BAB I PENDAHULUAN

Bab Pendahuluan memuat penjelasan mengenai gambaran topik penelitian yang terdiri atas beberapa bagian, yaitu latar belakang dari topik penelitian, perumusan masalah, tujuan penelitian, batasan-batasan dan manfaat dari peneliti.

I.1 Latar Belakang

Arsitektur *monolithic* saat ini bertahan sebagai salah satu pendekatan yang masih banyak digunakan untuk kebutuhan pengembangan aplikasi *website*. Secara tradisional, pengembangan aplikasi *website* dilakukan dengan menetapkan seluruh komponen sistem menjadi satu kesatuan yang utuh (Ren dkk., 2018). Namun, ketetapan dari arsitektur *monolithic* ini menjadi suatu kekurangan karena secara tidak langsung menghambat skalabilitas dan pemeliharaan karena komponen yang saling terikat, sehingga proses *deploy* tidak dapat dilakukan secara terpisah (Gos & Zabierowski, 2020). Sebuah studi sebelumnya dilakukan oleh Tapia dkk. (2020) yang membandingkan performa antara sistem *monolithic* dan *microservices* dalam pengembangan aplikasi *website* menunjukkan bahwa arsitektur *monolithic* sering kali mengalami hambatan performa, terutama karena sifatnya yang terpusat di mana satu titik kegagalan dapat memengaruhi keseluruhan sistem. Hal ini membuat aplikasi berbasis *monolithic* kurang efisien dalam menangani beban kerja tinggi atau kebutuhan skalabilitas yang dinamis (Tapia dkk., 2020). Selain itu, pernyataan ini juga didukung oleh studi yang dilakukan oleh Blinowski dkk. (2022) yang menunjukkan bahwa pendekatan *monolithic* cenderung menghasilkan fluktuasi performa yang signifikan, terutama ketika aplikasi sedang menangani peningkatan jumlah pengguna atau permintaan layanan secara simultan (Blinowski dkk., 2022). Studi tersebut menunjukkan bahwa arsitektur *monolithic* seringkali menghadapi performa akibat desain terpusat yang menciptakan satu titik kegagalan, sehingga respons dari sistem menurun dan terjadi fluktuasi performa saat beban pengguna atau permintaan layanan meningkat. Keterbatasan arsitektur ini dalam mengatasi permasalahan tersebut menjadi suatu urgensi untuk memilih pendekatan yang lebih modular dan terdistribusi untuk memastikan ketahanan dan konsistensi performa.

Arsitektur *microservices* merupakan salah satu pendekatan arsitektur yang semakin populer di kalangan industri maupun akademisi dalam mendukung ketangkasan dan pemeliharaan dalam pengembangan *software* dan *deployment* (Ahmad dkk., 2025; Lercher, 2024). Dalam arsitektur ini, sistem dipecah menjadi layanan-layanan kecil yang berjalan dan diskalakan secara independen, serta saling berkomunikasi melalui API (Surianarayanan dkk., 2019). Komunikasi antar layanan pada *microservices* yang dilakukan melalui API pada umumnya dibangun berbasis HTTP yang berperan sebagai media pertukaran data (Cleveland dkk., 2020). Terdapat beberapa jenis API yang dapat digunakan pada arsitektur *microservices*, diantaranya RabbitMQ, Kafka, gRPC, GraphQL, dan HTTP REST menjadi standar protokol yang paling umum digunakan (Cleveland dkk., 2020; Kazanavičius & Mažeika, 2023) (Leu dkk., 2024). GraphQL merupakan alternatif paradigma API yang telah banyak diadopsi di arsitektur *microservices* (Quiña-Mera dkk., 2023a). REST mendefinisikan kumpulan *endpoint* yang mewakili *resource* tertentu, sedangkan GraphQL memusatkan seluruh akses data pada satu *endpoint* dan memungkinkan *client* untuk membuat *query* yang fleksibel sesuai dengan kebutuhan (Guha & Shreyasi Majumder, 2020).

Salah satu aspek penting dalam menjaga performa sistem untuk selalu optimal pada arsitektur *microservices* adalah dengan pemilihan protokol komunikasi yang tepat. REST masih menjadi pendekatan protokol yang paling umum digunakan, namun alternatif seperti GraphQL mulai banyak diadopsi karena menawarkan fleksibilitas dalam pengambilan data. Sebuah studi dilakukan oleh Niswar dkk. (2024) dalam mengevaluasi performa tiga protokol komunikasi API yang umum digunakan pada arsitektur *microservices*, yaitu REST, GraphQL, dan gRPC. Hasil pengujian menunjukkan bahwa gRPC memiliki *response time* tercepat dalam semua skenario, diikuti oleh REST, sementara GraphQL menjadi yang paling lambat karena kompleksitas dalam memproses *query* dan pengambilan data bertingkat. Dari sisi penggunaan *resource*, GraphQL memiliki konsumsi CPU paling tinggi, sedangkan REST dan gRPC lebih efisien. GraphQL dinilai unggul dalam fleksibilitas pengambilan data yang kompleks, tetapi memerlukan *resource* yang lebih besar dan memiliki *response time* lebih lambat (Niswar dkk., 2024). Temuan ini membuktikan pentingnya evaluasi mendalam terhadap kebutuhan

sistem sebelum menentukan protokol komunikasi yang digunakan. Selain itu, studi lainnya dilakukan oleh Lawi dkk. (2021) evaluasi dua pendekatan API REST dan GraphQL dalam konteks sistem informasi yang besar dan kompleks. Untuk menilai performa secara menyeluruh dilakukan pengujian yang berdasarkan kepada empat metrik utama, yaitu *response time, throughput,* CPU *Load,* dan *memory utilization.* Hasil studi membuktikan bahwa REST unggul dalam kecepatan dan volume pemrosesan data, sementara GraphQL lebih efisien dalam penggunaan *resource* sistem, seperti CPU dan *memory* (Lawi dkk., 2021). Namun, penelitian oleh Vohra dan Manuaba (2022) memberikan sudut pandang lain terkait pengujian performa pada REST dan GraphQL dalam arsitektur *microservices*. Dalam studi ini, kedua protokol diuji pada sistem berbasis freelance marketplace dengan skenario nyata dan menggunakan alat uji JMeter. Hasilnya menunjukkan bahwa performa REST dan GraphQL hampir seimbang untuk operasi POST dan PUT. Namun, pada operasi GET yang melibatkan pengambilan data dari banyak layanan, GraphQL justru menunjukkan performa yang lebih baik. Temuan ini menunjukkan bahwa dalam situasi tertentu, terutama saat menangani data bertingkat, GraphQL dapat lebih efisien dibanding REST (Vohra & Kerthyayana Manuaba, 2022).

Meskipun berbagai studi telah membandingkan performa REST dan GraphQL, penelitian yang secara khusus memfokuskan analisis performa kedua API ini pada modul tertentu dalam sistem berbasis arsitektur microservices masih terbatas. Selain itu, variasi dalam lingkungan pengujian, bahasa pemrograman, dan alat yang digunakan dalam kajian-kajian sebelumnya menyulitkan generalisasi hasilnya. Di sisi lain, pada implementasi nyata seperti pada aplikasi *existing* SOFI, ditemukan adanya permasalahan performa berupa penurunan *response time* yang signifikan saat terjadi lonjakan beban pengguna, khususnya pada modul pendaftaran sidang yang memiliki kepadatan transaksi tertinggi selama periode puncak. Dari sisi praktis, fokus penelitian ini adalah mengukur metrik-metrik kunci kuantitatif, seperti *response time, throughput, error rate,* CPU *usage*, dan *memory usage* pada modul pendaftaran sidang SOFI. Hasil dari penelitian ini diharapkan dapat memberikan wawasan dan panduan konkret bagi pengembang dalam memilih protokol komunikasi API yang paling sesuai untuk mendukung

kebutuhan skalabilitas dan efisiensi operasional sistem yang dikembangkan. Ini akan berkontribusi pada peningkatan pemahaman dan keterampilan dalam pengelolaan API pada arsitektur *microservices*, serta mengasah kemampuan analisis untuk memberikan kontribusi pengetahuan kepada masyarakat. Dengan demikian, penelitian ini dapat menjadi bahan pertimbangan strategis bagi pengembang atau penelitian serupa dalam menentukan teknologi API yang optimal di masa depan, serta memberikan wawasan mengenai keunggulan dan kekurangan pengelolaan API REST dan GraphQL dalam membangun sistem yang lebih baik.

I.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dibahas sebelumnya, dapat disimpulkan pertanyaan yang mendasari dari tugas akhir ini adalah sebagai berikut:

a. Pemilihan API REST dalam transisi Aplikasi SOFI ke arsitektur *microservices* menimbulkan permasalahan terkait optimalisasi jangka panjang, di mana efisiensi sumber daya dan performanya belum terbukti lebih unggul dibandingkan dengan arsitektur alternatif, seperti GraphQL untuk menangani beban kerja spesifik aplikasi tersebut.
b. Belum tersedianya data perbandingan kuantitatif yang mengukur secara langsung metrik performa (*response time, error rate* dan *throughput*) dan efisiensi konsumsi sumber daya (CPU *usage* dan *memory usage*) antara kedua arsitektur API dalam menangani beban kerja spesifik pada modul pendaftaran sidang SOFI.
c. Diperlukannya sebuah pedoman pengambilan keputusan yang strategis dan berbasis bukti empiris untuk membantu tim pengembang dalam memilih arsitektur API (REST atau GraphQL) yang paling sesuai untuk pengembangan modul lainnya untuk menyeimbangkan prioritas antara latensi minimal dan efisiensi biaya infrastruktur.

I.3 Tujuan Tugas Akhir

Tujuan tugas akhir menjelaskan tentang alasan utama dan hasil yang ingin dicapai pada penelitian. Adapun tujuan dilakukannya tugas akhir ini adalah sebagai berikut:

a. Mengevaluasi secara kuantitatif performa dan efisiensi sumber daya antara API REST dan GraphQL untuk memvalidasi pilihan teknologi yang paling optimal bagi modul pendaftaran sidang Aplikasi SOFI.
b. Mengukur dan menganalisis secara mendalam metrik-metrik kunci, meliputi *response time, throughput, error rate,* CPU *usage,* dan *memory usage* dari kedua arsitektur API di bawah berbagai skenario beban untuk menghasilkan data perbandingan yang valid.
c. Merumuskan sebuah pedoman pengambilan keputusan yang strategis berdasarkan hasil analisis eksperimental, sebagai panduan bagi tim pengembang dalam memilih arsitektur API yang paling sesuai untuk kebutuhan modul *microservices* SOFI di masa depan.

I.4 Manfaat Tugas Akhir

Manfaat tugas akhir menjelaskan tentang kontribusi atau nilai tambah yang diharapkan tercapai pada penelitian. Adapun manfaat dari tugas akhir ini adalah sebagai berikut:

a. Bagi Universitas Telkom : Penelitian ini diharapkan dapat bermanfaat dalam meningkatkan produktivitas setiap entitas yang terlibat dalam proses bisnis di Fakultas Rekayasa Industri (FRI), khususnya dalam kegiatan kepengurusan administratif sidang bagi dosen dan mahasiswa akhir.
b. Bagi LAAK FRI : Menjadi bahan pertimbangan bagi *product owner* dan tim pengembang berikutnya dalam menentukan teknologi yang sesuai untuk penerapan API di Aplikasi Sidang Online Fakultas Rekayasa Industri (SOFI) di masa depan.
c. Bagi penulis : Penelitian ini bermanfaat dalam meningkatkan pemahaman dan keterampilan penulis dalam melakukan pengelolaan API pada arsitektur *microservices*, serta mengasah kemampuan penulis dalam melakukan analisis dan ikut berpatisipasi memberikan kontribusi pengetahuan kepada masyarakat.
d. Bagi peneliti lain : Memberikan wawasan kepada peneliti lain dalam menjelaskan keunggulan, serta kekurangan dari pengelolaan API REST

dan GraphQL pada analisis perbandingan aplikasi *website* berbasis *microservices* dalam membangun sistem yang lebih baik.

I.5 Batasan dan Asumsi Tugas Akhir

Batasan tugas akhir menjelaskan tentang ruang lingkup dalam penelitian, kondisi dan/atau asumsi yang telah ada pada rumusan masalah agar fokus dan terarah. Adapun batasan dan asumsi dari tugas akhir ini adalah sebagai berikut:

a. Penelitian ini hanya berfokuskan kepada API REST dan GraphQL sebagai objek penelitian perbandingan tanpa membandingkan jenis API lainnya, seperti SOAP (*Simple Object Access Protocol*), gRPC (*Google Remote Procedure Call*), dan lain-lainnya.
b. Penelitian ini dilakukan hanya berlaku dalam konteks pengembangan aplikasi *website* berbasis *microservices*, sehingga hasil penelitian tidak akan sepenuhnya relevan ketika diterapkan kepada jenis arsitektur lain dengan kebutuhan yang berbeda.
c. Analisis performa dan *resource* hanya difokuskan kepada aspek-aspek penting dan tidak menyinggung aspek lain, seperti *uptime, network* dan lainnya.
d. Semua kontainer *microservices* berjalan pada konfigurasi *hardware* yang konsisten, namun tidak menerapkan limitasi *resource* pada kontainer.

I.6 Sistematika Laporan

Secara sistematis isi dari laporan ini disusun sebagai berikut :

BAB I PENDAHULUAN

Bab ini menguraikan konteks penelitian, dimulai dari identifikasi masalah performa dan skalabilitas pada aplikasi SOFI yang berarsitektur *monolithic*. Diuraikan pula urgensi pemilihan protokol API yang efisien setelah transisi ke arsitektur *microservices*, sehingga dirumuskan masalah perbandingan kuantitatif antara performa API REST dan GraphQL pada modul pendaftaran sidang.

BAB II LANDASAN TEORI

Bab ini menyajikan landasan teoretis dan konseptual yang relevan. Di dalamnya dibahas secara mendalam prinsip-prinsip arsitektur *microservices*, karakteristik fundamental dari REST API , dan paradigma *declarative data fetching* pada

GraphQL. Bab ini juga dilengkapi dengan tinjauan sistematis terhadap penelitian-penelitian terdahulu yang membandingkan kedua teknologi, serta justifikasi pemilihan kerangka proses rekayasa perangkat lunak empiris sebagai metodologi penelitian.

BAB III METODE PENYELESAIAN MASALAH

Bab ini memaparkan metodologi penelitian yang diterapkan secara rinci. Dimulai dengan pemaparan kerangka berpikir yang memandu penelitian, kemudian diuraikan sistematika penyelesaian masalah yang mengadopsi lima tahapan proses eksperimental, yaitu *Scoping, Planning, Operation, Analysis & Interpretation*, dan *Presentation & Package*. Dijelaskan pula teknik pengumpulan data melalui wawancara dengan *product owner* dan analisis sistem *existing*, serta metode evaluasi data untuk merancang skenario pengujian yang terkontrol.

BAB IV PENYELESAIAN MASALAH

Bab ini berfokus pada realisasi teknis dari artefak eksperimen. Diuraikan proses perancangan kedua layanan API yang mengadopsi pola *Microservices Architecture*, desain basis data yang identik, serta definisi kontrak API untuk setiap *endpoint* REST dan *query* GraphQL. Dilanjutkan dengan paparan tahap pengembangan yang mencakup implementasi logika bisnis menggunakan bahasa Go dan GORM, serta orkestrasi lingkungan pengujian yang terisolasi menggunakan Docker dan Docker Compose.

BAB V ANALISIS, HASIL, DAN IMPLIKASI

Bab ini menyajikan hasil pengujian dan analisis mendalam terhadap data yang diperoleh. Dipaparkan perbandingan kuantitatif metrik performa layanan (*response time, throughput,* dan *error rate*) dan efisiensi konsumsi sumber daya (CPU *usage* dan *memory usage*) untuk kedua arsitektur API di bawah tiga skenario beban pengguna. Hasil tersebut kemudian dianalisis untuk mengidentifikasi adanya *trade-off* antara latensi dan efisiensi sumber daya, serta membahas implikasi praktis dari temuan tersebut bagi pengembang.

BAB VI KESIMPULAN DAN SARAN

Bab ini merangkum keseluruhan penelitian dengan menyajikan kesimpulan yang secara langsung menjawab rumusan masalah mengenai perbedaan performa dan

rekomendasi strategis pemilihan API. Diberikan pula saran untuk penelitian di masa depan, yang mencakup pengujian pada skala beban lebih besar, analisis pengaruh latensi jaringan dalam lingkungan terdistribusi, serta eksplorasi strategi optimisasi, seperti *caching* dan *batching*.