RANCANG BANGUN APPLICATION PROGRAMMING INTERFACE MENGGUNAKAN GRAPHQL PADA MULTI-PLATFORM APPLICATION

Gede Humaswara Prathama[1)] I Gusti Ngurah Darma Paramartha[2)]
Program Studi Teknologi Informasi[1) 2)]
Fakultas Teknik dan Informatika, Universitas Pendidikan Nasional, Denpasar, Bali[1) 2)]
huma@undiknas.ac.id[(1)] ngurahdarma@undiknas.ac.id[2)]

ABSTRACT

During the last decade the use of APIs has become a mandatory service for web-based application development, in addition to using APIs for communication between applications and database servers, APIs are also used as communication between applications to share data with each other in the B2B (Business to Business) concept. In the current API development, RESTful and SOAP architectures still dominate where RESTful produces JSON data format and SOAP produces XML data format. In the current development of applications where the use of APIs is used by various platforms such as web, mobile, desktop, smartwatch, IoT devices and other platforms in the same application or data so that API developers who utilize RESTful or SOAP architecture create many versions and API nodes to serve API requests from each of these platforms. Not a few developments with limited resources that do not create versions or API nodes for each platform at the expense of the performance of the API response both from latency, bandwidth and data provided. This research will apply GraphQL architecture in developing APIs that are able to serve applications from various platforms, GraphQL can overcome problems related to available data and application requirements, minimal use of latency and bandwidth according to the needs of each platform as well as documentation and version problems of the API by utilizing the features provided on the GraphQL architecture.
Keywords*: GraphQL, Multi-platform, API, RESTful, latency, bandwidth*.

ABSTRAK

Selama dekade terakhir penggunaan API menjadi layanan yang wajib disediakan dalam pengembangan aplikasi berbasis web, selain pemanfaatan API untuk komunikasi antara aplikasi dengan database server, API juga dimanfaatkan sebagai komunikasi antar aplikasi untuk saling berbagi data dalam konsep B2B (Business to Business). Dalam pengembangan API saat ini arsitektur RESTful dan SOAP masih mendominasi dimana RESTful menghasilkan format data JSON dan SOAP menghasilkan format data berbentuk XML. Dalam perkembangan aplikasi saat ini dimana penggunaan API digunakan berbagai macam platform seperti web, mobile, desktop, smartwatch, IoT devices dan platform lain dalam satu aplikasi atau data yang sama sehingga pengembang API yang memanfaatkan arsitektur RESTful atau SOAP membuat banyak versi dan node API untuk melayani permintaan API dari masing-masing platform tersebut. Tidak sedikit pengembangan dengan keterbatasan resource yang tidak mengkhususkan versi atau node API untuk masing-masing platform dengan mengorbankan performa dari response API tersebut baik dari latency, bandwidth maupun data yang disediakan. Penelitian ini akan menerapkan arsitektur GraphQL dalam mengembangkan API yang mampu melayani aplikasi dari berbagai platform, dimana GraphQL dapat mengatasi masalah terkait data yang tersedia dan dibutuhkan aplikasi, penggunaan latency dan bandwidth yang minimal sesuai kebutuhan masing-masing platform serta permasalahan dokumentasi dan versi dari API dengan memanfaatkan fitur-fitur yang disediakan pada arsitektur GraphQL.
Kata Kunci: *GraphQL, Multi-platform, API, RESTful, latency, bandwidth*.

PENDAHULUAN

Perkembangan internet saat ini merubah cara pengguna dalam menggunakan aplikasi. pengguna mengharapkan aplikasi lebih mudah diakses sebagai layanan yang menawarkan ketersediaan kepada siapapun, dimanapun dan kapan saja melalui hampir semua perangkat dan platform [1]. Pertumbuhan yang heterogenitas dari berbagai perangkat dan platform ini mendorong pengalaman pengguna yang beragam tergantung pada perangkat dan platform yang ingin digunakan [1]. Ini telah menjadi tantangan bagi pengembang untuk menyediakan layanan di seluruh perangkat dan platform yang ada.

Salah satu tantangan yang dihadapi oleh pengembang untuk menyediakan layanan di berbagai perangkat dan platform ini ada pada bagian UI/UX yang beragam. Dengan cara yang beragam ini tentu data yang ditampilkan dari setiap langkah pada aplikasi akan beragam pula, permasalahan yang muncul dari tantangan ini adalah pada API yang menyediakan data bagi aplikasi tersebut, pengembang didorong untuk menyediakan endpoint dari API yang lebih spesifik dari masing-masing UI/UX tersebut untuk menjaga performa aplikasi, terlebih lagi saat ini beberapa platform mulai mengawasi aplikasi pada platform mereka dalam penggunaan bandwidth sehingga platform bisa menawarkan pengguna untuk menghapus aplikasi yang terlalu banyak menggunakan bandwidth.

Evolusi dari API memiliki tantangan tersendiri yang terkait dengan versi, dokumentasi, dan komunikasi API [2]. Perubahan yang terjadi pada API dapat mempengaruhi aplikasi dan pengembang yang menggunakan API tersebut jika API tersebut merupakan public API tanpa adanya pengelolaan yang baik pada versi, dokumentasi dan komunikasi. Dari beberapa arsitektur API yang ditawarkan diantaranya SOAP, WSDL, RESTful, RPC dan lainnya, RESTful menjadi pilihan populer dalam satu dekade ini. RESTful menawarkan kemudahan dan keunggulan dalam hal performa [3]. Namun RESTful masih memiliki kendala pada pengelolaan versi, dokumentasi dan data yang disediakan dimana kebutuhan API saat ini digunakan bukan hanya untuk layanan kepada aplikasi pengembang itu sendiri tapi juga pengembang lain yang menggunakan data yang sama atau disebut dengan Business to Business application sehingga data yang dibutuhkan tidak bisa diprediksi terutama pada aplikasi yang memiliki data cukup besar. Dengan RESTful Pengembang API harus menyediakan endpoint yang cukup banyak untuk bisa menjaga fleksibilitas dari data yang akan diberikan. Dari permasalahan tersebut pada tahun 2016, Facebook merilis spesifikasi [4]. dan referensi implementasi dari GraphQL framework. Framework ini memperkenalkan arsitektur baru pada API berbasis web yang menghadirkan alternatif terhadap API berbasis RESTful.

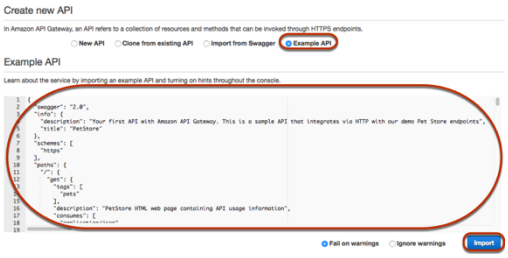
Dalam spirit bahasa query, GraphQL menggunakan skema untuk menggambarkan organisasi data dan bahasa query deklaratif untuk memungkinkan klien mengakses data yang dibutuhkan saja [5]. Permasalahan pada API agar mampu melayani permintaan data yang kompleks dan bisa menyesuaikan dengan berbagai UI/UX dari aplikasi multi-platform yang menggunakan API tanpa adanya overload data yang diterima dan kemampuan GraphQL dalam menangani permasalahan overload data, Maka dalam penelitian ini akan dikembangkan API berbasis web yang menerapkan arsitektur GraphQL untuk melayani aplikasi dari berbagai platform, dimana GraphQL diharapkan dapat mengatasi masalah fleksibilitas terkait data yang tersedia dan dibutuhkan UI/UX dari aplikasi, penggunaan latency dan bandwidth yang minimal sesuai kebutuhan masing-masing platform serta permasalahan dokumentasi dan versi dari API dengan memanfaatkan fitur-fitur yang disediakan pada arsitektur GraphQL.

TINJAUAN PUSTAKA

Application Programming Interface (API)

API adalah sebuah spesifikasi yang dimaksudkan untuk digunakan sebagai antarmuka dengan komponen perangkat lunak untuk berkomunikasi satu sama lain. API mungkin termasuk spesifikasi untuk rutinitas, struktur data, objek kelas, dan variable [6]. Spesifikasi API dapat mengambil banyak bentuk, termasuk standar internasional seperti Dokumentasi POSIX atau vendor seperti

Microsoft Windows API, atau *Library* bahasa pemrograman, misalnya standar *Template Library* di C ++ atau Java API.



Gambar 1. Contoh API (amazon.com, 2019)

GraphQL

GraphQL adalah bahasa query open source untuk API yang dikembangkan oleh Facebook Inc. *GraphQL* mengeksekusi *query server-side* dan hanya mengembalikan data yang didefinisikan oleh sistem tipe *Web Service* yang sesuai. *Variabel* dan *field* yang tersedia untuk *query* didefinisikan dalam skema yang disebut dengan sisi *server* (*GraphQL*). *Query* khusus dapat dibuat berdasarkan layanan *GraphQL* yang tersedia yang telah menentukan data yang tersedia untuk *query* yang memungkinkan satu titik akhir daripada beberapa titik akhir. Dengan mendeklarasikan bidang kita dapat misalnya hanya mengambil nama dan data dari artikel alih-alih menerima segala sesuatu yang berhubungan dengan artikel dan menyortir data dan hanya menampilkan bidang yang dibutuhkan, dimana query *GraphQL* meminta yang spesifik. orang dan bidang tertentu yang berhubungan dengan orang tersebut. Dengan menghilangkan jumlah *query* dan jumlah data yang ditransfer, kecepatan transfer data berpotensi ditingkatkan melalui beragam koneksi jaringan yang berbeda yang sedang digunakan saat ini [7].

```
schema {
  query: Query
  mutation: Mutation
}
type Mutation {
  createOffice(input: OfficeInput!): Office
}
type Query {
  company(id: ID!): Company
}
type Company {
  id: ID!
  name: String
  address: String
  age: Int @deprecated(reason: "No longer relevant.")
  offices(limit: Int!, after: ID): OfficeConnection
}
```

```
type OfficeConnection {
  totalCount: Int
  nodes: [Office]
  edges: [OfficeEdge]
}
type OfficeEdge {
  node: Office
  cursor: ID
}
type Office {
  id: ID!
  name: String
}
input OfficeInput {
  name: String!
}
```

Gambar 2. Contoh skema GraphQL dalam SDL

Penyedia GraphQL dapat menentukan skema baik secara programatik menggunakan library seperti Graphqljs, atau mereka dapat menentukan secara deklaratif menggunakan *Schema Definition Languange* (SDL). Gambar 2 menunjukkan skema contoh yang ditetapkan dalam SDL.

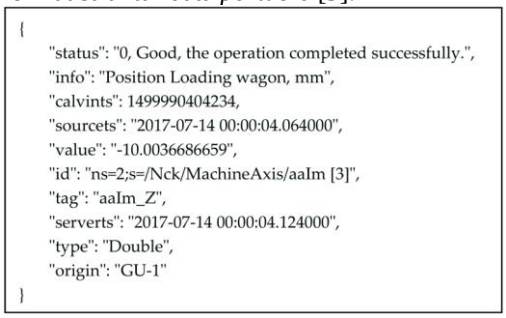
RESTful API

RESTful API didasarkan pada teknologi state transfer (representational state transfer / REST), gaya arsitektur dan pendekatan komunikasi yang sering digunakan dalam pengembangan layanan web.

Meskipun REST dapat digunakan di hampir semua protokol, tapi biasanya memanfaatkan HTTP ketika digunakan untuk Web API. Hal ini membantu pengembang web tidak perlu menginstal library atau perangkat lunak tambahan untuk memanfaatkan desain REST API. Design REST API pertama kali diperkenalkan oleh Dr. Roy Fielding dalam disertasi doktor tahun 2000-nya. REST API terkenal karena fleksibilitasnya yang luar biasa. Data tidak terikat dengan metode dan sumber daya, REST memiliki kemampuan untuk menangani beberapa jenis panggilan, mengembalikan format data yang berbeda dan bahkan mengubah secara struktural tentunya dengan implementasi yang benar. [8]

***JavaScript Object Notation* (JSON)**

JavaScript Object Notation (JSON) adalah format pertukaran data, berbasis teks, dan bersifat independen. Berasal dari Standar bahasa pemrograman ECMAScript. JSON mendefinisikan set aturan untuk representasi format struktur data *portable* [9].

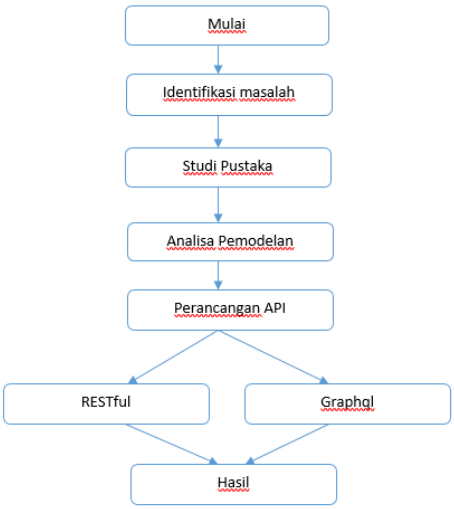


```
{
    "status": "0, Good, the operation completed successfully.",
    "info": "Position Loading wagon, mm",
    "calvints": 1499990404234,
    "sourcets": "2017-07-14 00:00:04.064000",
    "value": "-10.0036686659",
    "id": "ns=2;s=/Nck/MachineAxis/aaIm [3]",
    "tag": "aaIm_Z",
    "serverts": "2017-07-14 00:00:04.124000",
    "type": "Double",
    "origin": "GU-1"
}
```

Gambar 3. contoh JSON

METODE PENELITIAN

Pelaksanaan Perancangan Sistem

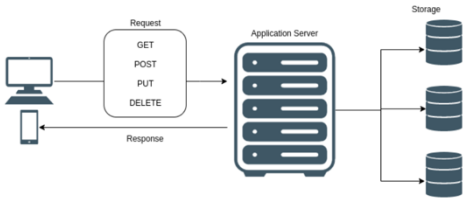


Gambar 4. Alur Proses Penelitian dengan Arsitektur RESTful dan GraphQL

Penelitian ini membandingkan performa sistem dengan menguji dua buah API yaitu, RESTful dan Graphql. Dalam pengujian ini dibuatkan sistem sederhana untuk menguji kinerja dari masing-masing API. Dari hasil pengujian akan diperoleh perbandingan performa dari RESTful dan Graphql.

RESTful

Paradigma desain arsitektur REST berpusat tentang menetapkan hubungan antara metode permintaan HTTP (GET, POST, PUT, PATCH, DELETE) dan titik akhir URL. Dalam arsitektur REST, setiap kombinasi metode dan titik akhir mengevaluasi ke fungsionalitas yang berbeda dan terenkapsulasi. Jika klien membutuhkan data yang tidak disediakan oleh endpoint / metode tertentu, permintaan tambahan mungkin diperlukan. Format data yang dikembalikan dari permintaan REST tergantung pada endpoint-tidak ada jaminan bahwa data ini akan diformat dengan cara yang dibutuhkan frontend untuk mengkonsumsinya. Untuk menggunakan data dari respons dalam format yang berbeda dari yang dikembalikan dari titik akhir secara default, penguraian dan manipulasi data harus ditulis sisi klien.

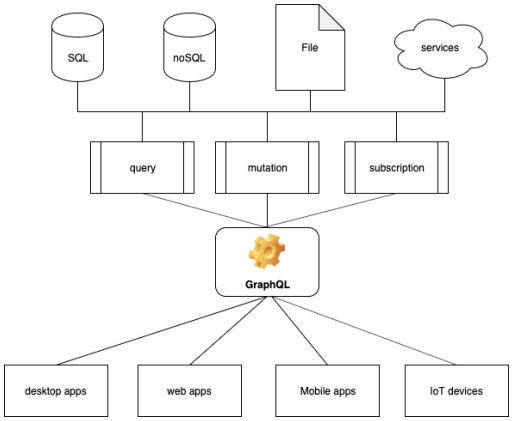


Gambar 5: *Pemodelan arsitektur RESTfull*

GraphQL

Arsitektur GraphQL ini digunakan untuk menghasilkan API berbasis web yang melayani apikasi multi-platform dengan 1 endpoint dan mengakomodasi berbagai permintaan data dari klien.

Dalam arsitektur GraphQL, single endpoint dapat mengakomodasikan permintaan berupa query untuk mengambil data, mutation untuk melakukan perubahan terhadap data dan subscription untuk streaming live data dari beragam sumber diantaranya data SQL, noSQL, file dan services lainnya untuk melayani kebutuhan dari beragam platform (desktop, web, mobile dan IoT devices) dengan ketersediaan data yang disesuaikan dari masing-masing platform.



Gambar 6. *Pemodelan arsitektur GraphQL*

Dari perbandingan arsitektur diatas, dapat dilihat beberapa perbedaan dalam proses pengambilan data. Masing-masing mempunyai kelebihan dan kekurangan dalam implementasinya. Dalam penelitian ini untuk

mengetahui perbandingannya maka akan dilakukan analisa dengan cara menguji : (1) Kompleksitas Implementasi, (2) Round Trip Time (RTT), (3) Bandwidth, (4) Fleksibilitas dan (5) Dokumentasi.

Kompleksitas Implementasi

Pengujian ini dimaksudkan untuk mengetahui seberapa kompleks dan efisienkah sebuah API, hasilnya nanti akan diketahui seberapa rumit dan kompleks kah algoritma yang ada didalam API tersebut. Pengujian ini dilakukan dengan melihat langsung pada barisan code yang digunakan dalam API tersebut.

Round Trip Time (RTT)

Pengujian ini akan dilakukan untuk mengetahui jumlah waktu yang diperlukan untuk mengirimkan sebuah data. Kedua API akan diuji untuk mengetahui masing-masing waktu yang dibutuhkan. Pengujian ini akan dilakukan dengan menggunakan tools yang ada di google chrome yang bernama Chrome DevTools.

Bandwidth

Pengujian ini akan dilakukan untuk mengetahui seberapa besar bandwith yang digunakan saat menjalan sistem. Pengujian ini akan dilakukan dengan menggunakan tools yang ada di google chrome yang bernama Chrome DevTools. Besarnya penggunaan bandwith dari kedua API akan terlihat dari tools tersebut.

Fleksibilitas

Pengujian ini akan dilakukan untuk mengetahui seberapa fleksibel sebuah API dapat digunakan saat mengakses suatu data. Hal ini dapat dilihat dari queri yang digunakan saat akan memanggil data, apakah bisa sederhana dengan langsung memanggil data yang dibutuhkan atau harus dilakukan pengecekan keseluruh data sebelum mendapatkan data yang akan dicari. Pengujian ini dapat dilakukan langsung dengan melihat kueri yang ada pada masing-masing API.

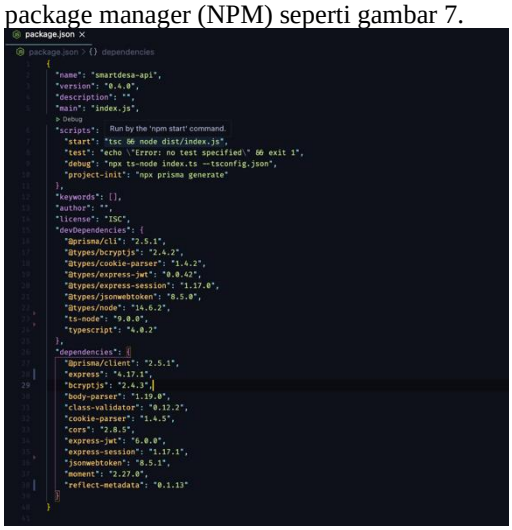
Dokumentasi

Pengujian ini akan dilakukan untuk mengetahui dokumentasi ketika ada perubahan dari sebuah kueri, dengan adanya dokumentasi, client atau front end developer akan mengetahui langkah apa yang harus dilakukan ketika adanya perubahan dalam suatu queri.

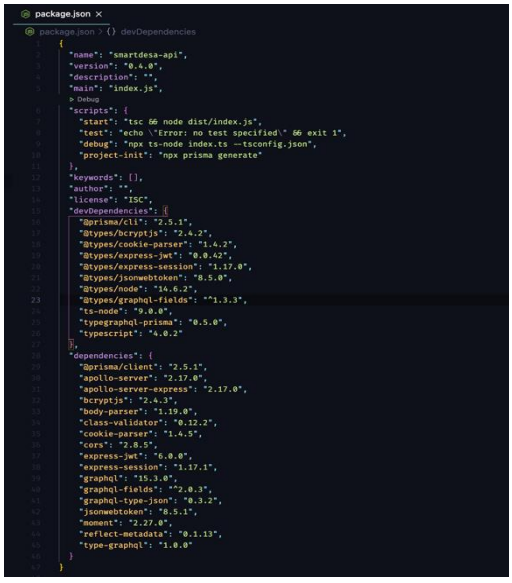
HASIL PELAKSANAAN PENELITIAN

Rancang Bangun Application Programming Interface

Pada penelitian ini API dibangun dengan 2 metode yaitu RESTful dan GraphQL dengan menggunakan bahasa pemrograman yang sama yaitu javascript dan pada runtime yang sama yaitu nodeJS (v8 javascript engine). RESTful API dibangun menggunakan library utama seperti Express dan Prisma ORM pada node package manager (NPM) seperti gambar 7.



Gambar 7 Library yang digunakan pada RESTful API



Gambar 8. Library yang digunakan pada RESTful API denganTypeGraphql

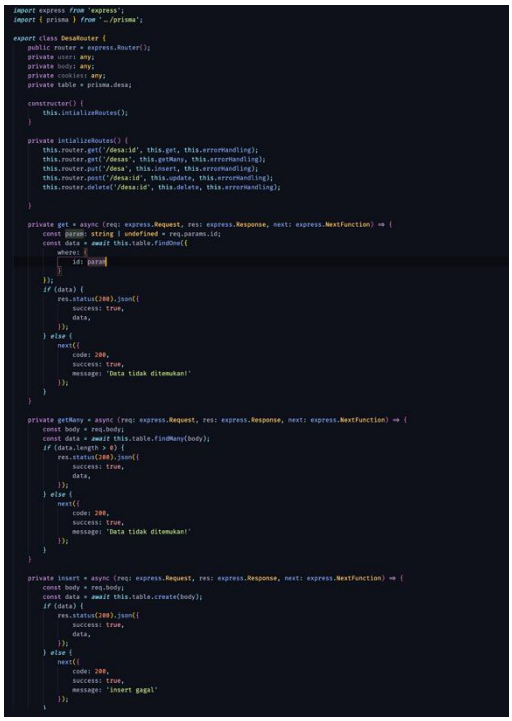
GraphQL API dibangun menggunakan library utama yang hampir sama dengan library pada RESTful API dengan tambahan TypeGraphql dan Apollo Server untuk mendukung GraphQL API seperti gambar 8..
Untuk dapat menguji dan membandingkan kemampuan kedua API tersebut dirancang sebuah database dengan 29 tabel dan saling berelasi menggunakan mysql engine.

Kompleksitas Implementasi

Pada RESTful API implementasinya bisa dibuat sesederhana mungkin, pada penelitian ini endpoint dari RESTful hanya berdasarkan http method (GET, PUT, POST, DELETE) pada masing-masing nama tabel dan beberapa tambahan seperti login, logout dan beberapa endpoint yang perlu proses khusus, hal ini menjadi keunggulan dari RESTful API dikarenakan pada RESTful API tidak perlu mendeskripsikan kembali struktur yang ada pada database. Sedangkan pada GraphQL API implementasi menjadi sangat kompleks dikarenakan GraphQL API menggunakan skema untuk memodelkan struktur database mulai dari table, field dan tipe datanya, resolver dari proses query, mutation dan subscription dimana hal ini menjadi kelemahan dari GraphQL API karena harus mengulang pemodelan struktur data pada kode program. pada gambar berikut terlihat contoh sederhana *file* yang digunakan dalam pengembangan RESTful API beserta contoh kode yang digunakan dari 1 tabel.

Gambar 9. *File router* dari RESTful API

***File router* dari RESTful API**



```
import express from 'express';
import { prisma } from '../prisma';

export class DesaRouter {
  public router = express.Router();
  private user: any;
  private body: any;
  private cookies: any;
  private table = prisma.desa;

  constructor() {
    this.intializeRoutes();
  }

  private intializeRoutes() {
    this.router.get('/desa:id', this.get, this.errorHandling);
    this.router.get('/desas', this.getMany, this.errorHandling);
    this.router.put('/desa', this.insert, this.errorHandling);
    this.router.post('/desa:id', this.update, this.errorHandling);
    this.router.delete('/desa:id', this.delete, this.errorHandling);
  }

  private get = async (req: express.Request, res: express.Response, next: express.NextFunction) => {
    const param: string | undefined = req.params.id;
    const data = await this.table.findOne({
      where: {
        id: parm
      }
    });
    if (data) {
      res.status(200).json({
        success: true,
        data,
      });
    } else {
      next({
        code: 200,
        success: true,
        message: 'Data tidak ditemukan!'
      });
    }
  }

  private getMany = async (req: express.Request, res: express.Response, next: express.NextFunction) => {
    const body = req.body;
    const data = await this.table.findMany(body);
    if (data.length > 0) {
      res.status(200).json({
        success: true,
        data,
      });
    } else {
      next({
        code: 200,
        success: true,
        message: 'Data tidak ditemukan!'
      });
    }
  }

  private insert = async (req: express.Request, res: express.Response, next: express.NextFunction) => {
    const body = req.body;
    const data = await this.table.create(body);
    if (data) {
      res.status(200).json({
        success: true,
        data,
      });
    } else {
      next({
        code: 200,
        success: true,
        message: 'insert gagal'
      });
    }
```

Gambar 10. Kode dari 1 tabel yang melayani RESTful API pada tabel desa

```
  updatedAt: DateTime
  createdBy: Int
  updatedBy: Int
  DesaDestionation: DesaUpdateOneWithoutWargaDatangInput
  User: UserUpdateOneWithoutWargaPindahInput
}

input WargaPindahUpdateWithoutUserDataInput {
  tanggal: DateTime
  createdAt: DateTime
  updatedAt: DateTime
  createdBy: Int
  updatedBy: Int
  DesaDestionation: DesaUpdateOneWithoutWargaDatangInput
  DesaOrigin: DesaUpdateOneWithoutWargaKeluarInput
}

input WargaPindahUpdateWithWhereUniqueWithoutDesaDestionationInput {
  where: WargaPindahWhereUniqueInput!
  data: WargaPindahUpdateWithoutDesaDestionationDataInput!
}

input WargaPindahUpdateWithWhereUniqueWithoutDesaOriginInput {
  where: WargaPindahWhereUniqueInput!
  data: WargaPindahUpdateWithoutDesaOriginDataInput!
}

input WargaPindahUpdateWithWhereUniqueWithoutUserInput {
  where: WargaPindahWhereUniqueInput!
  data: WargaPindahUpdateWithoutUserDataInput!
}

input WargaPindahUpsertWithWhereUniqueWithoutDesaDestionationInput {
  where: WargaPindahWhereUniqueInput!
  update: WargaPindahUpdateWithoutDesaDestionationDataInput!
  create: WargaPindahCreateWithoutDesaDestionationInput!
}

input WargaPindahUpsertWithWhereUniqueWithoutDesaOriginInput {
  where: WargaPindahWhereUniqueInput!
  update: WargaPindahUpdateWithoutDesaOriginDataInput!
  create: WargaPindahCreateWithoutDesaOriginInput!
}

input WargaPindahUpsertWithWhereUniqueWithoutUserInput {
  where: WargaPindahWhereUniqueInput!
  update: WargaPindahUpdateWithoutUserDataInput!
  create: WargaPindahCreateWithoutUserInput!
}

input WargaPindahWhereInput {
  AND: [WargaPindahWhereInput!]
  OR: [WargaPindahWhereInput!]
  NOT: [WargaPindahWhereInput!]
  id: IntFilter
  userId: IntNullableFilter
  origin: StringNullableFilter
  destination: StringNullableFilter
  tanggal: DateTimeNullableFilter
  createdAt: DateTimeNullableFilter
  updatedAt: DateTimeNullableFilter
  createdBy: IntNullableFilter
  updatedBy: IntNullableFilter
  DesaDestionation: DesaWhereInput
  DesaOrigin: DesaWhereInput
  User: UserWhereInput
}

input WargaPindahWhereUniqueInput {
  id: Int
  userId: Int
  origin: String
  destination: String
}
```

Gambar 11. contoh schema dari graphql

Pada GraphQL API schema harus dibuat terlebih dahulu untuk bisa menyediakan layanan API, schema yang dibuat tidak hanya berdasarkan tabel dan field dari tabel namun juga beserta tipe data dari masing-masing field. pada gambar berikut dapat dilihat contoh schema dari graphql berdasarkan database dari penelitian ini yang berjumlah 29 tabel dan pada schema graphql bisa mencapai 4019 baris kode yang dibuat.

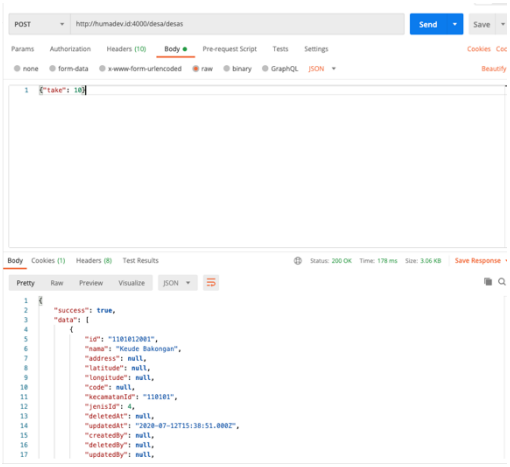
Ditambah dengan resolver untuk masing-masing *type* dari schema, resolver ini digunakan untuk melayani permintaan API dari client. pada 1 tabel rata-rata memiliki 10 resolver, sehingga file dan kode yang harus dibuat cukup besar untuk database yang sederhana. Berikut adalah contoh resolver dari tabel agama.

Gambar 12. contoh resolver dari tabel agama.

Round Trip Time (RTT)

Pada penelitian ini menguji coba 1 panggilan dengan permintaan data yang sama pada kedua API untuk menguji RTT dari masing-masing API. Pada RESTful API RTT yang dibutuhkan hampir sama dengan GraphQL API, hal ini sangat bergantung dari bentuk data yang dibutuhkan oleh aplikasi dan perangkat. Selain itu faktor penting seperti konten situs web (jumlah elemen / permintaan DOM dan ukurannya), lokasi server, media transmisi, data kecepatan transfer, jumlah node perantara, kepadatan lalu lintas, prioritas *event* / permintaan dan penundaan pemrosesan [10]. Berikut adalah gambar hasil dari pengujian.

Gambar 13. Hasil pengujian dari 1 panggilan API pada GraphQL API

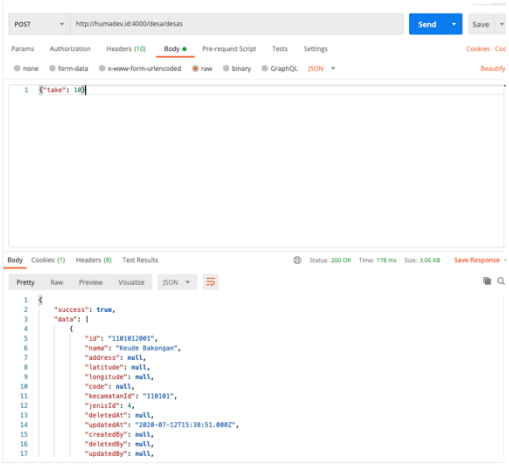


Gambar 14. Hasil pengujian dari 1 panggilan API pada GraphQL API

Bandwidth

Pada penelitian ini pengujian yang dilakukan pada bandwidth dilakukan pada data yang sama dengan pengujian pada round *trip time* dan didapatkan hasil pengujian pada *bandwidth* dari kedua API dengan hasil yang sama terhadap pengujian RTT yaitu kebutuhan terhadap bandwidth hampir sama diantara kedua metode, hal ini sangat bergantung dari bentuk data yang dibutuhkan oleh aplikasi dan perangkat. dan pada kedua metode bisa ditambahkan fitur caching yang mampu mengurangi bandwidth pada data yang sering diakses oleh perangkat dan disimpan sementara waktu pada perangkat. Berikut adalah hasil pengujian bandwidth terhadap kedua API.

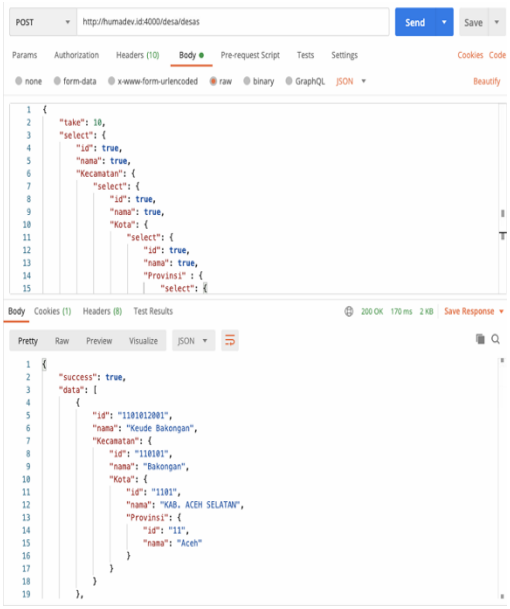
Gambar 15. Hasil pengujian bandwidth pada GraphQL API



Gambar 16. Hasil pengujian bandwidth pada GraphQL API

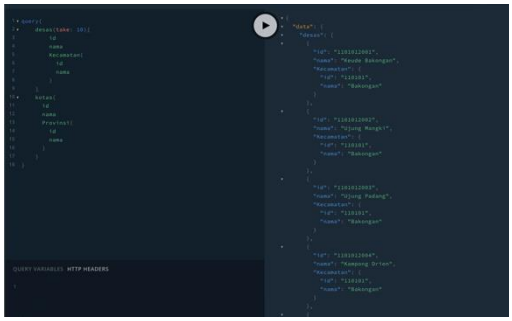
Fleksibilitas

Tujuan utama dikembangkannya GraphQL API adalah membuat panggilan API lebih fleksibel terutama pada data yang berelasi, pada RESTful API untuk memanggil data yang berelasi harus dengan GET/ user terlebih dahulu dan kemudian memanggil setiap relasinya secara individual melalui GET/user/:id/relation/:id endpoint, ini dapat menghasilkan kueri N + 1 dan merupakan kinerja yang diketahui akan menimbulkan masalah dalam API dan kueri database. Dengan kata lain, panggilan RESTful API dirangkai secara berurutan pada aplikasi sebelum representasi akhir dapat dibentuk untuk ditampilkan. GraphQL dapat mengurangi ini dengan mengaktifkan web server untuk mengumpulkan data untuk klien dalam satu kueri dengan kata lain, panggilan GraphQL API dirangkai secara berurutan pada sisi server sehingga mengurangi panggilan dari endpoint. Pada penelitian ini juga menguji coba kemampuan RESTful API untuk memanggil data yang berelasi dengan bantuan library prisma ORM kemampuan RESTful API untuk memanggil data yang berelasi menjadi hampir sama dengan GraphQL namun dalam hal memanggil endpoint dengan banyak entitas masih belum bisa dilakukan menggunakan RESTful API dan itu yang menjadi keunggulan dari GraphQL API. Berikut adalah contoh pemanggilan API menggunakan RESTful dan GraphQL API.



Gambar 17. Hasil panggilan API dengan GraphQL pada satu tabel dan relasinya

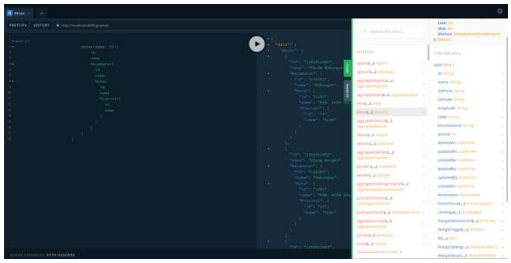
Gambar 18. Hasil panggilan API dengan GraphQL pada satu tabel dan relasinya



Gambar 19. Hasil panggilan API dengan GraphQL pada 2 tabel dan relasinya

Dokumentasi

Mempertahankan beberapa versi yang disediakan API dan memperluas fleksibilitas untuk klien API membutuhkan lebih banyak upaya dari pengembang API. Alhasil, tak semuanya versi lama bisa dipertahankan. Ketika klien API tertinggal jauh dibelakang dari versi terbaru, bermigrasi ke versi terbaru akan menjadi lebih sulit [1]. Dari sisi pengembang juga akan kesulitan tidak hanya dalam mengelola versi API tapi juga dalam mendokumentasikannya, karena dari segala pengembangan dan kode yang ditulis, semua harus ditulis ulang lagi dalam bentuk dokumentasi sehingga pengembang klien yang menggunakan API bisa mengetahui akses apa saja yang bisa dilakukan terhadap API. Pada GraphQL API ini menjadi keunggulan utamanya. pembuatan skema graphql yang kompleks seperti pada uji coba kompleksitas implementasi menjadi keunggulan pada dokumentasi. Ini dikarenakan dari kode tersebut graphql bisa menyediakan playground untuk melihat detail segala bentuk akses yang bisa dilakukan pada API seperti terlihat pada gambar berikut.



Gambar 20. Playground dari Graphql

pada playground dari GraphQL bisa dilihat bukan hanya entitas yang bisa diakses tetapi termasuk tipe datanya hal ini menjadi kemudahan bagi pengembang dalam mengembangkan API, karena hanya berdasarkan kode dan skema yang dibuat akan langsung menjadi dokumentasinya.

SIMPULAN

GraphQL API bisa menjadi teknologi baru yang menarik, tetapi penting untuk memahami besarnya kompleksitas implementasi sebelum membuat keputusan menggunakan metode ini. Beberapa API seperti yang memiliki sangat sedikit entitas dan hubungan antar entitas seperti API analitik mungkin tidak cocok untuk GraphQL. Sedangkan aplikasi dengan banyak objek domain yang berbeda seperti e-commerce di mana databasenya memiliki item, pengguna, pesanan, pembayaran, dan sebagainya mungkin dapat lebih memanfaatkan GraphQL. Keunggulan utama dari RESTful adalah API bisa dibangun dengan sederhana maupun kompleks sekalipun sedangkan pada GraphQL keunggulan utamanya adalah fleksibilitas dan dokumentasi.

DAFTAR PUSTAKA

[1] Thanh-Diane N, Vanderdonckt J, Seffah A. (2016). UIPLML: Pattern-based engineering of user interfaces of multi-platform systems. IEEE, Grenoble.

[2] Sohan S.M., Anslow C, Maurer F. (2015). A Case Study of Web Evolution. IEEE, New York.

[3] F. Belqasmi, J. Singh, S. Y. Bani Melhem and R. H. Glitho, "SOAP-Based vs. RESTful Web Services: A Case Study for Multimedia Conferencing," in *IEEE Internet Computing*, vol. 16, no. 4, pp. 54-63, July-Aug. 2012.

[4] Facebook,Inc.2016.GraphQL.WorkingDraft,Oct.2016.Onlineathttp://facebook. github.io/graphql, retrieved on Dec. 12, 2016. (Oct. 2016).

[5] Hartig O, Perez J. (2018). Semantics and Complexity of GraphQL. ACMDL, Lyon.

[6] Bloch, J. (2008). *Effective java (the java series)*. Prentice Hall PTR Upper Saddle River, NJ, USA.

[7] Helgason, Arnar Freyr. (2017).*Performance analysis of Web Services*: Comparison between RESTful & GraphQL web services.

[8] ___ Rest API Tutorial, Available at https://restfulapi.net/, diakses tanggal 7 Oktober 2019

[9] Bray, T. (2014). The JavaScript Object Notation (JSON) Data Interchange Format.

[10] N. Naik, P. Jenkins, P. Davies and D. Newell, "Native Web Communication Protocols and Their Effects on the Performance of Web Services and Systems," 2016 IEEE International Conference on Computer and Information Technology (CIT), Nadi, 2016, pp. 219-225, doi: 10.1109/CIT.2016.100