SKRIPSI

ANALISIS KINERJA REST API DAN GRAPHQL PADA TEKNOLOGI WEB *SERVICES*

Disusun dan diajukan oleh:

DARUL IKHSAN

D121181017

DEPARTEMEN TEKNIK INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS HASANUDDIN

MAKASSAR

2022

LEMBAR PENGESAHAN SKRIPSI

ANALISIS KINERJA REST API DAN GRAPHQL PADA TEKNOLOGI WEB SERVICES

Disusun dan diajukan oleh

DARUL IKHSAN

D121181017

Telah dipertahankan di hadapan Panitia Ujian yang dibentuk dalam rangka Penyelesaian Studi Program Sarjana Program Studi Teknik Informatika Fakultas Teknik Universitas Hasanuddin pada tanggal 30 November 2022 dan dinyatakan telah memenuhi syarat kelulusan.

Menyetujui,

Pembimbing Utama,	Pembimbing Pendamping,
Dr. Eng. Ir. Muhammad Niswar, S.T., M.InfoTech.	Iqra' Aswad, S.T., M.T.
Nip. 197309221999031001	Nip. 199011282019043001

Ketua Program Studi,

Dr. Indrabayu, S.T., M.T., M.Bus.Sys.
Nip. 19750716 200212 1 004

PERNYATAAN KEASLIAN

Yang bertanda tangan di bawah ini:

Nama : Darul Ikhsan

NIM : D121181017

Departemen : Teknik Informatika

Jenjang : S1

Menyatakan dengan ini karya tulisan saya berjudul:

ANALISIS KINERJA REST API DAN GRAPHQL PADA TEKNOLOGI *WEB SERVICES*

Adalah karya tulisan saya sendiri dan bukan merupakan pengambilalihan tulisan orang lain bahwa skripsi yang saya tulis ini benar-benar merupakan hasil karya saya sendiri.

Apabila di kemudian hari terbukti atau dapat dibuktikan bahwa sebagian atau keseluruhan skripsi ini hasil karya orang lain, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Makassar, 30 November 2022

Yang menyatakan,

Darul Ikhsan

KATA PENGANTAR

Segala puji dan syukur penulis panjatkan kehadirat Tuhan Yang Maha Pengasih lagi Maha Penyayang atas limpahan kasih dan karunia-Nya sehingga tugas akhir dengan judul **"ANALISIS KINERJA REST API DAN GRAPHQL PADA TEKNOLOGI WEB *SERVICES*"** dapat diselesaikan dengan baik. Penyusunan tugas akhir ini menjadi salah satu syarat dalam menyelesaikan jenjang strata-1 di Departemen Teknik Informatika Universitas Hasanuddin.

Penulisan tugas akhir ini tidak lepas dari doa, bimbingan, dan bantuan dari berbagai pihak. Pada kesempatan ini disampaikan terima kasih kepada semua pihak yang telah membantu dalam pembuatan tulisan ini, ucapan terima kasih yang sebesar-besarnya kepada:

1. Kedua orang tua penulis, Bapak Syamsir dan Ibu Hasniar yang atas kasih sayang, doa, didikan, dan semangat yang tak pernah putus dari mereka telah menuntun perjalan hidup penulis;
2. Bapak Dr. Eng. Ir. Muhammad Niswar, S.T., M.InfoTech. selaku pembimbing utama yang telah menyempatkan waktu, tenaga, dan pikiran untuk memberikan arahan dan masukan kepada penulis selama proses penyusunan tugas akhir;
3. Bapak Iqra Aswad, S.T., M.T., selaku pembimbing pendamping yang telah menyempatkan waktu, tenaga, dan pikiran untuk memberikan arahan dan masukan kepada penulis selama proses penyusunan tugas akhir;

4. Bapak A. Ais Prayogi Alimuddin, S.T., M.Eng. selaku pembimbing akademik yang telah banyak memberikan masukan, saran, dan bantuan kepada penulis selama perkuliahan hingga proses penyusunan tugas akhir;
5. Bapak Dr. Ir. Indrabayu, M.T., M.Bus.Sys., IPM. selaku ketua departemen Teknik Informatika Universitas Hasanuddin atas bimbingannya selama masa perkuliahan;
6. Bapak dan ibu dosen di Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin atas didikan dan arahannya selama masa perkuliahan;
7. Segenap Staf Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin yang telah banyak membantu selama proses perkuliahan dan hingga pada penyelesaian tugas akhir;
8. Semua Orang yang telah membantu dan mendukung penulis namun tidak sempat disebutkan.

Penulis menyadari bahwa tugas akhir ini mungkin masih terdapat kekurangan, oleh karena itu penulis berharap saran serta masukan yang membangun dari semua pihak. Semoga tugas akhir ini dapat memberikan manfaat bagi penulis maupun kepentingan bersama

Gowa, 30 November 2022

Penulis

ABSTRAK

Pertumbuhan kebutuhan untuk saling bertukar informasi mengakibatkan sistem-sistem saat ini dituntut untuk dapat saling terintegrasi satu sama lain. Web *Services* merupakan sistem yang dirancang untuk mendukung *software integration*, komunikasi dan pertukaran data antara perangkat lunak yang dilakukan secara *remote* dengan mekanisme komunikasi tertentu. Dua dari mekanisme tersebut adalah REST API dan GraphQL. REST API dan GraphQL merupakan metode yang mendukung komunikasi, pertukaran data, dan integrasi perangkat lunak secara *remote*. Oleh karena itu, pada penelitian ini dilakukan pengujian dan analisis terhadap kinerja Web *Services* yang dikembangkan menggunakan metode tersebut. Web *Services* yang dikembangkan adalah sistem yang mengadopsi fungsi-fungsi dari sistem SISTER Web *Services* PT 1.0.0. Web *Services* ini dibangun menggunakan *framework* Laravel 9 dengan *database* MySQL. Web *Services* ini terbagi menjadi dua *Services* dengan *database* masing-masing yang dibuat terpisah dari *Services* itu sendiri. Dua *Services* utama masing-masing menggunakan *instance AWS EC2* dan dua *Services database* masing-masing menggunakan *instance Amazon RDS*. Pengujian sistem ini mengukur parameter *Response Time, Throughput, CPU Utilization* dan *Page Load Time*. Hasil pengujian menunjukkan bahwa REST memiliki kinerja lebih baik untuk data yang tidak *nested* akan tetapi GraphQL menjadi unggul ketika integrasi dengan *client* yang melakukan *fetch* data *nested*. Hal ini terjadi karena pada GraphQL tidak terjadi n+1 *request* dan data yang diperoleh sesuai dengan yang dibutuhkan sehingga tidak terjadi *over-fetching* dan/atau *under-fetching* data pada aplikasi *client* pada saat integrasi.

Kata kunci: *Software Integration*, Web *Services*, REST API, GraphQL

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR LAMPIRAN

BAB I

PENDAHULUAN

1.1 Latar Belakang

Web *Services* merupakan sebuah teknologi yang dirancang untuk mendukung s*oftware integration*, komunikasi dan pertukaran data antara perangkat lunak atau program-to-program yang saling berinteraksi menggunakan mekanisme tertentu secara *remote* atau jarak jauh (Shi et al., 2017). Untuk melakukan sebuah komunikasi diperlukan sebuah metode atau mekanisme yang mengatur komunikasi tersebut. Hal ini dimaksudkan agar kedua belah pihak bisa saling mengerti sehingga dapat bertukar data dan informasi melalui jaringan internet baik itu dari *server-server* maupun *client-server*. Saat ini REST API adalah metode komunikasi yang populer digunakan oleh *developer* untuk mengembangkan teknologi web *services*.

Istilah REST diperkenalkan pada tahun 2000 dalam disertasi doktoral Roy Fielding salah satu penulis utama spesifikasi *Hypertext Transfer Protocol* (HTTP). REST atau *Representational State Transfer* adalah salah satu bentuk atau gaya arsitektur untuk mengembangkan Web *Services* yang memungkinkan aplikasi *client* dapat mengakses data dan fungsional yang telah didefinisikan oleh *Services* (Massé, n.d.). Selama sepuluh tahun terakhir, REST menjadi sangat populer dan telah menjadi semacam standar untuk merancang Web *Services*. REST menggunakan *Javascript Object Notation* atau *json* sebagai format pertukaran datanya. *Method* HTTP yang sering digunakan pada arsitektur REST yaitu, *GET, POST, PUT, PATCH,* dan *DELETE*. REST banyak digunakan oleh *programmer*

untuk mengembangkan sebuah Web *Services* karena kecepatan dan lebih ringan, serta mudah dalam pengembangannya (Putra, 2019). Setiap *resources* pada *service* REST di wakili oleh *endpoint*. Cukup dengan melakukan *request* ke e*ndpoint* menggunakan m*ethod* tertentu maka REST akan merespon dan menyajikan data dalam bentuk JSON dengan *resources* yang telah ditentukan oleh Web *Services*.

REST API tidak begitu fleksibel untuk mengikuti perubahan kebutuhan *client* yang begitu cepat. Contohnya pada kasus integrasi SISTER Web *Services* PT 1.0.0 dengan aplikasi Sinkron Sister yang digunakan oleh Universitas Hasanuddin. Dalam skenario ini, aplikasi c*lient* atau sinkron sister, membutuhkan data yang bersarang atau *nested*, menggunakan REST akan sedikit lebih rumit karena harus mengakses lebih dari satu e*ndpoint* dan melakukan *request* lebih dari sekali. Pada skenario data yang bersarang, menggunakan REST biasanya akan menimbulkan problem n+1 *request*. Hal ini terjadi karena saling ketergantungan antara data sehingga *client* membutuhkan *request* lain sebelum melakukan *request* yang sebenarnya. Pada REST juga kadang terjadi o*ver-fetching* dan *under-fetching*. Hal ini berarti bahwa e*ndpoint* tertentu memberikan data yang lebih dan atau kurang dari data yang dibutuhkan (GraphQL *is better than Rest*, n.d.). Hal ini kadang kala terjadi karena pada sisi penyedia layanan sulit untuk merancang API menyediakan data sesuai kebutuhan *client*, dan *client* tidak dapat menentukan data yang akan diterima.

Pada tahun 2012, perusahaan teknologi Facebook membuat GraphQL untuk kepentingan internal kemudian pada tahun 2015 Facebook merilis GraphQL secara publik. Facebook membuat GraphQL sebagai solusi dari masalah yang ditemukan

saat menggunakan REST (Maldonado, 2019). GraphQL adalah mekanisme untuk melakukan interaksi antara c*lient* dan *server* secara *remote* dengan mengirimkan *query* ke sebuah e*ndpoint* (*Introduction* to GraphQL, n.d.). Pada dasarnya GraphQL memberikan akses kepada *client* untuk melakukan *query* ke *database* yang direpresentasikan oleh s*chema* melalui sebuah e*ndpoint* (Brito et al., 2019). Dengan menggunakan GraphQL, *client* dapat menentukan sendiri data yang dibutuhkan melalui *query* yang dikirimkan ke sebuah e*ndpoint* sehingga tidak akan terjadi o*ver-fetching* dan u*nder-fetching*. Berkomunikasi menggunakan GraphQL hanya menggunakan satu e*ndpoint* saja, sehingga dapat mencegah kasus n+1 *request* pada kasus data yang bersarang.

Berdasarkan masalah di atas, penulis mengajukan penelitian untuk melakukan pengujian dan analisis perbandingan kinerja antara REST dan GraphQL pada Teknologi Web *Services*. Penelitian tersebut berjudul "Analisis Kinerja REST dan GraphQL pada Web *Services*". Pada penelitian ini, penulis akan mengembangkan sebuah Web *Services* menggunakan REST dan GraphQL yang akan diintegrasikan dengan aplikasi sinkron sister milik Universitas Hasanuddin. Kemudian dilakukan pengujian, pengukuran, dan analisis kinerja pada sistem tersebut menggunakan API c*lient* yaitu Apache Jmeter dan Chrome *DevTools*. Parameter yang diukur yaitu *Response Time*, *Throughput*, *CPU Utilization,* dan *Page Load Time*.

1.2 Rumusan Masalah

1. Bagaimana perbandingan kinerja Web *Services* yang di bangun menggunakan REST API dan GraphQL pada kasus *fetch* data tidak *nested* atau tidak bersarang?
2. Bagaimana perbadingan kinerja Web *Services* yang di bangun menggunakan REST API dan GraphQL pada kasus *upload* dokumen dengan *metadata*?
3. Bagaimana perbandingan kinerja Web *Services* yang di bangun menggunakan REST API dan GraphQL pada kasus *fetch* data *nested* ketika proses integrasi dengan aplikasi Sinkron Sister?

1.3 Tujuan Penelitian

1. Mengetahui perbandingan kinerja Web *Services* yang di bangun menggunakan REST API dan GraphQL pada kasus data tidak *nested* atau tidak bersarang.
2. Mengetahui perbandingan kinerja Web *Services* yang di bangun menggunakan REST API dan Graphql pada kasus *Upload* dokumen dengan *metadata*.
3. Mengetahui perbandingan kinerja Web *Services* yang di bangun menggunakan REST API dan GraphQL pada kasus *fetch* data *nested* atau bersarang ketika proses integrasi dengan aplikasi Sinkron Sister?

1.5 Batasan Masalah

1. Skenario pertama yang akan diuji yaitu pengukuran *fetch* data tidak *nested* atau tidak bersarang, diberikan c*oncurrent request* sebanyak 100, 120, 140, dan 160 dalam satu detik dan parameter yang diukur yaitu *response time*, *throughput*, dan *cpu utilization*. Setiap pengukuran akan dilakukan sebanyak 5 kali.
2. Kedua, skenario pengukuran *upload* dokumen *size 1MB* dengan *metadata* milik sdm, diberikan *concurrent request* sebanyak 100, 120, 140, dan 160 dalam satu detik dan parameter yang diukur yaitu *response time*, *throughput*, dan *cpu utilization*. Setiap pengukuran akan dilakukan sebanyak 5 kali.
3. Ketiga, skenario pengukuran *fetch* dan *upload* data tidak *nested* atau tidak bersarang, diberikan *request* sebanyak 100, 120, 140, dan 160 dalam waktu interval 5 menit setiap percobaan dan parameter yang diukur adalah *response time*.
4. Keempat, skenario pengukuran *fetch* data *nested* atau bersarang dengan cara mengintegrasikan sistem yang di bangun dengan aplikasi sinkron sister universitas hasanuddin. Parameter yang diukur adalah *page load time* pada *website* sinkron sister ketika melakukan *fetch* data *nested* dari sistem web *services* yang di bangun. Setiap pengukuran akan dilakukan sebanyak 5 kali.
5. Penelitian ini menggunakan studi kasus integrasi SISTER Web *Services* PT 1.0.0 dengan aplikasi Sinkron Sister yang digunakan oleh Universitas Hasanuddin.

6. GraphQL dan REST API akan dibangun pada satu aplikasi yang sama.
7. Sistem terdiri dari dua *service* dan terhubung ke *database* masing-masing secara *remote*.
8. Setiap *services* akan menggunakan *instance AWS EC2* dan *database* menggunakan *instance Amazon RDS*.
9. Virtual *machine* untuk *service* akan ditempatkan pada *public subnet* sedangkan virtual *machine* untuk *database* akan ditempatkan pada *private subnet*.
10. Sistem dibangun menggunakan Bahasa pemrograman PHP *framework* Laravel dan DBMS MySQL.
11. Pengukuran kinerja menggunakan Apache JMeter yang di install pada lokal komputer dan Chrome *DevTools*.

1.4 Manfaat Penelitian

1. Memberikan sumbangan kepada ilmu pengetahuan berupa data hasil analisis kinerja REST API dan GraphQL.
2. Hasil analisis dapat dijadikan sebagai referensi dalam proses pemilihan teknologi antara REST API dan/atau GraphQL dalam mengembangkan Web *Services*.

1.6 Sistematika Penulisan

Adapun sistematika penulisan penelitian adalah sebagai berikut:

BAB I PENDAHULUAN

Bab ini menguraikan latar belakang penelitian, rumusan masalah, tujuan dan manfaat penelitian, batasan masalah, serta sistematika penulisan.

BAB II TINJAUAN PUSTAKA

Bab ini membahas tentang landasan teori yang menunjang setiap proses pada penelitian yang dilakukan.

BAB III METODOLOGI PENELITIAN

Bab ini menjelaskan tentang tahap penelitian, alat dan bahan penelitian, gambaran sistem, implementasi sistem hingga skenario pengujian kinerja sistem yang dikembangkan.

BAB IV HASIL DAN PEMBAHASAN

Bab ini membahas secara menyeluruh hasil pengukuran dan analisis kinerja Web Services yang dikembangkan menggunakan REST dan GraphQL.

BAB V PENUTUP

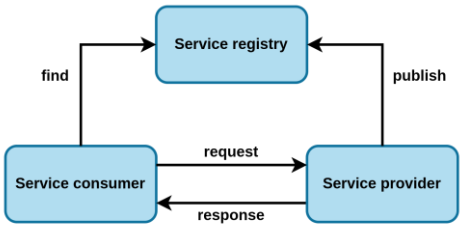
Bab ini berisi kesimpulan dari hasil penelitian dan saran-saran untuk pengembangan lebih lanjut.

BAB II

TINJAUAN PUSTAKA

2.1 Web *Services*

Web *Services* merupakan suatu *interface* yang menyediakan serangkaian fungsi yang dirancang untuk menghasilkan web transaksional yang mendukung s*oftware integration*, komunikasi dan pertukaran data antara perangkat lunak atau program-to-program yang saling berinteraksi menggunakan mekanisme tertentu secara *remote* atau jarak jauh yang di bangun berdasarkan standar-standar yang ada (Gottschalk, 2002).



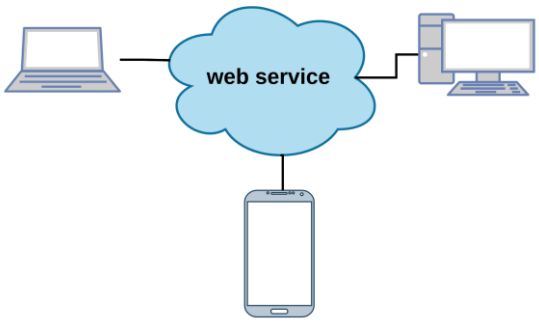
Gambar 2.1 Model Web *Services*

Model web *service* pada umumnya terdiri dari tiga entitas seperti gambar diatas yaitu *service provider*, *service registry*, dan *service consumer* (Dustdar & Schreiner, 2005).

1. *Service Provider* adalah entitas yang berfungsi untuk mengembangkan, menyediakan, dan menyajikan web *service*.
2. *Service registry* adalah entitas yang berisikan informasi-informasi tambahan dan detail teknik tentang layanan yang disajikan oleh *Services provider*.

3. Entitas yang terakhir adalah *Service consumer* atau yang biasa kita kenal dengan *client*. *Service consumer* mengambil informasi layananan dari *service registry* dan menggunakan informasi layanan yang diperoleh untuk mem-*binding* dan me-*request* layanan *Service provider*.

Teknologi Web *Services* tidak terikat pada satu bahasa pemrograman dan sistem operasi sehingga menciptakan lingkungan pengembangan yang permisif dan adaptif yang dapat mempermudah proses integrasi antar aplikasi. Dengan menggunakan web *service* meningkatkan *interoperable* yang memungkinkan adanya interaksi dan pertukaran informasi satu sama lain antar peranti dengan bahasa pemrograman dan sistem operasinya masing-masing melalui internet (Kumar Mungara, n.d.). Seperti gambar dibawah ini



Gambar 2. 2 Web *Services* tanpa terikat system operasi

2.2 SISTER Web *Services*

SISTER merupakan singkatan dari Sistem Informasi Sumber Daya Terintegrasi. Pada Tahun 2017 Direktorat Jenderal Sumber Daya Iptek Dikti meluncurkan aplikasi SISTER dengan tujuan untuk mendukung aktivitas kinerja dosen, menambah portofolio dan meningkatkan karir dosen pada perguruan tinggi

(Ragam Peran dan Tugas Pengguna di Aplikasi SISTER, n.d.). Aplikasi SISTER ini dapat diintegrasikan dengan aplikasi internal kampus menggunakan SISTER Web *Services*. SISTER Web *Services* berfungsi untuk memberikan kemudahan bagi universitas untuk mengakses data-data dari aplikasi SISTER. SISTER Web *Services* ini adalah sistem yang dapat digunakan oleh pengembang perangkat lunak perguruan tinggi untuk mengakses data terbaru pada SISTER (Sistem Informasi Sumber Daya Terintegrasi).

2.3 Sinkron Sister

Aplikasi Sinkron Sister merupakan aplikasi milik Universitas Hasanuddin yang berfungsi sebagai aplikasi penghubung aplikasi SISTER Web *Services* dengan aplikasi Dashboard Data Unhas. Aplikasi ini berfungsi untuk melakukan *mapping* data dan mengolah data dari SISTER Web *Services* sebelum digunakan oleh aplikasi Dashboard Data Unhas dan data yang telah diolah sesuai kebutuhan kemudian ditransmisikan atau dikirim ke aplikasi *Dashboard* Data Unhas.

2.4 PHP

PHP: Hypertext Preprocessor merupakan sebuah *scripting language* yang berjalan di *server-side* dan bersifat *open source*. Menggunakan PHP dapat menciptakan halaman *website* yang dinamis. PHP dibuat oleh Rasmus Lerdorf pada tahun 1995. Utamanya PHP dapat digunakan dalam tiga pendekatan yaitu *server-side scripting*, c*ommand-line scripting, client-side GUI Applications*:

1. *Server-side scripting*

   PHP pada mulanya didesain untuk menghasilkan konten *website* yang dinamis dan hingga kini masih cocok untuk tugas tersebut. Pada PHP terdapat PHP parser yang berfungsi untuk menghasilkan HTML. PHP Juga menjadi populer digunakan untuk menghasilkan dokumen XML, bagan, animasi, PDF, dan banyak lagi

2. *Command-line Scripting*

   PHP dapat menjalankan *script* dari *command line*, serupa Perl, awk, atau unix shell. anda mungkin menggunakan *script command-line* untuk tugas-tugas sistem *administor* seperti *backup* dan *log parsing*.

3. *Client*-side *GUI Applications*

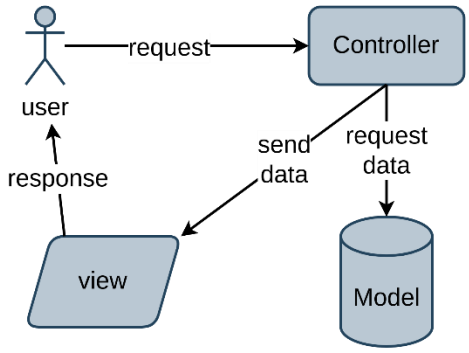
   Dengan menggunakan PHP-GTK, PHP dapat menulis aplikasi *GUI* lintas platform yang lengkap.

Bahasa pemrograman ini sangat fleksibel dan dapat berjalan pada semua sistem operasi, mulai dari Unix termasuk linux, FreeBSD, dan Solaris serta platform lain seperti Windows dan Mac OS X. Salah satu fitur PHP yang paling menonjol adalah dukungannya yang luas untuk *database*. PHP *supports* untuk semua *database* utama seperti MySQL, PostgreSQL, Oracle, dan ODBS - compliant *database*. Selain itu PHP juga memiliki banyak *library* yang dapat digunakan untuk melakukan tugas-tugas umum, misalnya *database abstraction*, *error handling*, dan sebagainya (Lerdorf et al., 2002).

2.5 Laravel

Laravel Adalah *framework* aplikasi yang dibangun menggunakan bahasa pemrograman PHP dengan sintaks yang ekspresif, elegan dan fungsi-fungsi yang terbaru serta segala kemudahan dalam menggunakannya tanpa perlu memikirkan hal-hal kecil (*The PHP Frameworks for Web Artisans*, n.d.). Laravel merupakan *framework* PHP yang paling banyak digunakan oleh pemula hingga profesional. Hal ini tidak lepas dari kemudahan yang ditawarkan oleh *framework* ini. Laravel menyediakan banyak modul-modul dan fitur-fitur canggih yang saling terkoneksi bersama-sama.

Laravel menggunakan arsitektur *MVC*. *MVC* adalah suatu *design pattern* yang mengambil penamaan berdasarkan tiga komponen utamanya yaitu *Model-View-Controller*.



Gambar 2. 3 Arsitektur *MVC* Laravel

Dalam proses pengembangan aplikasi komponen-komponen tersebut dirancang untuk menangani aspek *development* sesuai kebutuhan. *Controller* berfungsi sebagai perantara antara *view* dan *model*. *Controller* dapat menangani *business logic* yang mengatur jalannya aplikasi dan menangani data input dari *user*

serta memproses data tersebut. *Model* adalah komponen utama yang mewakili data yang akan ditransfer dari *controller* ke *view*. *Model* berisikan logika pemrograman yang berkaitan dengan koneksi ke *database*. *View* adalah komponen yang berkaitan dengan *user interface* yang akan menerima data dari *controller* (Subecz, 2021).

2.6 REST

REST adalah akronim dari *Representational State Transfer* yang merupakan sebuah *architectural style* untuk sistem terdistribusi yang menyediakan komunikasi antara *client* dan *server* menggunakan protokol HTTP (Prayogi et al., 2020). REST api sangat mudah digunakan dan bisa beradaptasi di semua bahasa pemrograman.

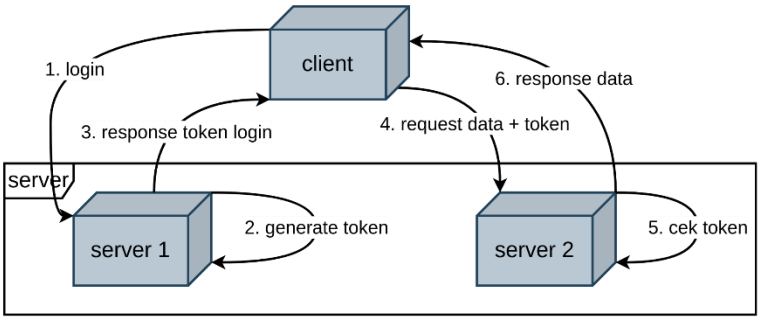
Sama halnya dengan *architectural style* yang lainnya, REST memiliki *design principles* dan *constraints*. Prinsip-prinsip ini harus dipenuhi jika *interface* layanan ingin disebut sebagai RESTful (Gupta, 2022). Prinsip-prinsip tersebut antara lain sebagai berikut.

1. *Client* Server

   Prinsip yang pertama adalah c*lient server*. Yaitu prinsip yang memisahkan kompleksitas data internal *server* dengan data yang akan di ekspos ke *client*. Hal ini demi dapat mendukung komponen *client* dan *server* berkembang secara independen sehingga meningkatkan portabilitas *user interface* diberbagai platform dan meningkatkan skalabilitas dengan dengan menyederhanakan komponen *server*. Dengan constraint tetap mempertahankan *interface*/kontrak antara *client* dan *server* tidak rusak.

2. *Stateless*

Prinsip kedua yaitu *stateless*, sederhanya adalah tidak menyimpan *state* atau data antar interaksi artinya tiap interaksi harus independen dan tidak tergantung pada interaksi sebelum atau setelahnya sehingga setiap interaksi ini *client* harus mengirim seluruh informasi yang dibutuhkan oleh *server* untuk menyelesaikan interaksi (Khannedy, 2021). Oleh karena itu *client* lah yang bertanggung jawab menyimpan dan melakukan manajemen *state*. Seperti pada gambar dibawah ini



Gambar 2. 4 Prinsip *Stateless* RESTful API

3. *Cacheable*

Prinsip ketiga ini dapat diartikan c*lient* dapat menyimpan data di lokal seperti di web *browser* atau *mobile app* secara sementara sehingga tidak perlu melakukan *request* untuk data yang telah disimpan sebelumnya, hanya perlu menggunakan data yang disimpan. Hal ini dapat dilakukan jika r*esponse* yang dikembalikan oleh *server* secara implisit atau eksplisit memberi label sendiri sebagai *cacheable*.

4. *Uniform Interface*

   Prinsip selanjutnya adalah *Uniform Interface* artinya prinsip yang mengamanatkan untuk menggunakan *interface* komunikasi yang seragam untuk semua pihak baik itu *client* atau *server* yang lain. Selain itu data yang dikembalikan harus memiliki standar yang *general*.

5. *Layered System*

   Layered System merupakan prinsip desain yang memungkinkan arsitektur sistem terdiri dari lapisan-lapisan hirarkis dengan membatasi perilaku dari setiap komponen tersebut. Misalnya dengan menerapkan *encapsulation* pada tiap-tiap komponen sehingga tidak dapat melihat melihat diluar lapisan langsung yang berinteraksi dengannya.

6. *Code on Demand* (Opsional)

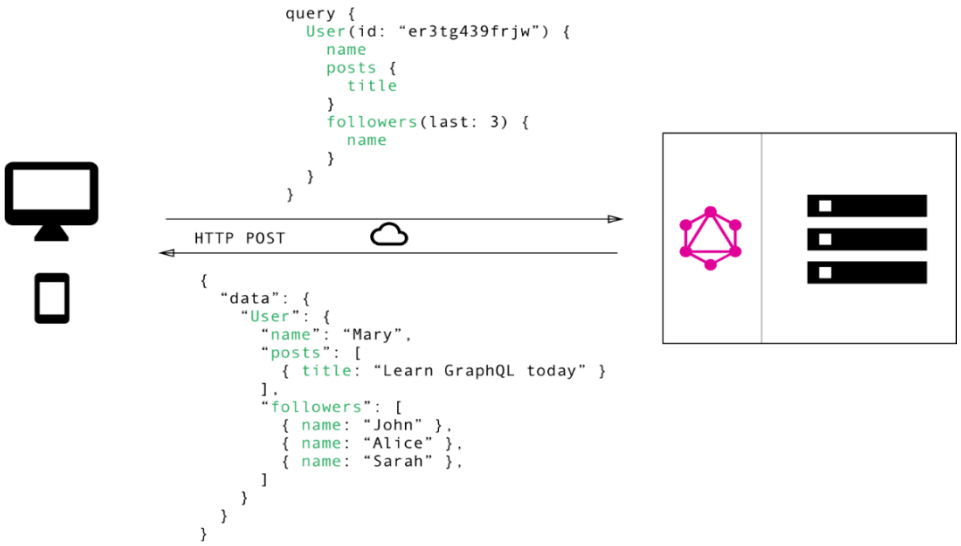
   Prinsip ini memungkinkan *server* untuk menyediakan fitur dan disampaikan dalam bentuk kode atau *script* ke *client*. Klien sistem mendapatkan r*esponse* berupa *code* atau *script* dapat langsung digunakan dengan cara mengeksekusi kode atau *script* tersebut.

2.7 GraphQL

GraphQL dikembangkan oleh Facebook pada tahun 2012 dan digunakan secara internal kemudian pada tahun 2015 Facebook merilis GraphQL menjadi *software open-source*. GraphQL adalah *query language* untuk API dan sekaligus menjadi *server-side runtime* untuk menjalankan *query* tersebut dengan data yang

tersedia. GraphQL dapat mendukung suatu sistem untuk melakukan integrasi dengan sistem lain secara *remote*. GraphQL memberikan gambaran yang komprehensif dan mudah dipahami, memberikan keleluasaan kepada *client* untuk menentukan sendiri model data yang mereka butuhkan sehingga GraphQL lebih fleksibel dan efisien (*A query language for your API*, n.d.).

GraphQL *query* selalu mengembalikan *response* yang dapat diperkirakan. Aplikasi menggunakan GraphQL lebih cepat karena dapat mengatur sendiri *resources* yang diinginkan sesuai kebutuhan. Pada GraphQL *client* dapat mendapatkan semua data yang dibutuhkan dengan hanya dalam satu permintaan.



Gambar 2. 5 *Request Response* GraphQL

GraphQL menyediakan sintaks yang fleksibel untuk menggambarkan *requirements* data dan interaksi untuk membangun aplikasi. GraphQL mengimplementasikan fitur-fitur bahasa dan karakteristik yang diperlukan yang dijelaskan dalam s*chema*. Bahasa tersebut menyediakan *type* sistem, menentukan

type dan ekspresi yang di dukung (Vogel et al., 2018). *Core* konsep pada GraphQL antara lain sebagai berikut.

1. *Schema Definition Language (SDL)*

   GraphQL menggunakan *type* sistem sendiri yang digunakan untuk mendefinisikan *schema* dari sebuah API. Sintaks untuk menulis s*chema* tersebut disebut *Schema Definition Language (SDL)*. *Schema* ini akan berisi kemampuan kolektif-*type* dari GraphQL *Service*, antara lain *type directives*, serta *type root operation* untuk setiap jenis operasi seperti *query*, *mutation*, dan *subscription* (Quiña-Mera et al., 2022).

2. *Fetching* Data *with Queries*

   Ketika menggunakan REST APIs, data akan diterima dari spesifik e*ndpoint*s dan setiap e*ndpoint* tersebut akan mengembalikan data yang *fixed*. Pendekatan ini diterapkan dalam GraphQL dengan sangat berbeda. GraphQL hanya mengekspose satu e*ndpoint*. Dengan satu e*ndpoint* tersebut data yang dikembalikan menjadi lebih fleksibel. Klien dapat menentukan sendiri data yang dibutuhkan.

3. *Writing* Data *with Mutations*

   Untuk membuat perubahan pada *server*, menggunakan GraphQL perubahan ini dapat dibuat dengan menggunakan yang disebut dengan *mutations*. Secara umum terdapat tiga *mutations* yaitu *create new* data, *updating existing* data, dan *deleting existing* data.

4. *Realtime Updates with Subscriptions*

GraphQL menyediakan fitur *subscriptions*. Ketika *client* melakukan *subscribe* terhadap suatu *event*, maka ia akan memulai dan mempertahankan koneksi yang stabil ke *server*. Setiap *event* yang terjadi *server* akan mendorong data yang sesuai untuk *client*.

2.8 Lighthouse

Lighthouse adalah sebuah pustaka tambahan untuk laravel yang dapat menyederhanakan proses pembuatan *interface* GraphQL. Lighthouse bekerjasama dengan GraphQL *models* dan hanya membutuhkan deskripsi *mutation* dan *query* untuk memperoleh data (Sultanov et al., 2021). Lighthouse menyediakan *directives* dan perlengkapan otomatis sehingga dapat mengotomasi pekerjaan *developer*. *Developer* hanya mengerjakan hal-hal yang esensial seperti perencanaan dan desain data (Madeja, 2022).

2.9 MySQL

MySQL adalah *relational database management system* yang cukup populer dan tersedia secara *open-source*. MySQL memungkinkan berbagai macam aplikasi komputer yang ditulis dengan berbagai bahasa pemrograman dapat mengakses *database* MySQL. Hal ini dapat terwujud berkat adanya fasilitas *API (Application Programming Interface)* yang dimiliki oleh MySQL (Komputer, 2010). Komponen-komponen utama dari MySQL antara lain sebagai berikut.

1. Tabel

   Tabel adalah objek dasar yang merupakan jantung dari *relational database*. Tabel bertujuan untuk menyimpan informasi-informasi suatu entitas. Informasi-informasi tersebut dibagi menjadi beberapa data terpisah secara logis kemudian disimpan kedalam tabel. *Database* dapat mengandung banyak tabel-tabel di dalamnya dalam jumlah yang tidak terbatas. Setiap baris tabel dapat mengandung nilai dengan tipe data dan ukurannya ditentukan sendiri sesuai yang didukung oleh MySQL.

2. *Query*

   Untuk mengambil informasi tertentu yang tersimpan pada tabel atau pada multi tabel dapat menggunakan *query*. Dengan menjalankan *query*, c*lient* dapat memperoleh data berupa *values* yang sesuai dengan kriteria yang diinginkan.

2.10 Laravel Sail

Laravel Sail adalah *command-line interface* yang ringan untuk berinteraksi dengan docker *development environment* bawaan laravel. Laravel Sail menyediakan *starting point* yang bagus untuk membangun aplikasi laravel menggunakan PHP, MySQL, dan Redis tanpa memerlukan pengalaman docker selumunya (*Laravel - The PHP Framework For Web Artisans*, n.d.).

Pada intinya sail adalah berkas docker-compose.yml dan sail *script* yang disimpan pada *root project*. Sail *script* menyediakan *command-line interface*

dengan metode yang mudah digunakan untuk berinteraksi dengan kontainer docker yang didefinisikan oleh berkas docker-compose.yml. Laravel Sail didukung oleh sistem operasi pada umumnya seperti macOS, Linux, dan Windows.

2.11 *AWS EC2*

Amazon Elastic Compute Cloud adalah salah satu platform komputasi *cloud* yang ditawarkan oleh *Amazon* Web *Service* yang dapat diperbaharui dan diperluas. Dengan *Amazon EC2* dapat mengeliminasi kebutuhan untuk biaya dalam komponen hardware fisik (*What Is Amazon EC2? - Amazon Elastic Compute Cloud*, n.d.). Penggunakan *Amazon EC2* dapat disesuaikan dengan yang dibutuhkan, mengkonfigurasi sendiri keamanan dan jaringan, serta mengelola penyimpanannya. Dengan *Amazon EC2* memungkinkan *server* untuk ditingkatkan atau diturunkan skalanya demi menangani perubahan atau lonjakan permintaan, sehingga mengurangi kebutuhan dalam memperkirakan trafik.

Layanan *EC2* terdiri atas tiga komponen utama yaitu pertama *instance EC2* yang merupakan mesin virtual yang berjalan pada lingkungan *EC2* dan melakukan tugas komputasi yang di biasanya dilakukan oleh *server* fisk. Selanjutnya adalah Lingkungan tempat *EC2* berjalan, lingkungan ini menyediakan *access control* yang dapat dikonfigurasi, data kontekstual, dan informasi lain yang dibutuhkan *instance* untuk melakukan tugas komputasi. Kemudian yang terakhir adalah *Amazon Machine* Images *(AMIs)* yang merupakan file *snapshot* lengkap dari *instance EC2* pada periode waktu tertentu, termasuk *software*-nya, konfigurasi, dan bahkan

datanya. *Image* ini berfungsi sebagai *disk boot* untuk *instance* yang akan dijalankan (Murty, 2008)

2.12 *Amazon RDS*

Amazon Relational *Database Service* merupakan layanan *database* yang dikelola oleh *AWS* sehingga mempermudah pengaturan, pengoptimalan, pengoperasian, dan penskalaan *database* relasional pada *AWS cloud* (*Apa Itu Amazon Relational Database Service (Amazon RDS? - Amazon Relational Database Service*, n.d.). *Amazon RDS* mendukung banyak mesin *DB* dengan basis relasional seperti MariaDB, Microsoft SQL Server, MySQL, Oracle, dan PostgreSQL.

Amazon RDS memiliki kemampuan pencadangan dan pemulihan. Pencadangan dapat dilakukan dengan memilih *snapshot* cadangan otomatis ataupun secara manual melakukan *snapshot*. *Amazon RDS* juga mendukung *Multi AZ* sehingga meningkatkan *durability* dan *availability* jika terjadi kegagalan komputasi (*DBaas Comparison*, n.d.), pada proses pembuatan *instance* lalu memilih *multi deployment* maka *AWS* akan membuat *instance* replika pada *availability zone* yang lain.

2.13 Apache Jmeter

Apache Jmeter adalah sebuah *software testing tools* yang bersifat *open-source* yang dibuat oleh Stefano Mazzocchi pada *Apache Software Foundation*. Fungsi utama dari apache jmeter adalah melakukan simulasi *load* tes dan mengukur

performa dari c*lient* ataupun *server* (National University of Sciences and Technology (Islāmābād et al., n.d.).

Jmeter akan bertindak sebagai *client* dari aplikasi yang akan diuji performanya. Jmeter dapat mengukur *response time* dan semua sumber daya lainnya seperti *CPU loads*, penggunaan memori, dan penggunaan *resources* (Halili, 2008). Jmeter dapat digunakan untuk menguji kinerja pada *resources* yang statis dan dinamis seperti file statis, *server* FTP, Java Object, *Database*, *script* Perl, *query*, dan banyak lagi. Untuk menguji dan mengukur ketahan *server* HTTP, penguji perlu memberikan simulasi beberapa dan berbagai jenis beban yang berbeda pada objek-objek pada sistem.

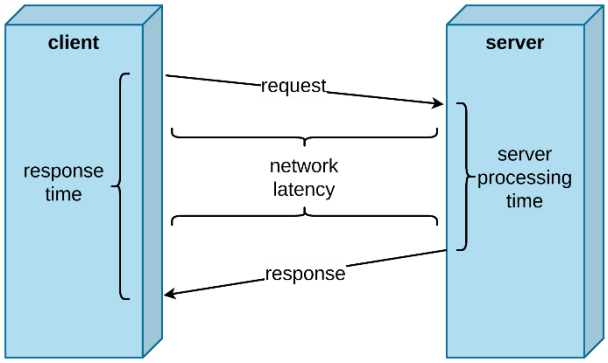
2.14 Chrome *DevTools*

Chrome *DevTools* adalah serangkaian alat pengukung web *development* yang dibangun langsung ke dalam *browser* google chrome, yang dapat digunakan untuk mendiagnosis suatu halaman, menganalisis kinerja halaman sehingga nantinya dapat tercipta halaman web yang lebih baik dan lebih cepat(*Chrome DevTools*, n.d.).

Chrome *DevTools* juga dapat melakukan simulasi untuk *mobile device*. Dengan menggunakan *virtual device* untuk menciptakan *mobile-first website* sehingga memudahkan pengguna dalam proses pengembangan *website* yang interaktif. Chrome *DevTools* ini dapat diakses dengan sangat mudah, yakni dengan melakukan klik kanan kemudian *inspect* pada tampilan *website* yang akan di analisis.

2.15 *Response Time*

Response Time adalah waktu yang diperlukan oleh aplikasi untuk merespon permintaan dari *user*. Hal ini diukur menggunakan satuan waktu detik atau milidetik sesuai dengan aplikasi (Murty, 2008).



Gambar 2. 6 Mekanisme Response Time

Untuk menghitung rata-rata *response time* dapat menggunakan persamaan berikut.

$$Average = \frac{1}{n} \sum_{i=1}^{n} \text{ atau } Average = \frac{1}{n}(a_1 + a_2 + a_3 + \cdots + a_n)$$

Keterangan:

n = jumlah *request*

i = nomor *request*

a = *response time*

2.16 *Throughput*

Throughput adalah jumlah data yang berhasil ditransmisikan pada jangka waktu tertentu. *Throughput* diukur dengan bit per sekon (bps), atau megabit per sekon (Mbps), atau Gigabit per sekon (Gbps) (*What Is Throughput?*, n.d.). *Throughput* sangat bergantung pada kemampuan *hardware server*, beban dari sistem, dan *latency* jaringan (Murty, 2008). Untuk menghitung *throughput* dapat menggunakan persamaan sebagai berikut (*Apache JMeter - User's Manual: Glossary*, n.d.).

$$Throughput = \frac{n}{t}$$

Keterangan:

n = jumlah *request*

t = total waktu yang digunakan untuk menangani semua *request*

2.17 *CPU Utilization*

CPU Utilization adalah persentase rata-rata penggunaan *Central Processing Unit* atau yang biasa dikenal dengan istilah prosesor. *CPU Utilization* menunjukkan seberapa persen prosesor yang digunakan untuk melakukan atau mengeksekusi suatu *task* (Rizvandi et al., 2014). *CPU Utilization* ini memberikan gambaran mengenai beban yang diterima oleh *server* saat menangani permintaan dari *client*.

2.18 *Page Load Time*

Page Load Time adalah jumlah waktu yang dilewati antara *browser* yang mengirim *request* ke *server* hingga halaman dimuat dan dirender sepenuhnya oleh *browser*.