

# **SKRIPSI**

## **Implementasi GraphQL pada *Web Service* Aplikasi Pelaporan Masyarakat**

Diajukan untuk memenuhi salah satu syarat mengerjakan dan menempuh ujian  
tugas akhir 2



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	Mengetahui: Dekan Fakultas Ilmu Komputer
<b>Fahri Firdausillah S.Kom, M.CS</b>	<b>Dr. Drs. Abdul Syukur MM</b>

## 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
PENGESAHAN DEWAN PENGUJI.....	ii
PERSEMBAHAN.....	iii
KATA PENGANTAR .....	iv
DAFTAR ISI.....	v
DAFTAR GAMBAR .....	vii
DAFTAR TABEL.....	viii
BAB I PENDAHULUAN .....	1
1.1 Latar Belakang.....	1
1.2 Rumusan Masalah.....	2
1.3 Batasan Masalah .....	2
1.4 Tujuan Penelitian .....	2
1.5 Manfaat Penelitian.....	3
BAB II TINJAUAN PUSTAKA DAN LANDASAN TEORI .....	4
2.1 Tinjauan Studi.....	4
2.2 Tinjauan Pustaka.....	7
2.3 Kerangka Pemikiran .....	12
BAB III METODE PENELITIAN.....	14
3.1 Instrumen Penelitian .....	14
3.2 Jenis dan Sumber Data .....	15
3.3 Teknik Pengumpulan Data .....	15
3.4 Metode Pengembangan Sistem.....	15
3.5 Metode Pengujian.....	17
BAB IV PERANCANGAN SISTEM DAN IMPLEMENTASI .....	18
4.1 Pengantar .....	18
4.2 Studi Kasus .....	18

4.3 Perancangan Sistem .....	19
4.4 Implementasi .....	33
BAB V HASIL PENELITIAN DAN PEMBAHASAN .....	43
5.1 Hasil Penelitian.....	43
5.2 Pengujian .....	46
BAB VI KESIMPULAN DAN SARAN .....	48
6.1 Kesimpulan.....	48
6.2 Saran .....	48
DAFTAR PUSTAKA .....	49

## DAFTAR GAMBAR

Gambar 2.1 Backend dan Frontend .....	8
Gambar 2.2 Contoh graphql schema .....	11
Gambar 2.3 Contoh relasi pada schema graphql .....	11
Gambar 2.4 Contoh graphql request dan response .....	12
Gambar 3.1 Extreme Programming .....	16
Gambar 4.1 Wireframe Halaman Utama .....	19
Gambar 4.2 Wireframe Daftar Insiden .....	20
Gambar 4.3 Wireframe Rincian dan Riwayat Insiden .....	21
Gambar 4.4 Wireframe Statistik Insiden .....	22
Gambar 4.5 Activity Diagram Pelaporan Insiden .....	23
Gambar 4.6 ERD Basis Data .....	24
Gambar 4.7 Skema tabel database Aplikasi Pelaporan Masyarakat .....	33
Gambar 5.1 GraphQL playground .....	45
Gambar 5.2 Koreksi input tipe data graphql playground .....	46



## DAFTAR TABEL

Tabel 4.1 Tabel users .....	25
Tabel 4.2 Tabel users_roles .....	25
Tabel 4.3 Tabel organizations.....	26
Tabel 4.4 Tabel organization_related_incident_label.....	26
Tabel 4.5 Tabel incident_label.....	27
Tabel 4.6 Tabel incidents.....	28
Tabel 4.7 Tabel incident_histories.....	29
Tabel 4.8 Tabel user_upvote_incident.....	29

# BAB I

## PENDAHULUAN

### 1.1 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”.

## 1.2 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 *underfetching* dan *overfetching*.
3. Bagaimana membangun *Web Service* API yang semantic.

## 1.3 Batasan Masalah

1. Metode *Web Service* penulis menggunakan 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

## 1.4 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

### **1.5 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

#### 2.1 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 menentukan 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 menyatakan 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 *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 menyatakan GraphQL dalam konseptual memiliki keunggulan dalam struktur sumber dayanya yang berupa <i>object graph</i> 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 <i>tools</i> Swagger Editor yang tersedia dengan <i>Open API Specification</i> untuk membuat <i>Table Interpretation</i> sehingga <i>value</i> 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	REST dan GraphQL	Pada penelitian ini menghasilkan, jika pengembang lebih memilih

			Client and a Cloud Environment		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.
--	--	--	--------------------------------	--	--

## 2.2 Tinjauan Pustaka

### 2.2.1 Aplikasi Modern

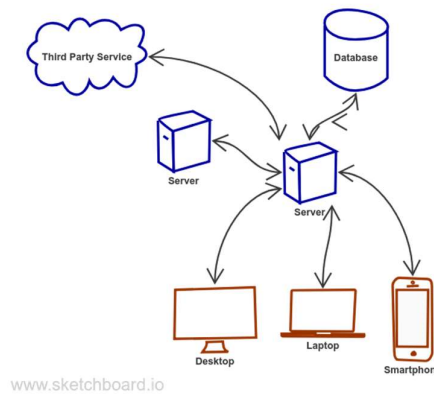
Perkembangan teknologi informasi modern ini, pembuatan sebuah aplikasi merupakan sebuah kesatuan sistem yang dibangun dengan berbagai sumber atau aplikasi yang berberda. Hal ini kemudian menjadi tuntutan dimana sebuah sistem harus bisa melakukan integrasi dengan sistem yang lain [11]. Berbagai aplikasi 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 seringkali 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.

### 2.2.2 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.

### 2.2.3 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.

### 2.2.4 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*.

### 2.2.5 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.

```
type User {  
  
  name: String!  
  
  username: String!  
  
}
```

*Gambar 2.2 Contoh graphql schema*

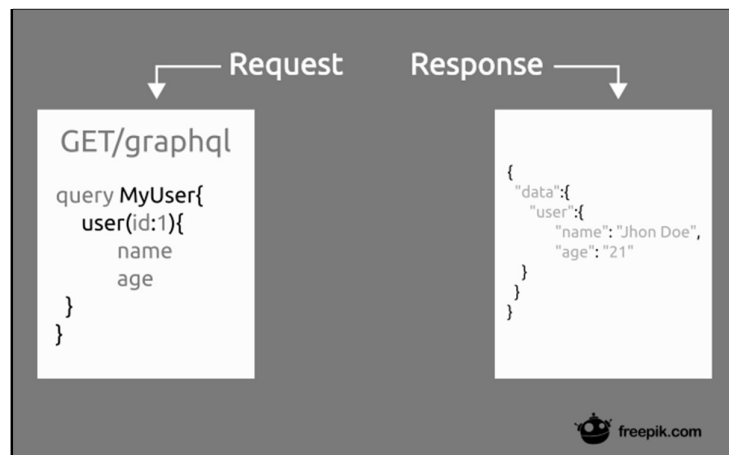
Contoh di atas merupakan *schema* User yang memiliki dua field bersifat *required* yang ditandai dengan tanda ! dan bertipe string.

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

```
type Document {  
  
  title: String!  
  
  content: String!  
  
  author: User!  
  
}
```

*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].

### 2.3 Kerangka Pemikiran

Kerangka pemikiran dalam penelitian ini dapat digambarkan sebagai berikut.

Permasalahan
Bagaimana membangun <i>Web Service</i> yang semantic dan dapat digunakan secara efektif

Pendekatan
Pembuatan aplikasi backend menggunakan GraphQL untuk <i>web service</i> API yang semantic

Pengembangan
--------------

Sisi Server : API berbasis GraphQL dengan NodeJS Sisi <i>Client</i> : web berupa <i>Single Page Application</i> dengan bahasa pemrograman javascript.
--

Pengujian
-----------

Menggunakan <i>unit test</i> untuk fungsi yang berjalan mandiri dan <i>integration test</i> untuk fungsi yang berhubungan dengan aplikasi lain
--

Hasil
-------

Aplikasi backend dengan <i>web service</i> GraphQL
--

## **BAB III**

### **METODE PENELITIAN**

#### **3.1 Instrumen Penelitian**

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

##### **3.1.1 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.

##### **3.1.2 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. Kapasitas RAM 12GB.
4. Penyimpanan SSD Samsung EVO 850 250GB.

### **3.2 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

### **3.3 Teknik Pengumpulan Data**

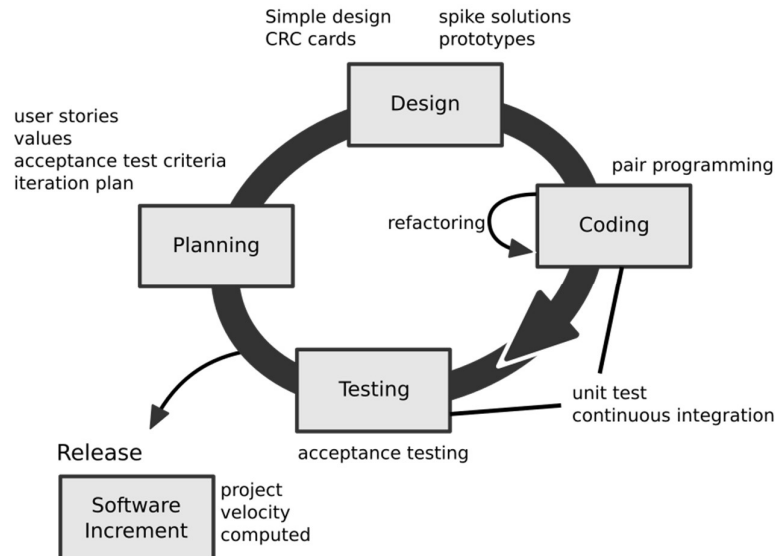
Metode yang digunakan untuk mengumpulkan data dalam melakukan penelitian ini adalah studi pustaka. Studi pustaka digunakan dengan cara mencari dan memahami teori-teori yang ada pada literatur terkait dengan penelitian yang dilakukan, salah satunya adalah penelitian “Toward Automatic Semantic API Descriptions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

### **3.4 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 pada Aplikasi Pelaporan Masyarakat.

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





Gambar 3.1 Extreme Programming

#### 3.4.1 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

#### 3.4.2 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 singkat dibanding dengan yang rumit. Pemrograman ekstrim merekomendasikan untuk

membuat prototype yang operasional untuk kemudian dievaluasi. Solusi ini disebut dengan *spike*.

### 3.4.3 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.

### 3.4.4 Pengujian

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

## 3.5 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. Penggunaan metode berbasis path ini memungkinkan untuk menentukan tingkat kompleksitas dalam level komponen, fungsi dan logic[20]. Dalam extreme programming, white-box testing dapat menggunakan unit test, yang bisa dilakukan menggunakan beragam *framework* yang sudah ada hingga pengujiannya dapat diotomatisasi.

## BAB IV

### PERANCANGAN SISTEM DAN IMPLEMENTASI

#### 4.1 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.

#### 4.2 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 desktop 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* disini 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.

### 4.3 Perancangan Sistem

#### 4.3.1 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.

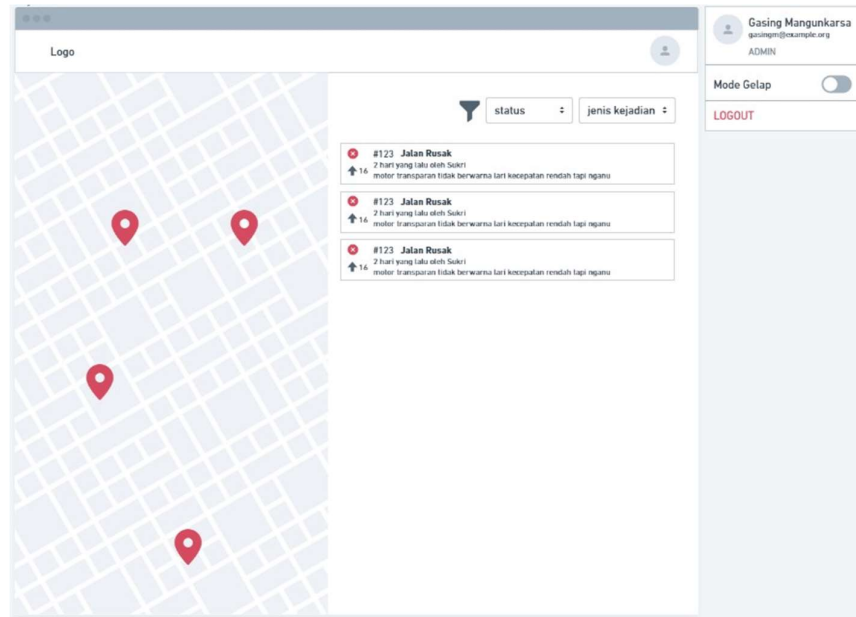
##### 4.3.1.1 Halaman awal atau halaman utama



*Gambar 4.1 Wireframe Halaman Utama*

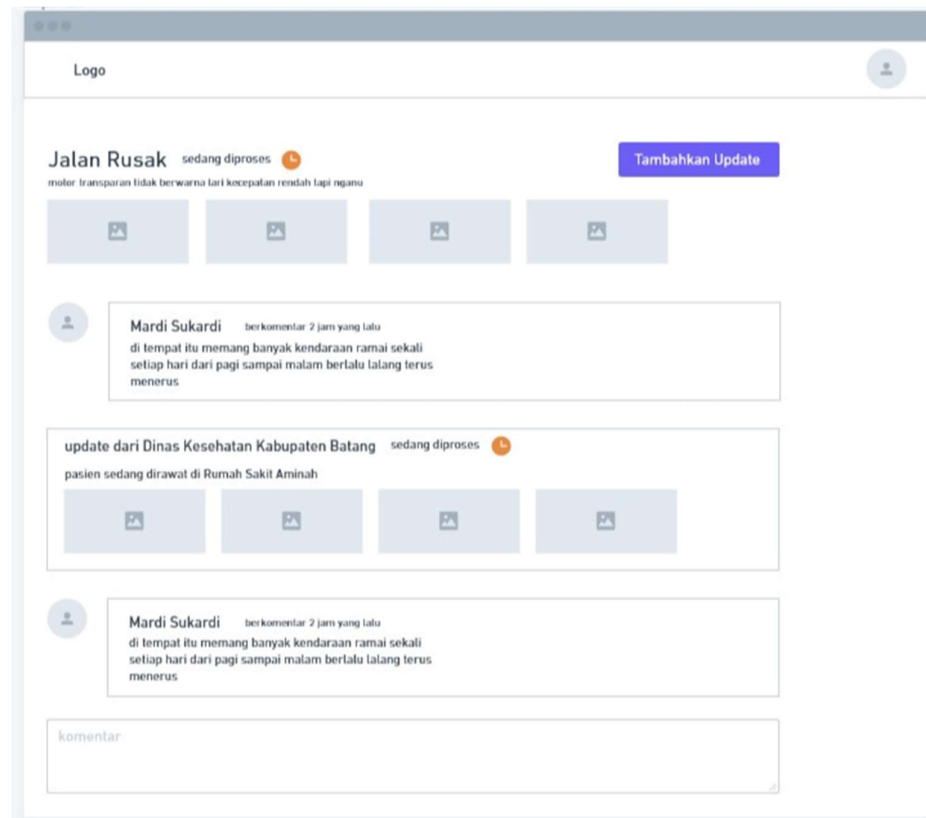
Halaman utama adalah halaman ketika web ini pertama kali diakses. Antarmukanya menampilkan label-label insiden yang bisa dipilih untuk user melaporkan insiden yang ditemui. Insiden yang dilaporkan akan dapat dilihat dan ditindaklanjuti oleh organisasi terkait sesuai dengan label insiden yang dilaporkan. 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.

#### 4.3.1.2 Halaman Insiden dari sisi petugas



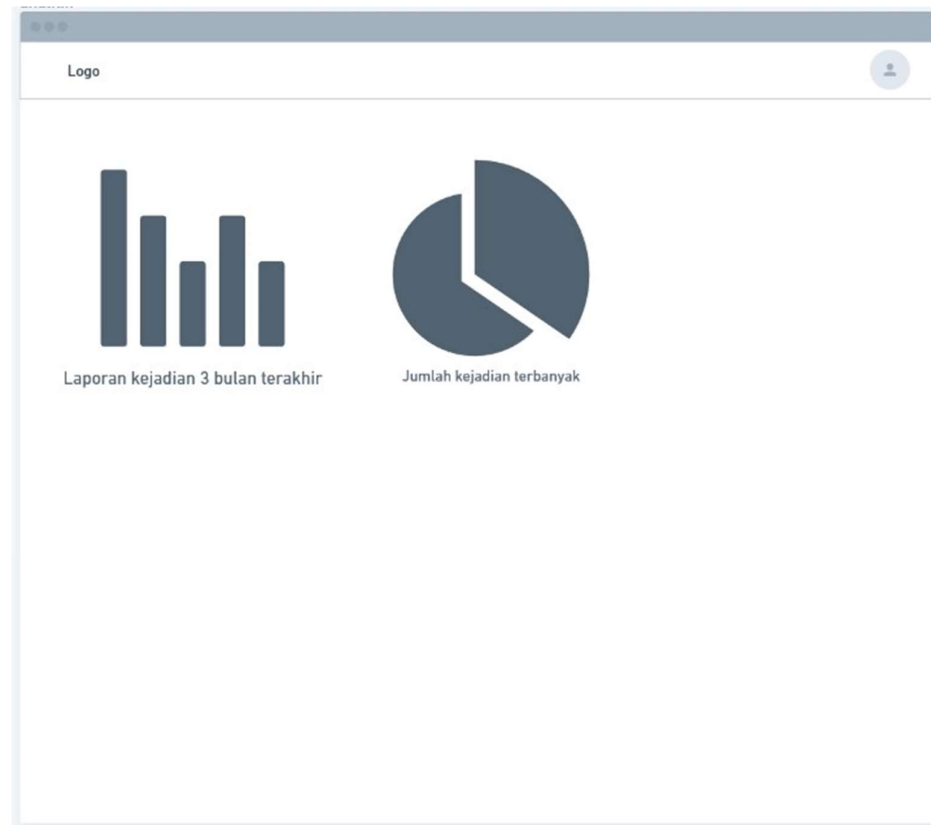
*Gambar 4.2 Wireframe Daftar Insiden*

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 detail lokasi kejadian yang dilaporkan.



*Gambar 4.3 Wireframe Rincian dan Riwayat Insiden*

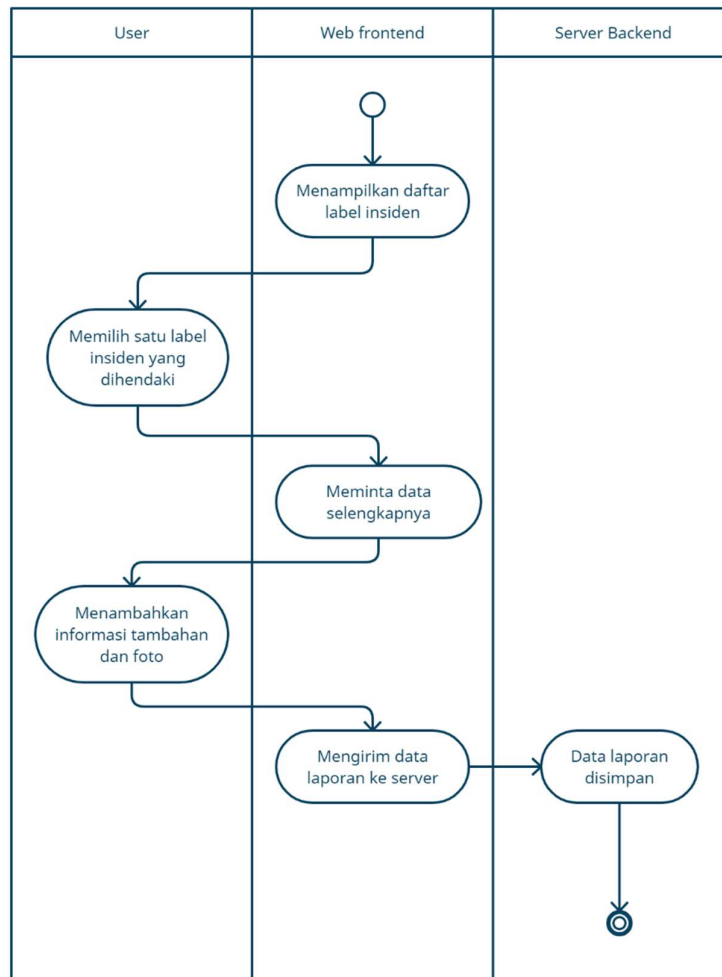
Di halaman detail insiden ini semua user termasuk petugas dapat melihat rincian insiden yang dipalorkan lengkap dengan foto dan riwayat komentar dan tindak lanjut dari petugas instansi terkait. Untuk petugas yang sedang membuka halaman ini juga bisa memberikan tindak lanjut yang telah dilakukan untuk 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.

### 4.3.2 Activity Diagram

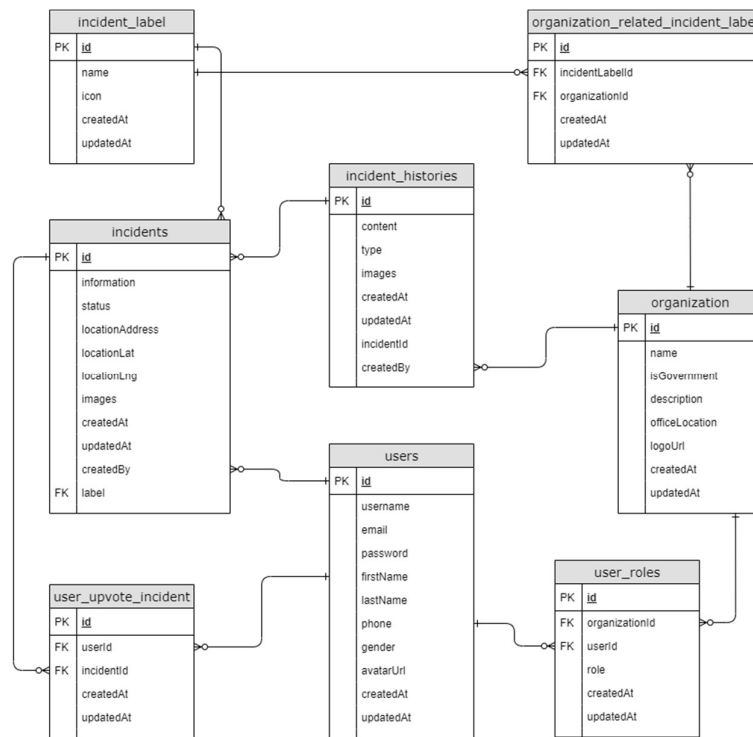


*Gambar 4.5 Activity Diagram Pelaporan Insiden*

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.



### 4.3.3 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.

#### 4.3.3.1 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.

#### 4.3.3.2 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.

#### 4.3.3.3 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 `incident_labels` sebagai label-label insiden yang akan ditindaklanjuti oleh organisasi. Dengan begitu label-label insiden yang ada akan memiliki organisasi-organisasi terkait yang akan menindaklanjuti laporan insiden yang dibuat oleh user. Tabel ini juga memiliki relasi ke tabel `user_roles` yang telah di jelaskan di atas sebelumnya.

#### 4.3.3.4 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 menghubungkan relasi many-to-many antara tabel `organizations`

dengan incident\_label. Setiap organisasi memiliki banyak label insiden dan setiap label insiden memiliki banyak organisasi.

#### 4.3.3.5 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.

#### 4.3.3.6 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	Foreign key, id user yang membuat laporan
11	label	integer	Foreign key, id label insiden pada tabel 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.

#### 4.3.3.7 Tabel incident\_histories

No	Nama Field	Tipe Data	Keterangan
1	id	integer	Primary key, auto increment
2	content	text	Text berisi 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	incidentId	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.

#### 4.3.3.8 Tabel user\_upvote\_incident

No	Nama Field	Tipe Data	Keterangan
1	incidentId	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 merupakan tabel pivot antara tabel users dan incidents.

#### 4.3.4 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.

##### 4.3.4.1 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;
```

##### 4.3.4.2 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`,`ima
ges`,`createdAt`,`updatedAt`,`createdBy`,`label`)
VALUES (DEFAULT,"tes laporan insiden", "OPEN",-
6.294732799999999,106.8433408,"[\"https://res.cloudinary.com/
sharofuddin/image/upload/v1621876173/rakyat62/fxdbkzpbza94hpd
p8f3q.png\"]", "2021-05-25 00:15:48", "2021-05-25
00:15:48",1,2);

```

#### 4.3.4.3 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;

```

#### 4.3.4.4 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`.`created
At` AS
`Organizations.organization_related_incident_label.createdAt`
,
`Organizations->organization_related_incident_label`.`updated
At` AS
`Organizations.organization_related_incident_label.updatedAt`
,
`Organizations->organization_related_incident_label`.`Inciden
tLabelId` AS
`Organizations.organization_related_incident_label.IncidentLa
belId`,
`Organizations->organization_related_incident_label`.`Organiz
ationId` AS
`Organizations.organization_related_incident_label.Organizati
onId`
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`.`Organiz
ationId`
) ON `IncidentLabel`.`id` =
`Organizations->organization_related_incident_label`.`Inciden
tLabelId`
WHERE `IncidentLabel`.`id` = 2;

```

#### 4.3.4.5 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”.

```

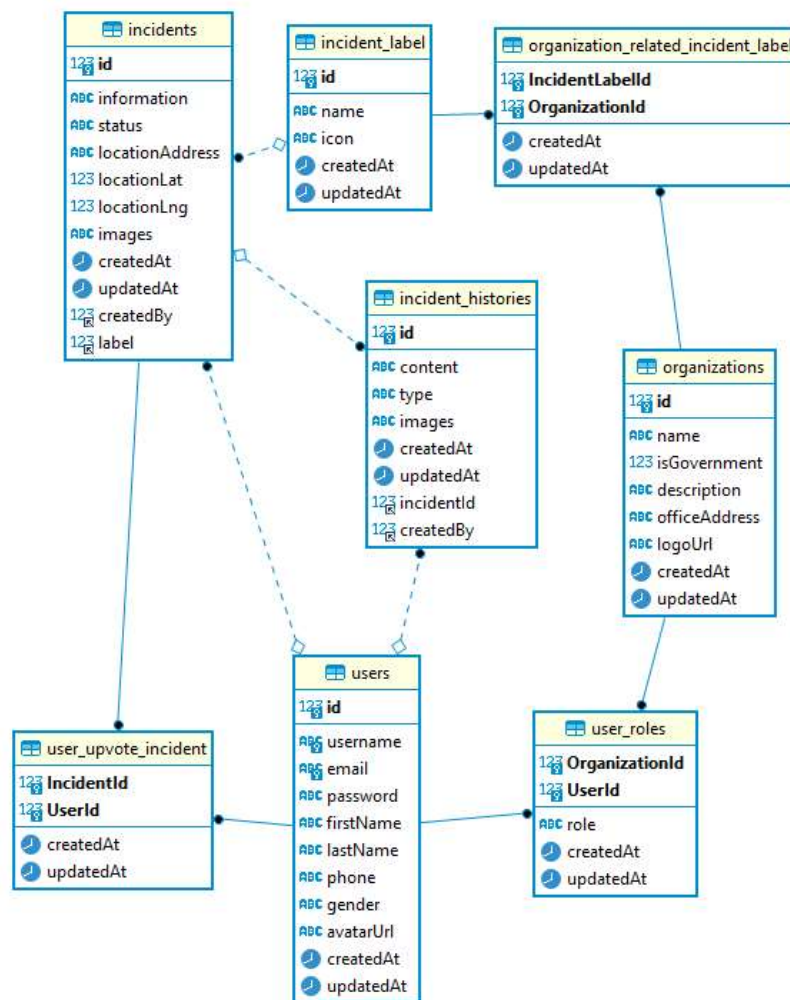
INSERT INTO `incident_histories`
(`id`,`content`,`type`,`images`,`createdAt`,`updatedAt`,`inci
dentId`,`createdBy`)
VALUES (DEFAULT,"tes komentar","COMMENT","[]","2021-05-25
00:52:16","2021-05-25 00:52:16",604,1);

```

## 4.4 Implementasi

### 4.4.1 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 database Aplikasi Pelaporan Masyarakat

### 4.4.2 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!
}
```

#### 4.4.3 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

##### 4.4.3.1 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*.



#### 4.4.3.2 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/fxdbkzpbza94hdp8f3q.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.

#### 4.4.3.3 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 dengan filter berdasarkan status dan label. Pada parameter label bisa menggunakan beberapa id label karena graphql mendukung tipe data *array*.

#### 4.4.3.4 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.

#### 4.4.3.5 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 PEMBAHASAN

#### 5.1 Hasil Penelitian

##### 5.1.1 Semantic API pada *Backend*

Dengan menggunakan graphql dalam pengembangan API *web service*, API yang dihasilkan memiliki arti dan aturan sebagaimana tujuan Facebook mengembangkan 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/fxdbkzpbza94hdp8f3q.png"
    ]
  }) {
    id
  }
}

```

Dari query tersebut, didapat respon data JSON seperti berikut

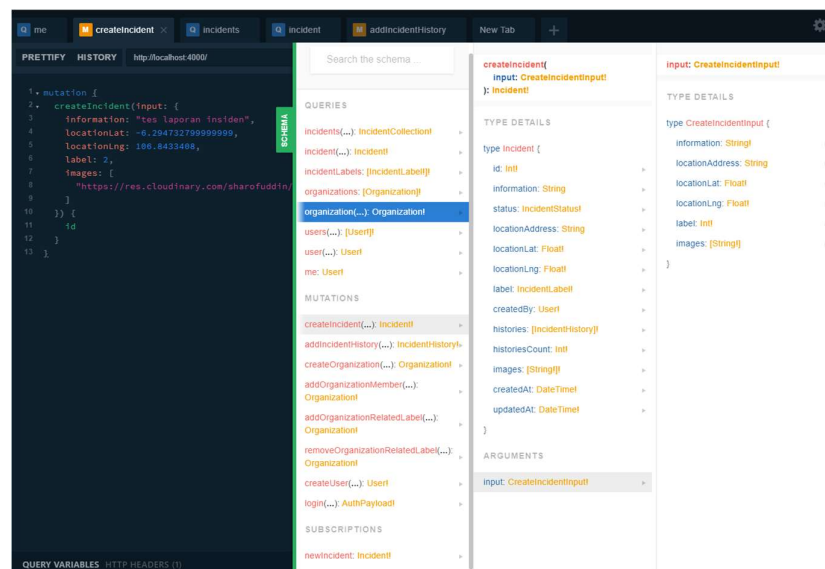
```

{
  "data": {
    "createIncident": {
      "id": 606
    }
  }
}

```

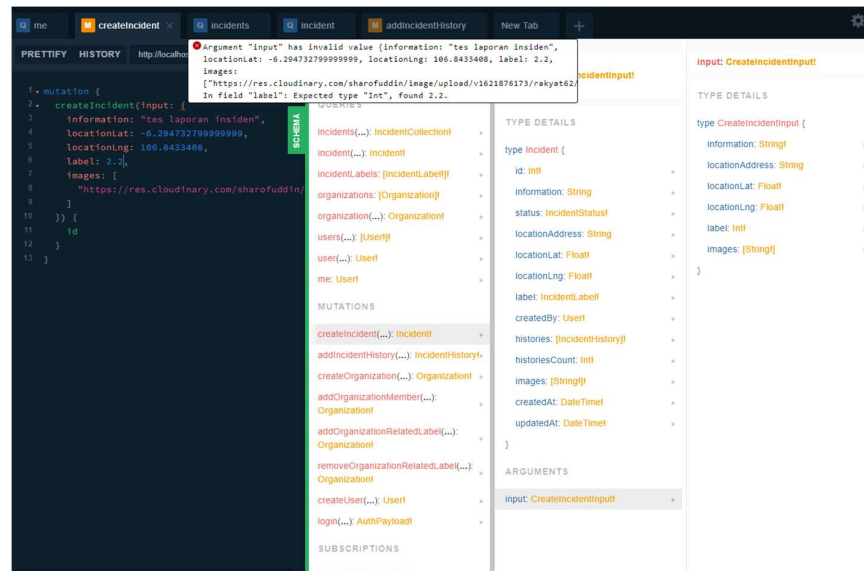
### 5.1.2 Developer tool

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

Dengan 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 diisi float, tidak sesuai dengan tipe yang seharusnya telah didefinisikan sebagai CreateIncidentInput.

## 5.2 Pengujian

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

### 5.2.1 White Box Testing

### 5.2.2 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 keterangan 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

### 5.2.3 Performa GraphQL



## BAB VI

### KESIMPULAN DAN SARAN

#### 6.1 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 GraphQL, API yang dibuat memiliki semantic yang terdefinisi dengan jelas dan hal ini memberi keleluasaan pengembang untuk menggunakan *tools* bantuan pihak ketiga yang dianggap bermanfaat.

#### 6.2 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.