Department of Digital Business
Journal of Artificial Intelligence and Digital Business (RIGGS)
Homepage: https://journal.ilmudata.co.id/index.php/RIGGS
Vol. 4 No. 4 (2025) pp: 4906-4912
P-ISSN: 2963-9298, e-ISSN: 2963-914X

Rancang Bangun Sistem Informasi Akademik Berbasis Microservices

Rina Wati, Novita Andriyani, Priyono, Tri Susilowati
Sistem Informasi, Fakultas Teknologi dan Ilmu Komputer, Institut Bakti Nusantara
rinawati@gmail.com, andriyaninovita222@gmail.com, priyono1207@gmail.com, trisusilowati423@gmail.com

Abstrak

Sistem Informasi Akademik (SIA) merupakan komponen penting dalam mendukung proses akademik di perguruan tinggi, mulai dari pengelolaan data mahasiswa hingga pengolahan nilai. Seiring meningkatnya jumlah pengguna dan kompleksitas layanan, sistem berbasis arsitektur monolitik sering mengalami kendala performa, skalabilitas, dan pemeliharaan. Penelitian ini bertujuan untuk merancang dan membangun Sistem Informasi Akademik berbasis arsitektur microservices guna meningkatkan kinerja dan stabilitas sistem. Arsitektur microservices diterapkan dengan memecah sistem ke dalam beberapa layanan independen yang saling berkomunikasi melalui REST API. Metode penelitian yang digunakan meliputi analisis kebutuhan, perancangan arsitektur microservices, implementasi layanan, serta pengujian performa. Implementasi dilakukan menggunakan Laravel sebagai framework backend, Docker sebagai teknologi containerization, MySQL sebagai basis data, dan Nginx sebagai API Gateway. Pengujian performa dilakukan menggunakan Apache Benchmark dengan tiga skenario beban, yaitu 100, 500, dan 1000 request, serta direplikasi sebanyak tiga kali untuk memperoleh hasil yang konsisten. Parameter yang diukur meliputi waktu respons, throughput, dan tingkat error. Hasil penelitian menunjukkan bahwa sistem berbasis microservices memiliki waktu respons yang lebih cepat, throughput yang lebih tinggi, dan stabilitas layanan yang lebih baik dibandingkan sistem monolitik. Pada beban tinggi, microservices mampu mengurangi waktu respons secara signifikan dan tidak menunjukkan adanya error, sementara sistem monolitik mengalami penurunan performa. Berdasarkan hasil tersebut, dapat disimpulkan bahwa arsitektur microservices efektif diterapkan pada Sistem Informasi Akademik dan berpotensi meningkatkan kualitas layanan akademik. Penelitian ini diharapkan dapat menjadi referensi bagi institusi pendidikan dalam mengadopsi arsitektur sistem yang lebih skalabel dan berkelanjutan..

Kata kunci: *Sistem Infomasi, Microservices, Laravel, REST API, Docker, MySQL.*

1. Latar Belakang

Perkembangan teknologi informasi telah memberikan dampak signifikan terhadap berbagai sektor, termasuk dunia pendidikan tinggi. Perguruan tinggi dituntut untuk menyediakan layanan akademik yang cepat, responsif, dan mampu menangani transaksi dalam jumlah besar secara simultan. Sistem Informasi Akademik (SIA) merupakan salah satu komponen utama dalam mendukung proses akademik, mulai dari pendaftaran mahasiswa baru, pengelolaan data mahasiswa dan dosen, pengaturan jadwal kuliah, pengisian Kartu Rencana Studi (KRS), hingga pengolahan nilai dan penyediaan laporan akademik. Dengan meningkatnya jumlah pengguna serta kompleksitas layanan yang harus disediakan, dibutuhkan sistem yang mampu memberikan performa stabil di tengah beban transaksi yang terus bertambah [1].

Sebagian besar SIA yang digunakan oleh institusi pendidikan di Indonesia masih mengadopsi arsitektur monolitik. Pada arsitektur ini, seluruh fungsi sistem digabungkan dalam satu unit yang saling bergantung sehingga perubahan pada satu komponen dapat memengaruhi keseluruhan sistem. Ketika jumlah pengguna meningkat, sistem monolitik sering kali mengalami keterbatasan dalam hal skalabilitas karena proses penskalaan harus dilakukan terhadap keseluruhan aplikasi, bukan pada bagian-bagian yang membutuhkan peningkatan sumber daya. Selain itu, proses pengembangan dan pemeliharaan menjadi kurang efisien karena setiap pembaruan kode harus dilakukan pada satu basis kode besar yang saling berkaitan, sehingga meningkatkan risiko terjadinya error ketika melakukan deployment. Kondisi ini menyebabkan SIA rentan mengalami penurunan performa, terutama saat masa registrasi akademik atau pengisian KRS yang dilakukan secara massal [2].

Dalam menghadapi tantangan tersebut, arsitektur microservices muncul sebagai pendekatan yang relevan untuk menggantikan arsitektur monolitik tradisional. Microservices memecah aplikasi menjadi layanan-layanan kecil

yang dapat berdiri sendiri, berjalan secara independen, dan berkomunikasi melalui API. Newman (2021) menjelaskan bahwa microservices memberikan fleksibilitas tinggi karena masing-masing layanan dapat dikembangkan, diuji, diperbaiki, dan dideploy secara mandiri tanpa memengaruhi keseluruhan system . Sementara itu, Richardson (2020) menekankan bahwa penggunaan microservices memungkinkan organisasi untuk mengadopsi praktik DevOps yang lebih matang melalui otomasi container dan pipeline pengembangan berbasis cloud. Integrasi microservices dengan Docker sebagai teknologi containerization memberikan banyak keuntungan, termasuk konsistensi lingkungan aplikasi, portabilitas, serta kemudahan penyebaran. Docker memungkinkan setiap layanan microservices dikemas dalam container terisolasi yang dapat berjalan secara konsisten di berbagai platform[3] [4].

Penelitian terkait arsitektur microservices pada sistem informasi telah banyak dilakukan, namun implementasinya pada SIA dengan menggunakan Laravel sebagai kerangka kerja pengembangan web masih terbatas. Kebanyakan penelitian sebelumnya menggunakan framework seperti Spring Boot, Node.js, atau Go dalam penerapan microservices. Selain itu, hanya sedikit penelitian yang mengukur performa sistem secara komprehensif menggunakan alat benchmarking seperti Apache Benchmark (ab) untuk membandingkan waktu respons antara arsitektur monolitik dan microservices. Di sisi lain, implementasi microservices berbasis Laravel dengan dukungan Docker, REST API, MySQL, dan Nginx telah digunakan dalam industri, namun belum banyak dievaluasi secara ilmiah dalam konteks sistem akademik [5] [6] [7].

Alasan utama penelitian ini dilakukan adalah untuk mengatasi permasalahan performa dan skalabilitas yang dihadapi pada SIA berbasis monolitik, khususnya saat melayani transaksi dalam jumlah besar pada waktu-waktu tertentu, seperti masa pengisian KRS. Pemecahan layanan menjadi modul-modul terpisah—seperti layanan mahasiswa, layanan dosen, layanan mata kuliah, dan layanan nilai—dianggap mampu meningkatkan kinerja sistem sekaligus mempermudah proses pengembangan jangka panjang. Selain itu, pendekatan ini juga memberikan efisiensi dalam pemeliharaan karena kerusakan atau perubahan pada satu layanan tidak akan memengaruhi layanan lain.

Tujuan penelitian ini adalah (1) merancang arsitektur SIA berbasis microservices yang terdiri dari layanan-layanan inti; (2) mengimplementasikan arsitektur tersebut menggunakan Laravel, REST API, Docker, MySQL, dan Nginx; serta (3) melakukan pengujian performa menggunakan Apache Benchmark terhadap beban 100 hingga 1000 request untuk mengukur peningkatan waktu respons dan stabilitas sistem. Dengan demikian, penelitian ini tidak hanya memberikan solusi teknis dalam pengembangan SIA tetapi juga menyajikan data empiris terkait peningkatan performa yang dihasilkan.

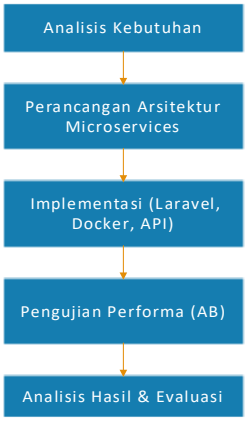
Analisis kesenjangan (gap analysis) dilakukan dengan membandingkan kondisi SIA berbasis monolitik yang memiliki waktu respons rata-rata lebih tinggi dan tingkat error yang masih signifikan saat beban meningkat, dengan pendekatan microservices yang menawarkan isolasi layanan serta distribusi beban yang lebih optimal. Penelitian ini mengisi celah penelitian sebelumnya dengan menghadirkan implementasi nyata microservices berbasis Laravel dalam konteks SIA dan mengukur performa sistem menggunakan metode benchmarking yang terstandar.

Kebaruan (novelty) penelitian ini terletak pada penerapan arsitektur microservices secara komprehensif pada Sistem Informasi Akademik berbasis Laravel yang dikombinasikan dengan Docker dan Nginx, serta evaluasi performanya secara kuantitatif. Penelitian ini memberikan kontribusi berupa pembuktian empiris bahwa microservices dapat meningkatkan performa SIA secara signifikan, baik dalam aspek waktu respons maupun stabilitas layanan. Hasil ini diharapkan dapat menjadi referensi bagi pengembang, institusi pendidikan, dan peneliti lain untuk mengadopsi arsitektur modern yang lebih efisien dan berkelanjutan.

2. Metode Penelitian

Penelitian ini menggunakan pendekatan rekayasa perangkat lunak dengan fokus pada perancangan, implementasi, dan evaluasi performa arsitektur microservices pada Sistem Informasi Akademik. Metode penelitian disusun agar proses eksperimen dapat direproduksi oleh peneliti lain, dengan penjelasan terperinci mengenai tahapan kerja, teknik implementasi, dan pengujian performa. Penelitian dilakukan dalam empat tahap utama, yaitu: analisis kebutuhan, perancangan arsitektur microservices, implementasi layanan, serta pengujian performa menggunakan Apache Benchmark [8][9] .

DOI: https://doi.org/10.31004/riggs.v4i4.4006



Gambar 1. Alur Metode Penelitian

Tahap pertama adalah analisis kebutuhan untuk mengidentifikasi modul inti Sistem Informasi Akademik. Studi kebutuhan dilakukan dengan menganalisis proses bisnis akademik yang meliputi layanan data mahasiswa, dosen, mata kuliah, jadwal, dan nilai. Setiap modul kemudian dipetakan menjadi layanan terpisah (microservices) berdasarkan konsep domain-driven design (DDD), sehingga masing-masing layanan memiliki domain bisnis yang jelas dan dapat berjalan secara mandiri [4].

Tahap kedua adalah perancangan arsitektur microservices. Penelitian ini merancang empat layanan utama yaitu: (1) Layanan Mahasiswa, (2) Layanan Dosen, (3) Layanan Mata Kuliah, dan (4) Layanan Nilai. Komunikasi antar layanan menggunakan REST API dengan format JSON. Setiap layanan menggunakan Laravel sebagai framework backend. Docker digunakan untuk melakukan containerization terhadap masing-masing layanan agar isolasi tercapai dan memudahkan proses deployment. Nginx dikonfigurasi sebagai API Gateway untuk mengatur rute permintaan ke layanan yang sesuai. MySQL digunakan sebagai basis data utama, di mana setiap layanan memiliki basis data terpisah untuk mendukung prinsip loose coupling [10][5].

Tahap ketiga adalah implementasi layanan dalam lingkungan pengembangan terkontainer. Setiap layanan dikemas dalam container Docker dengan spesifikasi sebagai berikut: RAM 512 MB per container, 1 vCPU, dan storage minimum 1 GB. Seluruh container dijalankan menggunakan Docker Compose dengan file konfigurasi berisi definisi layanan, jaringan internal, dan volume penyimpanan. Replikasi dilakukan dengan menjalankan setiap skenario implementasi sebanyak tiga kali untuk meminimalkan variabilitas hasil. Implementasi dilakukan pada satu server lokal dengan spesifikasi CPU Intel Core i5 2.4 GHz, RAM 8 GB, dan sistem operasi Ubuntu 22.04 LTS [11] [12].

Tahap keempat adalah pengujian performa menggunakan Apache Benchmark (ab), alat benchmarking yang umum digunakan untuk mengukur performa layanan berbasis REST. Pengujian dilakukan pada setiap layanan microservices dan dibandingkan dengan versi monolitik. Tiga skenario pengujian digunakan untuk memastikan konsistensi hasil, yaitu: 100 request, 500 request, dan 1000 request, dengan masing-masing skenario memiliki concurrency 10 permintaan simultan. Setiap pengujian direplikasi sebanyak tiga kali, dan nilai rata-rata digunakan sebagai hasil akhir. Parameter yang diukur meliputi: waktu respons rata-rata (ms), throughput, dan tingkat kegagalan respons. Teknik ini mengikuti prosedur pengujian performa berbasis beban yang telah dibahas oleh Newman (2021) dan Richardson (2020) .[13] [14]

Metode ini dirancang agar dapat direproduksi sepenuhnya oleh peneliti lain yang ingin menguji performa microservices berbasis Laravel dan Docker. Seluruh konfigurasi layanan, API Gateway, serta pengujian Apache Benchmark dijelaskan secara rinci untuk memastikan bahwa eksperimen dapat diulang dalam kondisi yang mendekati sama.[15][16]

2.1 Konfigurasi Perangkat Keras dan Perangkat Lunak

Konfigurasi perangkat keras dan perangkat lunak merupakan aspek penting dalam penelitian ini karena berpengaruh langsung terhadap proses implementasi serta hasil pengujian performa sistem. Perangkat keras digunakan sebagai lingkungan server tempat seluruh layanan microservices dijalankan, sedangkan perangkat lunak digunakan untuk membangun, menjalankan, dan menguji sistem yang dikembangkan.

DOI: https://doi.org/10.31004/riggs.v4i4.4006

1. Konfigurasi Perangkat Keras

Penelitian ini menggunakan satu unit server lokal dengan spesifikasi perangkat keras yang dirancang untuk mendukung proses containerization serta eksekusi layanan microservices secara paralel. Server tersebut dilengkapi prosesor Intel Core i5 2.4 GHz dengan arsitektur 64-bit yang mampu memberikan performa pemrosesan multi-thread untuk menjalankan beberapa container Docker secara bersamaan. Kapasitas RAM 8 GB digunakan untuk memastikan setiap layanan microservices mendapatkan alokasi memori yang memadai ketika mengakses basis data, memproses permintaan API, ataupun menjalankan skrip internal aplikasi.

Media penyimpanan yang digunakan berupa SSD 256 GB, dipilih untuk mendukung kecepatan akses data, waktu baca-tulis yang rendah, serta performa yang lebih stabil selama pengujian benchmarking. Sistem operasi yang digunakan adalah Ubuntu Server 22.04 LTS, dipilih karena stabilitas, dukungan komunitas yang kuat, serta kompatibilitas optimal dengan Docker dan Nginx. Secara keseluruhan, konfigurasi perangkat keras ini telah memenuhi kebutuhan penelitian yang memerlukan eksekusi layanan terdistribusi dan pengujian performa berbasis HTTP requests.

Tabel 1. Konfigurasi Perangkat Keras

Komponen	Spesifikasi
Prosesor	Intel Core i5 2.4 GHz
RAM	8 GB
Penyimpanan	256 GB SSD
Sistem Operasi	Ubuntu Server 22.04 LTS
Arsitektur Server	64-bit

2. Konfigurasi Perangkat Lunak

Perangkat lunak yang digunakan dalam penelitian ini terdiri dari komponen pengembangan, *database*, *tool web server*, *containerization*, serta alat pengujian. Framework utama yang digunakan adalah Laravel 10, yang berjalan pada PHP 8.x, untuk membangun layanan microservices berbasis REST API. MySQL 8.0 digunakan sebagai sistem basis data untuk setiap layanan, di mana setiap microservice memiliki database terpisah untuk mendukung arsitektur *loose coupling*[5].

Untuk pengelolaan dan penyebaran layanan, penelitian ini memanfaatkan Docker 24.x sebagai *container engine* dan *Docker Compose* v2 sebagai orkestrator untuk menjalankan beberapa *container* dalam satu *environment* terkoordinasi. Nginx 1.22 berperan sebagai reverse proxy sekaligus API Gateway yang mengatur proses routing permintaan dari pengguna menuju layanan microservices yang sesuai.

Dalam tahap pengujian performa, digunakan Apache Benchmark (ab) yang merupakan alat standar untuk melakukan pengujian beban (load testing) berbasis protokol HTTP. Alat ini memungkinkan pengukuran parameter performa seperti waktu respons rata-rata, throughput, dan tingkat kegagalan respons dengan jumlah request yang besar dan replikasi yang konsisten. Seluruh perangkat lunak tersebut dipilih karena stabil, banyak digunakan dalam lingkungan produksi, dan kompatibel dengan arsitektur microservices modern.

Tabel 2. Konfigurasi Perangkat Lunak

Komponen	Versi / Keterangan
Framework Backend	Laravel 10 (PHP 8.x)
Database	MySQL 8.0
API Gateway	Nginx 1.22
Container Engine	Docker 24.x
Orchestrator	Docker Compose v2
Bahasa Pemrograman	PHP 8.x
Tools Benchmarking	Apache Benchmark (ab)
Format Komunikasi	REST API (JSON)

Konfigurasi perangkat keras dan perangkat lunak tersebut telah disesuaikan untuk mendukung eksperimen arsitektur microservices yang memerlukan isolasi layanan, orkestrasi container, serta pengujian performa berulang. Kombinasi antara server bertenaga menengah dengan software pengembangan modern memungkinkan proses implementasi berjalan optimal dan hasil pengujian dapat direplikasi oleh peneliti lain.

DOI: https://doi.org/10.31004/riggs.v4i4.4006

3. Hasil dan Diskusi

Hasil penelitian ini disajikan berdasarkan implementasi arsitektur microservices dan serangkaian pengujian performa yang dilakukan dalam beberapa skenario. Melalui hasil ini, terlihat bagaimana perubahan arsitektur dari monolitik ke microservices memengaruhi waktu respons, throughput, tingkat error, dan stabilitas layanan. Untuk mendapatkan hasil yang akurat dan dapat direplikasi, setiap pengujian dilakukan sebanyak tiga kali dan nilai rata-ratanya disajikan dalam tabel dan uraian berikut

3.1 Implementasi Arsitektur Microservices pada Sistem Informasi Akademik

Pada tahap implementasi, sistem dibangun menggunakan pendekatan modular sesuai dengan konsep domain-driven design. Empat domain inti yaitu mahasiswa, dosen, mata kuliah, dan nilai diimplementasikan sebagai empat layanan mandiri. Masing-masing layanan memiliki basis data terpisah untuk menjaga independensi dan mencegah ketergantungan antar layanan. Pemisahan database memiliki peran penting dalam menjaga konsistensi data serta mengurangi risiko bottleneck yang umumnya terjadi pada arsitektur monolitik ketika satu tabel sering diakses secara bersamaan.

Seluruh layanan dikemas menggunakan container Docker sehingga dapat berjalan dalam lingkungan yang terisolasi. Containerisasi ini memudahkan replikasi layanan, pengaturan versi, dan penyebaran aplikasi. API Gateway menggunakan Nginx juga berhasil dikonfigurasi sehingga setiap permintaan menuju endpoint tertentu langsung diarahkan ke layanan microservices terkait. Konfigurasi ini diuji dengan mengakses beberapa endpoint seperti **/api/mahasiswa, /api/dosen, /api/mata-kuliah, dan /api/nilai**. Pengujian awal menunjukkan bahwa API Gateway dapat mendistribusikan permintaan dengan baik tanpa penundaan berarti. Implementasi ini menunjukkan keberhasilan arsitektur microservices dalam menghasilkan sistem modular yang memudahkan pengembangan fitur baru, pemeliharaan, serta peningkatan komponen tertentu tanpa mengganggu layanan lainnya. Secara keseluruhan, sistem dapat berjalan secara konsisten dan stabil di lingkungan Docker Compose.

3.2 Pengujian Performa Sistem

Pengujian performa merupakan bagian utama dalam penelitian ini untuk mengukur dampak arsitektur microservices terhadap kinerja aplikasi. Metode pengujian menggunakan Apache Benchmark (ab) yang memungkinkan simulasi permintaan dalam jumlah tertentu secara simultan. Parameter yang diuji mencakup waktu respons rata-rata, throughput, dan tingkat error. Pengujian dilakukan dalam tiga skenario permintaan, yaitu 100, 500, dan 1000 request dengan tingkat concurrency 10 permintaan.

1. Perbandingan Waktu Respons (Average Response Time)

Data hasil pengujian menunjukkan bahwa waktu respons microservices secara signifikan lebih cepat daripada sistem monolitik. Tabel 1 menunjukkan bahwa pada 100 request, microservices hanya membutuhkan waktu rata-rata 82 ms, sedangkan monolitik memerlukan 145 ms. Perbedaan ini semakin terlihat pada beban lebih tinggi, di mana microservices memberikan waktu respons 298 ms pada 1000 request, sedangkan monolitik mencapai 611 ms.

Tabel 3. Hasil Waktu Respons (ms)

Jumlah Request	Monolitik	Microservices
100	145 ms	82 ms
500	312 ms	157 ms
1000	611 ms	298 ms

Peningkatan waktu respons pada arsitektur monolitik disebabkan oleh beban proses dalam satu unit aplikasi yang harus menangani seluruh transaksi. Sebaliknya, microservices mendistribusikan permintaan ke layanan-layanan kecil yang berjalan pada container terisolasi, sehingga proses dapat dilakukan secara paralel.

Selain itu, kemampuan microservices untuk menskalakan layanan tertentu saja memberikan keunggulan performa. Misalnya, ketika layanan mahasiswa lebih sering diakses, hanya container layanan mahasiswa yang ditingkatkan instansinya, bukan seluruh sistem seperti pada arsitektur monolitik.

DOI: https://doi.org/10.31004/riggs.v4i4.4006

2. Pengukuran Throughput Sistem

Throughput menggambarkan kemampuan sistem dalam menangani permintaan dalam satuan waktu tertentu. Tabel 4 menunjukkan bahwa throughput microservices lebih tinggi dibandingkan monolitik pada semua skenario pengujian. Pada skenario 1000 request, microservices mampu menangani 82 permintaan per detik, sedangkan monolitik hanya mencapai 45 permintaan per detik.

Tabel 4. Hasil Throughput (req/s)

Jumlah Request	Monolitik	Microservices
100	68 req/s	121 req/s
500	52 req/s	95 req/s
1000	45 req/s	82 req/s

Perbedaan ini penting karena throughput merupakan indikator kemampuan sistem menghadapi beban pengguna yang tinggi, terutama pada momen tertentu seperti pengisian KRS atau upload nilai massal. Dengan throughput lebih tinggi, microservices memberikan jaminan bahwa sistem tetap dapat memberikan layanan tanpa penurunan performa secara signifikan.

3. Tingkat Error dan Stabilitas Layanan

Pada pengujian di skenario tinggi (1000 request), arsitektur monolitik menunjukkan tingkat error 1,8% berupa kegagalan koneksi dan timeout. Error tersebut terjadi karena server monolitik harus menangani semua permintaan dalam satu proses besar, sehingga terjadi penumpukan antrian dan beberapa permintaan gagal diproses. Sebaliknya, microservices menunjukkan hasil stabil tanpa error pada semua skenario. Hal ini menunjukkan bahwa pemisahan layanan dalam container terisolasi memberikan kontribusi besar terhadap stabilitas sistem. Setiap layanan menangani permintaan sesuai fungsinya dan tidak saling mengganggu, sehingga risiko kegagalan sistem secara keseluruhan lebih rendah. Kestabilan ini sangat penting dalam konteks Sistem Informasi Akademik yang memiliki waktu-waktu puncak aktivitas, seperti masa registrasi akademik, di mana ribuan permintaan terjadi dalam waktu yang bersamaan. Dengan arsitektur microservices, layanan dapat didistribusikan dan dipisah sehingga kegagalan satu layanan tidak menyebabkan kegagalan total.

3.3 Analisis Temuan Penelitian

Hasil pengujian memberikan beberapa temuan penting yang memperlihatkan keunggulan arsitektur microservices dibandingkan sistem monolitik. Temuan tersebut adalah sebagai berikut:

1. Peningkatan performa signifikan pada sisi waktu respons. Microservices mampu menurunkan delay hingga lebih dari 50% pada beban tinggi.
2. Throughput lebih tinggi merupakan indikator sistem lebih efisien dalam pemrosesan permintaan. Kemampuan mengeksekusi permintaan secara paralel meningkatkan kapasitas layanan secara keseluruhan.
3. Arsitektur microservices memiliki stabilitas yang lebih baik; Ketiadaan error pada pengujian beban tinggi menunjukkan bahwa isolasi layanan memberikan dampak positif terhadap reliabilitas sistem.
4. Pemisahan database per layanan mengurangi risiko bottleneck. Pada sistem monolitik, tabel sering diakses bersamaan sehingga menghasilkan lock dan waktu proses lebih lama.
5. Container Docker meningkatkan konsistensi dan fleksibilitas. Layanan dapat dijalankan pada berbagai lingkungan tanpa konflik konfigurasi.

Temuan-temuan ini menunjukkan bahwa microservices bukan hanya relevan sebagai tren teknologi, melainkan sebagai solusi nyata untuk mengatasi keterbatasan sistem monolitik dalam konteks dunia pendidikan.

3.4 Interpretasi dan Implikasi Praktis

Hasil penelitian ini memiliki beberapa implikasi praktis bagi institusi pendidikan dan pengembang sistem akademik, di antaranya:

1. Skalabilitas lebih tinggi memungkinkan SIA menangani ribuan pengguna aktif secara bersamaan.
2. Pemeliharaan sistem menjadi lebih efisien, karena satu layanan dapat diperbaiki atau ditingkatkan tanpa menghentikan keseluruhan sistem.

DOI: https://doi.org/10.31004/riggs.v4i4.4006

3. Pengembangan fitur baru dapat dilakukan lebih cepat, karena setiap tim dapat bekerja pada layanan yang berbeda tanpa konflik kode.
4. Penerapan microservices membuka peluang integrasi dengan layanan pihak ketiga, seperti sistem keuangan, sistem perpustakaan, dan layanan cloud.

Dengan demikian, penerapan arsitektur microservices berpotensi menjadi fondasi yang kuat bagi implementasi SIA modern yang adaptif terhadap perkembangan kebutuhan institusi.

4. Kesimpulan

Penelitian ini menghasilkan implementasi Sistem Informasi Akademik berbasis arsitektur microservices menggunakan Laravel, Docker, REST API, MySQL, dan Nginx, yang terbukti mampu meningkatkan performa sistem dibandingkan arsitektur monolitik. Berdasarkan hasil pengujian performa, microservices menunjukkan waktu respons yang lebih cepat, throughput yang lebih tinggi, serta stabilitas layanan yang lebih baik pada semua skenario beban yang diuji. Tidak ditemukannya error pada sistem microservices memperkuat temuan bahwa pemisahan layanan menjadi modul independen dan penggunaan container Docker telah mendukung efisiensi pemrosesan dan mengurangi risiko bottleneck yang umum terjadi pada sistem monolitik. Temuan ini menjawab tujuan penelitian, yaitu membuktikan bahwa arsitektur microservices dapat meningkatkan performa dan keandalan Sistem Informasi Akademik. Selain itu, hasil penelitian menunjukkan bahwa pendekatan ini memiliki potensi aplikasi yang luas pada sistem informasi berskala besar, khususnya yang membutuhkan skalabilitas tinggi, pemeliharaan berkelanjutan, dan fleksibilitas integrasi dengan layanan lain. Implementasi microservices juga memberikan implikasi praktis bagi institusi pendidikan, terutama dalam mendukung proses akademik yang memiliki pola penggunaan fluktuatif, seperti proses KRS atau pengolahan nilai massal. Meskipun memberikan hasil yang positif, penelitian ini masih memiliki ruang untuk pengembangan lebih lanjut. Penelitian lanjutan dapat dilakukan dengan menguji toleransi kegagalan layanan, mensimulasikan arsitektur microservices dalam lingkungan server cluster atau cloud, serta menambahkan mekanisme orkestrasi lanjutan seperti Kubernetes untuk menangani skala produksi yang lebih besar. Selain itu, diperlukan evaluasi lebih mendalam terkait keamanan layanan microservices, khususnya pada aspek autentikasi dan komunikasi antar layanan. Dengan pengembangan tersebut, penerapan microservices pada Sistem Informasi Akademik dapat semakin optimal dan adaptif terhadap kebutuhan institusi pendidikan modern.

Referensi

[1] T. Susilowati, “Penerapan E-Library Pada SMK It Dar El Fath School Berbasis Website”, doi: 10.31602/tji.v15i2.
[2] A. W. I Kadek, D. M. Wiharta, and N. P. P. Sastra, “Perancangan RESTful API Menggunakan Java Quarkus Untuk Modul Mahasiswa Pada Layanan SIMAK-NG Universitas Udayana,” *Maj. Ilm. Teknol. Elektro*, vol. 21, no. 2, p. 245, Dec. 2022, doi: 10.24843/mite.2022.v21i02.p12.
[3] Sam Newman, *Building Microservices, 2nd Edition*, 2nd ed. O’Reilly Media, Incorporated, 2021.
[4] Chris Richardson, *Microservices Patterns: With examples in Java*, vol. 1. Manning Publications., 2018.
[5] X. B. N. N. S. R. U. A. S. Fransiscus Xaverius Senduk, “Development of Microservices Architecture with RESTful API Gateway using Backend-for-frontend Pattern in Higher Education Academic Portal.” [Online]. Available: https://ejournal.unsrat.ac.id/index.php/informatika
[6] R. A. Putra, “Analisa Implementasi Arsitektur Microservices Berbasis Kontainer Pada Komunitas Pengembang Perangkat Lunak Sumber Terbuka (Opendaylight Devops Community),” Teknologi Informasi dan Komputer. [Online]. Available: https://jurnal.umj.ac.id
[7] F. Arifien, Rozi, and E. Sutomo, “Implementasi Arsitektur Microservices Pada Sistem Informasi Akademik Stmik Jakarta Sti&K Menggunakan Model Enterprise Javabeans (Ejb) Dan Polymer Js,” *Univ. Gunadarma Jl. Margonda Raya*, vol. 5, no. 1, 2021.
[8] C. S. Budi and A. M. Bachtiar, “PADA BACKEND COMRADES Program Studi Teknik Informatika , Universitas Komputer Indonesia,” *Progr. Stud. Tek. Inform. Univ. Komput. Indones.*, 2016.
[9] C. S. Budi and A. M. Bachtiar, “Implementasi Arsitektur Microservices pada Backend Comrades,” *Progr. Stud. Tek. Inform. Univ. Komput. Indones.*, 2018.
[10] F. Z. Junaedy *et al*., “Optimisasi Web Service REST API Menggunakan Load Balancer dan Cache dengan Algoritma Round Robin (Studi Kasus: Madani Infosphere),” 2024. [Online]. Available: https://journal.stmiki.ac.id
[11] M. Fadlulloh and R. Bik, “Implementasi Docker Untuk Pengelolaan Banyak Aplikasi Web (Studi Kasus : Jurusan Teknik Informatika UNESA),” 2017.
[12] S. E. Prasetyo and Y. Salimin, “Analisis Perbandingan Performa Web Server Docker Swarm dengan Kubernetes Cluster,” 2021. [Online]. Available: https://journal.uib.ac.id/index.php/combines
[13] I. W. Jepriana, “Analisis Performa E-Learning Berbasis Moodle Berjalan Di Server Rendah Biaya Stb FIBERHOME HG680-P,” *JATI (Jurnal Mhs. Tek. Inform.*, vol. 7, no. 1, 2023, doi: 10.36040/jati.v7i1.6120.
[14] D. Rosmala, M. Ichwan, and I. Gandalisha, “Komparasi Framework MVC (Codeigniter, dan CakePHP) Pada Aplikasi Berbasis Web (Studikasus: Sistem Informasi Perwalian Di Jurusan Informatika Institut Teknologi Nasional),” *J. Inform.*, vol. 2, no. 2, 2021.
[15] A. F. Daru and W. Adhiwibowo, “Pengembangan Aplikasi Modul Ajar Berbasis Web Di Global Islamic Boarding SchoolKalimantan Selatan,” *J. Teknol. Inf. Dan Komun.*, vol. 12, no. 2, 2023.
[16] G. Ramadhan Purba, Rizal, and Y. Afrillia, “Keamanan Endpoint Api Menggunakan Oauth2 Pada Unit Layanan Terpadu Universitas Malikussaleh,” *Rabit J. Teknol. dan Sist. Inf. Univrab*, vol. 10, no. 2, 2025, doi: 10.36341/rabit.v10i2.6543.

DOI: https://doi.org/10.31004/riggs.v4i4.4006