Analisis Uji Performa Aplikasi Dari Hasil Implementasi Refactoring Arsitektur Monolitik Ke Mikroservis dengan Decomposition dan Strangler Pattern

Riyanto
Computer Science
Institut Pertanian Bogor
Bogor, Indonesia
ryanmanriyanto@apps.ipb.ac.id

Irman Hermadi
Computer Science
Institut Pertanian Bogor
Bogor, Indonesia
rmanhermadi@apps.ipb.ac.id

Yani Nurhadryani [3]
Computer Science
Institut Pertanian Bogor
Bogor, Indonesia
yani_nurhadryani@apps.ipb.ac.id

Abstract—The SmartCampus application is still built with a monolithic architecture, where all components are tightly integrated into one unit. The increasing complexity of user scalability and service demands within the information system with a monolithic architecture is evident in the application's declining performance. In this research, a performance analysis is conducted by implementing refactoring from a monolithic architecture to microservices using the decomposition and strangler patterns. The Decomposition pattern divides the monolithic application into several business domains based on their main service categories, while the strangler pattern breaks down the business domains into microservices by replacing specific functions with new services through the stages of transform, co-exist, and eliminate. Once the new functionalities are ready, the old components are deactivated, and the new services are put into operation. The application's feasibility and quality considerations are assessed using the ISO/IEC 25010 model, which comprises eight characteristics: functionality suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, and portability. The performance of the resulting microservices application is tested using different performance testing types, such as load testing, spike testing, stress testing, and soak testing. Microservices showing satisfactory performance improvements will be isolated using container technology to optimize application resource efficiency and anticipate long-term needs

Keywords— Containers, Monolithic, Microservicing, Refactoring, Performance Testing

I. INTRODUCTION

Aplikasi *enterprise* sistem informasi di perguruan tinggi sebagian besar masih dibangun dengan arsitektur monolitik, dimana semua komponen menjadi satu kesatuan, sebagaimana halnya *SmartCampus* yang merupakan aplikasi monolitik di lingkungan IAIN Syekh Nurjati Cirebon. Dalam arsitektur *SmartCampus*, *server-side* akan menangani *request* kemudian menjalankan beberapa logika sesuai dengan domain, kemudian mengambil dan memperbarui data dari *database*, dan mengirim data tersebut ke sisi *client-side*, sehingga semakin meningkatnya tuntutan akan pelayanan penyelenggaraan pendidikan, *SmartCampus* dihadapkan pada hal- hal berikut : Pengguna yang meningkat menyebabkan peforma aplikasi menurun, skalabilitas yang semakin besar mengganngu waktu proses implementasi dari sisi *client*. Ketika akan merubah teknologi pada aplikasi maka akan merubah secara keselutuhan aplikasi dan pengujian *error* sangatlah sulit karena sistem tidak dapat dipisahkan dan dilokalisasikan. Sistem sulit untuk menyediakan fasilitas pengamanan (*security*). Setiap *request* yang spesifik kernel monolitik akan menjalankan seluruh layanan sehingga akan mempengaruhi waktu respon. Apabila adanya kesalahan pemrograman pada suatu bagian dari kernel, maka menyebabkan matinya seluruh sistem.

Beberapa pertimbangan dalam membangun ulang aplikasi dengan pola arsitektur yang sama (monolitik) dengan sistem warisan akan membawa beberapa dampak sebagai berikut : Waktu yang diperlukan dalam membangun ulang arsitektur monolitik relatif lebih lama karena bisnis proses pada sistem warisan harus secara utuh terbangun sehingga dapat digunakan. Proses migrasi antar server dalam proses transfer data yang besar

membutuhkan waktu dimana server sistem warisan harus berhenti beroperasi, sedangkan pelayanan kepada sivitas akademika dan stakeholder harus tetap berjalan. Seiring dengan peningkatan jumlah pengguna, permasalahan skalabilitas akan kembali terulang jika menerapkan pola arsitektur yang sama. Pola manajemen pengembangan aplikasi sulit dilakukan secara tim dikarenakan tidak semua anggota tim memiliki pemahaman yang baik terhadap sistem warisan. Inovasi dalam pemanfaatan teknologi server-side, database serta bahasa pemrograman tidak fleksibel.

Dari pertimbangan diatas hal yang memungkinkan adalah memperbaiki sistem warisan adalah dengan melakukan *refactoring* arsitektur monolitik menjadi layanan-layanan mikro secara bertahap dengan kondisi layanan sistem tetap berjalan. Menurut [1], *refactoring* adalah proses mengubah sistem dari suatu aplikasi tanpa mengubah perilaku eksternal aplikasi tersebut.

Penelitian [2] mengemukakan bahwa untuk menghindari masalah aplikasi monolitik kita dapat mengambil keuntungan dari beberapa manfaat arsitektur yang berorientasi pada servis atau dikenal dengan istilah Service Oriented Aplication (SOA). Mikroservis mengadopsi konsep SOA denga gaya arsitektur yang lebih terfokus pada achieving agility, kesederhanaan dan teknis pengembangan secara bertingkat, menghindari kompleksitas, dan memungkinkan tim pengembangan dapat dengan cepat menskalakan dan mengembangkan aplikasi secara terus-menerus sebagai solusi komputasi modern.

Penelitian [3] berpendapat bahwa konsep dan teknologi mikroservis bukan merupakan gaya arsitektur baru yang berbeda dari *Service Oriented Architecture* (SOA), tetapi lebih memenuhi syarat sebagai SOA yang dikembangkan lebih lanjut. Tinjauan literatur membuat jelas bahwa perbedaan antara mikroservis dan upaya sebelumnya untuk komputasi berorientasi layanan adalah gaya arsitektur. SOA tidak memperhatikan gaya arsitektur sebagaimana halnya mikroservis. Tinjauan literatur juga mengungkap bahwa, seperti halnya penjelmaan SOA, arsitektur mikroservis dihadapkan dengan sejumlah tantangan desain nontrivial yang intrinsik untuk sistem terdistribusi, termasuk integritas dan manajemen data, desain dan evolusi antarmuka layanan, dan aplikasi / layanan manajemen (termasuk manajemen keamanan aplikasi dan infrastruktur).

Penelitian [4] menyajikan proses pengembangan aplikasi perusahaan dengan layanan microser. Tahapan awal adalah mengidentifikasi fase konstruksi, praktik, dasar-dasar, metode dan alat yang digunakan dalam pengembangan aplikasi perusahaan berbasis layanan mikro. Setelah mengidentifikasi dan mengusulkan proses pengembangan, kemudian divalidasi dan diuji dalam pembangunan aplikasi yang disebut Sinplafut. Dengan menggunakan layanan microser, ukuran aplikasi perusahaan berkurang dan lebih mudah untuk mengelola, menyebarkan, menskalakan, menguji, dan mengganti menggunakan sumber daya yang ditawarkan oleh cloud computing. Dengan cara ini biaya dan upaya yang diperlukan untuk mempertahankan aplikasi di cloud sangat berkurang.

Penelitian [5] menyimpulkan bahwa arsitektur mikroservis dapat memberikan manfaat yang signifikan. Tata kelola desentralisasi dan manajemen data memungkinkan layanan menjadi independen, dan menghindari aplikasi untuk melakukan standarisasi pada satu teknologi tertentu. Arsitektur mikroservis sangat cocok untuk infrastruktur *cloud*, karena sangat menguntungkan dari elastisitas yang dimungkinkan cloud.

Beberapa penelitian tentang mikroservis telah banyak dilakukan. Pada [6] dalam penelitannya melakukan refactoring dengan decomposition pattern. Teknik ini dalam mengembangakan aplikasi kedalam layanan mikro memerlukan beberapa bucket sebagai dekomposisi yang dapat menampung seluruh aktifitas domain bisnis yang dikembangkan. Penelitian ini direkomendasikan untuk desainer arsitektur yang berniat untuk mengembangkan domain bisnis aplikasi kedalam desain layanan mikro. Pada [7] dalam penelitiannya melakukan proses refactoring aplikasi absensi dari arsitektur monolitik menjadi arsitektur mikroservis. Proses refactoring menghasilkan tujuh servis yang dikembangkan pada arsitektur microservis dan dilakukan pengujian dengan menggunakan load test. Disimpulkan bahwa arsitektur mikroservis yang telah dibangun lebih optimal

dibandingkan arsitektur monolitik pada saat jumlah penggunnya dinaikan. Pada [8] dalam penelitiannya merancang proses penyebaran layanan mikro pada cloud platform berbasis kontainer, hasilnya disimpulkan bahwa alokasi sumber daya dan fungsi pemantauan yang disediakan oleh teknologi kontainer dapat memenuhi prinsip independensi layanan mikro dan mendukung percepatan pengembangan layanan mikro. Pengembangan layanan mikro di cloud berbasis kontainer dapat mewujudkan otomatisasi dan integrasi berkelanjutan yang akan meningkatkan efisiensi pengembangan dan pemeliharaan aplikasi.

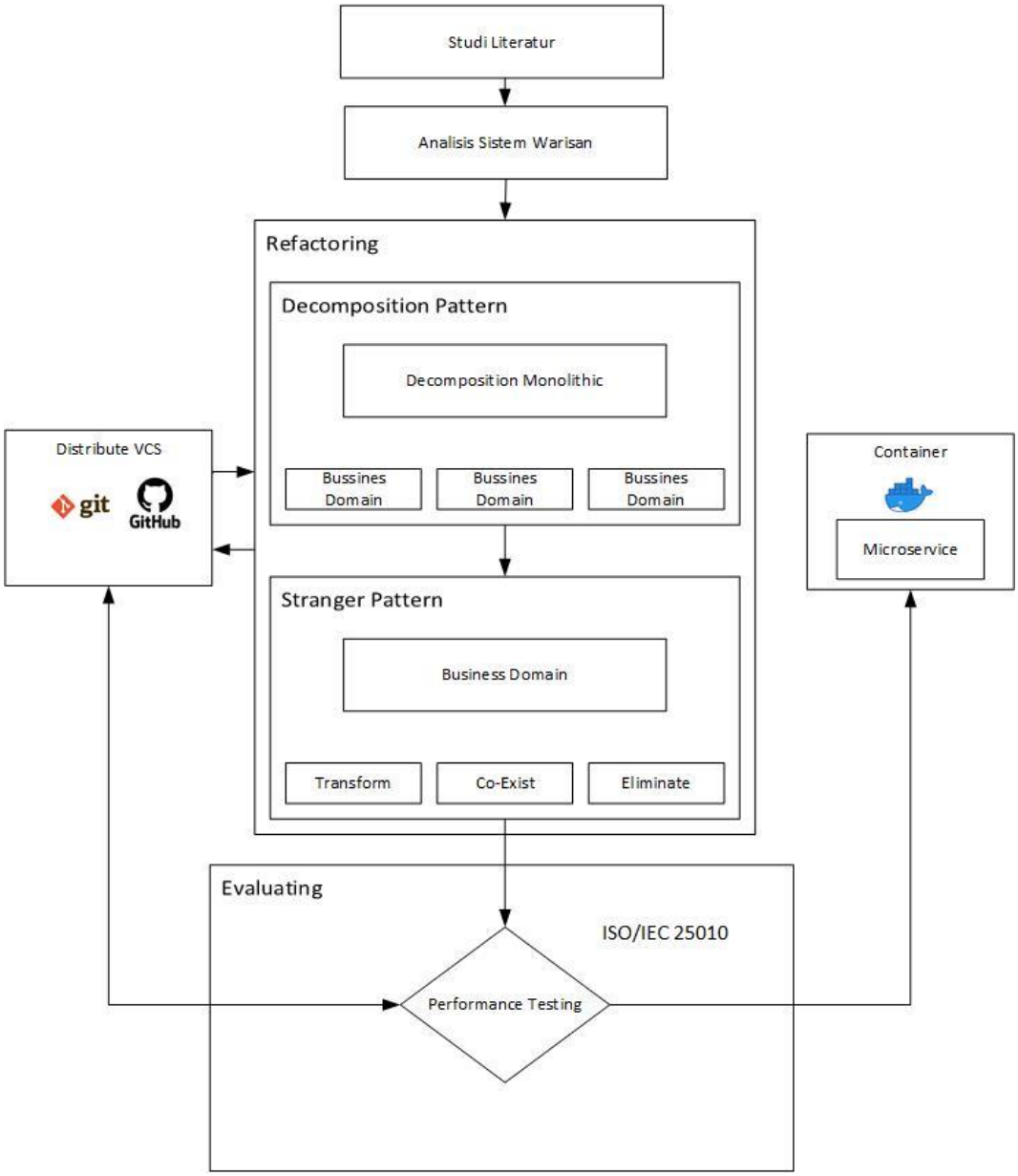
Penelitian ini berusaha melengkapi penelitian-penelitian terdahulu dengan mengimplementasikan proses *refactoring* arsitektur monolitik ke mikroservis dengan metode *refactoring* menggunakan *decomposition* dan *strangler pattern*, hasil implementasi berupa layanan-layanan mikro dilakukan alnalisis uji performa yang terdiri dari *load testing*, *spike testing*, *stress testing* dan *soak testing*. Hasil uji peningkatan performa servis-servis mikro yang layak akan diisolasi dengan teknologi kontainer sebagai upaya efisiensi dan fleksibilitas pengembangan jangka panjang. Tujuan dari penelitian ini adalah dapat membangun aplikasi perguruan tinggi yang *hight performance* dengan metode *refactoting* arsitektur monolitik sistem warisan ke arsitektur mikroservis dikombinasikan dengan memanfaatkan teknologi yang mendukung upaya efisiensi sumber daya aplikasi dan pengembangan berkelanjutan. Penelitian ini diharapkan dapat memberi kontribusi bagi para pengembang aplikasi dalam memberikan solusi permasalahan skalabilitas. Proses migrasi arsitektur monolitik ke mikroservis dalam studi kasus penelitian ini diharapkan dapat memberi gambaran sebuah teknik pengembangan aplikasi yang efektif untuk meminimalkan munculnya *software defect* sebagai solusi komputasi modern. Ruang lingkup penelitian membahas bagaimana mengatasi kelemahan arsitektur monolitik dengan melakukan *refactoring* menjadi aplikasi berarsitektur mikroservis. Pertimbangan kualitas aplikasi dikaji menggunakan karakteristik kualitas perangkat lunak ISO/IEC 25010 terdapat delapan karakteristik kualitas suatu perangkat lunak [9], yaitu ; *functionality suitability, performance efficiency, compability, usability, reliability, security, maintainability,* dan *portability*.

II. METHODS

Objek penelitian adalah aplikasi SmartCampus pada IAIN Syekh Nurjati Cirebon. Data yang digunakan adalah data realtime proses bisnis aplikasi *SmartCampus* pada IAIN Syekh Nurjati Cirebon. *Key performance indicator* (KPI) diantaranya *throughput*, yaitu berapa banyak unit informasi yang diproses sistem selama waktu tertentu. Selain itu adalah waktu respons atau latensi, yaitu jumlah waktu yang berlalu antara permintaan yang dimasukkan pengguna dan awal respons sistem terhadap permintaan itu.

Tahapan Penelitian

Prosedur kerja pada penelitian ini digambarkan pada diagram berikut ini :



Gambar 1. Tahapan Penelitian

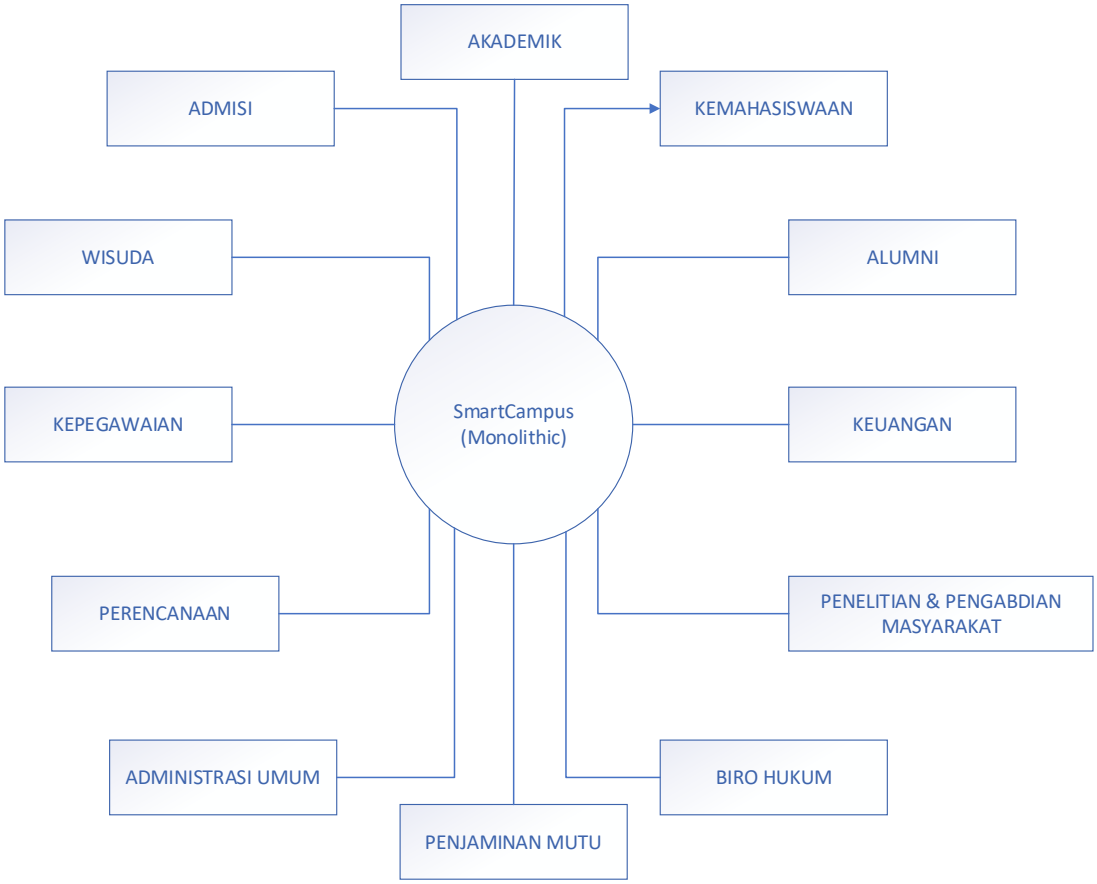
1. Studi Literatur

Studi literatur/ kajian teoritis, langkah ini dilakukan untuk memperkaya kajian teoritis terkait penerapan mikroservis sebagai landasan dalam melakukan pengembangan melalui eksperimen dengan melakukan *refactoring* dengan teknik *strangler pattern*. Studi meliputi kajian penelitian-penelitian terdahulu, kajian terkait terkait teknologi yang dapat dikembangkan serta kajian terkait bahasa pemrograman yang dapat dirterapkan.

2. Analisis Sistem Warisan

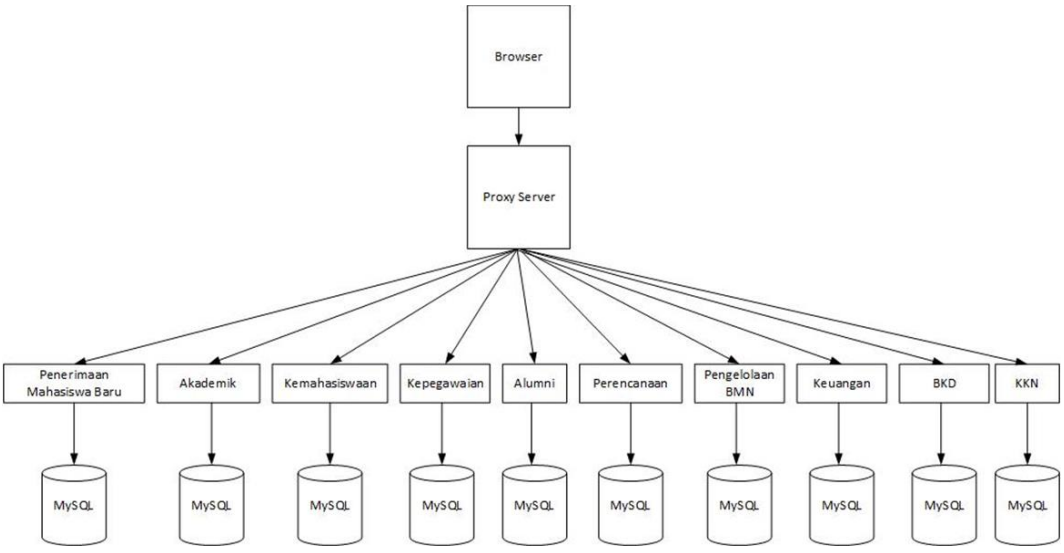
Analisa sistem warisan, langkah ini dilakukan sebagai dasar dalam pengembangan sistem. Analisa meliputi proses domain bisnis sistem bawaan, distribusi databasenya dari sisi *client-server*, serta skala prioritas pengembangan servisnya. Secara garis besar tahapan analisis system warisan dikelompokkan menjadi dua kategori berikut :

a) Pemecahan arsitektur monolitik berdasarkan proses bisnis atau dengan kata lain pemecahan suatu fungsi program atau sistem menjadi sub-fungsi yang lebih sederhana.



Gambar 2. Pemecahan berdasarkan proses bisnis

b) Pemecahan arsitektur monolitik berdasarkan distribusi database.



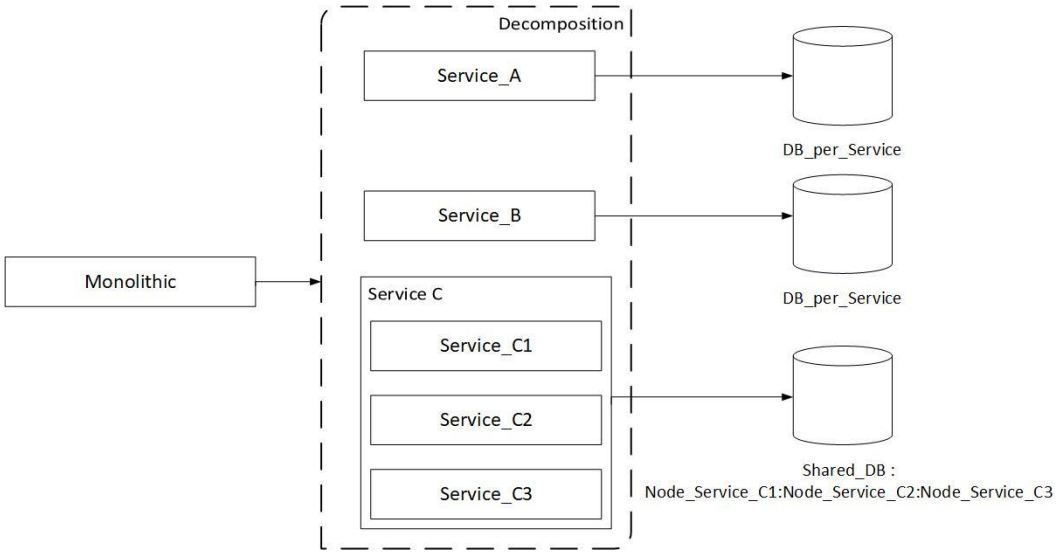
Gambar 3. Pemecahan berdasarkan distribusi database

3. Proses *Refactoring*

Implementasi *refactoring*, langkah ini dilakukan untuk memecah arsitektur sistem monolitik berdasarkan proses bisnis dan distribusi database, kemudian membangun

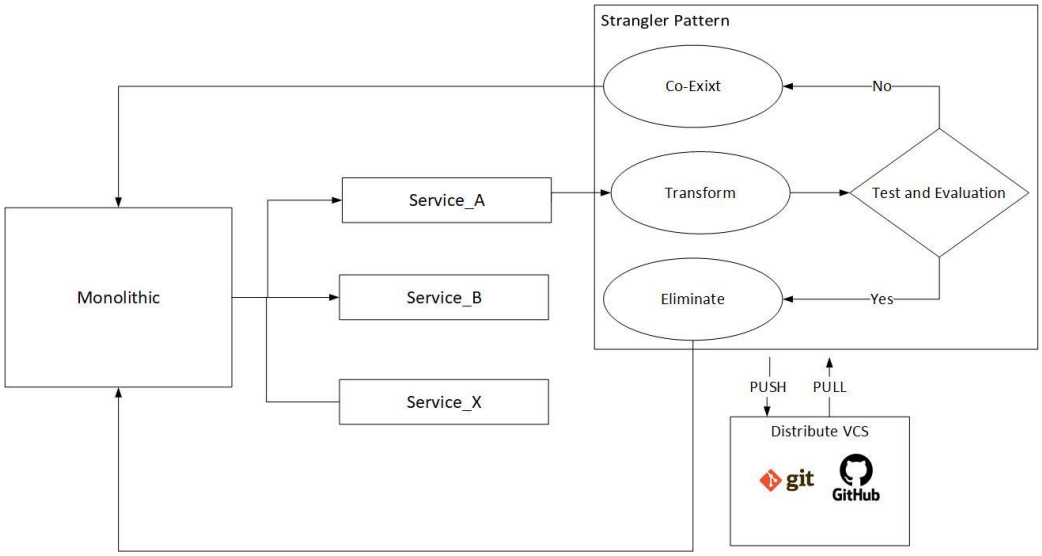
layanan-layanan mikroservis dari sisi *client-server*. Langkah awal dengan mendekomposisi domain bisnis berdasarkan ketaerkaitan proses bisnis dan distribusi database. Tahapan dekomposisi yang dilakukan adalah sebagai berikut:

a. Dekomposisi fungsi: Memecah suatu fungsi program atau sistem menjadi sub-fungsi yang lebih sederhana.
b. Dekomposisi data: Memecah suatu data atau informasi menjadi komponen-komponen yang lebih sederhana.
c. Dekomposisi proses: Memecah suatu proses menjadi tahapan-tahapan yang lebih sederhana.
d. Dekomposisi sistem: Memecah suatu sistem menjadi sub-sistem yang lebih sederhana.
e. Dekomposisi perangkat lunak: Memecah suatu aplikasi atau sistem perangkat lunak menjadi modul-modul yang lebih sederhana.



Gambar 4. Pola Dekomposisi Arsitektur Monolitik

Selanjutnya implementasi *strangler patten*, langkah ini untuk melakukan upaya bertahap dalam mengubah aplikasi monolitik menjadi layanan-layanan mikro dengan mengganti fungsi tertentu dengan layanan baru agar proses bisnis selama pengembangan sistem tetap berjalan. Selama proses *refactoring* diterapkan pola pengembangan berkelanjutan dengan beberapa pengembang aplikasi dengan pemanfaatan tools VCS. Selama proses *refactoring* semua lalu lintas aplikasi tetap dialihkan ke aplikasi lama, kemudian servis baru dibangun dengan memecah menjadi beberapa servis. Setelah servis baru dibangun maka dapat dilakukan pengujian fungsionalitas baru secara paralel terhadap kode monolitik yang ada. Baik monolitik dan servis yang baru dibangun harus berfungsi untuk jangka waktu tertentu. Proses eliminasi dilakukan ketika servis baru telah dikembangkan dan diuji secara bertahap sebelum dapat menggantikan aplikasi monolitik warisan.

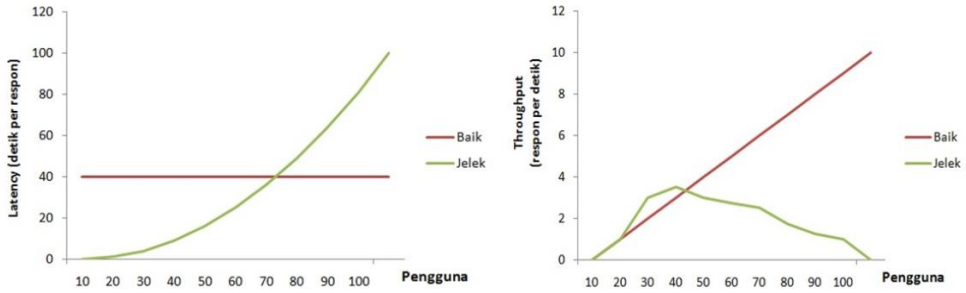


Gambar 5 Pola *Strangler Refactoring* Monolitik

4. Analisis dan validasi data

Validasi data dilakukan dengan membandingkan hasil uji performa sebelum dan sesudah dilakukannya implementasi *refactoring* arsitektur monolitik ke mikroservis. Data yang digunakan adalah data *realtime* proses bisnis sistem informasi IAIN Syekh Nurjati Cirebon dalam proses administrasi akademik. *Key performance indicator* (KPI) diantaranya *throughput*, yaitu berapa banyak unit informasi yang diproses sistem selama waktu tertentu. Selain itu adalah *latency*, yaitu jumlah waktu yang berlalu antara permintaan yang dimasukkan pengguna dan awal respons sistem terhadap permintaan itu.

Skalabilitas dalam persfektif *latency* mendefinisikan bahwa idealnya jika penambahan pengguna yang signifikan, aplikasi tersebut memberikan *latency* yang relatif konstan. Sedangkan persfektif t*hroughput* mendefinisikan bahwa idealnnya jika penambahan pengguna yang signifikan, aplikasi dapat meningkatkan kemampuan menangani jumlah request per detik secara linier.



Gambar 6. Skalabilitas dalam istilah *latency* dan *throughput*.

5. Evaluasi

Layanan mikro yang dihasilkan dari proses *refactoring* dilakukan peninjauan kelayakan dengan model sistem pengujian kualitas perangkat lunak (software testing) ISO/IEC 25010 dengan membandingkan dengan aplikasi sebelumnya. Karakteristik kualitas kelayakan terdiri dari 8 (delapan) aspek yaitu :

a. *Functional Suitability* merupakan karakteristik untuk mengukur sejauh mana produk atau sistem menyediakan fungsi yang memenuhi kebutuhan ketika digunakan dalam kondisi tertentu;

b. *Performance Efficiency* adalah karakteristik untuk mengukur kinerja relatif terhadap sumber daya yang digunakan dalam kondisi tertentu pada suatu sistem;
c. *Compatibility* adalah karakteristik untuk mengukur sejauh mana suatu sistem dapat bertukar informasi dengan sistem lain dan melakukan fungsi yang disyaratkan saat berbagi lingkungan perangkat keras atau perangkat lunak yang sama;
d. *Usability* adalah karakteristik untuk mengukur sejauh mana sistem dapat digunakan oleh pengguna untuk mencapai tujuan yang ditentukan dengan efektivitas, efisiensi, dan kepuasan dalam konteks penggunaan tertentu;
e. *Realiability* adalah karakteristik untuk mengukur sejauh mana sistem dapat melakukan fungsi dalam kondisi yang ditentukan untuk periode waktu terentu;
f. *Security* adalah karakteristik untuk mengukur suatu sistem dalam melakukan proteksi terhadap informasi dan data, sehingga sistem memiliki tingkat akses data sesuai dengan jenis dan tingkat otorisasi;
g. *Maintainability* adalah karakteristik untuk mewakili tingkat efektivitas dan efisiensi dalam proses modifikasi untuk perbaikan system sesuai dengan penyesuaian dan perubahan pada lingkungan operasional;
h. *Portability* adalah karakteristik untuk mewakili tingkat efektivitas dan efisiensi sistem dalam melakukan transfer dari satu perangkat ke perangkat lainnya.

Tes performa yang dilakukan sebagai bagian dari model pengujian perangkat lunak ISO/IEC 25010 diantaranya *load testing, spike testing, stress testing* dan *soak testing*.

a. *Load testing*
Load testing membantu pengembang memahami perilaku sistem di bawah nilai beban tertentu. Jenis pengujian ini membantu pengembang menentukan berapa banyak pengguna yang dapat ditangani oleh aplikasi atau sistem sebelum aplikasi atau sistem tersebut ditayangkan.

b. *Spike testing*
Spike testing menilai kinerja sistem di bawah peningkatan pengguna akhir yang disimulasikan secara tiba-tiba dan signifikan. Tes ini membantu menentukan apakah suatu sistem dapat menangani peningkatan beban kerja yang tiba-tiba dan drastis dalam waktu singkat, berulang kali.

c. *Stress testing*
Stress testing menempatkan sistem di bawah beban lalu lintas yang lebih tinggi dari perkiraan sehingga pengembang dapat melihat seberapa baik sistem bekerja di atas batas kapasitas yang diharapkan. Tes ini memungkinkan tim perangkat lunak memahami skalabilitas beban kerja.

d. *Soak testing*
Soak testing, mensimulasikan peningkatan pengguna akhir yang stabil dari waktu ke waktu untuk menguji keberlanjutan jangka panjang sistem. Uji ini juga menganalisis *throughput* dan waktu respons setelah penggunaan berkelanjutan untuk menunjukkan apakah metrik ini konsisten dengan statusnya di awal pengujian.

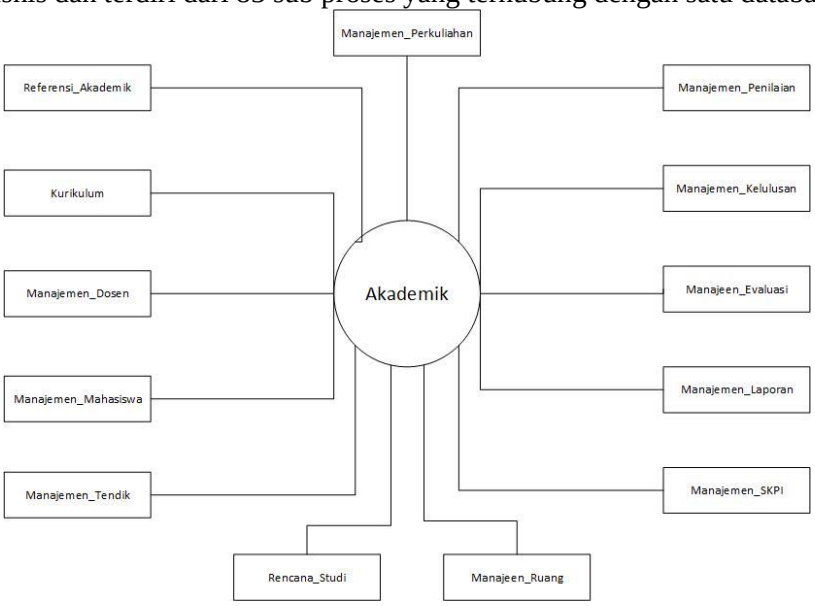
III. HASIL DAN PEMBAHASAN

Hasil Proses Refactoring Monolitik ke Mikroservis

Pola dekomposisi dari aplikasi berarsitektur monolitik ke mikroservis dilakukan dengan tahapan sebagai berikut :

a. Analisis Aplikasi Sistem Warisan

Dari hasil analisis aplikasi monolitik untuk proses kapabilitas bisnis akademik terdapat 13 proses bisnis dan terdiri dari 83 sub proses yang terhubung dengan satu database.



Gambar 7. Analisis SmartCampus (*Monolihtic*) berdasarkan kapabilitas bisnis Akademik

b. Identifikasi Layanan Potensial

Setelah memahami struktur aplikasi monolitik, identifikasi layanan-layanan potensial yang dapat diisolasi dan dipecah dari aplikasi utama. Layanan-layanan ini harus berfokus pada tugas-tugas atau fungsionalitas khusus yang dapat berdiri sendiri sebagai layanan mandiri. Dekomposisi servis struktur aplikasi monolitik dihasilkan 235 servis dari 83 layanan potensial yang dijabarkan pada table SVO berikut :

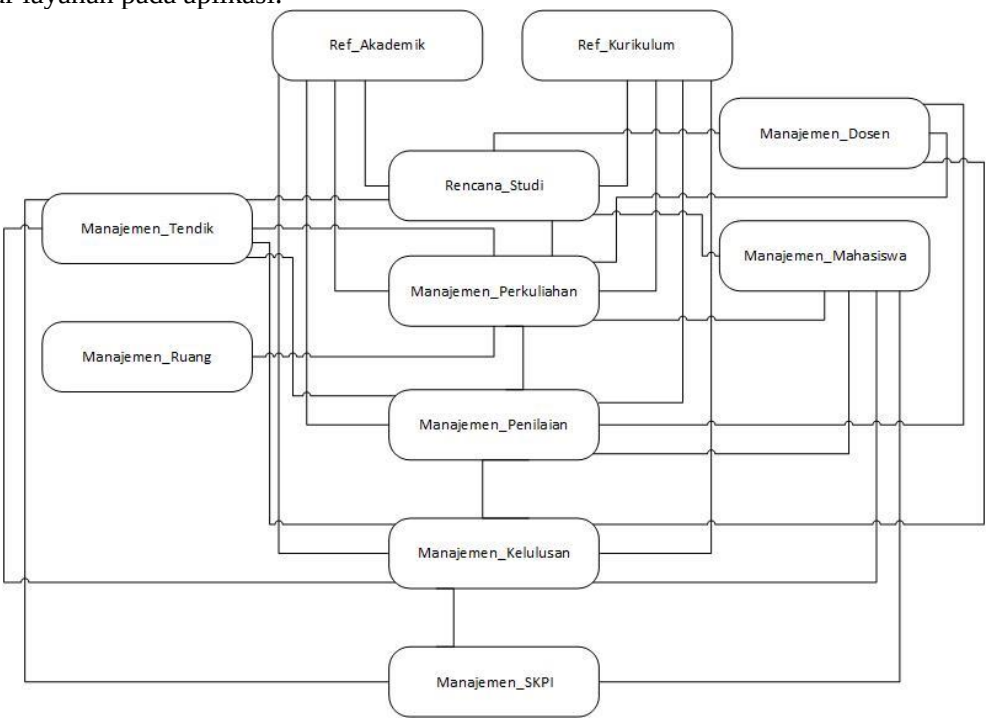
Tabel 1. Contoh Dekomposisi Layanan Potensial Menjadi Servis

Sub Proses	Servis	S	V	O
Managemen_Program_ Studi	- sinkronisasi_prodi	staf	update	data_prodi
	- tambah_prodi	staf	add	data_prodi
	- ubah_prodi	staf	update	data_prodi
	- lihat_prodi	staf	get	data_prodi

c. Batasan Domain Layanan

Langkah selanjutnya dalam proses dekomposisi adalah memastikan bahwa layanan-layanan yang diidentifikasi kemudian dikelompokkan berdasarkan prinsip fungsional yang memiliki persamaan antar proses bisnis. Identifikasi hubungan fungsional sistem dilakukan untuk mengetahui keterhubungan antara satu proses bisnis dengan proses bisnis yang lain. Proses analisa arsitektur servis dilakukan dengan mengambarkan keterhubungan antar servis. [10] menyimpulkan bahwa layanan mikroservis adalah sistem

terdistribusi yang sangat modular, dapat digunakan kembali melalui API (Application Programming Interface) yang diekspos jaringan. Fitur umum termasuk orientasi bisnis, teknik pemrograman, bahasa pemrograman, desentralisasi dan otomatisasi secara khusus ditekankan dalam pendekatan implementasi layanan mikroservis. Ketersediaan, kinerja, penskalaan otomatis, pengujian otomatis, integrasi dan penerapan berkelanjutan, keamanan, toleransi kesalahan, adalah fitur penting yang harus dimiliki oleh setiap aplikasi bisnis saat ini. Layanan mikroservis mengurangi kompleksitas penanganan fitur-fitur layanan pada aplikasi.



Gambar 8. Hubungan Antar Servis Hasil Dekomposisi Arsitektur Monolitik

Tahapan *Strangler Pattern* pada proses transform sebuah servis beberapa proses yang dilakukan adalah sebagai berikut :

a. Definisikan Kontrak Layanan

Setelah layanan-layanan potensial diidentifikasi, definisikan kontrak layanan untuk setiap layanan tersebut. Kontrak ini berisi detail tentang API dan tipe data yang akan digunakan untuk berkomunikasi dengan layanan tersebut.

b. Pemisahan Kode

Pemisahan kode dilakukan untuk memisahkan layanan-layanan potensial dari aplikasi monolitik. Anda bisa mulai dengan mengidentifikasi dan memisahkan fungsi-fungsi atau modul-modul tertentu ke dalam layanan-layanan yang mandiri.

c. Tambahkan Konektivitas Antar Layanan

Setelah layanan-layanan yang independen dibuat, selanjutnya adalah menambahkan konektivitas antar layanan untuk memungkinkan komunikasi antara layanan-layanan tersebut. Ini bisa dilakukan melalui protokol HTTP/REST.

d. Pengelolaan Data

Data akan dikelola di lingkungan mikroservis perlu dipertimbangkan antara menggunakan *database* terpisah untuk setiap layanan atau mempertahankan *database* bersama dan menyediakan antarmuka data yang terdefinisi dengan baik untuk layanan lainnya.

e. Pengujian

Dalam menguji setiap layanan secara terpisah untuk memastikan bahwa mereka berfungsi dengan baik dan sesuai dengan kontrak yang telah ditetapkan. Selain itu, perlu

juga dilakukan pengujian integrasi untuk memastikan bahwa komunikasi antar layanan berjalan dengan benar.

f. Penyebaran dan Skalabilitas:

Siapkan strategi penyebaran dan skalabilitas untuk masing-masing layanan. Layanan-layanan ini harus dapat dideploy secara independen dan dapat diubah skala sesuai kebutuhan.

g. Pemantauan dan Manajemen:

Setelah seluruh sistem berada dalam arsitektur mikroservis, pastikan memiliki mekanisme pemantauan dan manajemen untuk memastikan kinerja, ketersediaan, dan keandalan layanan-layanan tersebut.

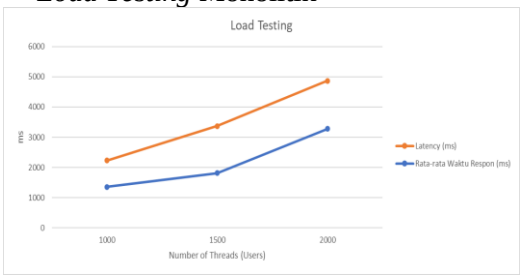
Hasil Uji Performa Aplikasi Mikroservis vs Monolitik

Hasil *load testing* menggunakan *GET method* dengan skenario uji menambahkan sejumlah akses user secara bersamaan dalam periode akses 1 detik dengan *looping* sebanyak 1 kali berbantuan JMeter, diperoleh grafik rata-rata waktu respon dan *latency* dari aplikasi mikroservis lebih konstan dibandingkan hasil *load testing* aplikasi monolitik. Hal ini dapat disimpulkan bahwa aplikasi mikroservis yang dihasilkan lebih mampu menangani banyak pengguna.

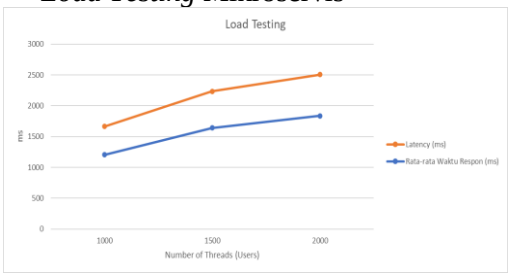
Tabel X. Perbandingan Hasil *Load Testing* Monolitik vs Mikroservis

Number of Threads (Users)	*SmartCampus*(Monolitik)			Mikroservis		
	Status	*Rata-rata Waktu Respon (ms)*	*Latency(ms)*	*Status*	*Rata-rata Waktu Respon (ms)*	*Latency(ms)*
1.000	99,13%	1353,800667	878,904	100%	1204,98	460,096
1.500	99,70%	1813,672	1559,516	100%	1640,249333	593,2733333
2.000	65,30%	3276,332	1587,4905	100%	1837,3275	668,3755

Load Testing Monolitik



Load Testing Mikroservis



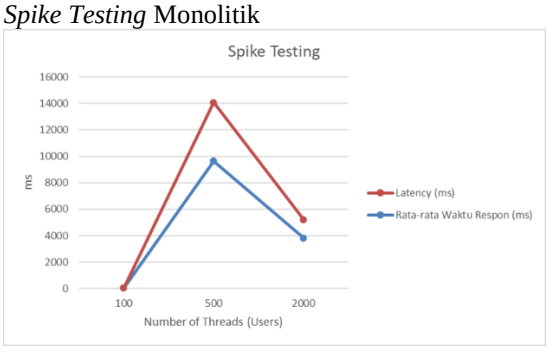
Gambar 9. Perbandingan Hasil *Load Testing* Monolitik vs Mikroservis

Hasil *spike testing* menggunakan *GET method* dengan skenario uji menambahkan sejumlah akses user secara bersamaan dalam periode akses 1 detik dengan looping sebanyak 5 kali berbantuan JMeter, diperoleh grafik rata-rata waktu respon dan *latency* dari aplikasi mikroservis yang dihasilkan lebih konstan dibandingkan dengan hasil *spike testing* aplikasi monolitik. Pada kategori jumlah akhir yang ditambahkan sebanyak 2000 user yang mengakses secara bersamaan, aplikasi mampu merespon sebanyak 76,52% *thread* atau sebanyak 7.652 *thread* dari total *10.000 thread* yang seharusnya mampu direspon. Hasil ini mengalami peningkatan dibandingkan pada *spike testing* aplikasi monolitik. Dapat disimpulkan bahwa kinerja aplikasi mikroservis di bawah peningkatan pengguna akhir yang disimulasikan secara tiba-tiba dan signifikan jauh lebih baik dibandingkan aplikasi monolitik sebelumnya.

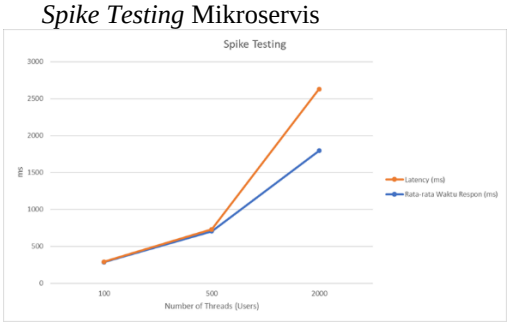
Tabel X. Perbandingan Hasil *Spike Testing* Monolitik vs Mikroservis

Number of Threads (Users)	*SmartCampus*(Monolitik)			Mikroservis		
	Status	*Rata-rata Waktu Respon (ms)*	*Latency(ms)*	*Status*	*Rata-rata Waktu Respon (ms)*	*Latency(ms)*
100	100%	32,88	9,108	100%	287,288	5,966
500	100%	9652,4664	4415,4516	100%	705,1444	26,1476
2.000	43,76%	3816,4022	1400,2153	76,52%	1794,751	834,1203

Spike Testing Monolitik



Spike Testing Mikroservis



Gambar 10. Perbandingan Hasil *Spike Testing* Monolitik vs Mikroservis

Hasil *stress testing* menggunakan *POST method* dengan skenario uji menambahkan sejumlah akses user secara bersamaan dalam periode akses 1 detik dengan *looping* sebanyak 1 kali berbantuan JMeter, diperoleh grafik rata-rata *throughput* aplikasi mikroservis lebih linier dibandingkan hasil *stress testing* aplikasi monolitik. Pada kategori jumlah user yang ditambahkan sebanyak 2000 user yang mengakses secara bersamaan, aplikasi hanya mampu merespon sebanyak 66,25%, namun nilai ini lebih baik dibandingkan hasil *stress testing* aplikasi monolitik. Hal ini menunjukan bahwa aplikasi mikroservis yang dihasilkan bekerja lebih baik di atas batas kapasitas yang diharapkan dibandingkan aplikasi monolitik sebelumnya.

Tabel X. Perbandingan Hasil *Stress Testing* Monolitik vs Mikroservis

Number of Threads (Users)	*SmartCampus*(Monolitik)		Mikroservis	
	Status	*Throughput*	*Status*	*Throughput*
1.000	68,30%	40,6/sec	88,90%	45,0/sec
1.500	59,33%	40,36/sec	75,73%	63,1/sec
2.000	44%	66,5/ sec	66,25%	75,5/sec

Stress Testing Monolitik

Stress Testing Mikroservis

Gambar 11. Perbandingan Hasil *Stress Testing* Monolitik vs Mikroservis

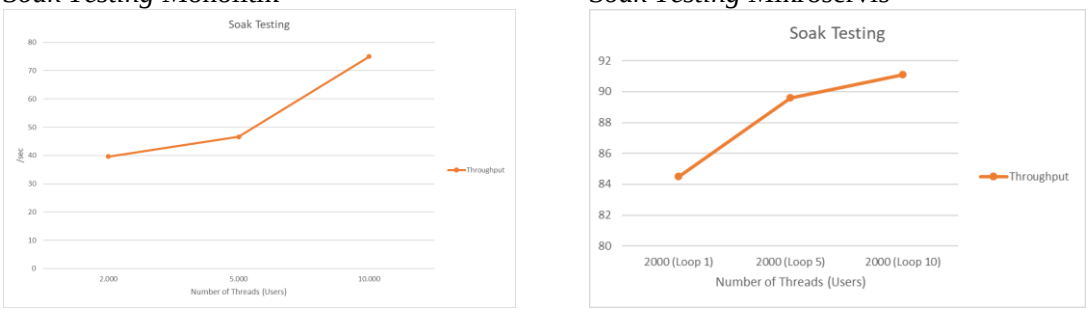
Hasil *soaks testing* menggunakan *POST method* dengan skenario uji menambahkan sejumlah akses user yang sama secara bersamaan dalam periode akses 1 detik dengan *looping* sebanyak 1 kali, 5 kali, dan 10 kali berbantuan JMeter, diperoleh grafik rata-rata *throughput* aplikasi mikroservis lebih linier dibandingkan *soak testing* aplikasi monolitik . Pada kategori jumlah user yang sebanyak 2000 user dengan jumlah lalu lintas looping yang ditambahkan hasilnya rata-rata *throughput* selalu meningkat dengan jumlah *threads user* yang mampu direspon menurun namun tidak signifikan. Hal ini menunjukan aplikasi mikroservis lebih mampu mempertahankan performa dalam keadaan *threads user* yang bertambah.

Tabel X. Perbandingan Hasil *Soak Testing* Monolitik vs Mikroservis

Number of Threads (Users)	SmartCampus(Monolitik)		Mikroservis	
	Status	*Throughput*	*Status*	*Throughput*
2000	40%	39,6/sec	61,20%	84,5/sec
2000	24,50%	56,6/sec	58,93%	89,6/sec
2000	5,90%	75/sec	56,03%	91,1/sec

Soak Testing Monolitik

Soak Testing Mikroservis



Gambar 12. Perbandingan Hasil *Soak Testing* Monolitik vs Mikroservis

Perbaikan Aspek Kelayakan Berdasararkan Karakteristik ISO/IEC 25010

Hasil pengujian *functional suitability* aplikasi mikroservis dihasilkan nilai persentasi 86,9% dengan kategori sangat baik, sedangkan aplikasi sebelumnya yang berarsitektur monolitik dihasilkan nilai persentasi 74%. Ditinjau dari aspek *compability*, aplikasi mikroservis yang dihasilkan diupayakan dapat memperbaiki kelemahan pada aplikasi monolitik system warisan khususnya dalam penggunaan bahasa pemrograman lebih fleksibel dalam pengembangan, khususnya ketika dibutuhkan penskalaan aplikasi agar dapat berkelanjutan. Pada aspek *maintainability*, arsitektur mikroservis yang dihasilkan diupayakan dapat menutupi kekurangan arsitektur monolitik sebelumnya. Beberapa pertimbangan aspek *maintainability* (kemudahan pemeliharaan) dari hasil kajian aplikasi berasitektur mikroservis diantaranya pemecahan servis, penggunaaan antarmuka terdefinisi, dan *Automasi dan Continuous Integration/Continuous Deployment* (CI/CD). Pertimbangan perbaikan aspek *portability* hal yang diupayakan dalam penerapan aplikasi

berarsitektur mikroservis diantaranya isolasi layanan, penggunaan teknologi yang beragam, dan peningkatan penanganan skalabilitas.

Berdasarkan hasil penelitian ini, implementasi *refactoring* arsitektur monolitik ke arsitektur mikroservis terbukti lebih efektif daripada membuat ulang aplikasi monolitik, karena seiring dengan peningkatan tuntutan kebutuhan akan aplikasi maka *codebase* aplikasi monolitik pasti akan bertambah besar dan kompleks. Hal ini sejalan dengan pernyataan [11] bahwa mengubah kode pada aplikasi monolitik bisa menjadi satu hal yang menyulitkan karena bisa jadi kode yang diubah menjadi dependensi di banyak tempat oleh beberapa komponen yang berbeda. Pilihan yang memungkinkan ketika aplikasi tidak mampu menangani kebutuhan karena lonjakan pengguna adalah dengan menaikkan sumber daya keseluruhan (*vertical scaling*) atau membuat beberapa *instance* dari aplikasi (*horizontal scaling*) karena pada aplikasi monolitik, kita tidak bisa mengatur skalabilitas di level fitur/ komponen. Hal ini karena tiap komponen bukanlah servis yang berjalan secara independen. Satu perubahan kecil bisa berpengaruh besar terhadap keseluruhan aplikasi

IV. KESIMPULAN

Hasil uji performa aplikasi mikroservis diantaranya *load testing, spike testing, stress testing* dan *soak testing* diperoleh terbukti lebih baik dibandingkan aplikasi *SmartCampus* yang berasitektur monolitik. Aplikasi mikroservis yang dihasilkan juga terbukti mampu memperbaiki kelemahan aplikasi monolitik pada sistem sebelumnya berdasarkan karakteristik kualitas perangkat lunak ISO/IEC 25010 dengan delapan karakteristik kualitas suatu perangkat lunak yaitu ; *functionality suitability, performance efficiency, compability, usability, reliability, security, maintainability,* dan *portability.* Keterbatasan dalam penelitian ini adalah belum memaksimalkan kajian lebih lanjut terkait pengukuran efisiensi pengisolasian layanan dengan pemanfaatan teknologi kontainer. Oleh karena itu, peneliti merekomendasikan penelitian lanjutan yang berfokus pada mengukur efisiensi penerapan isolasi layanan mikroservis dengan pemanfaatan teknologi kontainer.

REFERENCES

[1] M. Fowler, *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.

[2] M. Villamizar *et al.*, “Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud,” in *2015 10th Colombian Computing Conference, 10CCC 2015*, Institute of Electrical and Electronics Engineers Inc., Nov. 2015, pp. 583–590. doi: 10.1109/ColumbianCC.2015.7333476.

[3] O. Zimmermann, “Microservices tenets: agile approach to service development and deployment. Overview and vision paper, SummerSoC 2016,” *J. Comput. Sci. Res. Dev.(CSRD), Springer (to appear)*.

[4] F. H. Vera-Rivera, “A development process of enterprise applications with microservices,” in *Journal of Physics: Conference Series*, IOP Publishing, 2018, p. 012017.

[5] P. Di Francesco, P. Lago, and I. Malavolta, “Architecting with microservices: A systematic mapping study,” *Journal of Systems and Software*, vol. 150, pp. 77–97, 2019.

[6] M. Tusjunt and W. Vatanawood, “Refactoring orchestrated web services into microservices using decomposition pattern,” in *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, IEEE, 2018, pp. 609–613.

[7] R. Mufrizal and D. Indarti, "Refactoring Arsitektur Microservice Pada Aplikasi Absensi PT. Graha Usaha Teknik," *Jurnal Nasional Teknologi dan Sistem Informasi*, vol. 5, no. 1, pp. 57–68, Apr. 2019, doi: 10.25077/teknosi.v5i1.2019.57-68.

[8] J. Cao, "Design on deployment of microservices on container-based cloud platform," in *Journal of Physics: Conference Series*, IOP Publishing Ltd, Nov. 2020. doi: 10.1088/1742-6596/1624/6/062008.

[9] S. Kapcsolatos, I. Szabványok, Á. Nyári, and N. 1-K. András, "Safety and Security Sciences Review Biztonságtudományi Szemle," 2021.

[10] S. Baškarada, V. Nguyen, and A. Koronios, "Architecting microservices: Practical opportunities and challenges," *Journal of Computer Information Systems*, 2018.

[11] Sichuan Institute of Electronics and Institute of Electrical and Electronics Engineers, *2018 IEEE 4th International Conference on Computer and Communications (ICCC) : December 7-10, 2018, Chengdu, China*.