Analisis Kinerja Mikroservis dan Monolitik Menggunakan K6 pada Sistem Booking Tutor

M Dheo Fuady[1*], Nandang Sutisna[2]

[1*,2]Teknik Informatika, Sekolah Tinggi Ilmu Komputer Cipta Karya Informatika, Jakarta Timur, Jakarta, Indonesia.
Email: [1*]fuadydheo@gmail.com, [2]nandang.sutisna@gmail.com
[*)] Email Penulis Utama

Abstrak—Pertumbuhan pesat platform pendidikan digital, khususnya sistem *booking tutor*, menuntut arsitektur perangkat lunak yang skabel, andal, dan responsif. Arsitektur monolitik tradisional seringkali menghadapi kendala dalam skalabilitas dan pemeliharaan seiring dengan meningkatnya beban pengguna. Penelitian ini bertujuan untuk menganalisis dan membandingkan secara kuantitatif kinerja antara arsitektur mikroservis dan monolitik dalam konteks sistem booking tutor bahasa Jepang. Untuk mencapai tujuan ini, dua versi sistem identik diimplementasikan: satu menggunakan arsitektur monolitik dan satu lagi dengan arsitektur mikroservis berbasis *event-driven*. Kinerja kedua arsitektur dievaluasi secara ketat menggunakan *tool load testing* K6 pada beban 500 pengguna virtual. Metrik kinerja utama yang diukur meliputi waktu respons dan tingkat kesalahan. Hasil pengujian menunjukkan keunggulan kinerja yang signifikan pada arsitektur mikroservis. Pada operasi kritis seperti pembuatan pembayaran, arsitektur mikroservis mampu mempertahankan waktu respons 399 ms. Sebaliknya, arsitektur monolitik mengalami kegagalan fungsional dengan waktu respons pada endpoint yang sama mencapai 170 detik. Lebih lanjut, arsitektur monolitik menunjukkan ketidakstabilan sistem yang signifikan, dengan tingkat kegagalan puncak mencapai 0.81/s, jauh lebih tinggi dibandingkan tingkat kegagalan tertinggi pada mikroservis yang hanya 0.03/s. Studi ini menyimpulkan bahwa arsitektur mikroservis menawarkan keunggulan kinerja, skalabilitas, dan ketahanan yang jelas dibandingkan arsitektur monolitik untuk aplikasi *booking tutor*. Temuan ini memberikan bukti empiris yang kuat bagi para pengembang dan arsitek sistem dalam memilih fondasi arsitektur yang tepat untuk membangun aplikasi yang tangguh dan siap menghadapi pertumbuhan di masa depan.

Kata Kunci: Arsitektur Mikroservis, Arsitektur Monolitik, *Load Testing*, Waktu Respons, Tingkat Kegagalan

Abstract— *The rapid growth of digital education platforms, particularly tutor booking systems, demands a scalable, reliable, and responsive software architecture. Traditional monolithic architectures often face constraints in scalability and maintenance as user load increases. This study aims to quantitatively analyze and compare the performance of microservices and monolithic architectures within the context of a Japanese language tutor booking system. To achieve this, two identical system versions were implemented: one using a monolithic architecture and the other a microservices architecture based on an event-driven approach. The performance of both architectures was rigorously evaluated using the K6 load testing tool under a load of 500 virtual users. Key performance metrics measured included response time and failure rate.The test results reveal a significant performance advantage for the microservices architecture. In critical operations such as payment creation, the microservices architecture maintained a response time of 399 ms. Conversely, the monolithic architecture experienced a functional failure, with the response time for the same endpoint reaching 170 seconds. Furthermore, the monolithic architecture demonstrated significant system instability, with a peak failure rate reaching 0.81/s, considerably higher than the peak failure rate in the microservices architecture, which was only 0.03/s. This study concludes that the microservices architecture offers clear advantages in performance, scalability, and resilience compared to the monolithic architecture for tutor booking applications. These findings provide robust empirical evidence for developers and system architects in selecting the appropriate architectural foundation to build resilient applications prepared for future growth.*

Keywords: *Microservices Architecture, Monolithic Architecture, Load Testing, Response Time, Failure Rate*

1. PENDAHULUAN

Transformasi digital telah menjadi kekuatan pendorong utama yang membentuk ulang berbagai sektor industri di seluruh dunia, tidak terkecuali sektor pendidikan. Munculnya teknologi internet berkecepatan tinggi dan penetrasi perangkat pintar telah mengakselerasi adopsi platform pembelajaran digital atau *e-learning*. Pasar *e-learning* global telah menunjukkan pertumbuhan eksponensial, dengan proyeksi nilai yang terus meningkat seiring dengan pergeseran preferensi menuju metode pembelajaran yang lebih fleksibel dan dapat diakses kapan saja [1]. Dalam ekosistem ini, platform pemesanan tutor (*tutor booking system*) secara khusus muncul sebagai salah satu segmen dengan pertumbuhan paling pesat. Platform ini menjawab kebutuhan krusial akan pembelajaran yang terpersonalisasi, menghubungkan siswa dengan tutor ahli dari berbagai belahan dunia dalam berbagai disiplin ilmu, termasuk pembelajaran bahasa. Permintaan akan platform pembelajaran bahasa digital, seperti bahasa Jepang, terus meningkat, didorong oleh globalisasi dan minat budaya, yang menuntut ketersediaan sistem yang andal dan mampu melayani ribuan pengguna secara bersamaan [2].

Namun, di balik potensi pertumbuhan yang besar, terdapat tantangan teknis yang signifikan. Sistem pemesanan tutor modern tidak lagi hanya sekadar platform penjadwalan sederhana. Sistem ini telah berevolusi menjadi aplikasi kompleks yang mengintegrasikan berbagai fitur canggih, seperti penjadwalan *real-time*, pemrosesan pembayaran yang aman, perpesanan instan, kelas video interaktif, dan analisis kemajuan belajar siswa. Kompleksitas fungsional ini, ditambah dengan tuntutan untuk melayani volume pengguna yang tinggi dengan konkurensi (*concurrency*) yang dinamis, memberikan tekanan besar pada arsitektur perangkat lunak yang mendasarinya. Kinerja aplikasi, yang diukur dari kecepatan waktu respons (*response time*) dan keandalan (*reliability*), menjadi faktor kritis yang secara langsung memengaruhi pengalaman pengguna (*user experience*) dan tingkat retensi. Keterlambatan sepersekian detik atau kegagalan sistem pada saat jam sibuk dapat menyebabkan frustrasi pengguna dan kerugian bisnis yang signifikan [3].

Secara historis, banyak aplikasi web, termasuk generasi awal platform *e-learning*, dibangun menggunakan **arsitektur monolitik (*monolithic architecture*)**. Pendekatan ini menyatukan semua fungsionalitas aplikasi—mulai dari antarmuka pengguna, logika bisnis, hingga akses data—ke dalam satu basis kode (*codebase*) tunggal yang besar dan terintegrasi erat. Pada tahap awal pengembangan, arsitektur monolitik menawarkan keuntungan dalam hal kesederhanaan penerapan dan pengujian awal. Namun, seiring dengan pertumbuhan aplikasi dan penambahan fitur, pendekatan ini mulai menunjukkan kelemahan yang serius. Kesulitan dalam melakukan *scaling* secara efisien adalah salah satu kendala utama; jika satu fitur (misalnya, pemrosesan pembayaran) mengalami lonjakan beban, keseluruhan aplikasi harus diskalakan, yang mengakibatkan pemborosan sumber daya. Selain itu, pemeliharaan dan pembaruan menjadi sangat rumit dan berisiko tinggi. Perubahan kecil pada satu bagian dapat secara tidak terduga memengaruhi bagian lain, menciptakan siklus regresi yang panjang dan memperlambat laju inovasi. Keterikatan pada satu tumpukan teknologi (*technology stack*) juga menjadi penghambat adopsi teknologi baru yang lebih efisien [4].

Sebagai respons terhadap keterbatasan arsitektur monolitik, industri perangkat lunak telah bergerak menuju **arsitektur mikroservis (*microservices architecture*)**. Pendekatan arsitektural ini menstrukturkan aplikasi sebagai kumpulan layanan-layanan kecil yang independen, di mana setiap layanan bertanggung jawab atas satu kapabilitas bisnis tertentu [5]. Setiap layanan memiliki basis kodenya sendiri, dikelola oleh tim kecil, dan dapat diterapkan, diperbarui, serta diskalakan secara mandiri. Keunggulan utama dari mikroservis adalah peningkatan *scalability*, *resilience* (ketahanan), dan fleksibilitas. Jika layanan penjadwalan membutuhkan lebih banyak sumber daya, hanya layanan tersebut yang perlu diskalakan, tanpa memengaruhi layanan lain seperti autentikasi atau notifikasi. Lebih lanjut, kegagalan pada satu layanan dapat diisolasi sehingga tidak meruntuhkan seluruh aplikasi, sebuah konsep yang dikenal sebagai *fault isolation* [6].

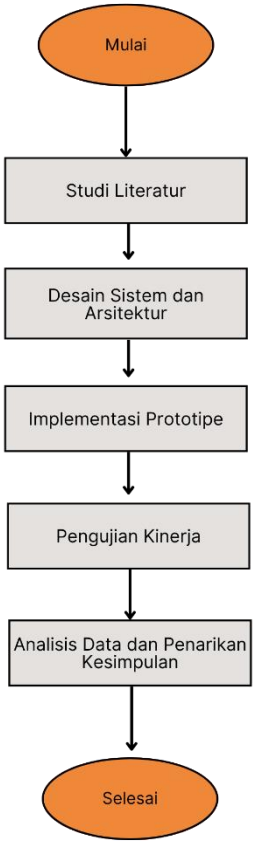
Dalam implementasi arsitektur mikroservis, pola komunikasi antar layanan menjadi aspek desain yang krusial. Komunikasi sinkron melalui *API-driven* (seperti REST API) umum digunakan, namun dapat menciptakan *coupling* (keterikatan) antar layanan yang pada akhirnya dapat membatasi skalabilitas pada beban yang sangat tinggi. Sebagai alternatif yang lebih maju, **arsitektur berbasis peristiwa (*Event-Driven Architecture* atau EDA)** menawarkan pendekatan asinkron yang mempromosikan *loose coupling* secara maksimal [7]. Dalam model EDA, layanan berkomunikasi melalui pertukaran *event* (peristiwa) tanpa perlu mengetahui satu sama lain secara langsung. Ketika sebuah peristiwa signifikan terjadi (misalnya, "pemesanan berhasil dibuat"), sebuah pesan *event* dipublikasikan ke broker pesan. Layanan lain yang tertarik pada *event* tersebut dapat berlangganan dan bereaksi sesuai kebutuhan. Pendekatan ini tidak hanya meningkatkan skalabilitas dan ketahanan sistem tetapi juga membuka peluang untuk fitur-fitur canggih seperti analitik *real-time* dan personalisasi berbasis AI. Meskipun banyak literatur teoretis dan studi kasus yang memuji keunggulan arsitektur mikroservis [8], [9], [10], masih terdapat kebutuhan akan bukti empiris yang lebih kuat melalui analisis kinerja kuantitatif. Banyak studi yang ada berfokus pada domain *e-commerce* atau media sosial [8], sementara data perbandingan kinerja yang spesifik untuk domain *e-learning* atau sistem booking masih terbatas. Selain itu, perbandingan langsung antara aplikasi monolitik dengan aplikasi mikroservis berbasis EDA yang identik secara fungsional dan diuji dalam kondisi beban yang terkontrol secara ketat masih jarang ditemukan. Kesenjangan inilah yang coba diisi oleh penelitian ini[11].

Oleh karena itu, penelitian ini bertujuan untuk menganalisis dan membandingkan secara kuantitatif kinerja antara arsitektur monolitik tradisional dengan arsitektur mikroservis berbasis *event-driven* dalam konteks sistem booking tutor. Untuk memastikan perbandingan yang adil, dua aplikasi fungsional identik dibangun dan diuji menggunakan *tool load testing* K6 untuk mensimulasikan beban 500 pengguna virtual[12]. Kontribusi utama dari penelitian ini adalah penyajian data empiris yang konkret mengenai metrik kinerja utama—yaitu waktu respons dan tingkat kegagalan—dari kedua arsitektur[13]. Temuan dari studi ini diharapkan dapat memberikan panduan praktis dan bukti kuat bagi para pengembang, arsitek sistem, dan pengambil keputusan teknis dalam memilih fondasi arsitektur yang paling tepat untuk membangun platform digital yang tangguh, skabel, dan siap menghadapi tantangan pertumbuhan di masa depan.

2. METODE PENELITIAN

2.1 Kerangka Penelitian

Penelitian ini dilaksanakan secara sistematis melalui serangkaian tahapan yang terstruktur untuk memastikan validitas dan reliabilitas hasil. Setiap tahapan dirancang untuk membangun fondasi bagi tahapan berikutnya, mulai dari perumusan masalah hingga analisis data akhir. Pendekatan eksperimental komparatif yang digunakan dalam studi ini menuntut adanya alur kerja yang jelas dan terkontrol agar perbandingan antara kedua arsitektur dapat dilakukan secara adil dan objektif. Secara visual, keseluruhan alur tahapan penelitian diilustrasikan pada Gambar 1.



Gambar 1. Alur Tahapan Penelitian

Penjelasan lebih rinci mengenai setiap tahapan pada Gambar 1 adalah sebagai berikut:

1. Studi Literatur : Tahap awal dimulai dengan studi literatur yang komprehensif untuk memahami secara mendalam konsep, kelebihan, serta kekurangan dari arsitektur monolitik dan arsitektur mikroservis. Dalam tahap ini, total 20 jurnal ilmiah relevan berhasil dikumpulkan untuk menjadi landasan teoretis dan identifikasi penelitian sebelumnya. Fokus khusus diberikan pada implementasi pola *event-driven* dalam mikroservis. Tinjauan juga dilakukan pada penelitian-penelitian tersebut untuk mengidentifikasi metodologi pengujian yang relevan dan celah penelitian (*research gap*) yang ada.

2. Desain Sistem dan Arsitektur: Berdasarkan studi literatur, tahap kedua adalah perancangan sistem. Pada tahap ini, dilakukan perancangan fungsionalitas sistem booking tutor yang akan diimplementasikan pada kedua arsitektur. Proses ini mencakup pembuatan *activity diagram* untuk alur proses bisnis dan *Entity-Relationship Diagram (ERD)* untuk struktur basis data. Selanjutnya, dirancang dua model arsitektur yang berbeda: arsitektur monolitik dan arsitektur mikroservis.

3. Implementasi Prototipe: Setelah desain arsitektur final, tahap ketiga adalah implementasi kedua prototipe sistem. Kedua sistem dibangun dengan fungsionalitas yang identik untuk memastikan bahwa perbandingan kinerja nantinya murni berdasarkan perbedaan arsitektur. Teknologi yang digunakan dipilih secara cermat agar dapat mendukung kedua model arsitektur dengan baik.

4. Pengujian Kinerja: Tahap keempat merupakan inti dari eksperimen penelitian ini. Lingkungan pengujian disiapkan secara identik untuk kedua prototipe. Pengujian beban (*load testing*) dilakukan menggunakan *tool K6* dengan skenario yang telah ditentukan, yaitu simulasi 500 pengguna virtual selama 5 menit. Selama pengujian, data kinerja seperti waktu respons dan tingkat kegagalan dikumpulkan secara sistematis.

5. Analisis Data dan Penarikan Kesimpulan: Tahap akhir adalah menganalisis data mentah yang diperoleh dari hasil pengujian. Data dari kedua arsitektur ditabulasi dan dibandingkan secara kuantitatif. Berdasarkan analisis ini, ditarik kesimpulan mengenai arsitektur mana yang menunjukkan kinerja, skalabilitas, dan stabilitas yang lebih unggul dalam konteks sistem *booking tutor*.

2.2 Desain Sistem dan Arsitektur

Bagian ini menguraikan perancangan teknis dari kedua sistem yang dibangun untuk penelitian: satu dengan arsitektur monolitik dan satu lagi dengan arsitektur mikroservis. Untuk memastikan perbandingan yang adil (*apple-to-apple comparison),* kedua sistem dikembangkan dengan fungsionalitas, alur proses bisnis, dan struktur basis data yang identik. Perbedaan fundamental antara keduanya hanya terletak pada arsitektur perangkat lunaknya.

2.2.1 Desain Fungsional dan Basis Data

Sistem booking tutor ini dirancang untuk memiliki beberapa fungsionalitas inti yang esensial bagi pengguna, yaitu:

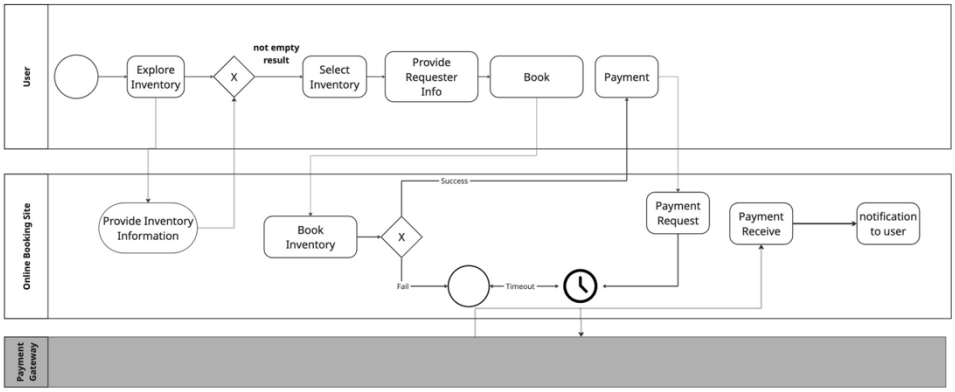
1. Manajemen Pengguna: Mencakup proses registrasi pengguna baru, proses autentikasi (login), dan kemampuan bagi pengguna untuk melihat data profilnya sendiri.

2. Manajemen Guru dan Jadwal: Pengguna dapat melihat daftar guru yang tersedia serta jadwal spesifik dari masing-masing guru.

3. Proses Booking: Pengguna dapat melakukan pemesanan sesi tutor pada jadwal yang tersedia.

4. Proses Pembayaran: Setelah melakukan booking, pengguna dapat melanjutkan ke proses pembayaran dan mendapatkan konfirmasi.

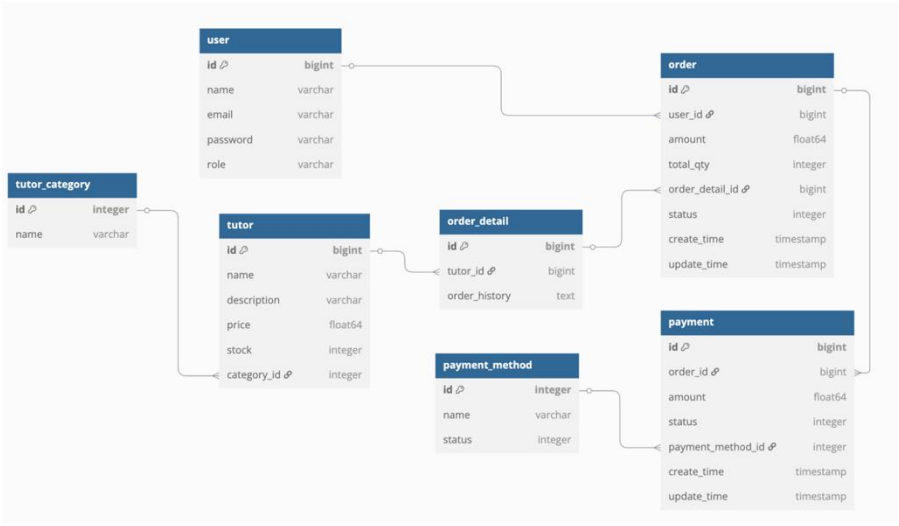
Alur proses bisnis utama, yaitu proses pemesanan oleh pengguna dari awal hingga akhir, dimodelkan menggunakan sebuah activity diagram seperti yang ditunjukkan pada Gambar 2.



Gambar 2. *Acvtivity Diagram*

Diagram tersebut menggambarkan langkah-langkah yang dilalui pengguna mulai dari memilih guru, melihat jadwal, melakukan pemesanan, hingga proses pembayaran berhasil divalidasi oleh sistem. Untuk mendukung fungsionalitas tersebut, struktur data dan hubungan antar entitas untuk keseluruhan sistem dirancang

menggunakan *Entity-Relationship Diagram* (ERD) yang disajikan pada Gambar 3. Skema basis data yang didefinisikan dalam ERD ini diterapkan secara konsisten pada kedua implementasi arsitektur untuk menjaga integritas dan kesamaan data.

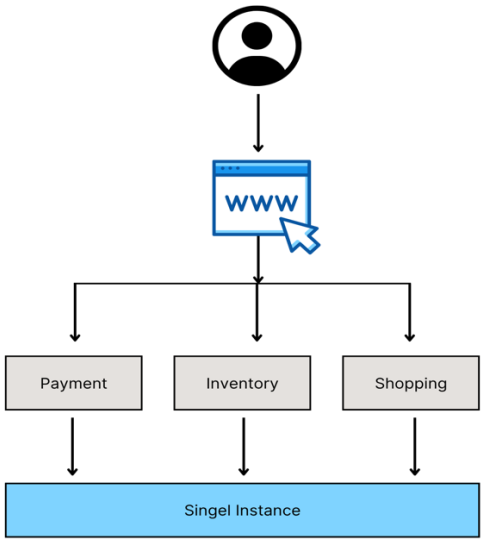


Gambar 3. *Entity-Relationship Diagram* (ERD)

Diagram ERD ini menunjukkan entitas-entitas utama seperti *Users*, *Teachers*, *Schedules*, *Bookings*, dan *Payments* beserta relasi kardinalitasnya yang menjadi fondasi bagi operasi aplikasi.

2.2.2 Arsitektur Monolitik

Arsitektur pertama yang dibangun untuk penelitian ini adalah arsitektur monolitik. Dalam pendekatan ini, seluruh fungsionalitas sistem booking tutor dikembangkan dan digabungkan ke dalam satu unit aplikasi tunggal (*single unit)*. Semua komponen, termasuk logika bisnis untuk manajemen pengguna, katalog guru, proses pemesanan (*shopping*), dan pembayaran (*payment*), berada dalam satu basis kode (*codebase*) yang sama dan dijalankan sebagai satu proses tunggal. Struktur dari arsitektur monolitik yang diimplementasikan divisualisasikan pada Gambar 4.



Gambar 4. Arsitektur Sistem Monolitik

Seperti yang diilustrasikan pada gambar, pengguna mengakses aplikasi melalui antarmuka web (WWW). Permintaan tersebut kemudian diproses oleh sebuah instansi tunggal (*single instance*) yang berisi semua modul

fungsional (*Payment, Inventory, Shopping*). Komponen-komponen ini saling terikat erat (*tightly coupled*), artinya perubahan pada satu modul berpotensi memengaruhi modul lainnya. Seluruh modul juga berbagi sumber daya komputasi yang sama dan terhubung ke satu skema basis data terpusat seperti yang telah dijelaskan pada Gambar 3. Model ini dipilih sebagai representasi arsitektur tradisional yang akan dijadikan dasar perbandingan (*baseline*) dalam eksperimen kinerja.

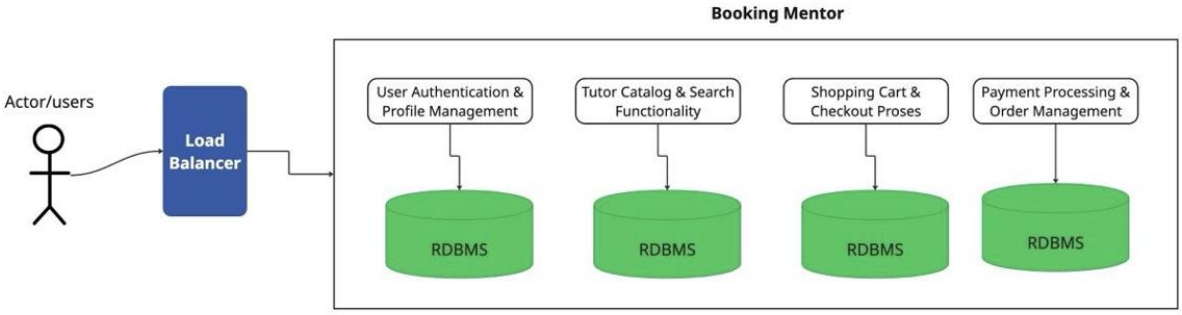
2.2.3 Arsirtektur Mikroservis

Arsitektur kedua yang dibangun dalam penelitian ini adalah arsitektur mikroservis. Pendekatan ini dirancang sebagai solusi untuk mengatasi berbagai kelemahan yang melekat pada arsitektur monolitik, terutama dalam hal skalabilitas, pemeliharaan, dan fleksibilitas. Dalam arsitektur ini, aplikasi booking tutor dipecah menjadi beberapa layanan (services) kecil yang independen, di mana setiap layanan bertanggung jawab penuh atas satu kapabilitas bisnis tunggal.

Layanan-layanan utama yang diimplementasikan meliputi:

1. Layanan Pengguna (User Service): Menangani semua fungsionalitas terkait pengguna, seperti registrasi dan autentikasi.
2. Layanan Guru (Teacher Service): Bertanggung jawab atas data guru dan jadwal.
3. Layanan Pemesanan (Booking Service): Mengelola proses pembuatan dan pengelolaan pesanan.
4. Layanan Pembayaran (Payment Service): Memproses semua transaksi pembayaran.

Untuk mencapai keterikatan yang longgar (loose coupling) dan skalabilitas yang superior, komunikasi antar layanan diimplementasikan menggunakan pola arsitektur berbasis peristiwa (event-driven). Dengan pola ini, layanan tidak saling memanggil secara langsung. Sebaliknya, sebuah layanan akan mempublikasikan event ke sebuah perantara pesan (message broker) ketika sebuah kejadian penting terjadi (contoh: Pesanan_Dibuat). Layanan lain yang berkepentingan dengan kejadian tersebut akan "mendengarkan" event dan bereaksi sesuai dengan fungsinya. Desain arsitektur mikroservis yang diimplementasikan dapat dilihat pada Gambar 5.



Gambar 5. Arsitektur Sistem Mikroservis *Event-Driven*

Gambar 5 mengilustrasikan bagaimana setiap layanan memiliki cakupan tugasnya sendiri. Meskipun dalam praktik terbaik setiap layanan dapat memiliki basis datanya sendiri, untuk tujuan perbandingan yang adil dalam penelitian ini, semua layanan tetap menggunakan skema basis data yang sama (seperti pada Gambar 3). Komunikasi asinkron melalui *event* memungkinkan setiap layanan untuk diskalakan, diperbarui, dan dikelola secara mandiri. Secara teoretis, pendekatan ini menawarkan ketahanan (*resilience*) dan kelincahan (*agility*) yang lebih besar dibandingkan model monolitik.

2.3 Teknologi dan Lingkungan Pengujian

Pemilihan teknologi serta standardisasi lingkungan pengujian merupakan faktor krusial dalam penelitian eksperimental komparatif ini. Untuk memastikan bahwa hasil perbandingan kinerja murni disebabkan oleh perbedaan arsitektural, kedua prototipe sistem (monolitik dan mikroservis) dibangun menggunakan tumpukan teknologi (*tech stack*) yang konsisten dan diuji pada lingkungan perangkat keras (*hardware*) yang sama persis. Spesifikasi lengkap dari perangkat lunak, *library*, dan *framework* yang digunakan dalam implementasi kedua sistem disajikan pada Tabel 1. Teknologi ini dipilih berdasarkan popularitas, dukungan komunitas, dan relevansinya dalam membangun aplikasi web modern yang berkinerja tinggi.

Tabel 1. Spesifikasi Teknologi dan Perangkat Lunak

Kategori	Teknologi	Keterangan
Bahasa Pemrograman	Golang	Bahasa utama untuk logika bisnis kedua aplikasi.
Basis Data	PostgreSQL	Sistem manajemen basis data relasional untuk semua data.
Web framework	Gin	*Framework* untuk membangun layanan API pada aplikasi Go.
Containerization	Docker	Untuk mengemas dan menjalankan aplikasi serta layanannya
Alat Pengujian Beban	K6	*Tool* untuk simulasi beban dan pengujian kinerja.

Pengujian kinerja untuk kedua arsitektur dijalankan pada satu mesin server yang sama untuk mengeliminasi variabilitas perangkat keras. Spesifikasi server yang digunakan dalam penelitian ini adalah sebagai berikut:

1. **CPU:** Intel Core i5
2. **RAM:** 32 GB DDR4
3. **Penyimpanan (*Storage*):** 512 GB NVMe SSD
4. **Sistem Operasi:** Ubuntu 22.04 LTS (Kernel 5.15)

Dengan standarisasi ini, semua perbedaan kinerja yang terukur dapat diatribusikan secara langsung pada kelebihan atau kekurangan dari masing-masing model arsitektur.

2.4 Skenario Pengujian Kinerja

Bagian ini mendeskripsikan protokol pengujian kinerja yang diterapkan secara identik pada kedua arsitektur (monolitik dan mikroservis). Tujuan dari skenario ini adalah untuk mensimulasikan beban pengguna yang signifikan dan mengukur bagaimana masing-masing sistem merespons, sehingga perbandingan yang objektif dapat dilakukan. Pengujian dilakukan secara otomatis menggunakan tool open-source K6. Tool ini dipilih karena kemampuannya dalam mensimulasikan lalu lintas pengguna yang realistis (virtual users) dan menghasilkan metrik kinerja yang akurat. Jenis pengujian yang dilakukan adalah pengujian beban (load testing), di mana sistem diberikan beban konstan dalam periode waktu tertentu untuk mengevaluasi kinerjanya di bawah tekanan.

Parameter utama yang digunakan dalam skenario pengujian ini adalah sebagai berikut:

1. Virtual Users (VUs): 500
2. Durasi (Duration): 5 menit

Skenario ini dirancang untuk mensimulasikan kondisi di mana 500 pengguna mengakses berbagai endpoint sistem secara bersamaan dan terus-menerus selama periode 5 menit. Selama pengujian berlangsung, dua metrik kinerja utama (variabel dependen) diukur, yaitu:

1. Waktu Respons (Response Time): Waktu yang dibutuhkan sistem untuk memproses dan menanggapi sebuah permintaan, diukur dalam milidetik (ms) atau detik (s). Waktu respons yang lebih rendah mengindikasikan kinerja yang lebih baik.
2. Tingkat Kegagalan (Failure Rate): Jumlah permintaan yang gagal diproses oleh sistem per detik (req/s). Tingkat kegagalan yang mendekati nol menunjukkan stabilitas sistem yang tinggi.

Protokol pengujian yang terstandarisasi ini memastikan bahwa data kuantitatif yang dihasilkan bersifat objektif dan dapat diandalkan untuk analisis perbandingan kinerja, yang akan dibahas secara mendalam pada bab selanjutnya.

3. HASIL DAN PEMBAHASAN

Bab ini menyajikan hasil kuantitatif dari pengujian kinerja yang telah dilakukan pada kedua arsitektur dan dilanjutkan dengan pembahasan mendalam untuk menganalisis temuan tersebut. Data yang disajikan menjadi dasar untuk perbandingan objektif antara arsitektur monolitik dan mikroservis dalam konteks penelitian ini.

3.1 Hasil Pengujian Kinerja

Pengujian kinerja dilakukan menggunakan skenario beban 500 *virtual users (VUs)* selama 5 menit. Metrik utama yang dicatat adalah waktu respons dan tingkat kegagalan dari setiap *endpoint* yang diuji. Hasil dari kedua arsitektur disajikan secara terpisah di bawah ini.

3.1.1 Kinerja Arsitektur Mikroservis

Hasil pengujian untuk arsitektur mikroservis menunjukkan kinerja yang bervariasi, di mana beberapa layanan sangat responsif sementara yang lain menunjukkan kelambatan pada proses bisnis yang kompleks. Data kinerja lengkap disajikan pada Tabel 2.

Tabel 2. Data Kuantitatif Arsitektur Mikroservis

Layanan	*Endpoint* Kunci	Waktu Respons	Tingkat Kegagalan
P*ayment*	/payment-methods	4 ms	0/s
Teacher	/schedules?page...	7 ms	0/s
User	/me	8 ms	0/s
Teacher	/schedule/filter-by-teacher	9 ms	0/s
Payment	/payments?page=1&limit=10	10 ms	0/s
Teacher	/schedule/teacher/{userId}...	15 ms	0/s
Teacher	/teachers	49 ms	0/s
Payment	/payments (Create)	399 ms	0/s
Booking	/bookings?page...	881 ms	0/s
Booking	/bookings (Create)	2 detik	0/s
Payment	/payments/callback	3 detik	0/s
User	/login	6 detik	0/s
User	/register	7 detik	0/s
Booking	/bookings/user/{userId} (Detail)	31 detik	0.03/s

Berdasarkan Tabel 2, terlihat bahwa mayoritas endpoint, terutama yang terkait dengan layanan *Teacher* dan operasi baca (read) pada P*ayment*, memiliki waktu respons sangat cepat di bawah 50 ms. Namun, *endpoint* yang melibatkan logika lebih kompleks seperti autentikasi (/login dan /register) serta pengambilan detail *Booking* menunjukkan waktu respons yang signifikan lebih tinggi. Tingkat kegagalan sistem secara keseluruhan sangat rendah, dengan nilai tertinggi hanya 0.03 req/s, menandakan stabilitas sistem yang baik di bawah beban.

3.1.2 Kinerja Arsitektur Monolitik

Selanjutnya, hasil pengujian untuk arsitektur monolitik pada skenario beban yang sama disajikan pada Tabel 3. Data pada Tabel 3 menunjukkan bahwa arsitektur monolitik mengalami degradasi kinerja yang parah di bawah beban pengujian. Dua endpoint mengalami kegagalan fungsional, yaitu /payments (create) dan /bookings/user/{userId} (detail), dengan waktu respons masing-masing 170 detik dan 57 detik. *Endpoint booking detail* tersebut juga mencatatkan tingkat kegagalan yang sangat tinggi sebesar 0.81 req/s, yang mengindikasikan ketidakstabilan sistem. Meskipun beberapa operasi sederhana pada layanan Teacher tetap cepat, mayoritas proses bisnis inti menunjukkan waktu respons yang sangat lambat, berada dalam hitungan detik hingga menit.

Tabel 3. Data Kuantitatif Arsitektur Monolitik

Layanan	*Endpoint* Kunci	Waktu Respons	Tingkat Kegagalan
P*ayment*	/payment-methods	17 ms	0/s
Teacher	/schedules?page...	35 ms	0/s
User	/me	35 ms	0/s
Teacher	/schedule/filter-by-teacher	51 ms	0/s
Payment	/payments?page=1&limit=10	2 detik	0/s
Teacher	/schedule/teacher/{userId}...	61 ms	0/s
Teacher	/teachers	43 ms	0/s
Payment	/payments (Create)	170 detik	0/s
Booking	/bookings?page...	2 detik	0/s
Booking	/bookings (Create)	4 detik	0/s
Payment	/payments/callback	7 detik	0/s
User	/login	8 detik	0/s
User	/register	8 detik	0/s
Booking	/bookings/user/{userId} (Detail)	57 detik	0.81/s

3.2 Pembahasan

Hasil pengujian secara jelas menunjukkan superioritas kinerja dan stabilitas arsitektur mikroservis di bawah beban tinggi. Perbedaan paling drastis terlihat pada operasi kritis seperti pembuatan pembayaran, di mana arsitektur mikroservis merespons dalam 399 ms, sementara arsitektur monolitik mengalami kegagalan fungsional dengan waktu respons mencapai 170 detik. Ketidakmampuan monolitik untuk menangani beban juga terbukti dari tingkat kegagalannya yang mencapai 0.81 req/s, mengindikasikan sistem yang runtuh. Sebaliknya, arsitektur mikroservis tetap stabil dengan tingkat kegagalan maksimal hanya 0.03 req/s, membuktikan kemampuannya untuk tetap beroperasi meskipun beberapa komponennya melambat. Temuan kuantitatif ini memberikan bukti empiris bagi konsep-konsep arsitektural kunci. Arsitektur monolitik menunjukkan efek domino, di mana satu bottleneck memperlambat dan merusak kinerja keseluruhan sistem. Berbeda secara fundamental, arsitektur mikroservis mendemonstrasikan prinsip isolasi kegagalan (fault isolation), di mana kelambatan pada satu layanan (misalnya, Booking Service) tidak memengaruhi responsivitas layanan lain yang independen (misalnya, Teacher Service). Oleh karena itu, dapat disimpulkan bahwa pendekatan mikroservis, khususnya yang berbasis event-driven, menawarkan fondasi yang jauh lebih tangguh, skabel, dan dapat diandalkan untuk aplikasi kompleks yang menargetkan pertumbuhan pengguna di masa depan. Implikasi dari temuan ini juga meluas ke siklus hidup pengembangan dan pemeliharaan. Ketika sebuah area masalah teridentifikasi seperti *Booking Service* tim pengembang pada arsitektur mikroservis dapat melakukan optimasi dan pembaruan secara terfokus hanya pada layanan tersebut. Hal ini mempercepat iterasi pengembangan dan mengurangi risiko. Sebaliknya, perbaikan pada arsitektur monolitik menuntut pengujian regresi yang menyeluruh pada seluruh aplikasi, sebuah proses yang lebih lambat dan berisiko, sehingga menghambat kelincahan (agility) tim dalam merespons kebutuhan pasar.

4. KESIMPULAN

Penelitian ini berangkat dari tantangan fundamental yang dihadapi oleh platform digital modern, khususnya sistem booking tutor, yaitu kebutuhan akan arsitektur perangkat lunak yang mampu mengakomodasi pertumbuhan pengguna dan kompleksitas fitur tanpa mengorbankan kinerja dan keandalan. Arsitektur monolitik yang dominan di masa lalu seringkali menjadi penghambat skalabilitas, sementara arsitektur mikroservis diusulkan sebagai solusi yang lebih fleksibel dan tangguh. Tujuan utama dari penelitian ini adalah untuk menyediakan analisis perbandingan kinerja yang kuantitatif dan objektif antara implementasi arsitektur monolitik tradisional dengan arsitektur mikroservis berbasis event-driven. Untuk mencapai tujuan tersebut, penelitian ini menggunakan metode eksperimental di mana dua prototipe sistem dengan fungsionalitas identik dibangun dan diuji dalam lingkungan yang terkontrol. Pengujian beban secara sistematis dilakukan menggunakan tool K6 dengan skenario simulasi 500 pengguna virtual untuk mengukur metrik kinerja kunci, yaitu waktu respons dan tingkat kegagalan.

Hasil pengujian secara konklusif menunjukkan bahwa arsitektur mikroservis memiliki kinerja yang jauh lebih unggul di bawah kondisi beban tinggi. Bukti paling signifikan terlihat pada operasi-operasi krusial. Sebagai contoh, pada proses pembuatan pembayaran, arsitektur mikroservis berhasil mempertahankan waktu respons pada

399 ms, sementara arsitektur monolitik mengalami kegagalan sistemik dengan waktu respons yang tidak dapat diterima yaitu 170 detik. Dari segi stabilitas, arsitektur monolitik terbukti tidak mampu menangani beban, yang ditandai dengan tingkat kegagalan yang tinggi mencapai 0.81 req/s. Hal ini berbanding terbalik dengan arsitektur mikroservis yang menunjukkan ketahanan luar biasa dengan tingkat kegagalan minimal hanya 0.03 req/s. Temuan ini secara empiris memvalidasi keunggulan teoretis dari arsitektur mikroservis, terutama dalam hal isolasi kegagalan (fault isolation), yang berhasil mencegah penyebaran masalah dari satu layanan ke layanan lainnya, berbeda dengan efek domino yang terlihat jelas pada arsitektur monolitik.

Berdasarkan bukti empiris yang terkumpul, penelitian ini menyimpulkan bahwa arsitektur mikroservis, khususnya dengan pendekatan event-driven, merupakan pilihan yang secara objektif lebih superior dibandingkan arsitektur monolitik untuk pengembangan sistem booking tutor atau aplikasi sejenis yang menargetkan skalabilitas dan keandalan tinggi. Temuan ini memberikan justifikasi berbasis data bagi para arsitek sistem dan tim pengembang untuk mengadopsi arsitektur mikroservis, tidak hanya berdasarkan klaim teoretis, tetapi juga berdasarkan bukti kinerja yang terukur. Implementasi arsitektur ini terbukti mampu menghasilkan sistem yang lebih tangguh, lebih mudah diskalakan, dan pada akhirnya memberikan pengalaman pengguna yang lebih baik.

Penulis menyadari adanya beberapa keterbatasan dalam penelitian ini. Hasil yang diperoleh spesifik untuk tumpukan teknologi (Go, PostgreSQL, RabbitMQ) dan konfigurasi perangkat keras yang digunakan. Kinerja dapat bervariasi dengan teknologi yang berbeda. Oleh karena itu, penelitian di masa depan dapat diarahkan pada beberapa hal. Pertama, melakukan studi perbandingan serupa menggunakan tumpukan teknologi lain (misalnya, Java/Spring, Python/Django) untuk menguji generalisasi temuan. Kedua, membandingkan berbagai pola komunikasi mikroservis (misalnya, event-driven vs. REST vs. gRPC) secara lebih mendalam. Ketiga, melakukan pengujian di lingkungan cloud terdistribusi menggunakan orkestrator seperti Kubernetes untuk menganalisis kinerja fitur auto-scaling secara dinamis. Arah penelitian di masa depan ini dapat lebih memperkaya pemahaman komunitas rekayasa perangkat lunak mengenai desain dan implementasi arsitektur sistem modern yang efisien.

UCAPAN TERIMAKASIH

Ucapan terima kasih terutama saya tujukan kepada semua pihak yang terlibat dalam penelitian ini. Penulis pertama juga ingin mengucapkan terima kasih yang sebesar-besarnya kepada istri dan keluarga saya atas dukungan, doa, serta motivasi yang tiada henti, yang telah menjadi sumber semangat dalam menyelesaikan penelitian ini. Tanpa bantuan dan dorongan dari mereka, penyusunan naskah publikasi ini tidak akan berjalan dengan lancar.

REFERENCES

[1] A. Chavan, "Exploring Event-Driven Architecture in Microservices – Patterns, Pitfalls and Best Practices," *International Journal of Science and Research Archive*, vol. 4, no. 1, pp. 229–249, 2021.

[2] S. Sharma, "The Impact of Microservices Architecture on System Scalability," *American Academic Scientific Research Journal for Engineering, Technology, and Sciences*, vol. 102, pp. 140–148, 2025.

[3] R. B. Y. Christian, "Studi Perbandingan Performa Aplikasi Web Monolitik dan Microservice Berbasis Apache Kafka," *Journal of Informatics and Computer Science*, vol. 3, pp. 79–88, 2021.

[4] E. Ok and J. Eniola, "A Comprehensive Guide to Event-Driven Architecture: Enhancing Microservices with Message Streaming," 2024.

[5] A. R. Kommera, "The Power of Event-Driven Architecture: Enabling Real-Time Systems and Scalable Solutions," *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, vol. 11, no. 1, pp. 1740–1751, 2020.

[6] Wawan Sismadi, Besar Agung Martono, Yodi Susanto, and Amin Muzaeni, "Implementasi Arsitektur Microservices pada Web Aplikasi Penerimaan Mahasiswa Baru," *EDUTECH: Jurnal Inovasi Pendidikan Berbantuan Teknologi*, vol. 4, no. 2, 2024.

[7] Hafi Ihza Farhana, Retno Mumpuni, and Fawwaz Ali Akbar, "Implementasi Arsitektur Mikroservis dan Orkestrasi Kubernetes dengan Paradigma DDD pada Website Freelancing," *CICES (Cyberpreneurship Innovative and Creative Exact and Social Science)*, vol. 11, pp. 10–23, Feb. 2025.

[8] E. D. Giovanni and I. B. K. Manuaba, "Event-driven approach in microservices architecture for flight booking simulation," *ICIC Express Letters*, vol. 16, no. 5, pp. 543–553, 2022.

[9] G. Richards, "Microservices and Event-Driven Architecture: Leveraging Messaging for Distributed Systems," *O'Reilly Media*, 2020.

[10] S. Newman, "Building Microservices: Designing Fine-Grained Systems," *O'Reilly Media*, 2021.

[11] R. S. Pressman and B. R. Maxim, "Software Engineering: A Practitioner's Approach," *McGraw-Hill Education*, 2015.

[12] A. M. Omer, "UML Diagrams for Modeling Microservices Architecture," *Int J Comput Appl*, vol. 177, pp. 1–7, 2020.

[13] N. Dragoni et al., "Microservices: Yesterday, Today, and Tomorrow," *Present and Ulterior Software Engineering*, pp. 195–216, 2017.