# BAB II
# TINJAUAN PUSTAKA

## 2.1 Penelitian Terdahulu

Penelitian terdahulu menjadi salah satu acuan yang dilakukan dalam penelitian sehingga dapat memperkaya teori yang digunakan dalam mengaji penelitian yang dilakukan. Beberapa penelitian terdahulu sebagai referensi pada penelitian ini. Penelitian-penelitian yang sejenis diantaranya telah dilakukan oleh peneliti terdahulu yaitu sebagai berikut:

Penelitian yang telah dilakukan oleh [1] dengan judul ***Evaluating GraphQL and REST API Services Performance in a Massive and Intensive Accessible Information System*** **yang** menggunakan dua gagasan utama untuk membandingkan kinerja dari kedua arsitektur. Gagasan yang pertama adalah evaluasi dari dua layanan yang dilakukan pada saat operasi dan gagasan kedua adalah hasil evaluasi kinerja yang diperoleh dari kualitas pelayanan berdasarkan waktu respons, throughput, beban CPU, dan penggunaan memori. Hasil penelitian menunjukkan REST masih lebih cepat hingga 50,50% pada response time dan 37,16% untuk throughput, sedangkan GraphQL sangat efisien dalam pemanfaatan sumber daya, yaitu 37,26% untuk beban CPU dan 39,74% untuk pemanfaatan memori. Oleh karena itu, GraphQL adalah pilihan yang tepat ketika kebutuhan data sering berubah, dan pemanfaatan sumber daya adalah pertimbangan yang paling penting. Perbedaan yang ada pada penelitian yang dilakukan oleh [1] dengan penelitian yang akan dilakukan adalah pada segi gagasan yang digunakan dikarenakan pada penelitian yang akan dilakukan hanya berfokus pada hasil evaluasi kinerja saja. Sedangkan untuk persamaan dari penelitian ini untuk penelitian yang akan dilakukan adalah sama-sama mengevaluasi dari segi kualitas pelayanan yang berupa waktu respon, throughput, beban CPU, dan penggunaan memori.

Penelitian yang telah dilakukan oleh [2] dengan judul penelitian ***A GraphQL approach to Healthcare Information Exchange with HL7 FHIR*** menjelaskan bahwa didalam penelitiannya menggunakan metode pemanfaatan penggunaan GraphQL dan HL7 FHIR untuk sistem informasi kesehatan yang didalamnya menyajikan algoritma untuk memetakan sumber daya HL7 FHIR ke skema GraphQL, dan membuat implementasi prototipe dari pendekatan tersebut dan membandingkannya dengan pendekatan REST. Hasil penelitian menunjukkan menunjukkan bahwa kombinasi API web berbasis GraphQL dan HL7 FHIR untuk sistem informasi kesehatan berkinerja baik, hemat biaya, skalabel, dan fleksibel untuk memenuhi persyaratan klien web dan

seluler. Persamaan pada penelitian ini dan penelitian yang akan dilakukan adalah sama-sama menggunakan REST dan GraphQL sebagai arsitektur sistem. Sedangkan untuk perbedaannya terletak pada pemanfaatan HL7 FHIR yang digunakan untuk memetakan sumber daya ke skema GraphQL yang dimana skema tersebut tidak akan diterapkan pada penelitian yang akan dilakukan.

Penelitian yang dilakukan oleh [3] dengan judul ***Comparative Analysis of Rest and GraphQL Technology on Nodejs-Based Api Development*** menggunakan metode yang berupa alur pengembangan dengan memanfaatkan metode Waterfall, salah satu siklus klasik dalam pengembangan perangkat lunak. Metode ini menggambarkan pendekatan yang cukup sistematis dan berurutan untuk pengembangan perangkat lunak. Hasil penelitian menunjukkan bahwa arsitektur REST memiliki performansi yang lebih baik dari GraphQL dalam kecepatan responnya. Di sisi lain, GraphQL juga unggul dalam penyajian data dengan permintaan aplikasi klien untuk mengoptimalkan bandwidth yang tersedia. Persamaan yang ada pada penelitian kali ini adalah sama-sama menggunakan arsitektur REST dan GraphQL pada pengembangan sistemnya. Sedangkan untuk perbedaan yang terletak pada penelitian kali ini adalah alur pengembangan yang memanfaatkan Agile sebagai metode pengembangan perangkat lunak.

## 2.2 API

*Application Programming Interface* atau biasa disebut dengan API merupakan sarana yang memungkinkan pengembang untuk mengintegrasikan dua bagian dari suatu aplikasi atau dengan aplikasi yang berbeda secara bersamaan [4]. Dalam proses perancangannya API akan memberi tahu masukan apa yang diharapkan dalam proses nya dan akan memberikan sebuah data sebagai balasan atas masukan yang ada. API dapat dikatakan memiliki konsep yang sederhana akan tetapi terlepas dari kesederhanaan konsep nya API merupakan ide yang patut diimplementasikan karena memungkinkan pengembang untuk membangun suatu alat yang berguna untuk memproses data dari sejumlah sumber yang menarik. API dikatakan baik menurut [5] ketika memenuhi beberapa kriteria yang ada yakni:

a. Ketika ada suatu kesalahan dalam pemrosesan data maka API harus memberikan informasi mengenai kesalahan yang terjadi kepada pengguna serta bagaimana cara memperbaiki kesalahan yang ada.
b. API juga harus membuang dokumen-dokumen pengecualian yang berada didalam sistem.

c. API juga harus mendokumentasikan dirinya sendiri, yang dimaksud dari mendokumentasikan dirinya sendiri adalah ketika inisialisasi nama variabel dan metode harus dilakukan secara jelas dan dapat dengan mudah dibaca oleh manusia.

API dapat mengambil berbagai bentuk, termasuk library perangkat lunak, protokol komunikasi, atau bahkan antarmuka web yang memungkinkan aplikasi beroperasi secara terintegrasi. Penggunaan API sangat umum dalam pengembangan perangkat lunak modern karena memungkinkan pengembang untuk memanfaatkan fungsionalitas yang sudah ada tanpa perlu membangun semuanya dari awal. Dalam konteks pengembangan web, salah satu jenis API yang umum digunakan adalah REST API (Representational State Transfer API). REST API menggunakan protokol HTTP untuk memungkinkan komunikasi antara aplikasi atau layanan. API ini biasanya digunakan untuk mengakses dan memanipulasi data, serta menjalankan operasi tertentu pada server. API juga dapat berfungsi sebagai jembatan antara aplikasi dan layanan pihak ketiga. Dengan mengizinkan akses terkontrol ke fungsionalitas atau data dari penyedia layanan eksternal, API memungkinkan pengembang untuk memperluas kemampuan aplikasinya tanpa harus membuat semuanya dari awal.

Penting untuk diingat bahwa keberhasilan integrasi melalui API bergantung pada dokumentasi yang baik. Dokumentasi API yang jelas dan komprehensif membantu pengembang memahami cara menggunakan API dengan benar, termasuk endpoint yang tersedia, parameter yang diperlukan, dan tanggapan yang diharapkan. Secara keseluruhan, API berperan kunci dalam menghubungkan dan menyatukan aplikasi perangkat lunak, memungkinkan pengembang untuk membangun solusi yang lebih kompleks dan efisien dengan memanfaatkan sumber daya yang sudah ada. Dalam era digital saat ini, di mana integrasi dan kolaborasi antar aplikasi semakin penting, peran API semakin menonjol dalam memfasilitasi konektivitas yang lancar antara berbagai sistem.

## 2.3 REST

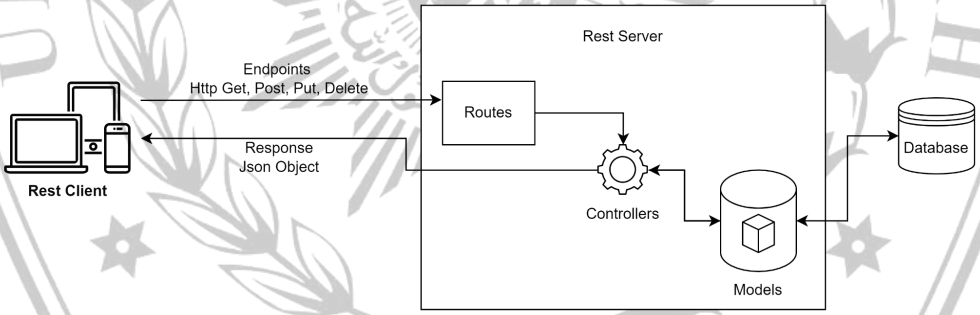
REST (Representational State Transfer) berkaitan dengan hubungan klien dan server dan bagaimana status disimpan. Arsitektur REST didasarkan pada gaya arsitektur klien/server. Dengan demikian, permintaan dan tanggapan dibangun berdasarkan proses transfer sumber daya. Secara umum arsitektur REST memiliki struktur yang lebih ringan dibandingkan dengan arsitektur yang

lain. Dalam hal implementasi desain yang memanfaatkan arsitektur REST memerlukan dua prinsip utama didalamnya sesuai dengan yang dijelaskan pada jurnal [6] yang meliputi:

a. *Addressability*, prinsip REST yang dimana sekumpulan data dimodelkan untuk beroperasi sebagai sumber daya.
b. *Statelessness*, prinsip yang berarti setiap transaksi yang ada didalam sistem harus independen dan tidak boleh terkait dengan transaksi sebelumnya, karena semua data yang diperlukan untuk melakukan dan memproses permintaan terdapat pada permintaan tersebut, sehingga server tidak perlu memelihara data sesi klien.

Disisi lain menurut yang dijelaskan oleh [2] Arsitektur REST mempunyai beberapa aturan dan batasan komprehensif yang dapat memberikan web service mempunyai akses terstruktur kedalam resource data. Tetapi aturan ini menjadi tidak fleksibel dan kompleks dikarenakan berbagai alasan yang meliputi:

a. Peningkatan kompleksitas sistem yang dibangun
b. Tuntutan kualitas layanan yang tinggi yang dibutuhkan oleh pengguna
c. Pengembangan sistem secara realtime
d. Pengambilan data dinamis oleh klien dari frontend maupun mobile.




*Gambar 1 Arsitektur REST*

Metode HTTP menjadi kunci dalam interaksi antara klien dan server dalam REST API. Metode ini mencerminkan operasi yang diinginkan terhadap sumber daya. Sebagai contoh, metode GET digunakan untuk mengambil data, POST untuk membuat sumber daya baru, PUT untuk memperbarui sumber daya, dan DELETE untuk menghapusnya. Dengan demikian, REST API memberikan semantik yang jelas dan intuitif dalam komunikasi antar-sistem. Sumber daya dalam REST diidentifikasi oleh URI, yang berfungsi sebagai alamat atau lokasi yang unik. URI memberikan cara untuk mengakses dan memanipulasi sumber daya tersebut. Representasi sumber

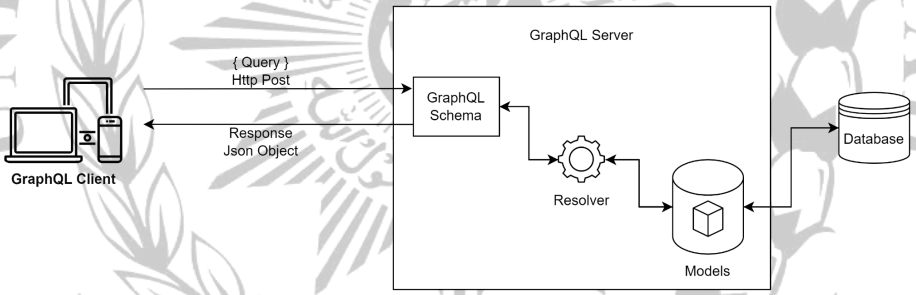
daya, seperti format data JSON atau XML, memfasilitasi pertukaran informasi antara klien dan server. Kesepakatan format data ini umumnya dilakukan di awal kerja sama.

REST API juga dikenal sebagai stateless, yang berarti setiap permintaan dari klien ke server harus mencakup semua informasi yang diperlukan untuk memahami dan memproses permintaan tersebut. Ini menghilangkan kebutuhan server untuk menyimpan status klien antar permintaan, mempermudah skalabilitas dan manajemen sistem. Salah satu konsep tambahan dalam REST API adalah HATEOAS (Hypermedia As The Engine Of Application State), yang melibatkan penyisipan hyperlink dalam representasi sumber daya. Dengan kata lain, server menyediakan link ke sumber daya terkait, memungkinkan klien untuk menavigasi aplikasi dengan lebih dinamis. Pemahaman mendalam tentang konsep-konsep dasar REST API, seperti resource URI, metode HTTP, representasi, statelessness, dan HATEOAS, menjadi kunci dalam merancang dan mengimplementasikan layanan web yang efektif, efisien, dan mudah dipahami. Dengan menggunakan pendekatan ini, pengembang dapat menciptakan antarmuka yang responsif dan intuitif untuk aplikasi mereka, meningkatkan interoperabilitas dan skalabilitas sistem secara keseluruhan[6].

## 2.4 GraphQL

Menurut (Oggier, 2020) dibandingkan dengan REST, GraphQL secara signifikan lebih baru dari versi sebelumnya. Proyek ini dimulai secara internal oleh Facebook pada tahun 2012 untuk memecahkan masalah terkait dengan meningkatnya permintaan untuk aplikasi seluler. Pada hari-hari awal ekosistem smartphone, perangkat sangat dibatasi oleh konektivitas. Aplikasi pengguna harus membuat permintaan sesedikit mungkin untuk mengoptimalkan kecepatan dan waktu muat. Namun, perusahaan seperti Facebook dengan aplikasi berat dan umpan berita berjuang untuk memenuhi standar ini[7]. Misalnya, beberapa kueri diperlukan untuk mengambil semua informasi yang terkait dengan sebuah postingan. Akibatnya, Facebook memutuskan untuk membuat standar kueri baru yang memungkinkan kami menggabungkan semua data yang kami butuhkan dalam satu kueri. GraphQL sangat intuitif untuk digunakan dikarenakan setiap permintaan yang ada berbasis skema sehingga dalam hal penggunaannya dapat dengan mudah dibaca dan dibangun. Dalam hal ini GraphQL juga menghindari banyaknya penggunaan logika dalam hal penyusunan ulang data seteah beberapa kali melewati proses pengambilan. GraphQL sendiri memiliki tiga konsep yang biasa digunakan yakni:

a. Arsitektur Simpel, Arsitektur ini menampilkan contoh server tunggal dari spesifikasi GraphQL. Server ini terhubung ke database dan hanya akan menjawab permintaan yang dibuat. Koneksi antara server dan klien dapat dibuat menggunakan protokol apa pun. Ini berarti bahwa transportasi dapat dilakukan melalui WebSockets, TCP, dan banyak opsi lainnya. Penting juga untuk dicatat bahwa server GraphQL dapat bekerja dengan semua jenis database, baik SQL atau NoSQL.
b. Arsitektur Gateway, server GraphQL akan berinteraksi dengan beberapa layanan seperti *REST endpoints*, layanan eksternal, instance GraphQL lainnya, dll. dalam hal ini, gateway GraphQL bertanggung jawab untuk membuat setiap sub-sistem berkolaborasi bersama dalam menyelesaikan layanan mana yang diperlukan untuk melayani permintaan.
c. Arsitektur Hybrid, Dalam arsitektur ini, gateway terletak di pusat sistem sebagai titik akhir utama. Salah satu alasan utama untuk mengadopsi struktur tersebut adalah kebutuhan untuk memperluas fitur akan sangat sering mengakibatkan penggunaan konfigurasi hybrid.




*Gambar 2 Arsitektur GraphQL*

Salah satu keunggulan utama GraphQL terletak pada kemampuannya memberikan kontrol lebih besar kepada klien terkait data yang diperlukan. Alih-alih mengambil semua data yang mungkin diperlukan, klien dapat menyusun query spesifik dengan hanya meminta informasi yang dibutuhkan. Hal ini menghindarkan masalah over-fetching (pengambilan data yang berlebihan) atau under-fetching (pengambilan data yang kurang) yang sering terjadi dalam penggunaan API tradisional. GraphQL menggunakan skema (schema) sebagai kontrak antara klien dan server, menjelaskan struktur data yang dapat diakses oleh klien. Ini membuat dokumentasi dan eksplorasi

API lebih mudah, dan memungkinkan pengembang untuk bekerja secara kolaboratif dengan lebih efisien[8].

Selain itu, GraphQL memungkinkan penggabungan beberapa permintaan dalam satu query, mengatasi masalah over-fetching dengan menggabungkan kebutuhan data dari berbagai bagian aplikasi dalam satu permintaan. Ini disebut dengan fitur batching, yang membantu mengoptimalkan kinerja aplikasi dan mengurangi jumlah permintaan ke server. Sebagai standar industri yang semakin populer, GraphQL telah diadopsi oleh banyak perusahaan teknologi dan menjadi pilihan dalam pengembangan API. Selain itu, beberapa kerangka kerja server populer, seperti Apollo Server dan Express.js, mendukung implementasi GraphQL, memudahkan integrasinya dalam berbagai proyek. Dengan kemampuan fleksibilitasnya, dokumentasi yang baik, dan evolusinya yang terus-menerus, GraphQL menjadi solusi yang menarik untuk kebutuhan pertukaran data antara klien dan server dalam pengembangan aplikasi modern.

## 2.5 Typescript

TypeScript adalah sebuah bahasa pemrograman yang dikembangkan oleh Microsoft. Ini adalah superset dari JavaScript, yang berarti semua fitur JavaScript juga dapat digunakan dalam TypeScript, namun TypeScript menambahkan fitur-fitur tambahan yang memungkinkan pengembangan perangkat lunak yang lebih aman, mudah dipelihara, dan bersifat objek. Salah satu fitur utama dari TypeScript adalah penambahan sistem tipe statis. Dalam JavaScript, tipe data suatu variabel ditentukan secara dinamis saat runtime, yang dapat menyebabkan kesalahan sulit dilacak selama pengembangan. TypeScript memungkinkan pengembang untuk mendeklarasikan tipe data variabel secara eksplisit, sehingga kesalahan dapat terdeteksi pada saat kompilasi. Ini meningkatkan keamanan kode dan memudahkan pengembangan dan pemeliharaan kode yang lebih besar. TypeScript juga mendukung konsep-konsep pemrograman berorientasi objek seperti kelas dan antarmuka. Pengguna dapat membuat struktur kode yang lebih terstruktur dan modular, mempermudah pengelolaan kompleksitas proyek[9].

Sintaksis TypeScript mirip dengan JavaScript, yang membuatnya mudah diadopsi oleh pengembang yang sudah terbiasa dengan JavaScript. Selain itu, TypeScript dapat dikompilasi menjadi JavaScript, sehingga dapat dijalankan di lingkungan JavaScript yang ada tanpa masalah. TypeScript banyak digunakan dalam pengembangan aplikasi web, terutama proyek-proyek berbasis Node.js dan aplikasi Angular. Angular, sebuah framework JavaScript yang juga dikembangkan oleh Microsoft, menggunakan TypeScript sebagai bahasa resmi pengembangannya.

Ini memberikan dukungan penuh terhadap fitur-fitur TypeScript dan memperkuat penggunaannya dalam pengembangan aplikasi web berskala besar. Dengan fitur-fitur seperti sistem tipe statis, dukungan untuk pemrograman berorientasi objek, dan kemampuan untuk diintegrasikan dengan ekosistem JavaScript yang luas, TypeScript menjadi pilihan yang populer bagi pengembang yang mencari solusi yang lebih aman dan terstruktur dalam pengembangan aplikasi web modern [9].

## 2.6 Node.JS

Node.js adalah runtime lingkungan JavaScript yang berbasis pada mesin JavaScript V8 dari Chrome. Dirancang untuk mengeksekusi kode JavaScript di sisi server, Node.js memungkinkan pengembang untuk menggunakan JavaScript untuk membuat aplikasi server-side dan menangani permintaan HTTP. Salah satu fitur kunci dari Node.js adalah kemampuannya untuk menangani banyak koneksi secara bersamaan dengan model non-blocking I/O, yang membuatnya sangat efisien dan cocok untuk pengembangan aplikasi real-time, seperti aplikasi web, permainan daring, dan alat kolaborasi. Node.js memanfaatkan model single-threaded dan non-blocking event loop, yang memungkinkan aplikasi untuk menangani sejumlah besar permintaan tanpa memblokir eksekusi kode. Ini berbeda dengan model tradisional server yang menggunakan thread terpisah untuk setiap koneksi, yang dapat menghasilkan overhead yang signifikan. Dengan Node.js, satu thread dapat menangani banyak koneksi secara simultan, mengoptimalkan kinerja dan responsifitas aplikasi. Node.js juga memiliki manajer paket bawaan yang disebut npm (Node Package Manager), yang memudahkan pengelolaan dependensi dan pustaka pihak ketiga dalam proyek. Dengan ekosistem npm yang sangat besar, pengembang dapat dengan mudah mengakses dan menggunakan ribuan paket pustaka yang telah dibuat oleh komunitas[10].

Selain digunakan untuk pengembangan server, Node.js juga umum digunakan untuk mengembangkan perangkat lunak frontend, seperti membangun alat pengembangan, bundlers, dan task runners. Kombinasi penggunaan Node.js di sisi server dan di sisi klien memungkinkan pengembang untuk menggunakan JavaScript secara end-to-end, menyederhanakan alur kerja pengembangan. Node.js telah menjadi salah satu teknologi yang sangat populer di dunia pengembangan perangkat lunak, terutama untuk aplikasi berbasis mikro layanan dan aplikasi real-time. Kecepatan, skalabilitas, dan ekosistem yang kaya dengan pustaka-pustaka siap pakai menjadikan Node.js pilihan utama bagi banyak pengembang yang ingin membangun aplikasi web yang efisien dan responsif [10].

## 2.7 Express.JS

Express.js adalah kerangka kerja (framework) aplikasi web yang dibangun di atas Node.js, dirancang untuk mempermudah pengembangan aplikasi web dengan menggunakan JavaScript di sisi server. Sebagai salah satu kerangka kerja paling populer dalam ekosistem Node.js, Express.js menyediakan struktur dan fungsionalitas tambahan untuk membangun aplikasi web dengan cara yang efisien dan terstruktur. Salah satu keunggulan Express.js adalah kesederhanaannya. Dengan desain yang minimalis dan filosofi "*middleware*", Express.js memungkinkan pengembang untuk dengan mudah menangani permintaan HTTP dan meresponsnya dengan cara yang fleksibel. Middleware dalam Express.js memungkinkan penggunaan berbagai fungsi untuk memproses permintaan sebelum atau sesudah mencapai rute yang ditentukan. Express.js menyediakan sistem rute yang kuat, memungkinkan pengembang untuk menentukan cara menangani permintaan untuk setiap endpoint atau URI tertentu. Ini mempermudah pembuatan API RESTful atau routing kompleks dalam aplikasi web. Selain itu, Express.js mendukung berbagai jenis tampilan (view engines) yang memungkinkan pengembang untuk menghasilkan tampilan HTML dinamis menggunakan template. Kemampuan untuk mengintegrasikan middleware pihak ketiga melalui ekosistem npm memperluas fungsionalitas Express.js. Middleware ini dapat digunakan untuk keperluan seperti otentikasi, logging, dan penanganan kesalahan, menjadikan Express.js sangat fleksibel dan dapat disesuaikan dengan kebutuhan proyek. Dengan konsep modularitas yang dianut Express.js, pengembang dapat membangun aplikasi yang sesuai dengan kebutuhan mereka, tanpa terlalu banyak beban atau kompleksitas yang tidak perlu. Dengan penggunaan yang luas dan dokumentasi yang baik, Express.js telah menjadi pilihan utama banyak pengembang dalam membangun aplikasi web yang efisien dan dapat diandalkan di atas platform Node.js [11].