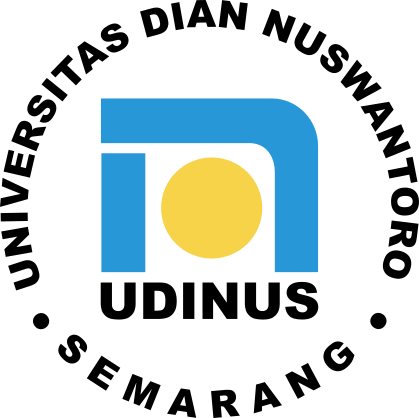
**SKRIPSI**

**Implementasi GraphQL pada *Web Service***

**Aplikasi Pelaporan Masyarakat**

Diajukan untuk memenuhi salah satu syarat memperoleh Sarjana Teknik Informatika



Disusun Oleh:

|  |  |
| --- | --- |
| Nama | : Muhammad Arji’ Sharofuddin |
| NIM | : A11.2014.08073 |
| Program Studi | : Teknik Informatika |

**FAKULTAS ILMU KOMPUTER**

**UNIVERSITAS DIAN NUSWANTORO**

**SEMARANG**

**2021**

# PERSETUJUAN SKRIPSI

Nama : Muhammad Arji' Sharofuddin

NIM : A11.2014.08073

Program Studi : Teknik Informatika

Fakultas : Ilmu Komputer

Judul Tugas Akhir : Implementasi GraphQL pada *Web Service* Aplikasi Pelaporan Masyarakat

Tugas Akhir ini telah diperiksa dan disetujui,

Semarang, …….

|  |  |
| --- | --- |
| Menyetujui:  Pembimbing  **Fahri Firdausillah S.Kom, M.CS** | Mengetahui:  Dekan Fakultas Ilmu Komputer  **Dr. Drs. Abdul Syukur MM** |

Pengesahan

# PENGESAHAN DEWAN PENGUJI

Nama : Muhammad Arji' Sharofuddin

NIM : A11.2014.08073

Program Studi : Teknik Informatika

Fakultas : Ilmu Komputer

Judul Tugas Akhir : Implementasi GraphQL pada *Web Service* Aplikasi Pelaporan Masyarakat

Tugas akhir ini telah diujikan dan dipertahankan dihadapan Dewan Penguji pada Sidang tugas akhir tanggal ….... Menurut pandangan kami, tugas akhir ini memadai dari segi kualitas maupun kuantitas untuk tujuan penganugrahan gelar Sarjana Komputer (S.Kom.)

Semarang, …….

Dewan Penguji:

|  |  |
| --- | --- |
| **Nama Dosen Penguji 1**  Anggota | **Nama Dosen Penguji 2**  Anggota |

**Nama Dosen Penguji 1**

Ketua Penguji

# PERSEMBAHAN

# KATA PENGANTAR

# DAFTAR ISI

[PERSETUJUAN SKRIPSI i](#_Toc74278561)

[PENGESAHAN DEWAN PENGUJI ii](#_Toc74278562)

[PERSEMBAHAN iii](#_Toc74278563)

[KATA PENGANTAR iv](#_Toc74278564)

[DAFTAR ISI v](#_Toc74278565)

[DAFTAR GAMBAR vii](#_Toc74278566)

[DAFTAR TABEL viii](#_Toc74278567)

[BAB I PENDAHULUAN 1](#_Toc74278568)

[1.1 Latar Belakang 1](#_Toc74278569)

[1.2 Rumusan Masalah 2](#_Toc74278570)

[1.3 Batasan Masalah 2](#_Toc74278571)

[1.4 Tujuan Penelitian 2](#_Toc74278572)

[1.5 Manfaat Penelitian 3](#_Toc74278573)

[BAB II TINJAUAN PUSTAKA DAN LANDASAN TEORI 4](#_Toc74278574)

[2.1 Tinjauan Studi 4](#_Toc74278575)

[2.2 Tinjauan Pustaka 7](#_Toc74278576)

[2.3 Kerangka Pemikiran 12](#_Toc74278577)

[BAB III METODE PENELITIAN 14](#_Toc74278578)

[3.1 Instrumen Penelitian 14](#_Toc74278579)

[3.2 Jenis dan Sumber Data 15](#_Toc74278580)

[3.3 Teknik Pengumpulan Data 15](#_Toc74278581)

[3.4 Metode Pengembangan Sistem 15](#_Toc74278582)

[3.5 Metode Pengujian 17](#_Toc74278583)

[BAB IV PERANCANGAN SISTEM DAN IMPLEMENTASI 18](#_Toc74278584)

[4.1 Pengantar 18](#_Toc74278585)

[4.2 Studi Kasus 18](#_Toc74278586)

[4.3 Perancangan Sistem 19](#_Toc74278587)

[4.4 Implementasi 34](#_Toc74278588)

[BAB V HASIL PENELITIAN DAN PAMBAHASAN 44](#_Toc74278589)

[5.1 Hasil Penelitian 44](#_Toc74278590)

[5.2 Pengujian 54](#_Toc74278591)

[BAB VI KESIMPULAN DAN SARAN 57](#_Toc74278592)

[6.1 Kesimpulan 57](#_Toc74278593)

[6.2 Saran 57](#_Toc74278594)

[DAFTAR PUSTAKA 58](#_Toc74278595)

# DAFTAR GAMBAR

[Gambar 2.1 Backend dan Frontend 8](#_Toc74278596)

[Gambar 2.2 Contoh graphQL schema 11](#_Toc74278597)

[Gambar 2.3 Contoh relasi pada schema graphQL 11](#_Toc74278598)

[Gambar 2.4 Contoh graphQL request dan response 12](#_Toc74278599)

[Gambar 3.1 Extreme Programming 16](#_Toc74278600)

[Gambar 4.1 Wireframe Halaman Utama 19](#_Toc74278601)

[Gambar 4.2 Wireframe Daftar Insiden 20](#_Toc74278602)

[Gambar 4.3 Wireframe Rincian dan Riwayat Insiden 21](#_Toc74278603)

[Gambar 4.4 Wireframe Statistik Insiden 22](#_Toc74278604)

[Gambar 4.5 Activity Diagram Pelaporan Insiden 23](#_Toc74278605)

[Gambar 4.6 ERD Basis Data 25](#_Toc74278606)

[Gambar 4.7 Skema tabel databse Aplikasi Pelaporan Masyarakat 34](#_Toc74278607)

[Gambar 5.1 GraphQL playground 46](#_Toc74278608)

[Gambar 5.2 Koreksi input tipe data graphQL playground 47](#_Toc74278609)

[Gambar 5.3 Tampilan halaman utama 50](#_Toc74278610)

[Gambar 5.4 Halaman daftar insiden pada petugas 51](#_Toc74278611)

[Gambar 5.5 Ukuran hasil kueri incidents atribut lengkap 53](#_Toc74278612)

[Gambar 5.6 Ukuran hasil kueri incidents tiga atribut 54](#_Toc74278613)

[Gambar 5.7 Potongan kode fungsi createIncident 55](#_Toc74278614)

[Gambar 5.8 Flow graph fungsi createIncident 55](#_Toc74278615)

# DAFTAR TABEL

[Tabel 4.1 Tabel users 26](#_Toc74278616)

[Tabel 4.2 Tabel users\_roles 26](#_Toc74278617)

[Tabel 4.3 Tabel organizations 27](#_Toc74278618)

[Tabel 4.4 Tabel organization\_related\_incident\_label 27](#_Toc74278619)

[Tabel 4.5 Tabel incident\_label 28](#_Toc74278620)

[Tabel 4.6 Tabel incidents 29](#_Toc74278621)

[Tabel 4.7 Tabel incident\_histories 30](#_Toc74278622)

[Tabel 4.8 Tabel user\_upvote\_incident 30](#_Toc74278623)

# BAB I PENDAHULUAN

## Latar Belakang

Dewasa ini pengembangan aplikasi web dibangun dengan komposisi dari berbagai komponen terpisah memanfaatkan teknologi web 2.0 untuk mengkombinasikan data dari *web service* yang dikembangkan secara terpisah baik secara internal maupun dari pihak lain [1]–[3]. Komposisi berbagai komponen ini berarti integrasi dengan web API dalam satu aplikasi. Pengembangan aplikasi seperti ini dinilai memiliki hambatan dalam hal heterogenitas semantic dari web API dan berkembangnya API tersebut [4].

Beberapa hal yang perlu dipertimbangkan dalam hal ini antara lain yang pertama yaitu sumber data yang berbeda kemungkinan memiliki cara pengambilan data (*fetching*) atau pemanipulasian data sendiri-sendiri. Maka integrasi ini membutuhkan cara yang terstandar dalam pengambilan data-data. Kemudian sumber data dari service memiliki berbagai model/entitas, yang ada juga yang saling berhubungan. Ini memungkinkan terjadi kesalahan dalam merepresentasikan data. Maka diperlukan mekanisme untuk memetakan model-model dari service yang sesuai dengan sistem.

Teknologi yang sudah ada seperti WADL (Web Application Description Language)[5] yang dikembangkan untuk *RESTful web service* mampu menyajikan representasi terstruktur dan semantic dari API web service. Namun dalam pengembangan aplikasi, API yang sudah mampu ter-representasikan strukturnya ini masih terdapat kekurangan dalam hal fleksibilitas ketika API berkembang semakin kompleks.

Arsitektur *web service* dalam REST berupa *endpoint-endpoint* yang masing-masing darinya mewakili sebuah model sumber data yang dapat diakses dengan HTTP request ke endpoint yang diinginkan. Pendekatan seperti ini mengakibatkan terjadinya percakapan bolak-balik antara client dengan server untuk meminta data yang di mana data tersebut dari server tersedia dalam endpoint yang berbeda, disebut juga *under fetching* [6].

Kombinasi dari semantic dan fleksibilitas, pada tahun 2016 muncul spesifikasi baru untuk web service, yaitu GraphQL. GraphQL di-*release* oleh Facebook sebagai cara baru dalam mengakses data pada aplikasi *client-server* dengan menggunakan bahasa *query* yang intuitif dan fleksibel [7]. Spesifikasi ini menjadi dasar untuk framework dan komunitas membangun *tools, library,* dan pengimplementasian dalam pengembangan web API, dimana data yang tersedia dari API disajikan dalam bentuk skema dan memberi kemudahan akses data dengan query yang memiliki sintaks mirip dengan JSON.

Dari penjelasan yang penulis paparkan di atas, maka penulis tertarik untuk melakukan penelitian mengenai “Implementasi GraphQL pada *Web Service* Aplikasi Pelaporan Masyarakat”.

## Rumusan Masalah

Berdasarkan latar belakang yang penulis sampaikan, maka dapat dirumuskan permasalahannya adalah:

1. Bagaimana implementasi GraphQL untuk membangun aplikasi dengan *backend API* dan *frontend* terpisah.
2. Apakah GraphQL dapat mengatasi *underf etching dan over fetching.*
3. Bagaimana membangun *Web Service* API yang semantic.

## Batasan Masalah

1. Metode *Web Service* penulismenggunakan GraphQL dengan hasil data berformat JSON.
2. Aplikasi yang penulis kembangkan merupakan *server-side* (beroperasi di bagian server) sebagai *web service* untuk aplikasi mobile dan web

## Tujuan Penelitian

Berdasarkan rumusan masalah yang telah dijelaskan sebelumnya, tujuan dari penelitian ini adalah

1. Membangun aplikasi dengan *backend API* dan *frontend* terpisah menggunakan GraphQL
2. Membangun *web service* dengan memanfaatkan GraphQL untuk mencegah *under fetching dan over fetching.*
3. Membangun *web service* berbasis GraphQL untuk API yang semantic

## Manfaat Penelitian

Adapun manfaat untuk pengembang adalah:

1. Mendapatkan alternatif solusi untuk cara pengambilan data ke server yang lebih fleksibel
2. Mendapatkan alternatif solusi untuk pembuatan *Web Service* API terdokumentasi

# BAB II TINJAUAN PUSTAKA DAN LANDASAN TEORI

## Tinjauan Studi

Penelitian pertama dari GraphQL adalah “An Initial Analysis of Facebook’s GraphQL Language” oleh Olaf Hartig dan Jorge Pérez. Penelitian ini bertujuan untuk memahami bahasa query dari GraphQL. Adapun GraphQL ini memiliki bentuk format query yang menyerupai JSON (Javascript Object Notation). Dari analisa penelitian ini menyatakan bahwa GraphQL memiliki konsep baru dalam mengakses data [7].

Penelitian “Efficient Data Communication Between a Web Client and a Cloud Environment” oleh Kit Gustavsson dan Erik Stenlund menjelaskan perbedaan arsitektur antara REST dan GraphQL [8]. Berdasarkan hasil penelitian ini, dalam pengembangan harus terlebih dahulu membuat model keputusan untuk mementukan teknik mana yang akan digunakan. Adapun efek ketika pengembang memutuskan untuk menggunakan GraphQL, pengembangan bergantung dengan *external dependencies* GraphQL. Sedangkan untuk REST, pengembang tidak harus bergantung dengan *external dependencies*. Di sisi lain dalam hal performa, dari penelitian ini menunjukkan GraphQL dapat mengurangi beban dari server maupun client.

Penelitian “Experiences on Migrating RESTful Web Services to GraphQL” menyajikan laporan berisi pengalaman Maximilian Vogel dan timnya dalam memigrasikan aplikasi *smart home* dari yang sebelumnya menggunakan REST API menjadi GraphQL. Hasil dari penelitian ini penyatakan GraphQL dalam konseptual memiliki keunggulan dalam struktur sumber dayanya yang berupa *object graph* dengan satu endpoint URI daripada banyak endpoint URI [9].

Penelitian selanjutnya adalah “Toward Automatic Semantic API Descriptions to Support Services Composition” oleh Marco Cremaschi dan Flavio De Paoli. Tujuannya adalah untuk memperkaya format *OpenAPI Specification* yang populer dengan anotasi semantic, dan menambahkan fungsionalitas dari anotasi semantic dan ke editor terkait (swagger editor). Pada pendahuluan penelitian ini menyatakan bahwa kebanyakan pendekatan yang dilakukan dalam mengintegrasikan Web Service mengalami masalah untuk membuat API berkomunikasi satu sama lain karena kurangnya kecocokan semantik antara data input dan output. Yang seperti ini membutuhkan keahlian khusus hanya untuk men-*deliver* *web service* yang semantic. Penelitian ini memanfaatkan *tools* Swagger Editor yang tersedia dengan *Open API Specification* untuk membuat *Table Interpretation* sehingga *value* dan properti API dapat “dipahami” oleh komputer. Tabel ini berformat relational lengkap dengan deskripsi API dengan tipe data, nama entitas dan relasi [10].

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No | Nama | Tahun | Judul | Metode | Hasil |
| 1 | Maximilian Vogel | 2018 | Experiences on Migrating RESTful Web Services to GraphQL | GraphQL | Hasil dari penelitian ini penyatakan GraphQL dalam konseptual memiliki keunggulan dalam struktur sumber dayanya yang berupa *object graph* dengan satu endpoint URI daripada banyak endpoint URI |
| 2 | Marco Cremaschi dan Flavio De Paoli | 2017 | Toward Automatic Semantic API Descriptions to Support Services Composition | REST | Penelitian ini memanfaatkan *tools* Swagger Editor yang tersedia dengan *Open API Specification* untuk membuat *Table Interpretation* sehingga *value* dan properti API dapat “dipahami” oleh komputer. Tabel ini berformat relational lengkap dengan deskripsi API dengan tipe data, nama entitas dan relasi |
| 3 | Olaf Hartig dan Jorge Pérez | 2017 | An Initial Analysis of Facebook’s GraphQL Language | GraphQL | Penelitian ini menghasilkan sebuah teknologi baru untuk mengakses data pada Web API menggunakan kueri berbentuk Graph. |
| 4 | Kit Gustavsson dan Erik Stenlund | 2016 | Efficient Data Communication Between a Web Client and a Cloud Environment | REST dan GraphQL | Pada penelitian ini menghasilkan, jika pengembang lebih memilih menggunakan GraphQL, pengembang harus bergantung pada external dependencies dengan pertimbangan performa yang lebih cepat ketimbang REST API. Tetapi jika pengembang lebih memilih menggunakan REST API pengembang bisa tidak bergantung pada external dependencies. |

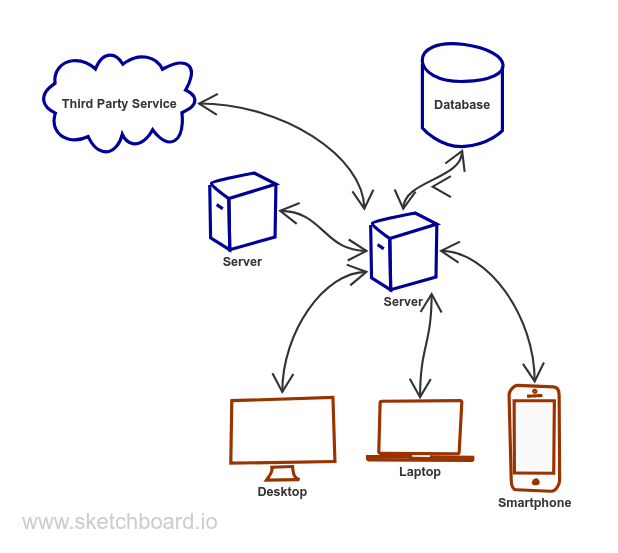
## Tinjauan Pustaka

### Aplikasi Modern

Perkembangan teknologi informasi modern ini, pembuatan sebuah aplikasi merupakan sebuah kesatuan sistem yang dibangun dengan berbagai sumber atau aplikasi yang berberda. Hal ini kemudian menjadi tuntutan dimana sebuah sistem harus bisa melakukan integrasi dengan sistem yang lain [11]. Berbagai aplikasi yang ada ini tentunya dibuat dengan teknologi yang beragam, mulai dari bahasa pemrograman hingga platform yang berbeda.

Sebagai contoh aplikasi *market place* tersedia aplikasi *mobile* untuk pengguna di dua platform platform yang berbeda yaitu android dan iOS. Ada juga aplikasi untuk mitra perusahaan *market place* tersebut. Semua aplikasi ini tentunya terhubung ke *database*, dimana untuk itu aplikasi yang berjalan di *smartphone* ini memerlukan aplikasi lain yang berjalan di sisi server.

Secara arsitektur, bagian-bagian dari sistem ini digolongkan menjadi backend dan frontend.



Gambar 2.1 Backend dan Frontend

Backend dan frontend di sini yang dimaksud adalah secara teknologi. Teknologi backend adalah segala macam teknologi yang berjalan di sisi server. Sebagai contoh untuk terhubung ke database, untuk mengirim email massal, untuk data analisis dan kebutuhan lain yang berjalan di sisi server. Teknologi backend seringnya berurusan dengan infrastruktur, seperti *scaling* database, integrasi dengan service lain, dan sebagainya. Adapun frontend adalah sebaliknya, yaitu berjalan di sisi client, dalam hal ini yang dimaksud adalah di *web* browser atau perangkat *mobile* yang digunakan oleh pengguna*.* Untuk aplikasi frontend sendiri bertanggung jawab terhadap optimasi di sisi *User Experience* [12] dan antarmuka pengguna.

Aplikasi frontend ini tidak terhubung langsung ke database karena berjalan di perangkat yang digunakan pengguna. Jadi untuk mengakses data yang dari database bisa memanfaatkan *web service* yang disediakan di aplikasi yang berjalan di server.

### Single Page Application

*Single Page Application* adalah aplikasi yang berjalan di *web browser* dan tidak memuat ulang halaman ketika digunakan. Perubahan tampilan tidak memuat halaman melainkan mengubah halaman yang sedang ditampilkan. Perubahan tampilan ini ditangani secara terprogram. Javascript merupakan salah satu yang banyak digunakan untuk membuat web Single Page Application. Cara kerja seperti ini membuat web terasa seperti aplikasi desktop. Contoh web yang Single Page Application antara lain gmail dan goole maps dari Google.

### Web Service

Web service merupakan standar pendistribusian layanan melalui internet [13]. Dalam penggunaan *web service*, client tidak perlu mengetahui tentang web service tersebut sebelum client benar-benar menggunakannya. Web service merupakan platform yang independen dan bersifat *loosely-coupled*. Karena sifatnya ini web service dapat diakses dengan mudah melalui berbagai macam platform, misalkan pada *smartphone* [14] dan service lain untuk pertukaran data antar service.

Karena *web service* ini akan digunakan untuk aplikasi saling berkomunikasi, dokumentasi untuk cara dan deskripsi *web service* ini sangatlah diperlukan.

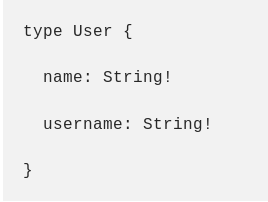
### Semantic Web Service

Dalam dokumentasi API *web service* menyajikan deskripsi, dan adanya arti dalam deskripsi-deskripsi inilah yang disebut semantic [10]. Misalnya adalah WSDL 2.0 (*Web Services Description Language*) [15]. WSDL menggunakan format XML untuk mendeskripsikan fungsionalitas *web service* secara abstrak dan juga rincian seperti bagaimana dan dimana fungsi-fungsi yang tersedia.WSDL mendukung untuk SOAP dan REST, meskipun untuk REST jarang digunakan. Kemudian ada WADL (*Web Application Description Language*)[5] yang mampu menyajikan representasi terstruktur dan semantic dari API web service berbasis REST. WADL menggunakan format XML dan juga *machine-readable* yang secara eksplisit memang ditujukan untuk layanan API. WADL juga diusulkan untuk standardisasi, namun tidak ada tindak lanjut.

Yang lebih baru, lebih *human-readable* memperkenalkan format metadata dan juga tersedia tools editor untuk memudahkan pengembang dalam membuat dokumentasi REST API. Yang populer misal OpenAPI Specification (OAS)[16] atau dikenal juga sebagai Swagger. Swagger menggunakan format berbasis YAML atau bisa juga dengan JSON. Format deskripsi ini sintaksis, yang berarti bahwa sedikit dukungan untuk bisa mengotomatisasi operasi seperti pengenalan model data, komposisi dan juga verifikasi atas model data dari *web service* yang tersedia. Sehingga untuk membuat deskripsi atau dokumentasi API membutuhkan pekerjaan manual diluar pembuatan API *web service.*

### GraphQL

Pada tahun 2016 GraphQL di-*release* oleh Facebook sebagai cara baru dalam mengakses data pada aplikasi *client-server* dengan menggunakan bahasa *query* yang intuitif dan fleksibel [7]. GraphQL menawarkan cara baru dalam membangun *web service*. *Web service* API dibuat dalam bentuk *schema* [17] yang merepresentasikan model data yang tersedia yang bisa diakses oleh client. Schema ini menyediakan deskripsi API yang lengkap dan mudah dimengerti. Berikut contoh *schema* graphQL.



Gambar 2.2 Contoh graphQL schema

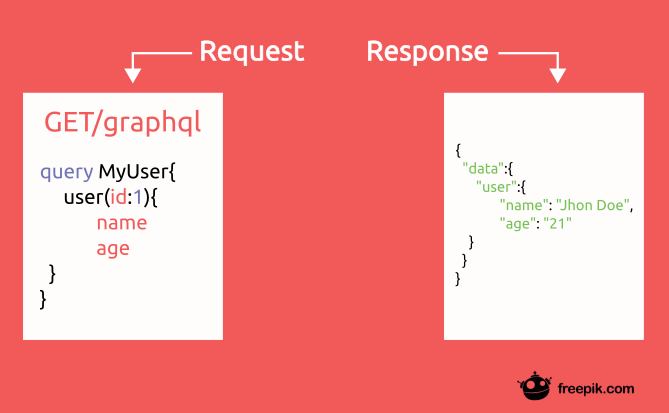
Contoh di atas merupakan *schema* User yang memiliki dua field bersifat *required* yang ditandai dengan tanda ! dan bertipe string.

GraphQL *schema* juga bisa berelasi. Contoh *schema* User di atas bisa berelasi dengan *schema* Document di bawah ini.



Gambar 2.3 Contoh relasi pada schema graphQL

Cara mengakses API menggunakan query dengan format menyerupai JSON dan akan menerima data sesuai dengan yang diminta dengan query tersebut.



Gambar 2.4 Contoh graphQL request dan response

Contoh lebih nyata ada pada Github API versi 4 dimana sebelumnya yakni pada versi 3 Github menggunakan REST untuk api versi 3-nya, kemudian memilih untuk menggunakan GraphQL untuk API versi 4 karena dinilai lebih fleksibel dan lebih *scalable*[18].

## Kerangka Pemikiran

Kerangka pemikiran dalam penelitian ini dapat digambarkan sebagai berikut.

|  |
| --- |
| Permasalahan |
| Bagaimana membangun *Web Service* yang semantic dan dapat digunakan secara efektif |

|  |
| --- |
| Pendekatan |
| Pembuatan aplikasi backend menggunakan GraphQL untuk *web service* API yang semantic |

|  |
| --- |
| Pengembangan |
| Sisi Server : API berbasis GraphQL dengan NodeJS  Sisi *Client* : web berupa *Single Page Application* dengan bahasa pemrograman javascript. |

|  |
| --- |
| Pengujian |
| Menggunakan *unit test* untuk fungsi yang berjalan mandiri dan *integration test* untuk fungsi yang berhubungan dengan aplikasi lain |

|  |
| --- |
| Hasil |
| Aplikasi backend dengan *web service* GraphQL |

# BAB III METODE PENELITIAN

## Instrumen Penelitian

Dalam melakukan penelitian yang dikerjakan ini diperlukan berbagai macam perangkat yang digunakan, yaitu:

### Kebutuhan Software

*Software* atau perangkat lunak yang digunakan untuk mendukung penelitian ini ialah:

1. Sistem Operasi yang digunakan adalah Manjaro Linux 64 bit.
2. Visual Studio Code versi 1.35 untuk menulis kode program
3. NodeJS sebagai runtime untuk menjalankan kode program berbasis javascript beserta *tools-tools* yang diperlukan.
4. Postgres sebagai penyimpanan data yang diperlukan.
5. *Nginx* sebagai mesin untuk menjalankan aplikasi.

### Kebutuhan Hardware

*Hardware* atau perangkat keras yang digunakan untuk mendukung penelitian ini ialah:

1. Komputer yang digunakan Laptop Thinkpad L440.
2. Prosesor Intel Core i5 generasi 4 vPro.
3. Kapasistas RAM 12GB.
4. Penyimpanan SSD Samsung EVO 850 250GB.

## Jenis dan Sumber Data

Penulis telah mengumpulkan beberapa jenis data sebagai acuan penelitian, data tersebut adalah data sekunder, yaitu data yang dijadikan landasan teori dan penunjang atau pelengkap data primer yang ada. Data sekunder didapatkan dari studi literatur dan dokumen penelitian terkait sebelumnya. Studi literatur digunakan untuk mencari penelitian *web service* yang menggunakan graphQL

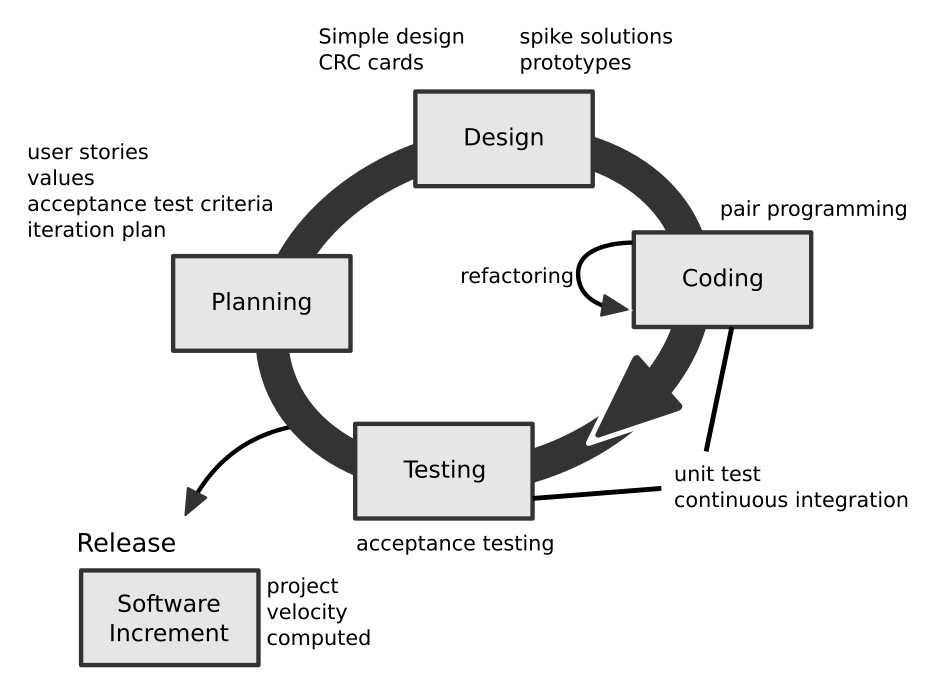
## Teknik Pengumpulan Data

Metode yang digunakan untuk mengumpulkan data dalam melakukan penelitian ini adalah studi pustaka. Studi pustaka digunakan dengan cara mencari dan memahami teori-teori yang ada pada literatur terkait dengan penelitian yang dilakukan, salah satunya adalah penelitian “Toward Automatic Semantic API Descriptions to Support Services Composition”. Penelitian ini membuat *web service* semantic dengan memanfaatkan swagger dengan API berbasis REST.

## Metode Pengembangan Sistem

Untuk mengembangkan suatu sistem dengan baik dibutuhkan sebuah metode pengembangan yang sesuai dengan kebutuhan. Berdasarkan tinjauan studi yang penulis lakukan, penulis memutuskan untuk menggunakan metode Pemrograman Ekstrim (Extreme Programming) sebagai metode pengembangan *web service* berbasis GraphQL padaAplikasi Pelaporan Masyarakat.

Extreme Programming merupakan metode pengembangan yang banyak digunakan dalam pengembangan aplikasi secara cepat, yang menyederhanakan tahapan-tahapan dalan proses pengembangan menjadi lebih adaptif. Metode ini memiliki empat kegiatan utama yaitu perencanaan, perancangan, pengkodean, dan pengujian.



Gambar 3.1 Extreme Programming

### Planning/Perencanaan

Tahap perencanaan dimulai dengan mengumpulkan berbagai kebutuhan untuk pengembangan perangkat lunak dan menentukan output yang diharapkan. Berikut kebutuhan-kebutuhannya:

1. Sistem yang dikembangkan menghasilkan *web service* API yang dapat diakses.
2. *Web Service* API dapat diakses dari beragam aplikasi *client-side* yang berbeda (web dan *mobile*).
3. Pengambilan dan pengoperasian data dari aplikasi *client* menggunakan query GraphQL.
4. Data yang diterima dari *web service* API berformat JSON

### Design/Perancangan

Tahap perancangan dilakukan untuk membuat alur perangkat lunak akan bekerja sesuai dengan kebutuhan pada tahap perencanaan. Perancangan dalam pemrograman ekstrim memiliki prinsip “penyederhanaan” atau *simplicity*. Rancangan yang sederhana akan memakan waktu pengembangan yang lebih sinkat dibanding dengan yang rumit. Pemrograman ekstrim merekomendasikan untuk membuat prototype yang operasional untuk kemudian dievaluasi. Solusi ini disebut dengan *spike.*

### Pengkodean

Pengkodean adalah penulisan kode program untuk membangun aplikasi sesuai dengan rancangan. Untuk ini penulis menggunakan bahasa pemrograman javascript dengan nodeJS sebagai runtime untuk menjalankan kode program javascript di server.

### Pengujian

Tahap pengujian ini berfungsi untuk menguji apakah kode program yang telah ditulis bisa bekerja sesuai yang diharapkan pada tahap perencanaan. Pegujian yang dilakukan adalah unit test dan *integration test*.

## Metode Pengujian

Metode pengujian yang digunakan penulis dalam penelitian ini adalah black-box testing. Black-box testing merupakan pengujian sistem berdasarkan fungsionalitas dari spesifikasi kebutuhan tanpa melakukan pengecekan kode. Black-box testing melakukan pengujian berdasarkan sudut pandang penggunaan[19]

Kemudian penulis juga menggunakan white-box testing. White-box testing merupakan pengujian yang berbasis path. Pengguaan metode barbasis path ini memungkinkan untuk menentukan tingkat kompeksitas dalam level komponen, fungsi dan logic[20]. Dalam extreme programming, white-box tesing dapat menggunakan unit test, yang bisa dilakukan menggunakan beragam *framework* yang sudah ada hingga pengujiannya dapat diotomatisasi.

# BAB IV PERANCANGAN SISTEM DAN IMPLEMENTASI

## Pengantar

Pada pembahasan kali ini penulis akan menjelaskan tentang rancangan sistem dan memaparkan hasil implementasi dari GraphQL masalah yang telah dibahas. Tujuan dari pembahasan ini adalah untuk melihat web service semantic yang dihasilkan dalam pengembangan menggunakan GraphQL.

## Studi Kasus

Aplikasi pelaporan masyarakat ini memiliki kebutuhan untuk diakses dari perangkat *smartphone* dan juga desktop untuk panel admin. Keterbatasan ukuran layar *smartphone* menjadikan tampilan antarmuka memuat rincian data yang lebih sedikit dibanding deskop sendiri. Untuk desktop dalam kasus ini digunakan petugas *call center* untuk melakukan pemantauan dan pengelolaan data sehingga untuk tampilan antarmuka akan memuat data yang lebih lengkap. *Web Service* disiini akan dimanfaatkan untuk melayani aplikasi *client side* berbasis web yang berjalan di web browser baik di platform desktop maupun *mobile*. Penggunaan REST yang berupa *endpoint-endpoint* dalam hal ini memungkinkan adanya perbedaan kebutuhan data antara *mobile* dengan desktop dalam hal rincian data. Dalam praktiknya hal ini bisa dengan menyediakan endpoint yang berbeda antara *mobile* dan desktop. Hal ini menyebabkan kompleksitas dari sisi pengembangan server melihat kebutuhan untuk entitas yang sama menjadikan *endpoint* yang berbeda dengan rincian data yang berbeda.

Dengan GraphQL, untuk hal rincian data yang berbeda seperti antara *mobile* dan desktop ini bisa tercukupi tanpa perlu membuat *endpoint* berlebih, karena cara kerja graphQL yang memungkinkan aplikasi dari sisi *client* melakukan request sendiri data yang dibutuhkan sesuai *schema* data yang tersedia dari server.

Sehingga dari sisi *mobile* yang kebutuhan data lebih sederhana tidak terjadi *overfetching* karena data yang diterima terlalu banyak melainkan bisa melakukan query sesuai dengan data yang dibutuhkan.

## Perancangan Sistem

### Wireframe

Pembuatan wireframe akan membantu seluruh stakeholder yang terlibat untuk memahami struktur sistem dan antarmuka aplikasi yang akan dikembangkan. Ini memiliki dua tujuan, yaitu menyediakan solusi dimana semua stakeholder bisa sekaligus memberi feedback, dan membantu semua pihak untuk bisa memperkirakan masalah dari segala perspektif.[21]. Supaya efisien dan memenuhi dua tujuan tersebut, wireframe bisa dibuat dengan sederhana menggunakan pensil dan kertas atau secara digital.

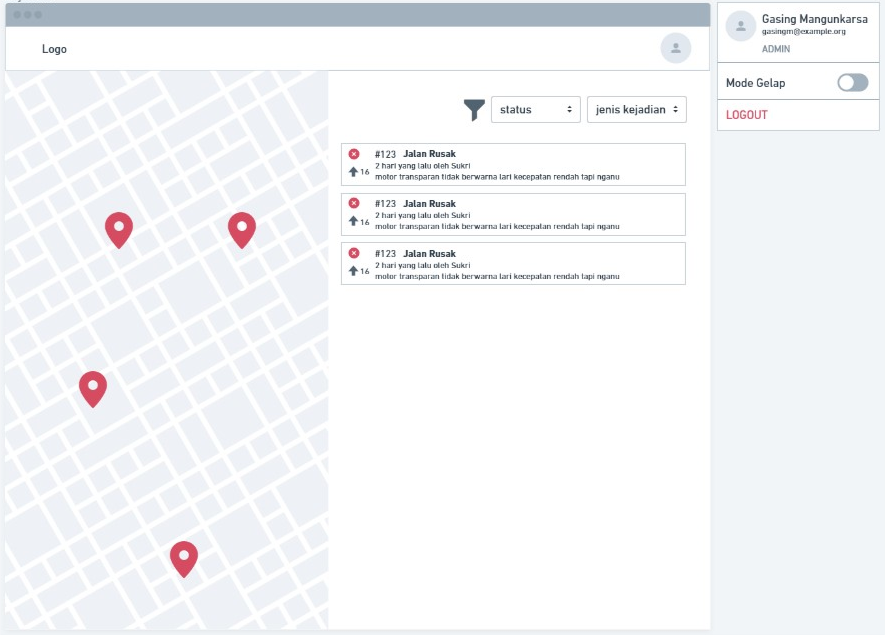
#### Halaman awal atau halaman utama



Gambar 4.1 Wireframe Halaman Utama

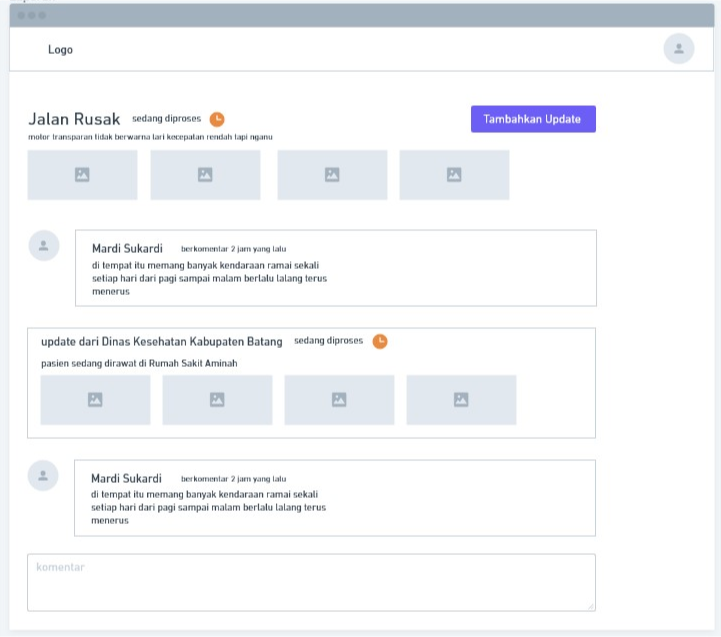
Halaman utama adalah halaman ketika web ini pertama kali diakses. Antarmukanya menampilkan label-label insiden yang bisa dipilih untuk user melaporkan insiden yang ditemui. Insiden yang dilaporkan akan dapat dilihat dan ditindaklanjuti oleh organisasi terkait sesuai dengan label insiden yang dilaporkan. Selain itu untuk menambah informasi bagi user, di sini juga menampilkan daftar insiden yang berada di dekat user. Insiden-insiden ini bisa dilihat dan dipantau oleh semua user.

#### Halaman Insiden dari sisi petugas



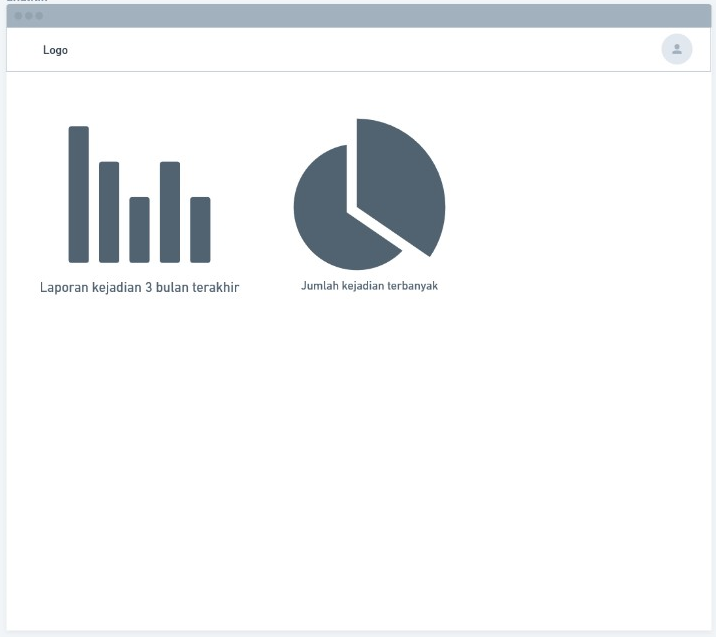
Gambar 4.2 Wireframe Daftar Insiden

Petugas yang login akan diperlihatkan halaman Daftar Insiden ini sesuai dengan organisasinya. Antarmuka menampilkan peta lokasi kejadian dengan menggunakan google map API sehingga mempermudah petugas untuk mencai tahu detil lokasi kejadian yang dilaporkan.



Gambar 4.3 Wireframe Rincian dan Riwayat Insiden

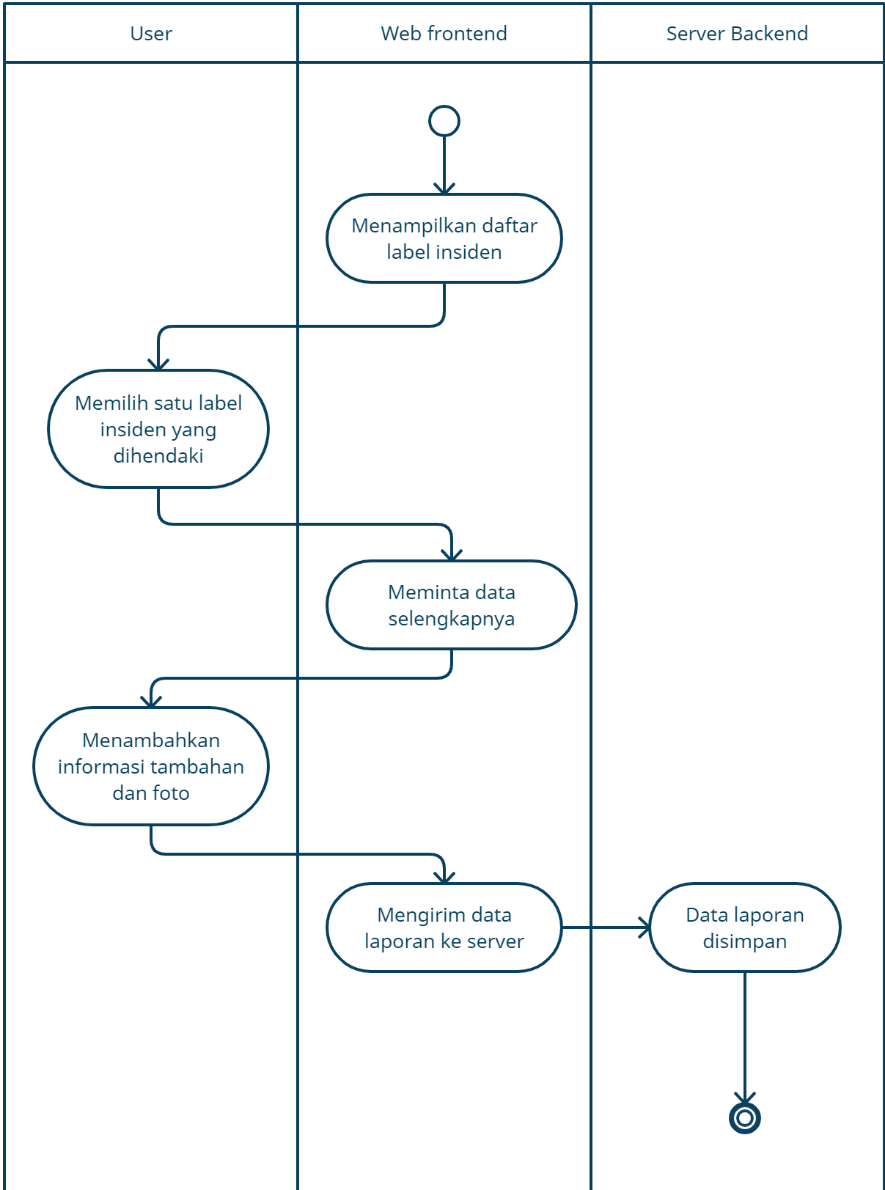
Di halaman detil insiden ini semua user termasuk petugas dapat melihat rincian insiden yang dipalorkan lengkap dengan foto dan riwayat komentar dan tindak lanjut dati petugas instansi terkait. Untuk petugas yang sedang membuka halaman ini juga bisa memberikan tindak lanjut yang telah dilakukan untuk menangani insiden yang telah dilaporkan berupa teks dan gambar.



Gambar 4.4 Wireframe Statistik Insiden

Untuk statis insiden ini menampilkan ringkasan insiden-insiden yang telah terjadi.

### Activity Diagram



Gambar 4.5 Activity Diagram Pelaporan Insiden

Activity Diagram di atas menggambarkan alur pembuatan laporan insiden oleh user. Laporan yang dibuat sesuai dengan label yang tersedia dan dilengkapi keterangan beserta foto untuk memperjelas laporan yang terjadi. Data laporan ini akan dikirimkan ke server beserta koordinat lokasi di mana user berada.

1. Menampilkan daftar insiden

Pada halaman web akan menampilkan daftar label insiden yang disediakan oleh sistem untuk pilihan sebelum user menginput data pelaporan yang akan dibuat.

1. Memilih satu label yang dikehendaki

User dapat memilih label yang ditampilkan pada halaman web dengan cara mengklik atau menekan sekali pada label yang dikehendaki.

1. Meminta data selengkapnya

Halaman web akan meminta user untuk menginputkan data lebih lengkap dengan menampilkan form dan tombol yang ketika diklik akan membuka kamera dari perangkat yang digunakan.

1. Menambahkan informasi tambahan dan foto

User mengisikan teks pada kolom yang tersedia dan mengambil gambar dari pengambil gambar yang terbuka ketika user memilih tombol untuk menambahkan gambar.

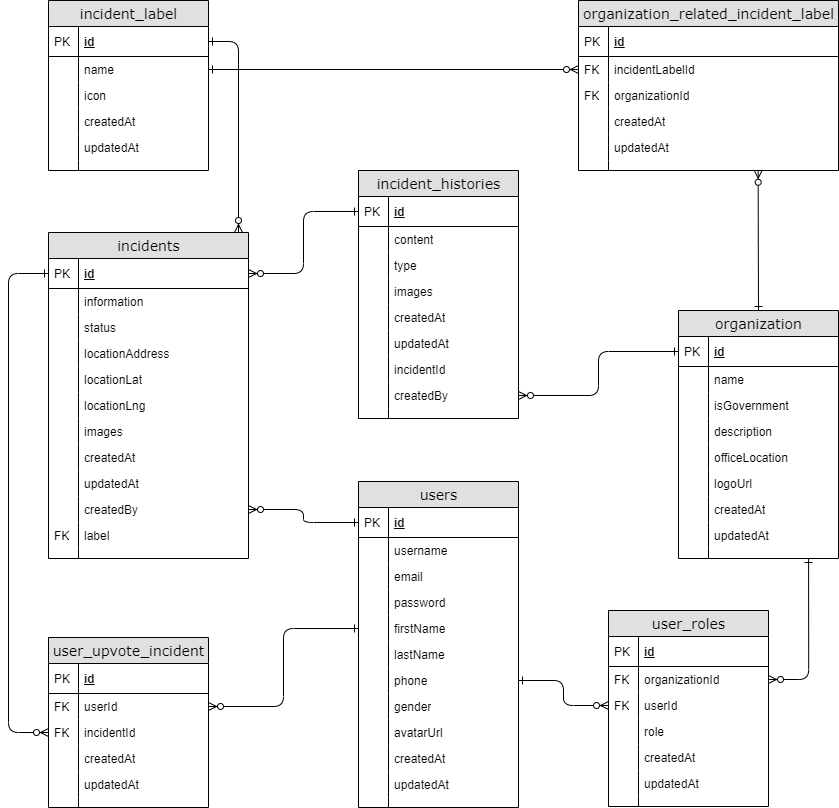
1. Mengirim data laporan ke server

Web akan memproses data yang *diinput* oleh *user*untuk kemudian dikirim ke server melalui protocol HTTP.

1. Data laporan disimpan

Data yang telah dikirim dari web browser akan diterima oleh server untuk kemudian diproses dan disimpan ke *database*.

### Perancangan Basis Data



Gambar 4.6 ERD Basis Data

Gambar di atas merupakan rancangan ERD dari aplikasi pelaporan masyarakat. Untuk mengurangi redundansi data pada system ini, penulis memisah tabel dan memberi relasi. Berikut penjelasan dari tabel-tabel dan relasinya.

#### Tabel users

|  |  |  |  |
| --- | --- | --- | --- |
| No | Nama Field | Tipe Data | Keterangan |
| 1 | id | Integer | Primary key, auto increment |
| 2 | username | varchar | username untuk autentikasi user |
| 3 | email | varchar | Email user |
| 4 | password | varchar | Password user untuk autentikasi |
| 5 | firstName | varchar | Nama depan user |
| 6 | lastName | varchar | Nama belakang user |
| 7 | phone | varchar | Nomor telepon/HP user |
| 8 | gender | varchar | Jenis kelamin user |
| 9 | avatarUrl | varchar | url gambar untuk foto profil user |
| 10 | createdAt | datetime | Tanggal user mendaftar |
| 11 | updatedAt | datetime | Tanggal terakhir data user diupdate |

Tabel 4.1 Tabel users

Tabel ini digunakan untuk menyimpan data user yang menggunakan aplikasi ini. Tabel ini memiliki beberapa relasi, pertama ke tabel user\_roles yang merupakan pivot table antara users dengan organizations. Kedua adalah relasi ke tabel incidents dimana ini digunakan untuk mengenali user sebagai pembuat(pelapor) insiden. Kemudian relasi ke tabel user\_upvote\_incident, jadi selain membuat laporan insiden, user juga bisa mem-*vote* insiden yang dibuat oleh user lain yang ditemui juga olehnya.

#### Tabel user\_roles

|  |  |  |  |
| --- | --- | --- | --- |
| No | Nama Field | Tipe Data | Keterangan |
| 1 | role | varchar | Role user dalam organisasi |
| 2 | organizationId | integer | Id organisasi |
| 3 | userId | integer | Id user |
| 4 | createdAt | datetime | Tanggal user ditambahkan role |
| 5 | updatedAt | datetime | Tanggal terakhir role diubah |

Tabel 4.2 Tabel users\_roles

Tabel ini merupakan pivot table many-to-many antara tabel users dan organizations. Field role mendefinisikan role suatu user di organisasi.

#### Tabel organizations

|  |  |  |  |
| --- | --- | --- | --- |
| No | Nama Field | Tipe Data | Keterangan |
| 1 | id | integer | Primary key, auto increment |
| 2 | name | varchar | Nama organisasi/instansi |
| 3 | isGovernment | tinyint | Penanda organisasi merupakan instansi pemerintah atau bukan |
| 4 | description | text | Deskripsi organisasi |
| 5 | officeAddress | text | Alamat kantor/basecamp organisasi |
| 6 | logoUrl | varchar | url file gambar logo organisasi |
| 7 | createdAt | datetime | Tanggal organisasi dibuat |
| 8 | updatedAt | datetime | Tanggal terakhir data organisasi diperbarui |

Tabel 4.3 Tabel organizations

Tabel organisasi merupakan tabel yang menyimpan data organisasi di aplikasi ini. Organisasi ini dapat berupa kelompok komunitas maupun instansi pemerintah. Tabel organisasi ini memiliki relasi ke tabel yang organization\_related\_incident\_label yang menghubungkan ke tabel icident\_labels sebagai label-label insiden yang akan ditindaklanjuti oleh organisasi. Dengan begitu label-label insiden yang ada akan memiliki organisai-organisasi terkait yang akan menindaklanjuti laporan isiden yang dibuat oleh user. Tabel ini juga memiliki relasi ke tabel user\_roles yang telah di jelaskan di atas sebelumnya.

#### Tabel organization\_related\_incident\_label

|  |  |  |  |
| --- | --- | --- | --- |
| No | Nama Field | Tipe Data | Keterangan |
| 1 | incidentLabelId | integer | Foreign key, id label insiden |
| 2 | organizationId | integer | Foreign key, id organisasi |
| 3 | createdAt | datetime | Tanggal organisasi ditambahkan ke label insiden |
| 4 | updatedAt | datetime | Tanggal terakhir data ini diperbarui |

Tabel 4.4 Tabel organization\_related\_incident\_label

Tabel organization\_related\_incident\_label merupakan tabel pivot yang menhubungkan relasi many-to-many antara tabel organizations dengan incident\_label. Setiap organisasi memiliki banyak label insiden dan setiap label insiden memiliki banyak organisasi.

#### Tabel incident\_label

|  |  |  |  |
| --- | --- | --- | --- |
| No | Nama Field | Tipe Data | Keterangan |
| 1 | id | integer | Primary key, auto increment |
| 2 | name | varchar | Nama label insiden |
| 3 | icon | varchar | Icon label insiden. Bisa digunakan pada antarmuka untuk menampilkan label insiden |
| 4 | createdAt | datetime | Tanggal label ini dibuat |
| 5 | updatedAt | datetime | Tanggal terakhir data label ini diperbarui |

Tabel 4.5 Tabel incident\_label

Tabel ini merupakan data label insiden yang digunakan untuk melabeli atau menglompokkan insiden yang dilaporkan oleh user, jadi ketika membuat laporan user harus memilih satu dari label yang ada. Tabel ini berrelasi ke tabel organisasi melalui tabel pivot organization\_related\_incident\_label sebagaimana dijelaskan pada bagian Tabel organizations di atas. Selain itu tabel ini juga berelasi ke tabel incidents, setiap insiden memiliki satu label insiden dan tiap label insiden memiliki banyak insiden.

#### Tabel incidents

|  |  |  |  |
| --- | --- | --- | --- |
| No | Nama Field | Tipe Data | Keterangan |
| 1 | id | integer | Primary key, auto increment |
| 2 | information | text | Teks tambahan dari user sebagai keterangan lebih lengkap insiden yang dilaporkan |
| 3 | status | varchar | Status proses tindak lanjut insiden |
| 4 | locationAddress | varchar | Keterangan alamat lokasi insiden |
| 5 | locationLat | decimal | Koordinat garis lintang lokasi |
| 6 | locationLng | decimal | Koordinat garis bujur lokasi |
| 7 | images | text | url gambar-gambar terkait insiden |
| 8 | createdAt | datetime | Tanggal laporan insiden dibuat |
| 9 | updatedAt | datetime | Tanggal terakhir data insiden diubah |
| 10 | createdBy | integer | Foregin key, id user yang membuat laporan |
| 11 | label | integer | Foreign key, id label insiden pada tabal incident\_label |

Tabel 4.6 Tabel incidents

Tabel incidents akan menampung data insiden yang dilaporkan oleh user. Tabel ini memiliki relasi ke tabel users sebagai user yang melaporkan insiden, incident\_label yang merupakan label insiden yang dipilih user sebagai label dari laporan insiden ini, dan incident\_histories yang merupakan riwayat tindak lanjut insiden yang ditangani oleh petugas organisasi.

#### Tabel incident\_histories

|  |  |  |  |
| --- | --- | --- | --- |
| No | Nama Field | Tipe Data | Keterangan |
| 1 | id | integer | Primary key, auto increment |
| 2 | content | text | Text bersi penjelasan tindak lanjut petugas organisasi yg terkait dengan insiden. Bisa juga diisi komentar dari user |
| 3 | type | varchar | Tipe riwayat, apakah berupa tindak lanjut petugas, komentar atau yang lain. |
| 4 | images | text | url gambar-gambar pendukung teks pada field content |
| 5 | createdAt | datetime | Tanggal data riwayat ini dibuat |
| 6 | updatedAt | datetime | Tanggal terakhir data riwayat ini diubah |
| 7 | incindentId | integer | Foreign key, id insiden |
| 8 | createdBy | integer | Foreign key, id user yang membuat data riwayat ini |

Tabel 4.7 Tabel incident\_histories

Tabel incident\_histories digunakan untuk menyimpan riwayat insiden, tentunya dengan relasi pada tabel insiden. Setiap insiden memiliki banyak riwayat. Data riwayat ini memiliki berbagai jenis, sesuai nilai pada field type.

#### Tabel user\_upvote\_incident

|  |  |  |  |
| --- | --- | --- | --- |
| No | Nama Field | Tipe Data | Keterangan |
| 1 | incindentId | varchar | Id insiden |
| 2 | userId | integer | Id user |
| 3 | createdAt | datetime | Tanggal user menambahkan upvote |
| 4 | updatedAt | datetime | Tanggal terakhir role diubah |

Tabel 4.8 Tabel user\_upvote\_incident

Tabel user\_upvote\_incident digunakan untuk menyimpan data vote yang dilakukan oleh user terhadap insiden yang dilaporkan oleh user lain. Tabel ini merupkan tabel pivot antara tabel users dan incidents.

### Kueri Basis Data

Dari ERD yang dibuat pada pembahasan sebelumnya, penulis dapat merancang kueri SQL yang akan diterjemahkan menjadi kueri graphQL. Berikut kueri SQL yang dirancang penulis.

#### Kueri data organisasi beserta role dari suatu user

Kueri di bawah digunakan untuk mengambil data suatu user berdasarkan id berikut organisasi dimana user tersebut di dalamnya beserta role user tersebut di tiap-tiap organisasi.

**SELECT** \* , `Organizations`.`id`, `Organizations`.`name`, `Organizations`.`isGovernment`, `Organizations`.`description`, `Organizations`.`officeAddress`, `Organizations`.`logoUrl`, `Organizations`.`createdAt`, `Organizations`.`updatedAt`, `Organizations->UserRole`.`role` **AS** `Organizations.UserRole.role`, `Organizations->UserRole`.`createdAt` **AS** `Organizations.UserRole.createdAt`, `Organizations->UserRole`.`updatedAt` **AS** `Organizations.UserRole.updatedAt`, `Organizations->UserRole`.`OrganizationId` **AS** `Organizations.UserRole.OrganizationId`, `Organizations->UserRole`.`UserId` **AS** `Organizations.UserRole.UserId`

**FROM** `users` **AS** `User`

**LEFT** **OUTER** **JOIN** (

`user\_roles` **AS** `Organizations->UserRole` **INNER** **JOIN** `organizations` **AS** `Organizations` **ON** `Organizations`.`id` = `Organizations->UserRole`.`OrganizationId`

) **ON** `User`.`id` = `Organizations->UserRole`.`UserId` **WHERE** `User`.`id` = 1;

#### Kueri membuat laporan insiden

Kueri di bawah berfungsi untuk melakukan *insert* data insiden ke tabel incidents. Kueri ini akan dijalankan ketika user melakukan pelaporan insiden dengan mengiripkan data yang diinput oleh user, label insiden yang dipilih oleh user dan lokasi yang didapatkan dari perangkat yang digunakan oleh user.

**INSERT** **INTO** `incidents` (`id`,`information`,`status`,`locationLat`,`locationLng`,`images`,`createdAt`,`updatedAt`,`createdBy`,`label`)

**VALUES** (**DEFAULT**,"tes laporan insiden", "OPEN",-6.294732799999999,106.8433408,"[\"https://res.cloudinary.com/sharofuddin/image/upload/v1621876173/rakyat62/fxdbkzpbza94hpdp8f3q.png\"]","2021-05-25 00:15:48","2021-05-25 00:15:48",1,2);

#### Kueri data insiden dengan filter status dan label

Kueri berikut mengambil data insiden dari tabel incidents dengan label tertentu. Ini akan dijalankan ketika petugas membuka halaman yang menampilkan daftar insiden sesuai label yang terkait dengan organisasi dari petugas di aplikasi ini. Ketentuan filter data yang dilakukan kueri ini adalah dengan satu pilihan status dan banyak pilihan label.

**SELECT** `id`, `information`, `status`, `locationAddress`, `locationLat`, `locationLng`, `images`, `createdAt`, `updatedAt`, `createdBy`, `label`, `label` **AS** `IncidentLabelId`

**FROM** `incidents` **AS** `Incident`

**WHERE** (`Incident`.`label` = 2) **AND** `Incident`.`status` = 'OPEN'

**ORDER** **BY** `Incident`.`createdAt` **DESC**;

#### Kueri data insiden beserta riwayat dan label insiden

Kueri berikut digunakan untuk menampilkan data insiden secara lebih lengkap pada halaman detil insiden. Data yang ditampilkan berupa riwayat insiden beserta label insidennya, lengkap dengan organisasi yang terkait dengan label dari insiden tersebut.

**SELECT** \* , `Organizations`.`id`, `Organizations`.`name`, `Organizations`.`isGovernment`, `Organizations`.`description`, `Organizations`.`officeAddress`, `Organizations`.`logoUrl`, `Organizations`.`createdAt`, `Organizations`.`updatedAt`, `Organizations->organization\_related\_incident\_label`.`createdAt` **AS** `Organizations.organization\_related\_incident\_label.createdAt`, `Organizations->organization\_related\_incident\_label`.`updatedAt` **AS** `Organizations.organization\_related\_incident\_label.updatedAt`, `Organizations->organization\_related\_incident\_label`.`IncidentLabelId` **AS** `Organizations.organization\_related\_incident\_label.IncidentLabelId`, `Organizations->organization\_related\_incident\_label`.`OrganizationId` **AS** `Organizations.organization\_related\_incident\_label.OrganizationId`

**FROM** `incident\_label` **AS** `IncidentLabel`

**LEFT** **OUTER** **JOIN** (

`organization\_related\_incident\_label` **AS** `Organizations->organization\_related\_incident\_label`

**INNER** **JOIN** `organizations` **AS** `Organizations` **ON** `Organizations`.`id` = `Organizations->organization\_related\_incident\_label`.`OrganizationId`

) **ON** `IncidentLabel`.`id` = `Organizations->organization\_related\_incident\_label`.`IncidentLabelId`

**WHERE** `IncidentLabel`.`id` = 2;

#### Kueri menambahkan riwayat insiden

Kueri berikut digunakan untuk menambahkan data riwayat dari suatu insiden tertentu. Data riwayat ini bisa berupa komentar atau tindak lanjut dari petugas jika yang menambahkan data adalah petugas dari organisasi yang terkait dengan label dari insiden yang dimaksud. Contoh berikut merupakan riwayat yang berupa komentar dari user, ditandai dengan field type “COMMENT”.

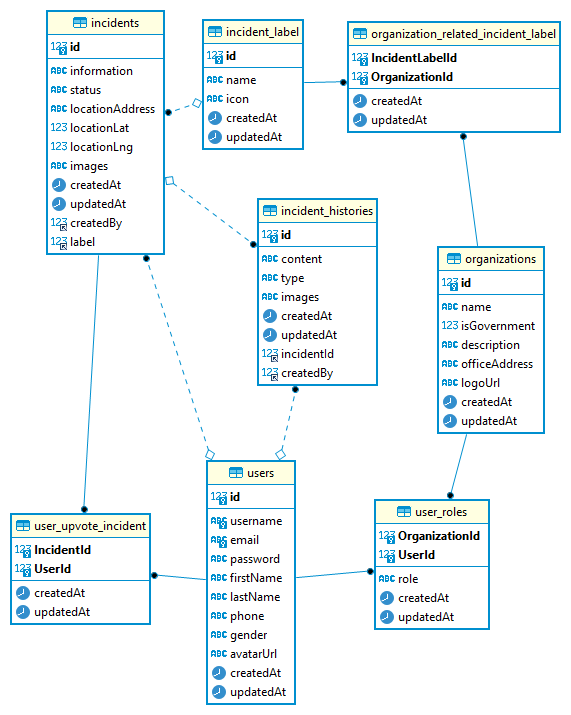
**INSERT** **INTO** `incident\_histories` (`id`,`content`,`type`,`images`,`createdAt`,`updatedAt`,`incidentId`,`createdBy`)

**VALUES** (**DEFAULT**,"tes komentar","COMMENT","[]","2021-05-25 00:52:16","2021-05-25 00:52:16",604,1);

## Implementasi

### Implementasi Basis Data

Dari perancangan yang dijelaskan penulis di atas, berikut adalah implementasi basis data Aplikasi Pelaporan Masyarakat. Implementasi basis data ini berupa *relational database* , digambarkan pada skema berikut yang didapatkan menggunakan bantuan aplikasi *dbeaver* untuk visualisasinya.



Gambar 4.7 Skema tabel databse Aplikasi Pelaporan Masyarakat

### Implementasi Skema GraphQL

Dalam mengembangkan API menggunakan graphQL, pengembang harus membuat skema graphQL terlebih dahulu. Skema ini adalah kontrak API yang memuat struktur, tipe, maupun relasi data sesuai spesifikasi yang ditentukan oleh Facebook selaku pengembang graphQL. Berikut adalah skema graphQL untuk Aplikasi Pelaporan Masyarakat.

schema {

  query: Query

  mutation: Mutation

  subscription: Subscription

}

type Query {

  incidents(

status: IncidentStatus,

labels: [Int!],

dateStart: DateTime,

dateEnd: DateTime

): IncidentCollection!

  incident(id: Int!): Incident!

  incidentLabels: [IncidentLabel!]!

  organizations: [Organization]!

  organization(id: Int!): Organization!

  users(keywords: String): [User!]!

  user(id: String, username: String): User!

  me: User!

}

type Mutation {

  createIncident(input: CreateIncidentInput!): Incident!

  addIncidentHistory(input: CreateIncidentHistoryInput!): IncidentHistory!

  createOrganization(input: CreateOrganizationInput!): Organization!

  addOrganizationMember(

organizationId: Int!, username: String!, role: UserRole

): Organization!

  addOrganizationRelatedLabel(

incidentLabelId: Int!, organizationId: Int!

): Organization!

  removeOrganizationRelatedLabel(

incidentLabelId: Int!, organizationId: Int!

): Organization!

  createUser(input: CreateUserInput!): User!

  login(input: LoginUserInput!): AuthPayload!

}

type Subscription {

  newIncident: Incident!

}

scalar MySqlFieldGroupBy

type IncidentCollection {

  nodes(offset: Int, limit: Int, orderBy: IncidentOrder): [Incident!]!

  stats(groupBy: MySqlFieldGroupBy!): [DataGroupStats]!

  totalCount: Int!

}

type DataGroupStats {

  count: Int!

  fieldGroup: MySqlFieldGroupBy!

}

input IncidentOrder {

  field: IncidentOrderField

  direction: OrderDirection

}

enum IncidentOrderField {

  createdAt

  label

  status

}

enum OrderDirection {

  ASC

  DESC

}

scalar DateTime

type Incident {

  id: Int!

  information: String

  status: IncidentStatus!

  locationAddress: String

  locationLat: Float!

  locationLng: Float!

  label: IncidentLabel!

  createdBy: User!

  histories: [IncidentHistory]!

  historiesCount: Int!

  images: [String!]!

  createdAt: DateTime!

  updatedAt: DateTime!

}

enum IncidentStatus {

  OPEN

  CLOSED

}

input CreateIncidentInput {

  information: String!

  locationAddress: String

  locationLat: Float!

  locationLng: Float!

  label: Int!

  images: [String!] = []

}

type IncidentLabel {

  id: Int!

  name: String!

  icon: String!

  incidents(

status: IncidentStatus, dateStart: DateTime, dateEnd: DateTime

): IncidentCollection!

  relatedOrganizations: [Organization]!

}

type IncidentHistory {

  id: Int!

  content: String

  type: IncidentHistoryType!

  images: [String!]!

  createdBy: User!

  createdAt: DateTime!

}

input CreateIncidentHistoryInput {

  content: String

  type: IncidentHistoryType!

  incidentId: Int!

  images: [String!] = []

}

enum IncidentHistoryType {

  FOLLOW\_UP

  COMMENT

}

type Organization {

  id: Int!

  name: String

  isGovernment: Boolean

  description: String

  officeAddress: String

  logoUrl: String

  members: [OrganizationMemberWithRole]

  relatedLabels: [IncidentLabel]

}

type OrganizationMemberWithRole implements UserType {

  id: Int!

  username: String!

  email: String!

  phone: String

  firstName: String!

  lastName: String!

  gender: UserGender!

  avatarUrl: String

  role: UserRole!

}

input CreateOrganizationInput {

  name: String!

  isGovernment: Boolean = false

  description: String = "*"*

  officeAddress: String!

}

type User implements UserType {

  id: Int!

  username: String!

  email: String!

  phone: String

  firstName: String!

  lastName: String!

  gender: UserGender!

  avatarUrl: String

  organizations: [UserOrganizationWithRole]

}

type UserOrganizationWithRole {

  id: Int!

  name: String

  isGovernment: Boolean

  description: String

  officeAddress: String

  logoUrl: String

  role: UserRole!

}

enum UserRole {

  OWNER

  MEMBER

  ADMIN

}

enum UserGender {

  MALE

  FEMALE

}

input CreateUserInput {

  username: String!

  email: String!

  phone: String

  firstName: String!

  lastName: String!

  gender: UserGender!

  password: String!

}

interface UserType {

  id: Int!

  username: String

  email: String

  phone: String

  firstName: String

  lastName: String

}

type AuthPayload {

  token: String!

  user: User!

}

input LoginUserInput {

  username: String!

  password: String!

}

### Implementasi Kueri GraphQL

Implementasi graphQL dilakukan berdasarkan kasus yang telah penulis jelaskan pada bagian kueri di atas. Berikut ini adalah bagaimana kueri kasus tersebut diimplementasikan dalam bentuk kueri graphQL

#### GraphQL data organisasi beserta role dari suatu user

{

  me {

    id

    username

    firstName

    lastName

    avatarUrl

    organizations {

      id

      name

      logoUrl

      role

    }

  }

}

Query graphQL di atas digunakan untuk mengambil data user yang sedang aktif ter-*login* di aplikasi. Query ini akan mengambil data dari tabel users di database berikut serta organisasi dan role dari user tersebut. Yang perlu dicatat dalam hal ini adalah pada query ini tidak ada id user, dikarenakan query ini dikirim melalui HTTP *request* server bisa mendapatkan id user dari proses authorisasi.

#### GraphQL membuat laporan insiden

mutation {

  createIncident(input: {

    information: "tes laporan insiden",

    locationLat: -6.294732799999999,

    locationLng: 106.8433408,

    label: 2,

    images: [

      "https://res.cloudinary.com/sharofuddin/image/upload/v1621876173/rakyat62/fxdbkzpbza94hpdp8f3q.png"

    ]

  }) {

    id

  }

}

Mutation graphQL di atas digunakan untuk menambah data laporan insiden yang dilakukan oleh user. Dalam kasus ini juga dalam query tidak menyertakan id user melainkan dengan *authorisasi* server dapat mengetahui id user dari query graphQL yang dijalankan.

#### GraphQL data insiden dengan filter status dan label

query {

  incidents(status: OPEN, labels: [2]) {

    nodes(orderBy: {field: createdAt, direction: DESC}) {

      id

      information

      locationLat

      locationLng

      status

      createdAt

      label {

        id

        name

      }

      createdBy {

        id

        username

      }

    }

  }

}

Query graphQL di atas digunakan untuk mendapatkan data insiden degan filter berdasarkan status dan label. Pada parameter label bisa menggunakan beberapa id label karena graphQL mendukung tipe data *array.*

#### GraphQL data insiden beserta riwayat dan label insiden

query {

  incident(id: 604) {

    id

    information

    status

    images

    histories {

      id

      content

      type

      images

      createdAt

      createdBy {

        id

        username

        avatarUrl

      }

    }

    label {

      id

      name

      relatedOrganizations {

        id

        name

        logoUrl

      }

    }

    createdBy {

      id

      username

      avatarUrl

    }

  }

}

Query graphQL di atas digunakan untuk mengambil satu data insiden berdasarkan id-nya berikut serta riwayat, label dan user yang membuat laporan insiden.

#### GraphQL menambahkan riwayat insiden

mutation {

  addIncidentHistory(input: {

    content: "tes komentar",

    type: COMMENT,

    incidentId: 604,

    images: []

  }) {

    id

    content

  }

}

Mutation graphQL di atas digunakan untuk menambahkan data riwayat insiden.

# BAB V HASIL PENELITIAN DAN PAMBAHASAN

## Hasil Penelitian

### Semantic API pada *Web Service*

Dengan menggunakan graphQL dalam pengembangan API *web service*, API yang dihasilkan memili arti dan aturan sebagaimana tujuan Facebook menembangkan graphQL. Ini dapat dilihat dari skema yang telah penulis buat pada bab 4 implementasi skema graphQL. Berikut penjelasan semantic API pada pelaporan insiden di Aplikasi Pelaporan Masyarakat.

type Mutation {

  createIncident(input: CreateIncidentInput!): Incident!

}

Di atas adalah bagian dari skema yang penulis sebutkan pada bagian Implementasi Skema GraphQL. Bagian tersebut merupakan mutation *createIncident* yang dibuat untuk digunakan menambahkan data insiden. Kemudian mutation ini menerima parameter bernama input yang bertipe CreateIncidentInput dan mengembalikan data dengan tipe Incident. Berikut ini potongan skema untuk melihat lebih jelas semantic untuk tipe CreateIncidentInput dan Incident

input CreateIncidentInput {

  information: String!

  locationAddress: String

  locationLat: Float!

  locationLng: Float!

  label: Int!

  images: [String!] = []

}

scalar DateTime

type Incident {

  id: Int!

  information: String

  status: IncidentStatus!

  locationAddress: String

  locationLat: Float!

  locationLng: Float!

  label: IncidentLabel!

  createdBy: User!

  histories: [IncidentHistory]!

  historiesCount: Int!

  images: [String!]!

  createdAt: DateTime!

  updatedAt: DateTime!

}

Skema tersebut dibuat sesuai dengan spesifikasi graphQL yang dibuat oleh Facebook. Dengan begitu, cara mengakses API untuk menambahkan data insiden adalah dengan mengirim query graphQL seperti berikut

mutation {

  createIncident(input: {

    information: "tes laporan insiden",

    locationLat: -6.294732799999999,

    locationLng: 106.8433408,

    label: 2,

    images: [

      "https://res.cloudinary.com/sharofuddin/image/upload/v1621876173/rakyat62/fxdbkzpbza94hpdp8f3q.png"

    ]

  }) {

    id

  }

}

Dari query tersebut, didapat respon data JSON seperti berikut

{

    "data": {

        "createIncident": {

            "id": 606

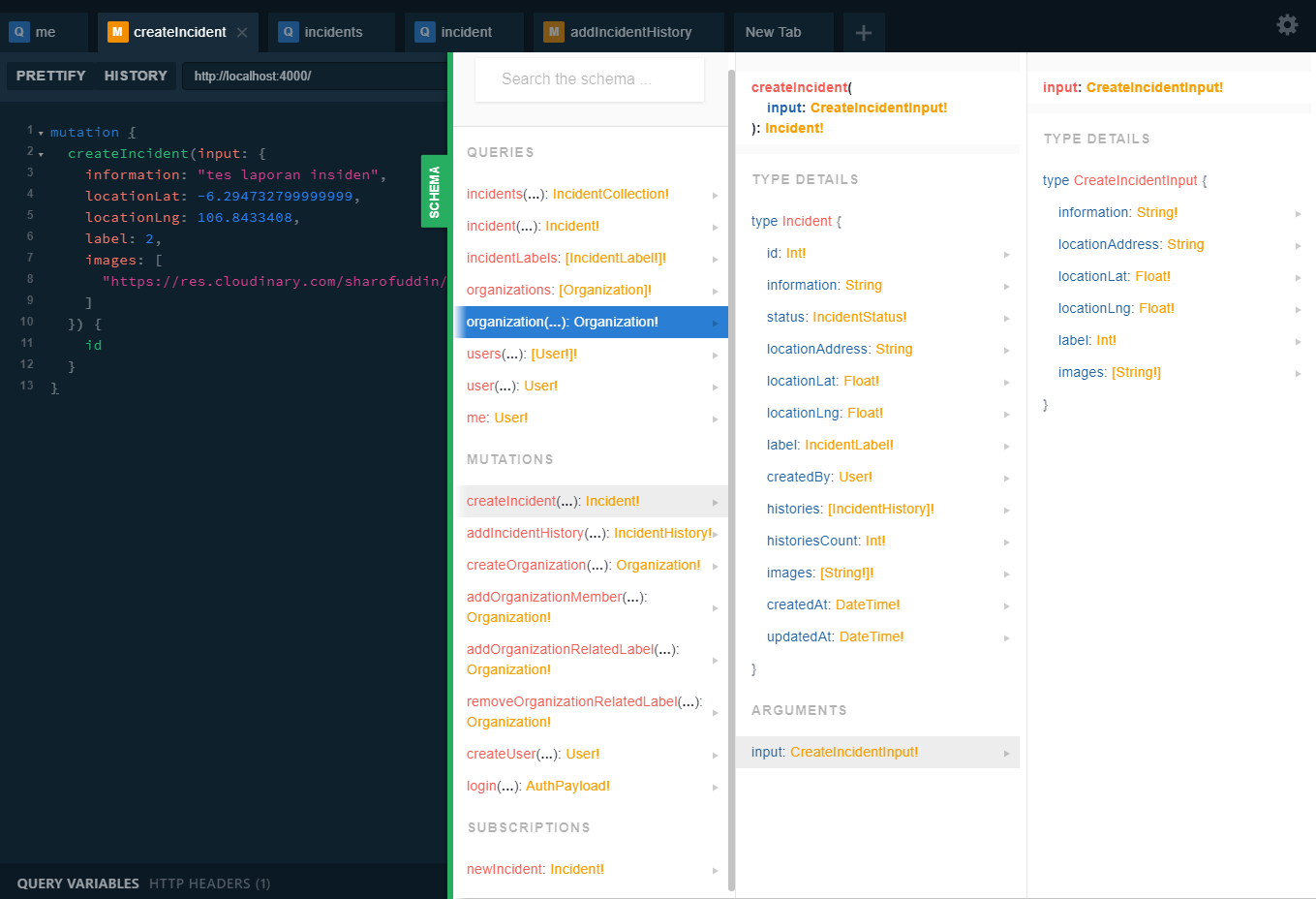
        }

    }

}

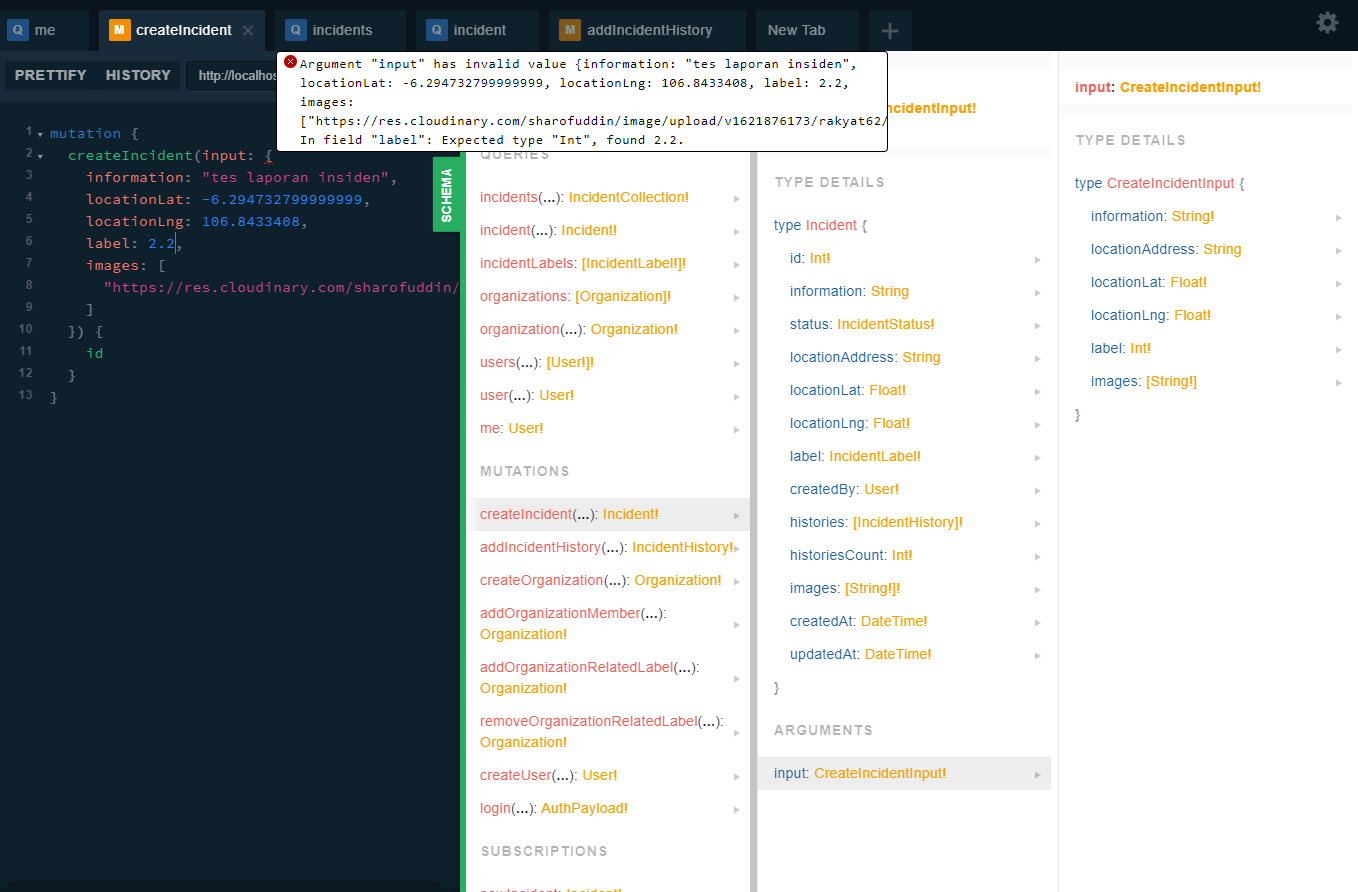
### *Developer* *tool*

Dengan API semantic seperti yang telah penulis jelaskan sebelumnya, pengembangan menggunakan graphQL ini memungkinkan penggunaan *tools* buatan pihak ketiga untuk mempermudah proses pengembangan. Salah satunya adalah graphQL playground.



Gambar 5.1 GraphQL playground

Degnan graphQL playground, pengembang bisa melihat visualisasi dari graphQL schema yang sesuai dengan semantic yang telah dijelaskan pada bagian sebelumnya dalam bentuk web yang interaktif. GraphQL playground juga bisa digunakan untuk mensimulasikan query *request* dan JSON *response* dengan bantuan pengkoreksian tipe data, sekali lagi ini juga sesuai dengan semantic dari graphQL schema yang ada.



Gambar 5.2 Koreksi input tipe data graphQL playground

Seperti contoh pada gambar di atas, graphQL playground menampilkan pesan error yang diakibatkan kesalahan tipe data field label yang seharusnya integer namun diisikan float, tidak sesuai dengan tipe yang seharusnya telah didefinisikan pada CreateIncidentInput.

### GraphQL Client

Yang dimaksudkan *graphQL* *client* adalah aplikasi yang akan mengkonsumsi *web service* yang telah dibuat dengan graphQL. Dalam penelitian ini penulis membuat aplikasi dalam bentuk web *single page application* dimana pemanggilan API graphQL dilakukan menggunakan bahasa pemrograman javascript yang berjalan di lingkungan peramban web.

Pada umumnya untuk mengkonsumsi API *web service* adalah dengan *HTTP* *request*. Demikian juga untuk *web service* graphQL, kueri dimuat pada *HTTP body* dengan *HTTP method* POST, kemudian kueri tersebut akan diproses oleh server dan dikembalikan dalam sebagai JSON dengan data sesuai dengan kueri yang dikirimkan. Dalam lingkungan javascript di peramban web, ini dapat dilakukan dengan menggunakan fungsi *fetch*[22]. Berikut contoh melakukan *fetch* kemudian datanya ditampilkan ke konsol.

const*response*=*await fetch*({

*url*:'apiurl.com',

*body*:{

*query*:`*mutation {*

*createIncident(input: {*

*information: "tes laporan insiden",*

*locationLat: -6.294732799999999,*

*locationLng: 106.8433408,*

*label: 2,*

*images: [*

*"https://res.cloudinary.com/sharofuddin/image/upload/v1621876173/rakyat62/fxdbkzpbza94hpdp8f3q.png"*

*]*

*}) {*

*id*

*}*

*}*`

}

});

const*data*=*await response.json*();

*console.log*(data);

Hasil dari fungsi di atas akan menampilkan data balikan dari kueri graphQL seperti berikut.

{

    "data": {

        "createIncident": {

            "id": 606

        }

    }

}

Data berformat JSON yang merupakan format data yang dapat secara natural diproses dalam struktur data program berbasis javascript.

Dalam implementasi di Aplikasi Pelaporan Masyarakat yang penulis kembangkan, penggunaan graphQL ini memungkinkan penggunaan *library* pihak ketiga berkode sumber terbuka yang mendukung fitur tambahan selain pemanggilan HTTP, yaitu melakukan *cache* data dari hasil kueri graphQL. *Library* tersebut adalah *apollo-graphql*[23]*.* Berikut contoh pemanggilan kueri seperti di atas tetapi dengan menggunakan *apollo*.

const*data*=*await this.$apollo.mutate*({

*mutation*:`*mutation {*

*createIncident(input: {*

*information:*${information}*,*

*locationLat:*${*userLocation.lat*}*,*

*locationLng:*${*userLocation.lng*}*,*

*label: 2,*

*images:*${*JSON.stringify*(images)}

*}) {*

*id*

*}*

*}*`,

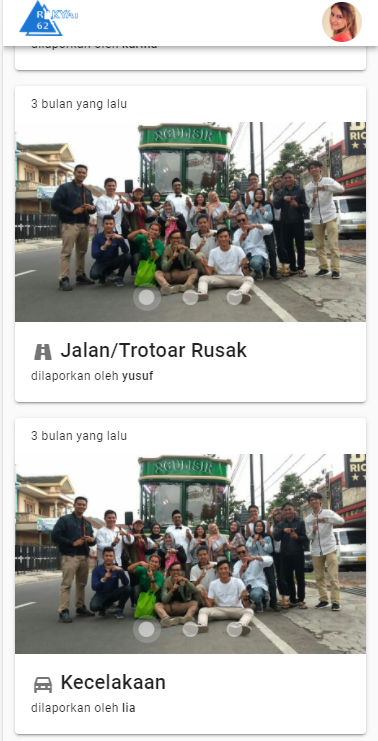
});

*console.log*(data);

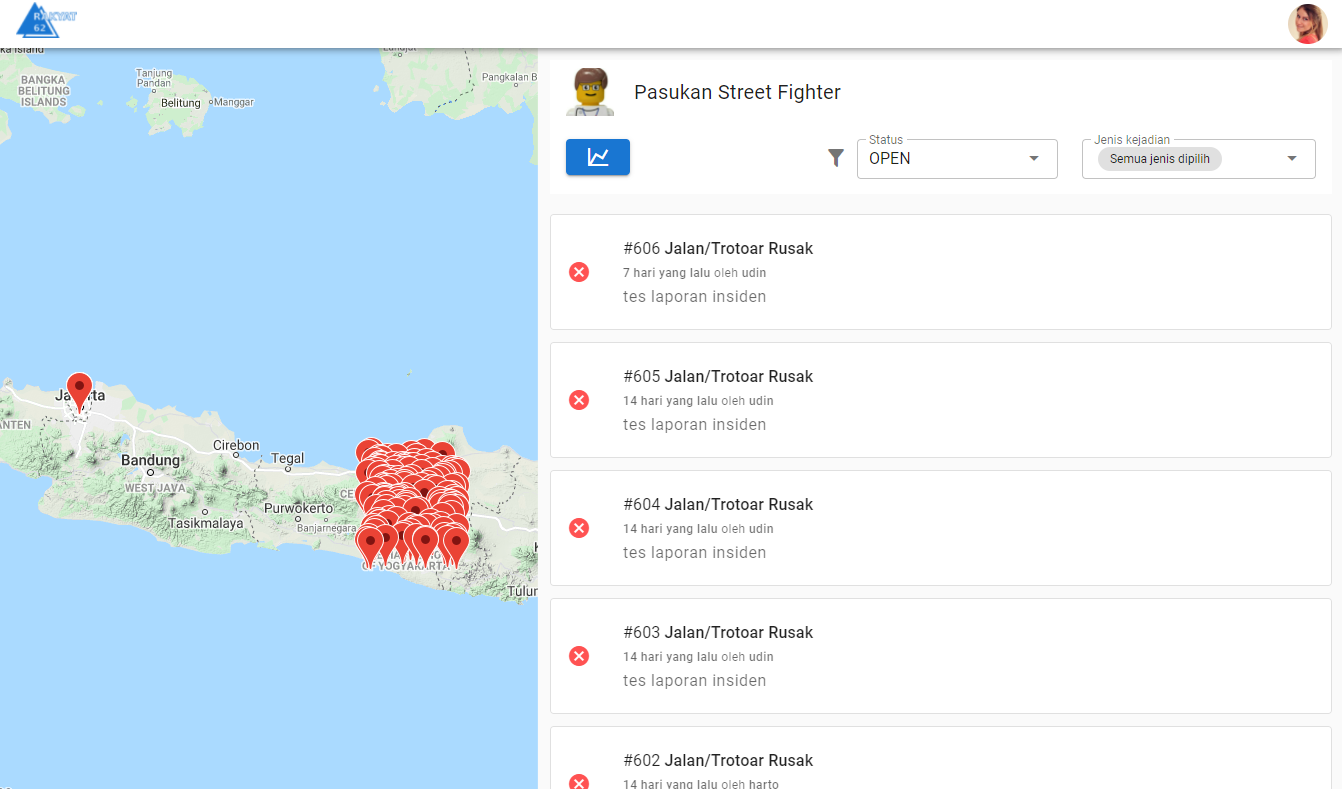
Apollo menyediakan objek *$apollo* dan method *mutate()* yang bisa digunakan untuk memanggil graphQL mutation. Pada potongan kode di atas, terlihat pemanggilan kueri graphQL yang sama dengan sebelumnya di atas yang penulis contohkan dengan *fetch*. Perbedaannya dengan apollo ini ketika memanggil kueri, hasil data balikan dari kueri tersebut akan secara otomatis disimpan dalam *cache*.

### Fleksibilitas GraphQL

Dalam pembahasan berikut ini penulis akan mengambil salah satu *function* yang digunakan pada dua halaman yang berbeda. Penulis menggunakan *query incidents* untuk melakukan mercobaan ini, dengan *fields* yang berbeda menyesuaikan kebutuh tampilan kedua halaman tersebut untuk menghasilkan balikan JSON yang sesuai juga. Di bawah ini merupakan halaman utama dan daftar insiden untuk petugas pada Aplikasi Pelaporan Masyarakat.



Gambar 5.3 Tampilan halaman utama



Gambar 5.4 Halaman daftar insiden pada petugas

Pada gambar 5.5 di atas terlihat halaman utama yang menampilkan daftar insiden. Pada daftar tersebut terlihat pada daftar insiden yang ditampilkan adalah tanggal laporan insiden itu dibuat, gambar, label, dan user yang melaporkan. Sehingga *query* yang dibutuhkan adalah sebagai berikut.

{

  incidents {

    nodes (

      limit: 20

      orderBy: { field: createdAt, direction: DESC }

    ) {

      id

      images

      label {

        id

        name

        icon

      }

      createdBy {

        id

        username

      }

      createdAt

    }

  }

}

Adapun untuk halaman daftar insiden untuk petugas yang terlihat pada gambar 5.6 di atas, dibutuhkan property data insiden yang lebih lengkap untuk menampilkan daftar beserta lokasi insiden untuk ditunjukkan pada peta sedemikian hingga *query*-nya adalah sebagai berikut.

{

  incidents(status: OPEN, labels: [2]) {

    nodes(orderBy: {field: createdAt, direction: DESC}) {

      id

      information

      locationLat

      locationLng

      status

      createdAt

      label {

        id

        name

      }

      createdBy {

        id

        username

      }

    }

  }

}

Dengan demikian dapat disimpulkan bahwa *web service* yang dikembangkan dengan graphQL memiliki fleksibilitas untuk pengkuerian data dengan hanya memanggil fungsi-fungsi yang diperlukan saja.

### Performa GraphQL

Dalam pembahasan kali ini penulis akan menggunakan salah salah satu fungsi dari Aplikasi Pelaporan Masyarakat ini untuk melihat ukuran data yang didapatkan dari API *web service* yang berfungsi untuk mengambil data daftar insiden, yaitu kueri *incidents*. Dengan melakukan kueri dengan atribut yang berbeda, penulis membandingkan ukuran data dari hasil kueri, sehingga terlihat selisih ukuran data dimana ukuran ini bergantung sesuai kebutuhan kueri data seperti yang penulis jelaskan pada 5.1.4. Berikut kueri *incidents* untuk mendapatkan data daftar insiden dengan atribut lengkap.

{

  incidents {

    nodes {

      id

      information

      locationLat

      locationLng

      status

      images

      label {

        id

        name

        icon

      }

      createdBy {

        id

        username

      }

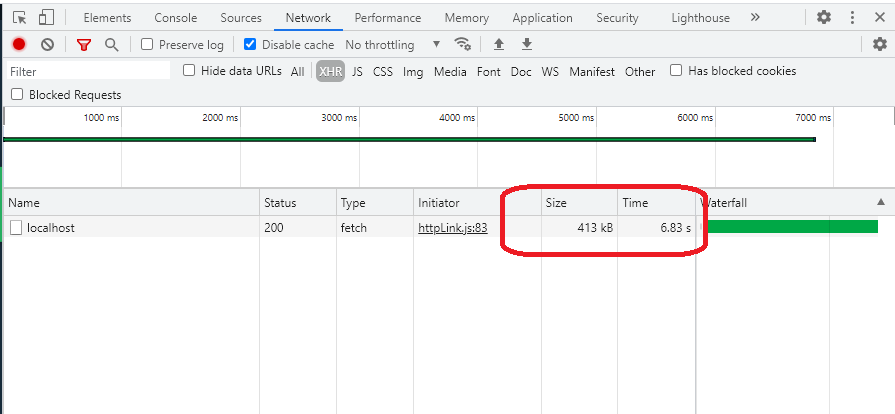
      createdAt

    }

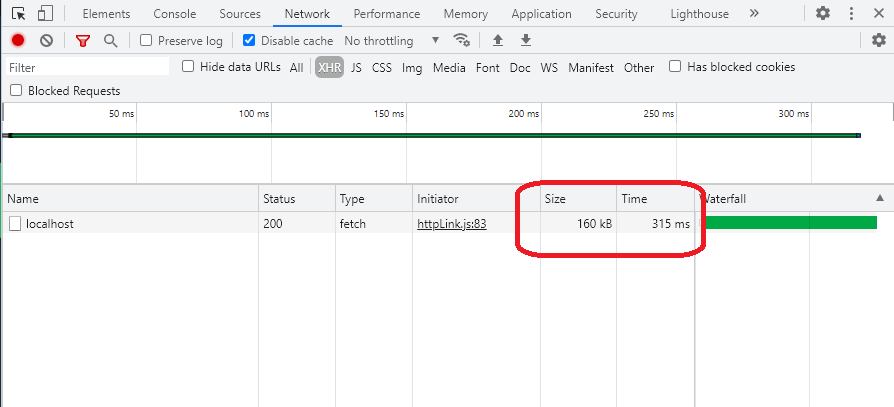
  }

}

Penulis menjalankan kueri dua kali, yang pertama dengan atribut lengkap seperti di atas dan yang kedua hanya membawa atribut id, information dan status. Penulis akan melihat perbedaan ukuran hasil dari perbedaan atribut dari kueri tersebut.



Gambar 5.5 Ukuran hasil kueri incidents atribut lengkap



Gambar 5.6 Ukuran hasil kueri incidents tiga atribut

Dari gambar di atas pada gambar 5.5 dapat dilihat ukuran hasil kueri *incidents* dengan atribut lengkap adalah sebesar 413 kB dan memakan waktu 6,83 detik. Berikutnya dilakukan kueri *incidents* lagi namun tidak dengan atribut lengkap dan hasilnya pada gambar 5.6 ukuran data yang didapat adalah sebesar 160kB dan waktu 0,315 detik. Dengan ini dapat diketahui bahwa dengan meringkas atribut kueri graphQL dapat memperkecil beban jaringan dan kecepatan pengambilan data, karena *server* tidak perlu mengirim data yang tidak diminta oleh aplikasi dari sisi *client.*

## Pengujian

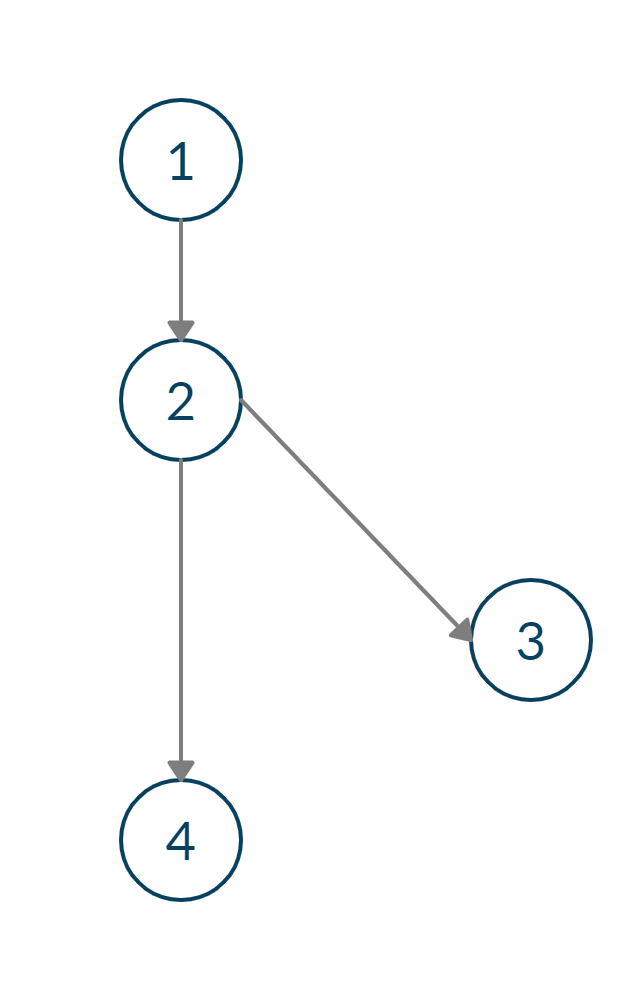
Pengujian dilakukan untuk memastikan apakah aplikasi yang telah dibangun dapat bekerja dengan baik sesuai spesifikasi yang telah ditentukan.

### White Box Testing

Pengujian dengan *white box testing*  ini akan penulis gambarkan notasi sederhana untuk representasi *flow graph* dan menentukan kompleksitas siklomatik dari *flow graph* tersebut. Pengujian ini dilakukan di bagian fungsi yang paling penting, yaitu fungsi untuk membuat laporan insiden.



Gambar 5.7 Potongan kode fungsi createIncident



Gambar 5.8 Flow graph fungsi createIncident

Kompleksitas Siklomatik:

V(G) = E – N + 2

V(G) = 3 – 4 + 2

V(G) = 1

### Black Box Testing

Di bawah ini merupakan tabel hasil pengujian Aplikasi Pelaporan Masyarakat menggunakan *black box testing*.

|  |  |  |  |
| --- | --- | --- | --- |
| No | Modul | Hasil yang diharapkan | Hasil Pengujian |
| 1 | Halaman utama | Menampilkan label-label insiden yang bisa dilaporkan, dan daftar insiden yang berlokasi di dekat user | Sesuai |
| 2 | Form buat laporan insiden | User dapat membuat laporan insiden dengan menginput keterangnan insiden, koordinat lokasi user dan label insiden yang telah dipilih dari halaman utama | Sesuai |
| 3 | Halaman daftar insiden pada dashboard petugas | Menampilkan daftar insiden sesuai dengan organisasi perugas yang sedang login pada aplikasi. | Sesuai |
| User dapat memilih label-label dan status untuk menyesuaikan daftar insiden yang ingin ditampilkan | Sesuai |
| 4 | Halaman rincian dan riwayat insiden | Menampilkan rincian insiden beserta riwayat komentar dan tindak lanjut dari petugas dalam urutan sesuai waktu. | Sesuai |
| User petugas dapat meng-klik tombol untuk tindak lanjut yang tampil di halaman ini. | Sesuai |
| 5 | Form tambahkan tindak lanjut insiden | User dapat menambah tindak lanjut insiden dengan data teks yang diinputkan dengan form. | Sesuai |

# BAB VI KESIMPULAN DAN SARAN

## Kesimpulan

Setelah mengimplementasikan graphQL untuk *web service API* pada Aplikasi Pelaporan Masyarakat, kesimpulan dari penulis adalah

* 1. Dari pembahasan semantic API pada backend aplikasi pelaporan masyarakat, dapat ditarik kesimpulan bahwa dengan menggunakan gaphQL, API yang dibuat memiliki semantic yang terdefinisi dengan jelas dan hal ini memberi keleluasaan pengembang untuk menggunakan *tools* bantuan pihak ketiga yang dianggap bermanfaat.

## Saran

Dalam penelitian ini tentunya penulis tidak bisa mencapai kesempurnaan. Oleh karena itu berikut saran untuk acuan penelitian selanjutnya.

* + 1. Pada proses *resolve* dari *query* graphQL ke pemanggilan *query* SQL, terdapat beberapa kasus seperti pengulangan pengiriman *query* SQL ke mesin database. Hal ini bisa lebih dioptimalkan dengan cara query SQL yang sekarang ini langsung dijalankan pada resolver, dibuat agregat dulu baru kemudian dikirim ke database.

# DAFTAR PUSTAKA

[1] N. Vesyropoulos, C. K. Georgiadis, dan P. Katsaros, “Ensuring business and service requirements in enterprise mashups,” *Inf Syst E-Bus Manage*, vol. 16, no. 1, Art. no. 1, Feb 2018, doi: 10.1007/s10257-017-0363-x.

[2] V. Hoyer dan M. Fischer, “Market Overview of Enterprise Mashup Tools,” dalam *Service-Oriented Computing – ICSOC 2008*, 2008, hlm. 708–721.

[3] G. Ghiani, F. Paternò, L. D. Spano, dan G. Pintori, “An environment for End-User Development of Web mashups,” *International Journal of Human-Computer Studies*, vol. 87, hlm. 38–64, Mar 2016, doi: 10.1016/j.ijhcs.2015.10.008.

[4] D. Bianchini, V. De Antonellis, dan M. Melchiori, “Semantics-Enabled Web API Organization and Recommendation,” dalam *Advances in Conceptual Modeling. Recent Developments and New Directions*, 2011, hlm. 34–43.

[5] M. J. Hadley, “Web Application Description Language (WADL),” Jan 2009.

[6] K. Soames dan J. Lind, *Detecting Cycles in GraphQL Schemas*. 2019. Diakses: Jul 11, 2019. [Daring]. Tersedia pada: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-156174

[7] O. Hartig dan J. Pérez, “An initial analysis of Facebook’s GraphQL language,” dalam *AMW 2017 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web, Montevideo, Uruguay, June 7-9, 2017.*, 2017, vol. 1912.

[8] E. Stenlund dan K. Gustavsson, “Efficent data communication between a webclient and a cloud environment,” 2016.

[9] M. Vogel, S. Weber, dan C. Zirpins, “Experiences on Migrating RESTful Web Services to GraphQL,” dalam *Service-Oriented Computing – ICSOC 2017 Workshops*, 2018, hlm. 283–295.

[10] M. Cremaschi dan F. De Paoli, “Toward Automatic Semantic API Descriptions to Support Services Composition,” dalam *Service-Oriented and Cloud Computing*, 2017, hlm. 159–167.

[11] P. R. ANDY, “Implementasi Webservice untuk Sinkronisasi Data Trankskrip Nilai di Tata Usaha Fakultas Universitas Dian Nuswantoro,” *Skripsi,Fakultas Ilmu Komputer*, 2016, Diakses: Jul 11, 2019. [Daring]. Tersedia pada: http://eprints.dinus.ac.id/18956/

[12] C. Lallemand, G. Gronier, dan V. Koenig, “User experience: A concept without consensus? Exploring practitioners’ perspectives through an international survey,” *Computers in Human Behavior*, vol. 43, hlm. 35–48, Feb 2015, doi: 10.1016/j.chb.2014.10.048.

[13] Snehal Mumbaika dan Puja Padiya, “Web services based on soap and rest principles,” *International Journal of Scientific and Research Publications (IJSRP)*, vol. 3, no. 5, Art. no. 5, Mei 2013.

[14] F. N. Rofiq dan A. Susanto, “Implementasi RESTful Web Service untuk Sistem Penghitungan Suara Secara Cepat pada Pilkada,” Art. no. 2, Mar 2017, Diakses: Jul 15, 2019. [Daring]. Tersedia pada: https://eksplora.stikom-bali.ac.id/index.php/eksplora/article/view/116

[15] R. Chinnici, J. J Moreau, A. Ryman, dan S. Weerawarana, “Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language,” *W3C working draft*, vol. 26, Jan 2004.

[16] “Home,” *OpenAPI Initiative*, Jul 15, 2019. https://www.openapis.org/ (diakses Jul 15, 2019).

[17] “GraphQL: A query language for APIs.,” Jul 15, 2019. http://graphql.org/ (diakses Jul 15, 2019).

[18] “The GitHub GraphQL API,” *The GitHub Blog*, Sep 14, 2016. https://github.blog/2016-09-14-the-github-graphql-api/ (diakses Jul 15, 2019).

[19] S. Nidhra, “Black Box and White Box Testing Techniques - A Literature Review,” *International Journal of Embedded Systems and Applications*, vol. 2, hlm. 29–50, Jun 2012, doi: 10.5121/ijesa.2012.2204.

[20] “(PDF) A Comparative Study of White Box, Black Box and Grey Box Testing Techniques,” Sep 23, 2019. https://www.researchgate.net/publication/270554162\_A\_Comparative\_Study\_of\_White\_Box\_Black\_Box\_and\_Grey\_Box\_Testing\_Techniques (diakses Sep 23, 2019).

[21] T. Ceglarek dan X. Kong, “webXstream - an optimal methodology for web development,” `, 2004.

[22] “Fetch API - Web APIs | MDN.” https://developer.mozilla.org/en-US/docs/Web/API/Fetch\_API (diakses Jun 11, 2021).

[23] “Introduction to Apollo Client,” *Apollo GraphQL Docs*. https://www.apollographql.com/docs/react/ (diakses Jun 11, 2021).