Analisis Response Metrics Terhadap Arsitektur Monolithic dan Microservices dalam Implementasi Aplikasi Kompen

Luqman Affandi[1], Meyti Eka Apriyani [2], Anggara Mahendra Putra [3]
[1,2,3]Teknik Informatika, Teknologi Informasi, Politeknik Negeri Malang, Indomesia
Korespodensi author: [1]laffandi@polinema.ac.id

Info Artikel

Diajukan: 3 September 2020
Diterima: 24 September 2020
Diterbitkan: 1 Oktober 2020

Keywords:
monolithic, microservice, refactoring, load test

Kata Kunci: *monolithic, microservice, refactoring, tes beban*

Abstract

Currently, information systems are faced with significant development, where policy changes or additional features are required immediately according to the needs of business rules, both internal and external to the system. For example, when an information system that is already running needs to develop certain modules or features to suit internal needs and policies, it is hoped that the process of adding these features in developing business rules will not interfere with the performance of system processes that are already running. The JTI Polinema Compensation application currently still adopts a monolithic architecture, where if a failure occurs during an update or addition of new features to the application, all application features will experience system failure. The next problem is that the number of students in the Polinema Information Technology Department increases over time, and under certain conditions such as at the end of the semester or graduation period for final year students who are completing their compensation obligations. So that when accessing the system simultaneously, the number of requests processed on the system increases so that application performance decreases. Given the problems described above, it is necessary to refactor the application from a monolithic architecture to a microservice architecture. The results of the refactoring carried out on the JTI Polinema Kompen application produced four services that can be developed using a microservice architecture. Based on test results using load tests, the microservice architecture that has been built is more optimal than monolithic architecture.

Abstrak

Saat ini sistem informasi dihadapkan pada pengembangan yang signifikan, dimana seketika dituntut perlu ada perubahan-perubahan kebijakan ataupun penambahan fitur-fitur sesuai kebutuhan business rule baik internal maupun eksternal sistem. Semisal ketika suatu sistem informasi yang telah berjalan lalu perlu dilakukan pengembangan modul ataupun fitur tertentu untuk menyesuaikan kebutuhan dan kebijakan internal, maka proses penambahan fitur tersebut dalam pengembangan business rules diharapkan tidak mengganggu kinerja dari proses sistem yang sudah berjalan. Aplikasi Kompen JTI Polinema saat ini masih mengadopsi arsitektur monolithic, dimana jika terjadi kegagalan pada saat pembaruan atau penambahan fitur baru pada aplikasi, maka seluruh fitur aplikasi akan mengalami kegagalan sistem. Permasalahan selanjutnya yakni seiring dengan bertambahnya jumlah mahasiswa Jurusan Teknologi Informasi Polinema dari waktu ke waktu, dan dalam kondisi tertentu seperti pada akhir semester atau masa kelulusan mahasiswa tingkat akhir yang sedang menyelesaikan tanggungan kompen. Sehingga akan mengakses sistem secara bersamaan, maka jumlah request yang diproses pada sistem semakin meningkat sehingga kinerja aplikasi semakin menurun. Dengan adanya permasalahan sebagaimana yang dijelaskan diatas, diperlukan proses refactoring aplikasi dari arsitektur monolithic menjadi arsitektur microservice. Hasil refactoring yang dilakukan pada aplikasi Kompen JTI Polinema menghasilkan empat service yang dapat dikembangkan pada arsitektur microservice. Berdasarkan hasil pengujian dengan menggunakan load test, arsitektur microservice yang telah dibangun lebih optimal dibandingkan arsitektur monolithic.

Cara mensitasi artikel:
L. Affandi, M. K. Apriyani, and A. M. Putra, "Analisis Response Metrics Terhadap Arsitektur Monolithic dan Microservices dalam Implementasi Aplikasi Kompen," *Jurnal Teknologi Informasi: Teori, Konsep, dan Implementasi (JTI-TKI)*, vol. 11, no. 2, pp. 48-53, October 2020, doi: 10.36382/jti-tki.v11i2.495

PENDAHULUAN

Perkembangan teknologi informasi telah berkembang sangat cepat mengikuti kebutuhan zaman yang memerlukan kecepatan dan ketepatan disegala aspek kehidupan. Perkembangan mengikuti segi perangkat keras, perangkat lunak maupun dari segi sumber daya manusia yang mengoperasikannya. Pada saat ini hampir semua bidang kehidupan baik di bidang ekonomi, politik, kebudayaan, seni, hingga pendidikan memerlukan teknologi informasi. Dengan komputer kita dapat melakukan pengolahan data dan penyimpanan data, *input* data, *edit* data, simpan, hapus dan lain-lain. Sehingga data

yang dikelola lebih efektif dan efisien. Hal tersebut akan memicu peningkatan jumlah pengguna teknologi. Beberapa situs bahkan telah menerima ratusan ribu koneksi dari *client* secara simultan, meningkatnya jumlah koneksi dapat mempengaruhi kinerja sebuah aplikasi yang telah di bangun.

Sebagian aplikasi yang telah dirancang dan dibangun sekarang masih menggunakan arsitektur *monolithic*, arsitektur *monolithic* adalah suatu arsitektur yang menggambarkan sebuah aplikasi yang menjalankan semua logika dalam satu server aplikasi. Peningkatan jumlah pengguna aplikasi dapat mempengaruhi proses pemeliharaan aplikasi, kinerja aplikasi dan kompleksnya proses pembaruan aplikasi yang dibangun dengan arsitektur *monolithic*[1][2][3]. Proses pemeliharaan aplikasi yang dibangun dengan arsitektur *monolithic* yang berukuran besar sangat sulit dikarenakan kompleksitasnya[4].

Saat ini sistem informasi dihadapkan pada pengembangan yang signifikan, dimana seketika dituntut perlu ada perubahan-perubahan kebijakan ataupun penambahan fitur-fitur sesuai kebutuhan *business rule* baik internal maupun eksternal sistem. Semisal ketika suatu sistem informasi yang telah berjalan lalu perlu dilakukan pengembangan modul ataupun fitur tertentu untuk menyesuaikan kebutuhan dan kebijakan internal, maka proses penambahan fitur tersebut dalam pengembangan *business rules* diharapkan tidak mengganggu kinerja dari proses sistem yang sudah berjalan [5].

Sistem Informasi Kompen Jurusan Teknologi Informasi Polinema merupakan aplikasi berbasis website yang dikembangkan untuk memonitor dan mengakumulasi absensi mahasiswa Jurusan Teknologi Informasi Polinema. Terdapat beberapa fitur dari Sistem Informasi Kompen diantaranya adalah untuk memudahkan dosen dan staff akademik Jurusan Teknologi Informasi Polinema, untuk melakukan rekapitulasi absensi, memantau kedisiplinan mahasiswa, memudahkan pembuatan laporan data absensi mahasiswa, dan sanksi kedisiplinan absensi.

Sistem Informasi Kompen Jurusan Teknologi Informasi Polinema saat ini masih mengadopsi arsitektur *monolithic*, dimana jika terjadi kegagalan pada saat proses pembaruan atau penambahan fitur baru pada aplikasi, maka seluruh fitur aplikasi akan mengalami kegagalan sistem [6]. Permasalahan selanjutnya yakni seiring dengan bertambahnya jumlah mahasiswa Jurusan Teknologi Informasi Polinema dari waktu ke waktu, dan dalam kondisi tertentu seperti pada akhir semester atau masa kelulusan mahasiswa tingkat akhir yang sedang menyelesaikan tanggungan kompen. Sehingga akan mengakses sistem secara bersamaan, maka jumlah *request* yang diproses pada sistem semakin meningkat sehingga kinerja aplikasi semakin menurun.

Saat ini muncul sebuah arsitektur baru yaitu arsitektur *microservice*, arsitektur *microservice* adalah gaya arsitektur perangkat lunak yang memerlukan pemecahan aplikasi secara bisnis [7]. Dengan adanya permasalahan sebagaimana yang dijelaskan diatas, maka dilakukan *refactoring* arsitektur *monolithic* menjadi arsitektur *microservice* pada Sistem Informasi Kompen Jurusan Teknologi Informasi Polinema. *Refactoring* merupakan proses mengubah *software system*, tapi tidak merubah tingkah laku atau *behaviour* dari kode tersebut tetapi membuat struktur didalamnya menjadi lebih baik [8]. Pengembangan aplikasi dengan menggunakan arsitektur *microservice* dapat memperbaiki permasalahan diatas, yaitu adanya pemisahan *service-service* menjadi kecil. Dibutuhkan pengukuran terhadap *performance* antara aplikasi yang menerapkan arsitektur *monolithic* dengan arsitektur *microservices* agar dapat mengetahui arsitektur mana yang lebih unggul dan lebih optimal, sehingga aplikasi dapat bekerja dengan baik tanpa ada kendala. Dalam hal ini digunakanlah response metric yaitu istilah parameter yang akan di gunakan dalam pengukuran di antaranya *throughput, respond time*, dan *error rate*.

Microservices adalah gaya arsitektur perangkat lunak yang memerlukan pemecahan aplikasi secara bisnis, arsitektur *microservices*[9]–[13] adalah sebuah arsitektur baru dimana pengembangan aplikasi dilakukan dalam bentuk *web service* kecil yang saling berkomunikasi satu sama lain. *microservices* memperlakukan "*services*" sebagai sebuah komponen. Dari konsep inilah, kemudian muncul beberapa paradigma pengembangan terbaru. Salah satunya ialah pengelolaan yang *terdesentralisasi*. Pengelolaan *terdesentralisasi*, dalam hal ini *microservices* dapat menjadi solusi. Jika sentralisasi dari aplikasi *monolithic* cenderung membuatnya lebih memilih menggunakan satu *database logis* untuk menghasilkan data yang lebih seragam, atau demi kemudahan dalam *lisensi*, maka *desentralisasi* yang digunakan *microservices* justru cenderung melakukan pendekatan yang disebut dengan *Polyglot Persistence*, yakni membiarkan masing-masing *services* untuk mengelola *database* mereka sendiri [14]

Monolithic adalah suatu arsitektur yang mengambarkan sebuah aplikasi yang menjalankan semua logika dalam satu *server* aplikasi [15]. Aplikasi yang dibangun dengan arsitektur *monolithic* hanya dijalankan dengan menggunakan satu server aplikasi sehingga hanya memerlukan proses pemeliharaan aplikasi pada satu *server* aplikasi. arsitektur *monolithic* merupakan aplikasi perangkat lunak yang modulnya tidak dapat dijalankan secara independen. Dengan menggunakan arsitektur *monolithic*, setiap aplikasi dijalankan secara bersamaan dikarenakan seluruh modul-modul aplikasi terdapat di dalam sebuah aplikasi yang sangat besar.

Docker adalah sebuah *platform* terbuka untuk siapapun yang bertujuan menggunakan sebuah platform untuk membangun, mendistribusikan dan menjalankan aplikasi dimanapun seperti *laptop, data center, virtual machine* ataupun *cloud. Docker* menggunakan arsitektur *client-server, docker client* menghubungi *Docker daemon*, yang melakukan pekerjaan berat, menjalankan, dan mendistribusikan *Docker container*. Kedua *Docker client* dan *daemon* dapat berjalan pada sistem yang sama. *Docker client* dan *daemon* berkomunikasi *via sockets* atau lewat *API* yang disediakan *Docker*.

REST (Representational State Transfer) merupakan standar arsitektur komunikasi berbasis *web* yang sering diterapkan dalam pengembangan layanan berbasis *web*. Penggunaan *HTTP (Hypertext Transfer Protocol)* sebagai protokol untuk komunikasi data. *REST* pertama kali diperkenalkan oleh Roy Fielding pada tahun 2000. Pada arsitektur *REST*, *REST server* menyediakan sumber data dan *REST client* mengakses dan menampilkan *resource* untuk penggunaan selanjutnya. Setiap sumber data diidentifikasi oleh *URI (Universal Resource Identifiers)* atau *global ID*. Sumber data direpresentasikan dalam bentuk format teks, *JSON* atau *XML*.

METODE PENELITIAN

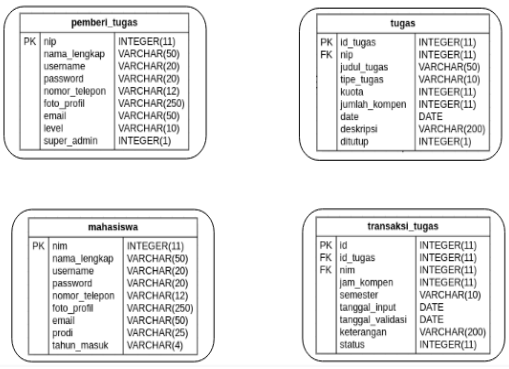
A. Pengumpulan Data

Observasi, observasi dilakukan di admin pengembang Sistem Informasi kompen jurusan teknologi informasi Polinema untuk mendapatkan informasi tentang, alur kerja sistem, str*uktur database.*

Studi Literatur, Studi literatur dilakukan guna menunjang pembuatan sistem berasitektur *microservices* dengan cara membaca, mencatat, serta memahami sumber – sumbur yang berkaitan, yang dipelajari adalah studi mengenai arsitektur *monolithic*, *microservices* dan literatur seperti jurnal, buku, dan sumber ilmiah yang terdapat pada internet yang bersangkutan dengan topik penelitian.

B. Pengolahan Data

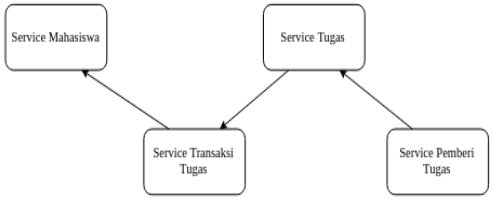
Data yang didapat dalam hal ini, data arsitektur *monolithic* dipecah menjadi bagian kecil yakni *microservices* berdasarkan proses bisnis dan kepemilikan data, yang nantinya akan dilakukan pengujian dan pengukuran dengan parameter *response metric*.



Gambar 1. Struktur tabel yang telah dilakukan *refactoring*

Refactoring database, yakni mengubah skema *database* dan memindahkan data. Proses pengubahan skema *database* dilakukan dengan memecah tabel berdasarkan kepemilikan data. Selanjutnya menentukan pola *database*, dalam penelitian ini dipilih *schema-per-service* yaitu semua *service* memiliki *database* masing-masing akan tetapi *database* tersebut berada dalam satu node *database server* agar jika terdapat salah satu *database* yang mati maka *service* yang lain dengan *database* yang masih menyala akan tetap bisa dijalankan. Setelah melakukan proses mengubah skema *database* dan menentukan pola *database*, tahapan selanjutnya memindahkan data dari skema *database* yang lama ke skema database yang baru. Hasil *refactoring database* nantinya setiap satu *database* akan mewakili satu *service*

Hubungan antar *service*, Berdasarkan proses *refactoring database* dihasilkan 4 *service* yaitu *service* pemberi tugas, *service* tugas, *service* transaksi tugas, dan *service* mahasiswa, Untuk mengetahui hubungan antar *service*, maka dibuatlah sebuah diagram keterhubungan antar *service*. Hubungan antar sever tersebut ditunjukkan oleh Gambar 2.



Gambar 2. hubungan antar *service*

HASIL DAN PEMBAHASAN

A. Performance pada arsitektur aplikasi

Pengujian ini digunakan untuk menguji kinerja antara aplikasi yang menggunakan arsitektur *monolithic* dengan aplikasi yang menggunakan arsitektur *microservice*, pengujian akan dilakukan dengan tiga skenario yaitu 500, 1000, 5000 jumlah *request.* Pengujian ini dilakukan pada beberapa *service* yaitu service mahasiswa, pemberi tugas, tugas dan transaksi tugas.

Tabel 1. Hasil pengujian service pemberi tugas

Tugas	Sample	Average	Min	Max	error Rate(%)	Throughput /Sec
monolith	500	12645	299	27953	0	17.8
microservice	500	3555	49	8447	0	53.2
Monolith	1000	35276	139	25195	11.2	4.0
microservice	1000	4218	341	27317	0	35.8
monolith	5000	50571	0	167510	42.8	29.8
microservice	5000	21994	0	151663	6.4	32.6

Pada tabel 1 menunjukan hasil perbandingan antara aplikasi yang menggunakan service monolitik dengan aplikasi dengan menggunakan arsitektur microservice, pada semua pengujian *sample* didapatkan nilai rata-rata *response microservice* jauh selalu lebih baik dari pada *monolithic*, dan *error* pada *microservice* didapatkan nilai lebih kecil dari pada *monolithic.*

Tabel 2. Hasil pengujian service mahasiswa

Mahasiswa	Sample	Average	Min	Max	error Rate(%)	Throughput /Sec
monolith	500	15450	1013	51606	3.8	9.5
microservice	500	3710	851	103723	0	4.8

Monolith	1000	14391	150	67908	1.3	14.5
microservice	1000	5185	334	26534	0	36.9
monolith	5000	76081	0	327691	36.7	15.2
microservice	5000	32618	0	152846	13.1	32.6

Pada tabel 2 menunjukan hasil perbandingan antara aplikasi yang menggunakan *service* monolitik dengan aplikasi dengan menggunakan arsitektur *microservice*, pada semua pengujian *sample* didapatkan nilai rata-rata *response microservice* jauh selalu lebih baik dari pada *monolithic,* dan *error* pada *microservice* didapatkan nilai lebih kecil dari pada *monolithic.*

Tabel 3. Hasil pengujian service tugas

Tugas	Sample	Average	Min	Max	error Rate(%)	Throughput /Sec
monolith	500	7062	105	21253	0.2	232
microservice	500	2687	376	9058	0	502
Monolith	1000	24189	576	117990	20	85
microservice	1000	8138	612	18251	0	527
monolith	5000	66253	0	177268	57.1	281
microservice	5000	38663	0	805636	12.9	6.2

Pada tabel 3 menunjukan hasil perbandingan antara aplikasi yang menggunakan *service* monolitik dengan aplikasi dengan menggunakan arsitektur *microservice*, pada semua pengujian *sample* didapatkan nilai rata-rata *response microservice* jauh selalu lebih baik dari pada *monolithic,* dan *error* pada *microservice* didapatkan nilai lebih kecil dari pada *monolithic.*

Tabel 4. Hasil pengujian service transaksi tugas

Tugas	Sample	Average	Min	Max	error Rate(%)	Throughput /Sec
monolith	500	15046	372	38187	16.6	13.0
microservice	500	1985	364	6814	0	66.7
Monolith	1000	50115	937	140005	47.5	7.1
			5			
microservice	1000	5737	214	17344	0	57.3
monolith	5000	69502	0	132214	72.7	37.5
microservice	5000	18708	0	119556	12.9	41.5

Pada tabel 4 menunjukan hasil perbandingan antara aplikasi yang menggunakan *service monolithic* dengan aplikasi dengan menggunakan arsitektur *microservice*, pada semua pengujian *sample* didapatkan nilai rata-rata *response microservice* jauh selalu lebih baik dari pada *monolithic,* dan *error* pada *microservice* didapatkan nilai lebih kecil dari pada *monolithic.*

B. Hasil Pengujian performance pada pola database

Pengujian ini digunakan untuk menguji kinerja antara aplikasi *microservice* yang menggunakan pola *database schema-per-service* dengan pola *database Private-table-per-service* untuk mengetahui apakah pola dari *database* dapat mempengaruhi kinerja aplikasi, pengujian dilakukan dengan mengakases aplikasi dengan 5000 *request.* Pengujian ini dilakukan pada beberapa *service* yaitu *service* mahasiswa, pemberi tugas, tugas dan transaksi tugas.

Tabel 5. Hasil pengujian service pemberi tugas

Pemberi Tugas	Sample	Average	Min	Max	error Rate(%)	Throughput /Sec
Schema per service	500	1729	232	8252	0.00	430.8
Private table per service	500	973	200	392	0.00	650.4

Pada tabel 5 menunjukan hasil perbandingan pada *service* pemberi tugas antara aplikasi *microservice* yang menggunakan pola *database schema-per-service* dengan pola *database Private-table-per-service*, pada pengujian *sample* didapatkan nilai rata-rata *response time schema-per-service* jauh lebih baik dari pada *Private-table-per-service,* dan *error* pada *Private-table-per-service* didapatkan nilai lebih kecil.

Tabel 6. Hasil pengujian service mahasiswa

Mahasiswa	Sample	Average	Min	Max	error Rate(%)	Throughput /Sec
Schema per service	500	1498	109	7806	0.00	482.5
Private table per service	500	1040	51	3949	0.00	613.5

Pada tabel 6 menunjukan hasil perbandingan pada *service* mahasiswa antara aplikasi *microservice* yang menggunakan pola *database schema-per-service* dengan pola *database Private*-table-*per-service*, pada pengujian *sample* didapatkan nilai rata-rata *response time schema-per-service* jauh lebih baik dari pada *Private-table-per-service,* dan *error* pada *Private-table-per-service* didapatkan nilai lebih kecil.

Tabel 7. Hasil pengujian service tugas

Mahasiswa	Sample	Average	Min	Max	error Rate(%)	Throughput /Sec
Schema per service	500	1040	80	7124	0.00	601.0
Private table per service	500	1488	98	7412	0.00	564.5

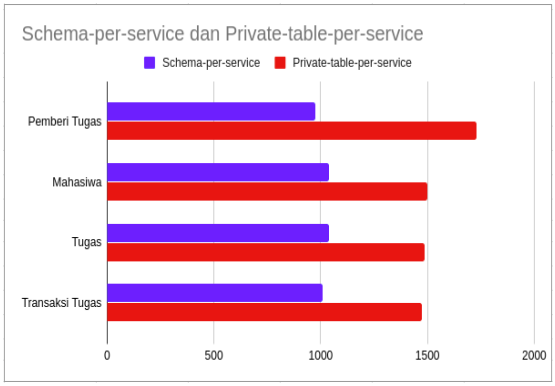
Pada tabel 7 menunjukan hasil perbandingan pada *service* tugas antara aplikasi *microservice* yang menggunakan pola *database schema-per-service* dengan pola *database Private-table-per-service*, pada pengujian *sample* didapatkan nilai rata-rata *response time schema-per-service* jauh lebih baik dari pada *Private-table-per-service,* dan *error* pada *Private-table-per-service* didapatkan nilai lebih kecil.

Tabel 8. Hasil pengujian service transaksi tugas

Transaksi Tugas	Sample	Average	Min	Max	error Rate(%)	Throughput /Sec
Schema per service	1472	173	7322	0	538.4	1472

Private table per service	1011	99	7432	0	493.1	1011

Pada tabel 8 menunjukan hasil perbandingan pada *service* transaksi tugas antara aplikasi *microservice* yang menggunakan pola *database schema-per-service* dengan pola *database Private-table-per-service*, pada pengujian *sample* didapatkan nilai rata-rata *response time schema-per-service* jauh lebih baik dari pada *Private-table-per-service*, dan *error* pada *Private-table-per-service* didapatkan nilai lebih kecil.



Gambar 3. Perbandingan rata-rata *response time*

Pada gambar 3 menunjukkan hasil rata-rata pada pengujian *schema-per-service* dengan *private-table-per-service.* pada pengujian didapatkan rata-rata *resonse time* untuk *schema-per-service* lebih baik dibandingkan *private-table-per-service.* Tapi terdapat kekurangan pada pola *database private-table-per-service* yakni jika *database* mati maka semua *service* akan ikut mati. Berbeda dengan pola *database schema-per-service* jika terdapat salah satu *database* yang mati maka *service* yang lain dengan *database* yang masih menyala akan tetap bisa dijalankan.

Tabel 9. Hasil pengujian fungsionalitas sistem

no	service	Pengujian	Berhasil
1	Service mahasiswa	Menampilkan data mahsiswa	ok
		Menginputkan data mahasiswa	ok
		Mengupdate data mahasiswa	ok
		Menghapus data mahasiswa	ok
		Menampilkan transaksi tugas untuk mahasiswa	ok
2	service pemberi tugas	Menampilkan data pemberi tugas	ok
		Menginputkan data pemberi tugas	ok
		Mengupdate data pemberi tugas	ok
		Menghapus data pemberi tugas	ok
		Menampilkan tugas untuk pemberi tugas	ok
3	service tugas	Menampilkan data tugas	ok
		Menginputkan data tugas	ok
		Mengupdate data tugas	ok
		Menghapus data tugas	ok
4	service tugas	Menampilkan data transaksi tugas	ok
		Menginputkan data transaksi tugas	ok
		Mengupdate data transaksi tugas	ok
		Menghapus data transaksi tugas	ok

KESIMPULAN

Berdasarkan proses pengujian dan analisis yang telah dilakukan, dapat diambil kesimpulan bahwa Aplikasi menggunakan arsitektur *microservice* mendapatkan waktu *response time* lebih singkat dari pada arsitektur *monolithic.* Arsitektur *microservice* dapat digunakan untuk meningkatkan *peformance* aplikasi. Pola *database schema-per-service* lebih unggul dibidang *performance* dari pada *private-table-per-service*, akan tetapi memiliki kekurangan yakni jika *database* mati maka semua *service* akan ikut mati.

REFERENSI

[1] B. P. Putra and Y. A. Susetyo, "Implementasi Api Master Store Menggunakan Flask, Rest Dan Orm Di Pt Xyz," *Sistemasi*, vol. 9, no. 3, p. 543, 2020, doi: 10.32520/stmsi.v9i3.899.

[2] R. Mufrizal and D. Indarti, "Refactoring Arsitektur Microservice Pada Aplikasi Absensi PT. Graha Usaha Teknik," *J. Nas. Teknol. dan Sist. Inf.*, vol. 5, no. 1, pp. 57–68, 2019, doi: 10.25077/teknosi.v5i1.2019.57-68.

[3] S. Daya et al. (2015, August). Microservices from Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach. (1 st edition). [On-Line]. Available:http://www.redbooks.ibm.com/abstracts/sg248275.html [January 15, 2019].

[4] N. Dragoni et al. (2017, September 6). Microservices: Yesterday, Today, and Tomorrow. [On-Line]. Available: https://link.springer.com/chapter/10.1007/978-3-319-67425-4_12 [January 15, 2019].

[5] P. L. L. Belluano, P. Purnawansyah, B. L. E. Panggabean dan H. Herman. 2020. "Sistem Informasi Program Kreativitas Mahasiswa berbasis Web Service dan Microservice".

[6] K. Katuwal. "Microservices: A Flexible Architecture for the Digital Age Version 1.0". American Journal of Computer Science and Engineering, Volume 3, September 2016, Pages 20-24.

[7] A. Messina, R. Rizzo, P. Storniolo, and A. Urso. "A Simplified Database Pattern for the Microservice Architecture" In Conference: DBKDA, 2016.

[8] R. Mufrizal dan D. Indarti. 2019. "Refactoring Arsitektur Microservice Pada Aplikasi Absensi PT. Graha Usaha Teknik

[9] G. Munawar dan A. Hodijah. 2018. "Analisis model Arsitektur Microservice pada Sistem Informasi DPLK".

[10] E. Rubio-Drosdov, D. Diaz-Sanchez, A. Marin-Lopez, and F. Almenares, "A Framework for Microservice Migration and Performance Assesment," *Intell. Environ. 2020*, vol. 28, no. June 2020, pp. 291–300, 2020, doi: 10.3233/AISE200053.

[11] S. Hassan, N. Ali, and R. Bahsoon, "Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity," *Proc. - 2017 IEEE Int. Conf. Softw. Archit. ICSA 2017*, no. April 2018, pp. 1–10, 2017, doi: 10.1109/ICSA.2017.32.

[12] A. Furda, C. Fidge, O. Zimmermann, W. Kelly, and A. Barros, "Migrating Enterprise Legacy Source Code to Microservices: On Multitenancy, Statefulness, and Data Consistency," *IEEE Softw.*, vol. 35, no. 3, pp. 63–72, 2018, doi: 10.1109/MS.2017.440134612.

[13] M. Ali Temoor and B. Muzaffar Ali, "Architecture for Microservice Based System. A Report," no. January, 2020, doi: 10.13140/RG.2.2.17340.16004/1

[14] H. Suryotrisongko. 2017. "Arsitektur Microservice untuk Resiliensi Sistem Informasi".
[15] I. Fadila Putra. 2019. "Penerapan Arsitektur Microservice Pada Sistem Informasi Sertifikasi".