1* pada *query*.
2. Menguji kemampuan sistem dalam melakukan *unit testing*.
3. Menguji kemampuan sistem dalam menangani proses unggah *file* melalui aplikasi.
4. Menguji kemampuan sistem dalam melakukan agregasi data.

6.2. Kriteria Pengujian

Penilaian atas pencapaian tujuan pengujian didapatkan dengan memperhatikan beberapa hasil yang diharapkan berikut :

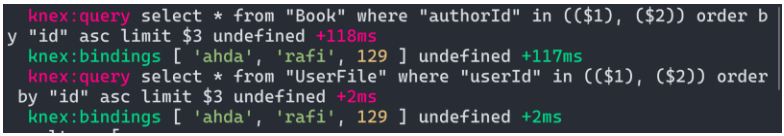
1. Kemampuan sistem untuk mengatasi permasalahan *N+1* pada *query*.
2. Kemampuan sistem untuk melakukan *unit testing*.
3. Kemampuan sistem untuk melayani *upload file* dari aplikasi.
4. Kemampuan sistem untuk melakukan agregasi.

6.3. Skenario Pengujian

Skenario pengujian dilakukan dengan melakukan peran sebagai *user* yang akan menjalankan fitur-fitur.

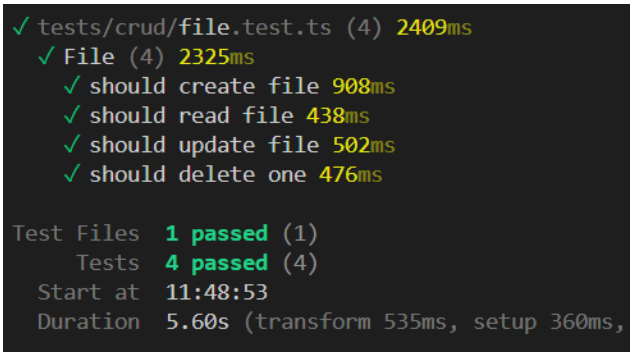
Langkah-langkah untuk setiap kebutuhan fungsionalitas yaitu sebagai berikut :

1. Pengguna bisa menggunakan fitur *query* tanpa adanya permasalahan *N+1*.
   Gambar 6.1 menunjukkan hasil pengujian yang telah dilakukan menggunakan *knex* untuk membuktikan penyelesaian *N+1 query*, *response* dari *knex* menunjukkan penggunaan *IN* ketika melakukan *query* buku dan *file* yang dimiliki oleh banyak *user*.

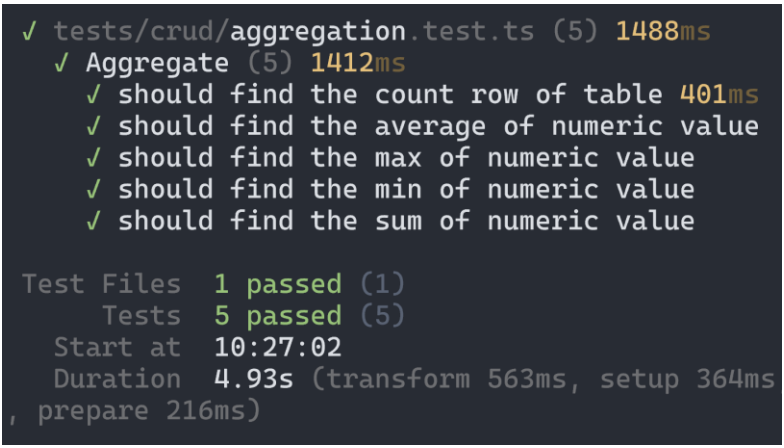


Gambar 6.1. Hasil Debug Knex untuk N+1 Problem

2. Pengguna bisa menggunakan fitur *unit testing* pada sistem.
   Gambar 6.2 dan Gambar 6.3 adalah hasil pengujian yang telah dilakukan menggunakan *vitest* berdasarkan *test files* yang telah dibuat, yakni testing untuk CRUD *file* dan agregasi (*count*, *average*, *max*, *min*, dan *sum*).



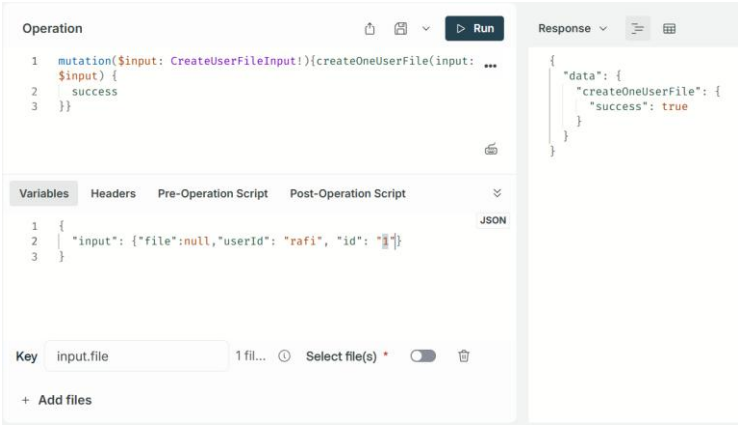
Gambar 6.2. Hasil Unit Testing tipe data File



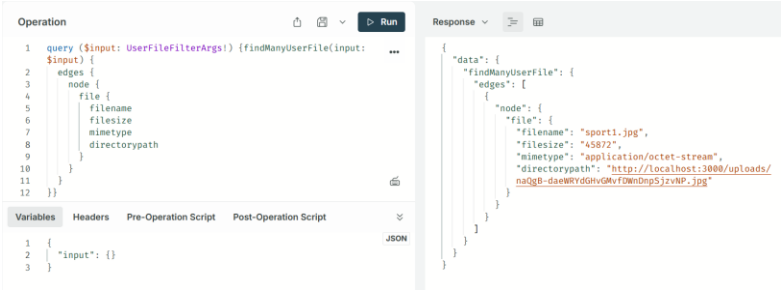
Gambar 6.3. Hasil Unit Testing Fitur Agregasi

3. Pengguna bisa menggunakan fitur tipe *data file* pada implementasi API.

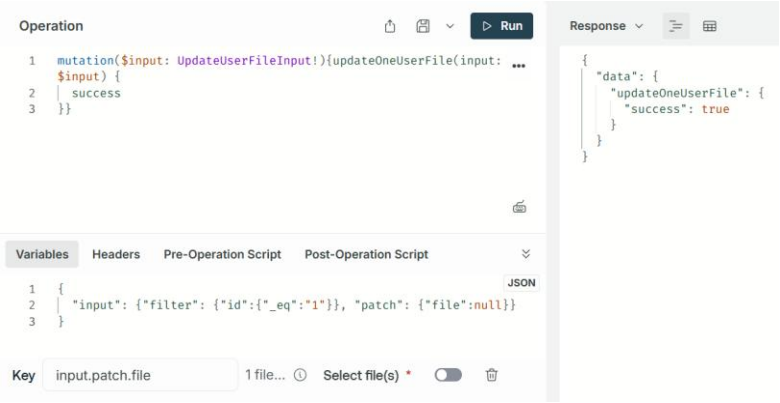
   Gambar 6.4 menggambarkan hasil percobaan *Create* menggunakan tipe *data file*. Gambar 6.5 menggambarkan hasil percobaan *Read* menggunakan tipe *data file*. Gambar 6.6 menggambarkan hasil percobaan *Update* menggunakan tipe *data file*. Gambar 6.7 menggambarkan hasil percobaan *Delete* menggunakan tipe *data file*.



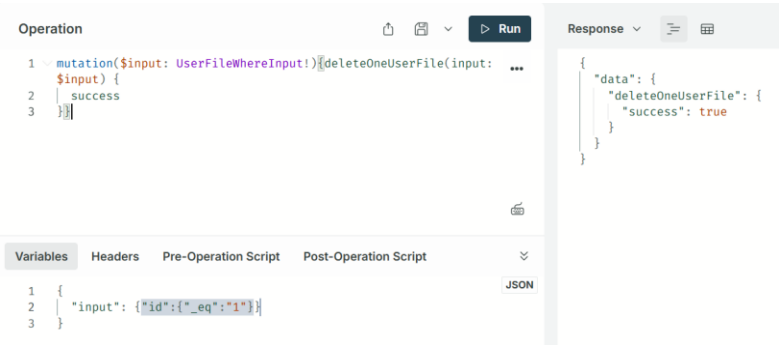
Gambar 6.4. Create Tipe data file



Gambar 6.5. Read Tipe data file



Gambar 6.6. Update Tipe data file



Gambar 6.7. Delete Tipe data file

4. Pengguna bisa menggunakan fitur agregasi.

   Gambar 6.8 menggambarkan contoh penggunaan *query* agregasi *average* diikuti dengan respons yang didapatkan, dapat dilihat bahwa jika *user* meminta rata-rata id yang bukan *numeric*, *output* yang diterima adalah "*not a numeric column*". Kemudian, Gambar 6.9 merupakan contoh penggunaan *query* agregasi *count*, Gambar 6.10 merupakan contoh penggunaan *query* agregasi *sum*, Gambar 6.11 merupakan contoh

penggunaan *query* agregasi *min*, Gambar 6.12 merupakan contoh penggunaan *query* agregasi *max*.

```
query ($input: BookWhereManyInput!) {
  averageFromBook(input: $input) {
    id
    price
  }
}
```

```
"data": {
  "averageFromBook": {
    "id": "not a numeric column",
    "price": 30000
  }
}
```

Gambar 6.8. Contoh Penggunaan Fitur Average

```
query ($input: BookWhereInput!) {
  countRowBook(input: $input) {
    total
  }
}
```

```
"data": {
  "countRowBook": {
    "total": 2
  }
}
```

Gambar 6.9. Contoh Penggunaan Fitur Count

```
query ($input: BookWhereManyInput!) {
  sumFromBook(input: $input) {
    price
  }
}
```

```
"data": {
  "sumFromBook": {
    "price": 60000
  }
}
```

Gambar 6.10. Contoh Penggunaan Fitur Sum

```
query ($input: BookWhereManyInput!) {
minFromBook(input: $input) {
  id
  price
}
}
```

```
"data": {
  "minFromBook": {
    "id": "not a numeric column",
    "price": 10000
  }
}
```

Gambar 6.11. Contoh Penggunaan Fitur Min

Gambar 6.12. Contoh Penggunaan Fitur Max

6.4. Evaluasi Pengujian

Hasil pengujian dilakukan terhadap pengamatan mengenai perilaku sistem Homura terhadap kasus skenario uji coba. Tabel 6.1 di bawah ini menjelaskan hasil uji coba terhadap sistem yang telah dibuat.

Tabel 6.1. Hasil Evaluasi Pengujian

Kriteria Pengujian	**Hasil Pengujian**
sistem dapat mengatasi permasalahan *N+1* pada *query*.	Terpenuhi
sistem memiliki *unit testing* yang sesuai dengan fitur yang ada.	Terpenuhi
Sistem mengimplementasikan tipe data *file*.	Terpenuhi
Sistem mengimplementasikan fitur agregasi.	Terpenuhi

[Halaman ini sengaja dikosongkan]

BAB VII
KESIMPULAN DAN SARAN

7.1. Kesimpulan

Kesimpulan yang didapat setelah melakukan implementasi fitur pada sistem aplikasi Homura adalah sebagai berikut :

1. Sistem Homura berhasil mengatasi permasalahan *N+1* pada *query*, meningkatkan efisiensi dan performa sistem.
2. Implementasi *unit testing* berjalan sesuai dengan fitur yang ada.
3. Fitur pengelolaan tipe *data file* telah berhasil diimplementasikan untuk mendukung kebutuhan sistem.
4. Fungsi agregasi berhasil diterapkan, memungkinkan sistem untuk mengelola dan menganalisis data secara lebih efektif.

7.2. Saran

Saran untuk implementasi fitur pada sistem kerangka kerja Homura adalah sebagai berikut :

1. Pada implementasi fitur tipe data *file*, sebaiknya ditambahkan *resource server* untuk mengatur penyimpanan *file* alternatif pada *storage* yang terpisah dari *repository* sistem, misalnya seperti *cloud storage*.
2. Pada implementasi fitur agregasi sebaiknya metode *query* dilakukan dengan cara menuliskan nama tabel diikuti dengan metode agregasinya.

[Halaman ini sengaja dikosongkan]

DAFTAR PUSTAKA

[1] Github GraphQL API v4 2017. https://developer.github.com/v4/. (2017).

[2] Departemen Teknik Informatika, 2024. *Tentang Informatika*. [online] Available at: https://www.its.ac.id/informatika/tentang-kami/ [Accessed at 21 December 2024].

[3] Laboratorium Komputasi Berbasis Jaringan, 2024. *Laboratorium Komputasi Berbasis Jaringan*. [online] Available at: https://www.its.ac.id/informatika/id/fasilitas/laboratorium/laboratorium-komputasi-berbasis-jaringan/ [Accessed at 21 December 2024].

[4] Alzamzami, M. N., & Azizah, F. N. (2022). GraphQL-based Backend Service Development Tool for CRUD Operations, Authentication, and Authorization. Proceedings of 2022 International Conference on Data and Software Engineering, ICoDSE 2022, 77–82. https://doi.org/10.1109/ICoDSE56892.2022.9971818

[5] GraphQL Users 2017. http://graphql.org/users/. (2017).

[6] Sharma, A., Kumar, R., & Mansotra, V. (2016). Proposed Stemming Algorithm for Hindi Information Retrieval. International Journal of Innovative Research in Computer and Communication Engineering (An ISO Certified Organization), 3297(6), 11449–11455. https://doi.org/10.15680/IJIRCCE.2016

[7] N+1 Queries, 2024. https://docs.sentry.io/product/issues/issue-details/performance-issues/n-one-queries/ [Accessed at 22 December 2024].

[8] Data Loader di balik layar, 2024. https://www.apollographql.com/tutorials/dataloaders-dgs/03-data-loaders-under-the-hood [Accessed at 22 December 2024].

[Halaman ini sengaja dikosongkan]

LAMPIRAN

Lampiran 1. Implementasi Data Loader

Kode untuk *data loader*.

```
import {DataLoader} from "../lib.js";
import {AdvancedMap} from "../utils/advanced-map.js";

export class AnyDataLoader extends DataLoader<unknown[],
AdvancedMap> {}
export class AnyDataLoader2 extends DataLoader<unknown[],
AdvancedMap[]> {}
```

```
getDataloader2(keys: string[], filter?: FilterManyObject):
AnyDataLoader2 {
    keys.sort();
    const key = this.generateDataloaderKey(keys);

    return this.dataloaderMap2.getValueOrSetFn(
      key,
      () =>
        new AnyDataLoader2(async (values) => {
          const ds = this.forceGetDataSource();

          if (this.dataloaderMap2.contains(key)) {
            this.dataloaderMap2.remove(key);
          }

          let results: AdvancedMap<unknown>[];

          if (filter) {
            results = await ds.findIn(
              keys,
              values.map((v) => [v]),
              filter,
```

```
            );
          } else {
            results = await ds.findIn(
              keys,
              values.map((v) => [v]),
            );
          }
          console.log("results: ", results);

          const resultsMap = new Map<string,
AdvancedMap<unknown>[]>();

          for (const result of results) {
            const keyValue = keys
              .map((key) => result.getValueOrThrow(key))
              .join("-");

            if (!resultsMap.has(keyValue)) {
              resultsMap.set(keyValue, [result]);
            } else {
              resultsMap.get(keyValue)?.push(result);
            }
          }
          console.log(resultsMap);

          return values.map((value) => {
            const keyValue = keys.map((key, index) =>
value[index]).join("-");
            return resultsMap.get(keyValue) || [];
          });
        }),
    );
  }
```

Lampiran 2. Penggunaan Data Loader pada Relay.ts

Kode untuk penggunaan *data loader*.

```
export const createConnectionToManyResolver = (
  ctx: ConnectionBuildContext,
  {connectionField, targetModel}: ConnectionResolverContext,
): ActionResolver => {
  const targetDs = targetModel.forceGetDataSource();
  const resolver: ActionResolver = async (
    parent,
    _ctx,
    _input,
  ): Promise<ConnectionResult | Record<string, unknown> | null>
=> {
    // input query (e.g. where, limit, orderby, first, after,
last, before)
    let input = filterManyInputValidator.parse(
      getInstance(_input, HObject)?.toRaw(),
    );

    //? Hard-coding limit to HARD_MAX_LIMIT
    let limit = input.first ?? HARD_MAX_LIMIT;
    let cursorDirection = 0;
    let orderBy: AdvancedMap<number>;
    if (input.orderBy) {
      orderBy = AdvancedMap.fromRecord(input.orderBy);
    } else {
      orderBy = AdvancedMap.fromEntries(
        targetModel.getPrimaryKeys().map((key) => [key, 1]),
      );
      input.orderBy = orderBy.toRecord();
    }

    let connectionWhere = null;
    if (parent && connectionField) {
```

```
      connectionWhere = {
        keys: connectionField.attr.targetFields, // misal di
query manyUser, saat masuk query book on each user, keys
bernilai authorId
        filter: {
          _eq: connectionField.attr.fields.map((v) =>
            parent.getValueOrThrow(v),
          ), // misal di skenario yang sama, nilai filter._eq
akan berisikan nama pemilik buku
        },
      };
    }
    console.log("ini cl fields",
connectionField?.attr.fields.length);

    // jika terdapat filtering after dan before
    if (input.after) {
      if (input.first !== undefined) {
        // console.log("masuk input first 10");
        limit = input.first;
      }
      const inputAfter = decodeCursor(input.after);
      console.log("ini input after: ", inputAfter);
      makeMultiReady(inputAfter);
      if (inputAfter.orderBy) {
        orderBy = AdvancedMap.fromRecord(inputAfter.orderBy);
      }
      const gtMap = AdvancedMap.fromRecord(
        inputAfter.orderBy ?? {},
      ).filterByValue((v) => v > 0);
      if (!gtMap.isEmpty()) {
        inputAfter.where!._multi.push({
          keys: gtMap.toKeys(),
          filter: {
```

```
            _gt: gtMap.toKeys().map((v) => inputAfter.value[v]),
          } satisfies FilterField,
        });
      }
      const ltMap = AdvancedMap.fromRecord(
        inputAfter.orderBy ?? {},
      ).filterByValue((v) => v < 0);
      if (!ltMap.isEmpty()) {
        inputAfter.where!._multi.push({
          keys: ltMap.toKeys(),
          filter: {
            _lt: ltMap.toKeys().map((v) => inputAfter.value[v]),
          } satisfies FilterField,
        });
      }
      input = inputAfter;
      console.log("ini final input setelah di after: ", input);
      cursorDirection = 1;
    } else if (input.before) {
      if (input.last !== undefined) {
        limit = input.last;
      }
      const inputBefore = decodeCursor(input.before);
      makeMultiReady(inputBefore);
      if (inputBefore.orderBy) {
        orderBy = AdvancedMap.fromRecord(inputBefore.orderBy);
      }
      const gtMap = AdvancedMap.fromRecord(
        inputBefore.orderBy ?? {},
      ).filterByValue((v) => v < 0);
      if (!gtMap.isEmpty()) {
        inputBefore.where!._multi.push({
          keys: gtMap.toKeys(),
          filter: {
```

```
            _gt: gtMap.toKeys().map((v) =>
inputBefore.value[v]),
          } satisfies FilterField,
        });
      }
      const ltMap = AdvancedMap.fromRecord(
        inputBefore.orderBy ?? {},
      ).filterByValue((v) => v > 0);
      if (!ltMap.isEmpty()) {
        inputBefore.where!._multi.push({
          keys: ltMap.toKeys(),
          filter: {
            _lt: ltMap.toKeys().map((v) =>
inputBefore.value[v]),
          } satisfies FilterField,
        });
      }
      input = inputBefore;
      cursorDirection = -1;
      orderBy = orderBy.map((v) => (v > 0 ? -1 : 1));
    }

    if (limit > HARD_MAX_LIMIT) {
      throw new Error(`The maximum limit is
${HARD_MAX_LIMIT}.`);
    }

    input.limit = limit + 1;
    // makeMultiReady(input);
    const where = input.where!;
    if (connectionWhere) {
      // where multi push somehow mempengaruhi output query
manyUser dan testing, tepatnya saat buildFilterMany terpanggil.
      // where._multi.push(connectionWhere);
```

```
    }

    let res: AdvancedMap<unknown>[] = [];
    const loaderKeys: string[] = connectionWhere?.keys ?? [];
    const dataLoader = targetModel.getDataloader2(loaderKeys, {
      where: input.where,
      limit: input.limit,
      orderBy: orderBy.toRecord(),
    });

    if (connectionWhere) {
      // dalam skenario manyUser, connectionWhere akan true jika
sudah memasuki query book on each user

      const keyValues = connectionWhere?.filter._eq;

      if (keyValues) {
        const rawResults = await dataLoader.load(keyValues); //
load data loader
        // console.log("Fetched raw results:", rawResults);

        // Flatten the array and filter out any errors
        const filteredResults = rawResults
          .flat()
          .filter(
            (item): item is AdvancedMap<unknown> => item
instanceof AdvancedMap,
          );

        // Assign the filtered results to res
        res = filteredResults;
      }
    } else {
      // dalam skenario yang sama, connectionWhere masih false
```

```
jika memasuki query ManyUser.
      res = await targetDs.findMany({
        where: input.where,
        limit: input.limit,
        orderBy: orderBy.toRecord(),
      });

      if (!res || res.length === 0) {
        console.warn("No results found for the provided
filter:", input.where);
        // Return an empty result set instead of null
        return {
          edges: [],
          pageInfo: {
            hasPreviousPage:
              cursorDirection > 0 ||
              (cursorDirection < 0 && res.length > limit),
            startCursor: null,
            endCursor: null,
            hasNextPage:
              cursorDirection < 0 ||
              (cursorDirection >= 0 && res.length > limit),
          },
        };
      }
    }

    if (cursorDirection < 0) {
      res.reverse();
    }
    let startSlice = 0;
    let endSlice = res.length;
    if (res.length > limit) {
      startSlice = cursorDirection >= 0 ? 0 : 1;
```

```
      endSlice = cursorDirection >= 0 ? limit : limit + 1;
    }

    const edges = res
      .slice(startSlice, endSlice)
      .map((v) => v.toRecord())
      .map((node) => ({
        node,
        cursor: encodeToCursor(input, node),
      }));

    const oRes: ConnectionResult = {
      edges,
      pageInfo: {
        hasPreviousPage:
          cursorDirection > 0 || (cursorDirection < 0 &&
res.length > limit),
        startCursor: edges.at(0)?.cursor ?? null,
        endCursor: edges.at(-1)?.cursor ?? null,
        hasNextPage:
          cursorDirection < 0 || (cursorDirection >= 0 &&
res.length > limit),
      },
    };
    return oRes;
  };

  return resolver;
};
```

Lampiran 3. Mapping GraphQLUpload untuk tipe data *file*

Kode untuk *mapping* GraphQLUpload untuk tipe data *file*.

```
const outputMappers: [Clazz<HFieldDef>, GraphQLOutputType][] = [
  [HBooleanDef, GraphQLBoolean],
  [HIntDef, GraphQLInt],
  [HFloatDef, GraphQLFloat],
  [HStringDef, GraphQLString],
  [HJSONObjectDef, GraphQLJSONObject],
  [HFileDef, GraphQLFileOutput],
];

const inputMappers: [Clazz<HFieldDef>, GraphQLInputType][] = [
  [HBooleanDef, GraphQLBoolean],
  [HIntDef, GraphQLInt],
  [HFloatDef, GraphQLFloat],
  [HStringDef, GraphQLString],
  [HFileDef, GraphQLUpload],
];
```

Lampiran 4. HFileDef dan HFile sebagai Tipe data File

Kode untuk *HFileDef* dan *HFile* sebagai tipe data *file*.

```
import {unknown, z} from "zod";
import {Define} from "../../props.js";
import {HFieldDef, HValue, ParseContext} from "../base.js";
import {Readable} from "stream";
import {NanoIDFieldProp} from "../../props/fields.js";
import {nanoid} from "nanoid/non-secure";
import {uploadFile} from "../../../utils/file/upload.js";

// directory saja yang dirandom, nama file tetap

interface UploadFile {
  filename: string;
  mimetype: string;
  encoding: string;
```

```
  createReadStream: () => Readable;
}

@Define(`@homura/file`)
export class HFileDef extends HFieldDef<HFile> {
  public static get DEFAULT_IDENTIFIER() {
    return `File`;
  }

  private static _instance = new
HFileDef(HFileDef.DEFAULT_IDENTIFIER, {});

  static getInstance(): HFileDef {
    return this._instance;
  }

  constructor(
    public identifier: string,
    public attr: {maxSize?: number; allowedTypes?: string[]} =
{},
  ) {
    super(identifier);
  }

  override async parse(value: unknown): Promise<HFile> {
    let buffer: Buffer;
    let filename = "";
    let mimetype = "";
    const directorypath = "http://localhost:3000/uploads";

    // console.log("masuk file.ts ini value:", value);

    if (Buffer.isBuffer(value)) {
      buffer = value;
```

```
    } else if (
      value &&
      typeof (value as UploadFile).createReadStream ===
"function"
    ) {
      const uploadFile = value as UploadFile;
      const stream = uploadFile.createReadStream();
      filename = uploadFile.filename;
      mimetype = uploadFile.mimetype;
      buffer = await this.streamToBuffer(stream);
    } else if (typeof value === "object" && value !== null) {
      // Handle object case here
      return new HFile({value: value as JSON});
    } else {
      throw new Error("Uploaded file must be a Buffer or a
Readable stream.");
    }

    const preBuffer = z.instanceof(Buffer).parse(buffer);
    const filesize = preBuffer.byteLength.toString();

    const randName = nanoid(32);

    const getFileExtensionFromMimetype = (mimetype: string) => {
      const parts = mimetype.split("/");
      return parts[1] ? parts[1].split("+")[0] : "bin";
    };

    let fileExtension = getFileExtensionFromMimetype(mimetype);

    if (mimetype === "application/octet-stream") {
      fileExtension = filename.split(".").pop()!.toLowerCase();
    }
```

```
    uploadFile({
      data: preBuffer,
      filename: randName + "." + fileExtension,
      mimetype: mimetype,
    });
    return new HFile({
      value: <JSON>(<unknown>{
        filename: filename,
        directorypath: directorypath + "/" + randName + "." +
fileExtension,
        filesize: filesize,
        mimetype: mimetype,
      }),
    });
  }

  // Helper function to convert a Readable stream into a Buffer
  private async streamToBuffer(stream: Readable):
Promise<Buffer> {
    return new Promise((resolve, reject) => {
      const chunks: Buffer[] = [];

      stream.on("data", (chunk) => {
        // Ensure the chunk is a Buffer, convert it if necessary
        if (Buffer.isBuffer(chunk)) {
          chunks.push(chunk);
        } else {
          chunks.push(Buffer.from(chunk)); // Convert to Buffer
if necessary
        }
      });

      stream.on("end", () => {
        // Concatenate all chunks into a single Buffer once the
```

```
stream ends
        resolve(Buffer.concat(chunks as unknown as
Uint8Array[]));
      });

      stream.on("error", (err) => {
        // Reject the promise if an error occurs while reading
the stream
        reject(err);
      });
    });
  }

  override parseFromString(value: string, ctx: ParseContext):
HFile {
    throw new Error("File cannot be parsed from a string");
  }
}

export class HFile extends HValue<JSON> {}
```

Lampiran 5. CRUD untuk tipe data file

Kode untuk operasi CRUD tipe data *file*.

```
import * as fs from "fs";
import * as path from "path";
import {HModelField} from "../../core/homura.js";
import {HFileDef} from "../../core/data-
types/primitives/file.js";
import {HPrimitiveField} from "../../core/data-
types/primitive.js";
import {HModelRefField} from "../../core/data-types/ref.js";
import {SafeDataSourceWrapper} from "../../core/data-
source/safe-wrapper.js";
```

```
import {BuildContext} from "../../core/crud/build-context.js";
import {HValue} from "../../core/data-types/base.js";

type CrudContext = BuildContext & {
  output: HModelRefField;
  ds: SafeDataSourceWrapper;
};

export async function deleteOneFile(ctx: CrudContext, input:
HValue<unknown>) {
  const fileFields: string[] = [];
  ctx.model.attr.modelFieldList.forEach((field: HModelField) =>
{
    const isFile = HPrimitiveField.is(field.attr.field,
HFileDef);
    if (isFile) {
      const attr = field.attr.fieldName;
      //   console.log("ini field: ", attr);
      fileFields.push(attr);
    }
  });

  const value = await ctx.ds.findOne(input.toRaw());
  value?.getValue;

  const valuedata = value?.toRecord();
  //   console.log("ini value:", valuedata);

  if (!valuedata || typeof valuedata !== "object") {
    console.error("Valuedata is undefined or not an object.");
    return;
  }

  for (const field of fileFields) {
```

```
    const fileData = valuedata[field] as {directorypath?:
string} | undefined;
    if (fileData && fileData.directorypath) {
      // Extract the relative path from the URL
      const relativePath = fileData.directorypath.replace(
        "http://localhost:3000/",
        "",
      );
      const fullPath = path.join(process.cwd(), relativePath);

      // Delete the file
      try {
        fs.unlinkSync(fullPath);
        console.log(`Deleted file: ${fullPath}`);
      } catch (error) {
        console.error(`Failed to delete file ${fullPath}:`);
      }
    }
  }
}

export async function deleteFileMany(ctx: CrudContext, input:
HValue<unknown>) {
  // console.log("ini input many: ", input.toRaw());
  const value1 = await ctx.ds.findMany({where: input.toRaw()});
  const valuedataQuery = value1.map((v) => v.toRecord());

  const fileFields: string[] = [];
  ctx.model.attr.modelFieldList.forEach((field: HModelField) =>
{
    const isFile = HPrimitiveField.is(field.attr.field,
HFileDef);
    if (isFile) {
      const attr = field.attr.fieldName;
```

```
      fileFields.push(attr);
    }
  });

  for (const data of valuedataQuery) {
    for (const key of fileFields) {
      const fileData = data[key] as {directorypath?: string} |
undefined;

      if (fileData && fileData.directorypath) {
        const relativePath = fileData.directorypath.replace(
          "http://localhost:3000/",
          "",
        );
        const fullPath = path.join(process.cwd(), relativePath);
        try {
          fs.unlinkSync(fullPath);
          console.log(`Deleted file: ${fullPath}`);
        } catch (error) {
          console.error(`Failed to delete file ${fullPath}:`,
error);
        }
      }
    }
  }
}

export async function updateOneFile(ctx: CrudContext, input:
HValue<unknown>) {
  const fileFields: string[] = [];
  ctx.model.attr.modelFieldList.forEach((field: HModelField) =>
{
    const isFile = HPrimitiveField.is(field.attr.field,
HFileDef);
```

```
    if (isFile) {
      const attr = field.attr.fieldName;
      fileFields.push(attr);
    }
  });
  const i = input.toRaw();
  // @ts-expect-error
  const value = await ctx.ds.findOne(i.filter);
  value?.getValue;

  const valuedata = value?.toRecord();

  const fileDef = HFileDef.getInstance();

  const fieldkeylist: string[] = [];

  // @ts-expect-error
  for (const fieldKey of Object.keys(i.patch)) {
    // @ts-expect-error
    const fieldValue = i.patch[fieldKey];

    if (fieldValue instanceof Promise) {
      try {
        // @ts-expect-error
        i.patch[fieldKey] = await fieldValue;
        fieldkeylist.push(fieldKey);
      } catch (error) {
        console.error(`Error resolving ${fieldKey}:`, error);
        continue;
      }
    }

    // @ts-expect-error
    const resolvedValue = i.patch[fieldKey];
```

```
    if (
      resolvedValue &&
      typeof resolvedValue === "object" &&
      "createReadStream" in resolvedValue
    ) {
      try {
        const parsedFile = await fileDef.parse(resolvedValue);
        // @ts-expect-error
        i.patch[fieldKey] = parsedFile.attr.value;
      } catch (error) {
        console.error(`Error processing ${fieldKey} as a file:`,
error);
      }
    } else {
      console.log(`${fieldKey} is not a file, it has value:`,
resolvedValue);
    }
  }

  deleteFileByFieldKey(valuedata, fieldkeylist);
  return i;
}

export async function updateFileMany(ctx: CrudContext, input:
HValue<unknown>) {
  const fileFields: string[] = [];
  ctx.model.attr.modelFieldList.forEach((field: HModelField) =>
{
    const isFile = HPrimitiveField.is(field.attr.field,
HFileDef);
    if (isFile) {
      const attr = field.attr.fieldName;
      fileFields.push(attr);
```

```
  }
});
const i = input.toRaw();

// @ts-expect-error
const value1 = await ctx.ds.findMany({where: i.filter});
const valuedataQuery = value1.map((v) => v.toRecord());

const fileDef = HFileDef.getInstance();

const fieldkeylist: string[] = [];

// @ts-expect-error
for (const fieldKey of Object.keys(i.patch)) {
  // @ts-expect-error
  const fieldValue = i.patch[fieldKey];

  if (fieldValue instanceof Promise) {
    try {
      // @ts-expect-error
      i.patch[fieldKey] = await fieldValue;
      fieldkeylist.push(fieldKey);
    } catch (error) {
      console.error(`Error resolving ${fieldKey}:`, error);
      continue;
    }
  }

  // @ts-expect-error
  const resolvedValue = i.patch[fieldKey];

  if (
    resolvedValue &&
    typeof resolvedValue === "object" &&
```

```
      "createReadStream" in resolvedValue
    ) {
      try {
        const parsedFile = await fileDef.parse(resolvedValue);
        // @ts-expect-error
        i.patch[fieldKey] = parsedFile.attr.value;
      } catch (error) {
        console.error(`Error processing ${fieldKey} as a file:`,
error);
      }
    } else {
      console.log(`${fieldKey} is not a file, it has value:`,
resolvedValue);
    }
  }

  deleteFileByFieldKeyMany(valuedataQuery, fieldkeylist);

  return i;
}

function deleteFileByFieldKey(
  valuedata: {[key: string]: unknown} | undefined,
  fieldKeys: string[],
) {
  if (!valuedata || typeof valuedata !== "object") {
    console.error("Valuedata is undefined or not an object.");
    return;
  }

  for (const fieldKey of fieldKeys) {
    const fileData = valuedata[fieldKey] as
      | {directorypath?: string}
      | undefined;
```

```
    if (!fileData || typeof fileData !== "object" ||
!fileData.directorypath) {
      console.error(`No valid directory path found for field
key: ${fieldKey}`);
      continue; // Skip to the next fieldKey
    }

    if (typeof fileData.directorypath !== "string") {
      console.error(
        `Directory path is not a string for field key:
${fieldKey}`,
      );
      continue;
    }

    // Extract the relative path from the directory path
    const relativePath = fileData.directorypath.replace(
      "http://localhost:3000/",
      "",
    );
    const fullPath = path.join(process.cwd(), relativePath);

    try {
      // Delete the file
      if (fs.existsSync(fullPath)) {
        fs.unlinkSync(fullPath);
        console.log(
          `Deleted file for field key "${fieldKey}":
${relativePath}`,
        );
      } else {
        console.log(
          `File does not exist for field key "${fieldKey}":
```

```
${fullPath}`,
        );
      }
    } catch (error) {
      console.error(
        `Failed to delete file for field key "${fieldKey}" at
path ${fullPath}:`,
        (error as Error).message,
      );
    }
  }
}

function deleteFileByFieldKeyMany(
  valuedataQuery: {[key: string]: unknown}[] | undefined,
  fieldKeys: string[],
) {
  if (!valuedataQuery) {
    console.log("No data to process.");
    return;
  }

  // console.log("ini valuedata", valuedataQuery);
  // console.log("ini fieldkey", fieldKeys);

  for (const entry of valuedataQuery) {
    for (const key of fieldKeys) {
      if (entry[key]) {
        const field = entry[key] as {directorypath: string};
        const relativePath = field.directorypath.replace(
          "http://localhost:3000/",
          "",
        );
        const fullPath = path.join(process.cwd(), relativePath);
```

```
        try {
          if (fs.existsSync(fullPath)) {
            fs.unlinkSync(fullPath);
            console.log(`Deleted file at path: ${fullPath}`);
          } else {
            console.log(`File does not exist at path:
${fullPath}`);
          }
        } catch (error) {
          console.error(`Error deleting file at path
${fullPath}:`, error);
        }
      } else {
        console.log(`Key "${key}" not found in entry:`, entry);
      }
    }
  }
}
```

Lampiran 6. Fungsi UploadFile untuk tipe data file

Kode untuk fungsi *uploadfile* untuk tipe data *file*.

```
interface SignedUrlResponse {
  signedUrl: string;
}
interface RetrieveFile {
  data: Buffer;
  filename: string;
  mimetype: string;
}

export async function uploadFile(input: RetrieveFile) {
  try {
```

```
    // console.log("ini action input------:", input);

    // Step 1: Generate Signed URL
    const {filename} = input;
    const signedUrlResponse = await fetch(
      `http://localhost:3000/generate-signed-
url?filename=${filename}`,
    );

    if (!signedUrlResponse.ok) {
      throw new Error("Failed to generate signed URL");
    }

    // Parse the response as JSON
    const data: unknown = await signedUrlResponse.json();

    // Type guard to check if data is a valid SignedUrlResponse
    if (typeof data === "object" && data !== null && "signedUrl"
in data) {
      const signedUrlData = data as SignedUrlResponse;

      // Step 2: Convert file data array to Uint8Array, then
Blob
      const arrayBuffer = Array.from(input.data);

      const fileData = new Uint8Array(arrayBuffer); //
input.data is the file's byte array
      const fileBuffer = new Blob([fileData], {type:
input.mimetype}); // Convert to Blob

      // Step 3: Upload the Blob
      const uploadResponse = await
fetch(signedUrlData.signedUrl, {
        method: "POST",
```

```
        headers: {
          "Content-Type": input.mimetype,
        },
        body: fileBuffer,
      });

      if (!uploadResponse.ok) {
        throw new Error("Failed to upload file");
      }

      const responseData = await uploadResponse.json();
      // console.log(responseData);
    } else {
      throw new Error("Invalid response format");
    }
  } catch (error) {
    console.error("Error during upload:", error);
  }
}
```

Lampiran 7. Server REST API untuk tipe data file

Kode untuk *server* REST API untuk tipe data *file*.

```
async serve(args: {port: number}): Promise<ApolloServer> {
    const server = this.createApolloServer();
    console.log("masuk server");

    const app = express(); // Create an Express application
    const UPLOADS_DIR = process.cwd() + "/uploads";
    // Use file upload middleware with express
    app.use(
      graphqlUploadExpress({
        maxFileSize: 10000000, // 10 MB
        maxFiles: 10,
```

```
      }),
    );
    console.log("masuk app use");

    // Middleware to parse JSON request bodies
    app.use(express.json());
    app.use(cors({origin: true}));

    // Generate signed URL endpoint
    app.get("/generate-signed-url", (req: Request, res:
Response) => {
      const filename = req.query.filename as string;

      const token = jwt.sign({filename}, process.env.JWT_KEY!, {
        expiresIn: "15m",
      });

      const signedUrl = `http://localhost:${args.port}/upload-
local?token=${token}`;

      res.json({signedUrl});
    });

    // Endpoint to handle file upload via signed URL
    app.post("/upload-local", (req: Request, res: Response) => {
      const token = req.query.token as string;
      // Verify the token
      jwt.verify(token, process.env.JWT_KEY!, async (err,
decoded) => {
        if (err) return res.status(401).json({message:
"Unauthorized"});

        // Type assertion for decoded as JwtPayload
        const payload = decoded as JwtPayload;
```

```
        // Check if filename exists in the payload
        const filename = payload.filename;

        if (!filename) {
          return res
            .status(400)
            .json({message: "Filename is required in the
token"});
        }

        // Create a write stream to save the uploaded file
        if (!fs.existsSync(UPLOADS_DIR)) {
          fs.mkdirSync(UPLOADS_DIR, {recursive: true});
        }

        const filepath = path.join(UPLOADS_DIR, filename);
        const fileStream = fs.createWriteStream(filepath);

        // Pipe the incoming data to the file
        req.pipe(fileStream);

        // Handle stream events
        fileStream.on("finish", () => {
          // console.log(`File uploaded successfully:
${filepath}`);
          res.json({message: "File uploaded successfully",
filepath});
        });

        fileStream.on("error", (err) => {
          console.error(`Error uploading file: ${err.message}`);
          res.status(500).json({message: "Error uploading
file"});
```

```
        });
      });
    });

    app.use("/uploads", express.static(UPLOADS_DIR));

    // Start Apollo Server
    await server.start();

    // Use Apollo Server's Express middleware to handle GraphQL
requests

    app.use("/graphql", expressMiddleware(server));

    // Start the Express server
    const httpServer = app.listen(args.port, () => {
      console.log(
        `🚀 Express server is running at
http://localhost:${args.port}`,
      );
    });

    console.log(
      `🚀 GraphQL server is ready at
http://localhost:${args.port}/graphql`,
    );

    return server;
  }
```

Lampiran 8. Implementasi Agregasi pada Lapisan Database

Kode untuk operasi *query* agregasi *count* (*countRow*) dan agregasi berdasarkan *method* (*aggregateColumns*)

```
async countRow(filter: FilterObject):
Promise<AdvancedMap<unknown> | null> {
    const res = await buildFilter(this.table(), filter).count("*
as total");

    if (!res) {
      return null;
    }

    const result = new AdvancedMap({
      total: Number(res[0].total),
    });

    return result;
  }

 async aggregateColumns(
    filter: FilterObject,
    method: string,
 ): Promise<AdvancedMap<unknown> | null> {
    const tableInfo = await  this.table().columnInfo();

    const query = buildFilter(this.table(), filter);
    const columnResults: Record<string, string | number> = {};

    for (const [column, info] of Object.entries(tableInfo)) {
      if (["integer", "float", "decimal",
"real"].includes(info.type)) {
        switch (method) {
          case "avg": {
            await query.avg(`${column} as ${column}`);
            break;
          }
          case "sum": {
            await query.sum(`${column} as ${column}`);
            break;
```

```
          }
          case "max": {
            await query.max(`${column} as ${column}`);
            break;
          }
          case "min": {
            await query.min(`${column} as ${column}`);
            break;
          }
          default: {
            throw new Error("invalid method");
          }
        }
      } else {
        columnResults[`${column}`] = "not a numeric column";
      }
    }

    const res = await query;

    if (!res || res.length === 0) {
      return null;
    }

    const finalRes = new AdvancedMap<unknown>();
    for (const [column, info] of Object.entries(tableInfo)) {
      const resKey = `${column}`;
      if (["integer", "float", "decimal",
"real"].includes(info.type)) {
        const avgValue = Number(res[0][resKey]);
        finalRes.setValue(
          resKey,
          info.type === "integer" ? Math.round(avgValue) :
avgValue,
        );
      } else {
```

```
        finalRes.setValue(
          resKey,
          info.type === "boolean" ? false : "not a numeric
column",
        );
      }
    }

    return finalRes;
  }
```

Lampiran 9. Implementasi Agregasi pada Lapisan Crud

Kode untuk generate *action* agregasi

```
export type CountInput = {
  input: FilterObject;
};
const gen_count = (ctx: CrudContext) => {
  return new Action({
    type: "read",
    identifier: `countRow`,
    actionName: `countRow${ctx.model.attr.modelName}`,
    input: {
      field: createFilterObjectField(ctx, true),
      pipelineKey: null,
    },
    output: {
      field: countField,
      union: new HNullField(),
      pipelineKey: pipelines.ON_RESULT,
    },
    resolve: async (parent, req, input) => {
      const value = await ctx.ds.countRow(input.toRaw());
```

```
      const output = value ? {total:
Number(value.toRecord().total)} : null;
      return output;
    },
  });
};

export type AverageInput = {
  input: FilterObject;
};
const gen_average = (ctx: CrudContext) => {
  return new Action({
    type: "read",
    identifier: `averageFrom`,
    actionName: `averageFrom${ctx.model.attr.modelName}`,
    input: {
      field: createFilterObjectField(ctx, false),
      pipelineKey: null,
    },
    output: {
      field: ctx.output,
      union: new HNullField(),
      pipelineKey: pipelines.ON_RESULT,
    },
    resolve: async (parent, req, input) => {
      const value = await ctx.ds.aggregateColumns(input.toRaw(),
"avg");

      return value?.toRecord() ?? null;
    },
  });
};

export type SumInput = {
  input: FilterObject;
};
```

```
const gen_sum = (ctx: CrudContext) => {
  return new Action({
    type: "read",
    identifier: `sumFrom`,
    actionName: `sumFrom${ctx.model.attr.modelName}`,
    input: {
      field: createFilterObjectField(ctx, false),
      pipelineKey: null,
    },
    output: {
      field: ctx.output,
      union: new HNullField(),
      pipelineKey: pipelines.ON_RESULT,
    },
    resolve: async (parent, req, input) => {
      const value = await ctx.ds.aggregateColumns(input.toRaw(),
"sum");
      return value?.toRecord() ?? null;
    },
  });
};

export type MaxInput = {
  input: FilterObject;
};
const gen_max = (ctx: CrudContext) => {
  return new Action({
    type: "read",
    identifier: `maxFrom`,
    actionName: `maxFrom${ctx.model.attr.modelName}`,
    input: {
      field: createFilterObjectField(ctx, false),
      pipelineKey: null,
    },
    output: {
      field: ctx.output,
```

```
      union: new HNullField(),
      pipelineKey: pipelines.ON_RESULT,
    },
    resolve: async (parent, req, input) => {
      const value = await ctx.ds.aggregateColumns(input.toRaw(),
"max");
      return value?.toRecord() ?? null;
    },
  });
};

export type MinInput = {
  input: FilterObject;
};
const gen_min = (ctx: CrudContext) => {
  return new Action({
    type: "read",
    identifier: `minFrom`,
    actionName: `minFrom${ctx.model.attr.modelName}`,
    input: {
      field: createFilterObjectField(ctx, false),
      pipelineKey: null,
    },
    output: {
      field: ctx.output,
      union: new HNullField(),
      pipelineKey: pipelines.ON_RESULT,
    },
    resolve: async (parent, req, input) => {
      const value = await ctx.ds.aggregateColumns(input.toRaw(),
"min");
      return value?.toRecord() ?? null;
    },
  });
};
```

Lampiran 10. Implementasi Unit Testing untuk tipe data file

Kode untuk *generate unit testing* tipe data *file* menggunakan *snapshot*

```
import {createReadStream} from "fs";
import {setupServer} from "../server.js";
import {filesSchema} from "./schema.js";
import FormData from "form-data";
import fetch from "node-fetch";

describe(`File`, () => {
  beforeEach(async () => {
    await setupServer(filesSchema());
    await t.server.executeOperation({
      query: gql`
        mutation ($input: CreateUserInput!) {
          createOneUser(input: $input) {
            success
          }
        }
      `,
      variables: {
        input: {
          username: "a",
        },
      },
    });
    const filePath = process.cwd() + "/tests/file_test/a.jpg";

    // Create FormData
    const formData = new FormData();
    formData.append(
      "operations",
```

```
      JSON.stringify({
        query: `
              mutation ($input: CreateUserFileInput!) {
                createOneUserFile(input: $input) {
                  success
                }
              }
            `,
        variables: {
          input: {
            id: "1",
            userId: "a",
            file: null,
            file2: null,
          },
        },
      }),
    );
    formData.append(
      "map",
      JSON.stringify({
        "0": ["variables.input.file"],
        "1": ["variables.input.file2"],
      }),
    );
    formData.append("0", createReadStream(filePath), "a.jpg");
    formData.append("1", createReadStream(filePath), "a.jpg");

    // Send request
    const response = await
fetch("http://localhost:3000/graphql", {
      method: "POST",
      body: formData,
    });
```

```
});

it("should create file", async () => {
  const filePath = process.cwd() + "/tests/file_test/a.jpg";

  // Create FormData
  const formData = new FormData();
  formData.append(
    "operations",
    JSON.stringify({
      query: `
            mutation ($input: CreateUserFileInput!) {
              createOneUserFile(input: $input) {
                success
              }
            }
          `,
      variables: {
        input: {
          id: "2",
          userId: "a",
          file: null,
          file2: null,
        },
      },
    }),
  );
  formData.append(
    "map",
    JSON.stringify({
      "0": ["variables.input.file"],
      "1": ["variables.input.file2"],
    }),
  );
```

```
    formData.append("0", createReadStream(filePath), "a.jpg");
    formData.append("1", createReadStream(filePath), "a.jpg");

    // Send request
    const response = await
fetch("http://localhost:3000/graphql", {
      method: "POST",
      body: formData,
    });
    const result = await response.json();

    // Assert
    expect(result).toMatchSnapshot();

    expect(
      await t.server.executeOperation({
        query: gql`
          query ($input: UserFileWhereInput!) {
            findOneUserFile(input: $input) {
              id
              userId
              file {
                filename
                filesize
                mimetype
              }
              file2 {
                filename
                filesize
                mimetype
              }
            }
          }
        `,
```

```
        variables: {
          input: {
            id: {
              _eq: "2",
            },
          },
        },
      }),
    ).toMatchSnapshot();
  });

it("should read file", async () => {
    const filePath = process.cwd() + "/tests/file_test/a.jpg";

    // Create FormData
    const formData = new FormData();
    formData.append(
      "operations",
      JSON.stringify({
        query: `
              mutation ($input: CreateUserFileInput!) {
                createOneUserFile(input: $input) {
                  success
                }
              }
            `,
        variables: {
          input: {
            id: "2",
            userId: "a",
            file: null,
            file2: null,
          },
        },
```

```
      }),
    );
    formData.append(
      "map",
      JSON.stringify({
        "0": ["variables.input.file"],
        "1": ["variables.input.file2"],
      }),
    );
    formData.append("0", createReadStream(filePath), "a.jpg");
    formData.append("1", createReadStream(filePath), "a.jpg");

    // Send request
    const response = await
fetch("http://localhost:3000/graphql", {
      method: "POST",
      body: formData,
    });
    const result = await response.json();

    expect(
      await t.server.executeOperation({
        query: gql`
          query ($input: UserFileWhereInput!) {
            findOneUserFile(input: $input) {
              id
              userId
              file {
                filename
                filesize
                mimetype
              }
              file2 {
                filename
```

```
                filesize
                mimetype
              }
            }
          }
        `,
        variables: {
          input: {
            id: {
              _eq: "2",
            },
          },
        },
      }),
    ).toMatchSnapshot();
  });


  it("should update file", async () => {
    const filePath = process.cwd() + "/tests/file_test/b.pdf";

    // Create FormData
    const formData = new FormData();
    formData.append(
      "operations",
      JSON.stringify({
        query: `
              mutation($input:
UpdateUserFileInput!){updateOneUserFile(input: $input) {
                success
              }}
            `,
        variables: {
          input: {
```

```
            filter: {id: {_eq: "1"}},
            patch: {file: null},
          },
        },
      }),
    );
    formData.append(
      "map",
      JSON.stringify({
        "0": ["variables.input.patch.file"],
      }),
    );
    formData.append("0", createReadStream(filePath), "b.jpg");

    // Send request
    const response = await
fetch("http://localhost:3000/graphql", {
      method: "POST",
      body: formData,
    });
    const result = await response.json();

    // Assert
    expect(result).toMatchSnapshot();

    expect(
      await t.server.executeOperation({
        query: gql`
          query ($input: UserFileWhereInput!) {
            findOneUserFile(input: $input) {
              id
              userId
              file {
                filename
```

```
              filesize
              mimetype
            }
            file2 {
              filename
              filesize
              mimetype
            }
          }
        }
      `,
      variables: {
        input: {
          id: {
            _eq: "1",
          },
        },
      },
    }),
  ).toMatchSnapshot();
});

it(`should delete one`, async () => {
  expect(
    await t.server.executeOperation({
      query: gql`
        mutation ($input: UserFileWhereInput!) {
          deleteOneUserFile(input: $input) {
            success
          }
        }
      `,
      variables: {
        input: {
```

```
          id: {
            _eq: "1",
          },
        },
      },
    }),
  ).toMatchSnapshot();

  expect(
    await t.server.executeOperation({
      query: gql`
        query ($input: UserFileWhereInput!) {
          findOneUserFile(input: $input) {
            id
            userId
            file {
              filename
              filesize
              mimetype
            }
            file2 {
              filename
              filesize
              mimetype
            }
          }
        }
      `,
      variables: {
        input: {
          id: {
            _eq: "1",
          },
        },
```

```
        },
      }),
    ).toMatchSnapshot();
  });
});
```

Lampiran 11. File Snapshot untuk Unit Testing untuk tipe data file

Hasil *snapshot* dari *unit testing* tipe data *file*

```
// Vitest Snapshot v1, https://vitest.dev/guide/snapshot.html

exports[`File > should create file 1`] = `
{
  "data": {
    "createOneUserFile": {
      "success": true,
    },
  },
}
`;

exports[`File > should create file 2`] = `
{
  "body": {
    "kind": "single",
    "singleResult": {
      "data": {
        "findOneUserFile": {
          "file": {
            "filename": "a.jpg",
            "filesize": "10645",
            "mimetype": "image/jpeg",
```

```
          },
          "file2": {
            "filename": "a.jpg",
            "filesize": "10645",
            "mimetype": "image/jpeg",
          },
          "id": "2",
          "userId": "a",
        },
      },
      "errors": undefined,
    },
  },
  "http": {
    "headers": Map {
      "cache-control" => "no-store",
    },
    "status": undefined,
  },
}
`;

exports[`File > should delete one 1`] = `
{
  "body": {
    "kind": "single",
    "singleResult": {
      "data": {
        "deleteOneUserFile": {
          "success": true,
        },
      },
      "errors": undefined,
    },
```

```
  },
  "http": {
    "headers": Map {
      "cache-control" => "no-store",
    },
    "status": undefined,
  },
}
`;

exports[`File > should delete one 2`] = `
{
  "body": {
    "kind": "single",
    "singleResult": {
      "data": {
        "findOneUserFile": null,
      },
      "errors": undefined,
    },
  },
  "http": {
    "headers": Map {
      "cache-control" => "no-store",
    },
    "status": undefined,
  },
}
`;

exports[`File > should read file 1`] = `
{
  "body": {
    "kind": "single",
```

```
    "singleResult": {
      "data": {
        "findOneUserFile": {
          "file": {
            "filename": "a.jpg",
            "filesize": "10645",
            "mimetype": "image/jpeg",
          },
          "file2": {
            "filename": "a.jpg",
            "filesize": "10645",
            "mimetype": "image/jpeg",
          },
          "id": "2",
          "userId": "a",
        },
      },
      "errors": undefined,
    },
  },
  "http": {
    "headers": Map {
      "cache-control" => "no-store",
    },
    "status": undefined,
  },
}
`;

exports[`File > should update file 1`] = `
{
  "data": {
    "updateOneUserFile": {
      "success": true,
```

```
    },
  },
}
`;

exports[`File > should update file 2`] = `
{
  "body": {
    "kind": "single",
    "singleResult": {
      "data": {
        "findOneUserFile": {
          "file": {
            "filename": "b.jpg",
            "filesize": "42393",
            "mimetype": "application/pdf",
          },
          "file2": {
            "filename": "a.jpg",
            "filesize": "10645",
            "mimetype": "image/jpeg",
          },
          "id": "1",
          "userId": "a",
        },
      },
      "errors": undefined,
    },
  },
  "http": {
    "headers": Map {
      "cache-control" => "no-store",
    },
    "status": undefined,
```

```
  },
}
`;
```

Lampiran 12. Test File untuk Fitur Agregasi

Kode untuk *unit testing* fitur *agregasi* menggunakan *snapshot*

```
describe(`Aggregate`, () => {
  beforeEach(async () => {
    await setupServer(simpleSchema());
    await t.server.executeOperation({
      query: gql`
        mutation ($input: [CreateModelAInput!]!) {
          createManyModelA(input: $input) {
            success
          }
        }
      `,
      variables: {
        input: [
          {
            a: "string1",
            b: 100,
            c: 3.5,
            d: true,
          },
          {
            a: "string2",
            b: 250,
            c: 1.5,
            d: true,
          },
          {
            a: "string3",
```

```
          b: 143,
          c: 1.5,
          d: true,
        },
        {
          a: "string4",
          b: 306,
          c: 4.56,
          d: true,
        },
      ],
    },
  });
});

it(`should find the count row of table`, async () => {
  expect(
    await t.server.executeOperation({
      query: gql`
        query ($input: ModelAWhereInput!) {
          countRowModelA(input: $input) {
            total
          }
        }
      `,
      variables: {
        input: {},
      },
    }),
  ).toMatchSnapshot();
});

it(`should find the average of numeric value`, async () => {
  expect(
    await t.server.executeOperation({
      query: gql`
```

```
        query ($input: ModelAWhereManyInput!) {
          averageFromModelA(input: $input) {
            a
            b
            c
            d
          }
        }
      `,
      variables: {
        input: {},
      },
    }),
  ).toMatchSnapshot();
});

it(`should find the max of numeric value`, async () => {
  expect(
    await t.server.executeOperation({
      query: gql`
        query ($input: ModelAWhereManyInput!) {
          maxFromModelA(input: $input) {
            a
            b
            c
            d
          }
        }
      `,
      variables: {
        input: {},
      },
    }),
  ).toMatchSnapshot();
});
```

```
it(`should find the min of numeric value`, async () => {
  expect(
    await t.server.executeOperation({
      query: gql`
        query ($input: ModelAWhereManyInput!) {
          minFromModelA(input: $input) {
            a
            b
            c
            d
          }
        }
      `,
      variables: {
        input: {},
      },
    }),
  ).toMatchSnapshot();
});

it(`should find the sum of numeric value`, async () => {
  expect(
    await t.server.executeOperation({
      query: gql`
        query ($input: ModelAWhereManyInput!) {
          sumFromModelA(input: $input) {
            a
            b
            c
            d
          }
}
      `,
      variables: {
        input: {},
      },
```

```
      }),
    ).toMatchSnapshot();
  });
});
```

Lampiran 13. Snapshot File untuk Fitur Agregasi

Hasil *snapshot* untuk *unit test* fitur agregasi

```
// Vitest Snapshot v1, https://vitest.dev/guide/snapshot.html

exports[`Aggregate > should find the average of numeric value
1`] = `
{
  "body": {
    "kind": "single",
    "singleResult": {
      "data": {
        "averageFromModelA": {
          "a": "not a numeric column",
          "b": 200,
          "c": 2.7649999856948853,
          "d": false,
        },
      },
      "errors": undefined,
    },
  },
  "http": {
    "headers": Map {
      "cache-control" => "no-store",
    },
    "status": undefined,
  },
}
```

```
`;

exports[`Aggregate > should find the count row of table 1`] = `
{
  "body": {
    "kind": "single",
    "singleResult": {
      "data": {
        "countRowModelA": {
          "total": 4,
        },
      },
      "errors": undefined,
    },
  },
  "http": {
    "headers": Map {
      "cache-control" => "no-store",
    },
    "status": undefined,
  },
}
`;

exports[`Aggregate > should find the max of numeric value 1`] =
`
{
  "body": {
    "kind": "single",
    "singleResult": {
      "data": {
        "maxFromModelA": {
          "a": "not a numeric column",
          "b": 306,
```

```
          "c": 4.56,
          "d": false,
        },
      },
      "errors": undefined,
    },
  },
  "http": {
    "headers": Map {
      "cache-control" => "no-store",
    },
    "status": undefined,
  },
}
`;

exports[`Aggregate > should find the min of numeric value 1`] =
`
{
  "body": {
    "kind": "single",
    "singleResult": {
      "data": {
        "minFromModelA": {
          "a": "not a numeric column",
          "b": 100,
          "c": 1.5,
          "d": false,
        },
      },
      "errors": undefined,
    },
  },
  "http": {
```

```
    "headers": Map {
      "cache-control" => "no-store",
    },
    "status": undefined,
  },
}
`;

exports[`Aggregate > should find the sum of numeric value 1`] =
`
{
  "body": {
    "kind": "single",
    "singleResult": {
      "data": {
        "sumFromModelA": {
          "a": "not a numeric column",
          "b": 799,
          "c": 11.059999,
          "d": false,
        },
      },
      "errors": undefined,
    },
  },
  "http": {
    "headers": Map {
      "cache-control" => "no-store",
    },
    "status": undefined,
  },
}
`;
```

[Halaman ini sengaja dikosongkan]

BIODATA PENULIS I

Nama	: Muhammad Rafi Sutrisno
Tempat, Tanggal Lahir	: Jember, 02 Maret 2003
Jenis Kelamin	: Laki-laki
Telepon	: +6281350361135
Email	: mrafis2003@gmail.com

AKADEMIS

Kuliah	: Departemen Teknik Informatika – FTEIC , ITS
Angkatan	: 2021
Semester	: 8 (Delapan)

BIODATA PENULIS II

Nama	: Ahda Filza Ghaffaru
Tempat, Tanggal Lahir	: Medan, 23 Mei 2003
Jenis Kelamin	: Laki-laki
Telepon	: +6287888246021
Email	: afilzag@gmail.com

AKADEMIS

Kuliah	: Departemen Teknik Informatika – FTEIC , ITS
Angkatan	: 2021
Semester	: 8 (Delapan)