# *Refactoring* Arsitektur *Microservice* pada Aplikasi Management Information System of *LP3I* Menggunakan *Strangler Pattern*

**Haisyam Maulana**[1]
Sistem Informasi
Universitas Galuh
Ciamis, Indonesia
e-mail: [1]haisyammaulana22@gmail.com

Diajukan: 12 April 2023; Direvisi: 14 Agustus 2023; Diterima: 18 September 2023

***Abstrak***

*Perubahan proses bisnis dalam sebuah kelembagaan atau perusahaan mempengaruhi sistem informasi. Perubahan tersebut berdampak terhadap sistem informasi yang dikembangkan. Namun, Management System of LP3I di Politeknik LP3I Kampus Tasikmalaya arsitektur yang digunakan menggunakan monolitik. Arsitektur tersebut dianggap sulit dikembangkan, maka dilakukan refactoring menjadi arsitektur microservice. Refactoring microservice memiliki tujuan untuk memudahkan pengembangan sistem tanpa mengganggu sistem yang sudah berjalan. Strangler pattern dijadikan metode dalam 10 tahap proses refactoring arsitektur dan dua tahap refactoring database yaitu mengubah skema database kemudian memindahkan data. Setelah itu, kemudian diuji mengunakan pendekatan heuristic usability dengan nilai diatas 73%. Hasilnya arsitektur microservice menggunakan strangler pattern berhasil dirancang dan direkomendasikan berupa blueprint.*

***Kata kunci:*** *Refactoring, Arsitektur microservice, Metode strangler pattern.*

***Abstract***

*Changes in business processes in an institution or company affect the information system. These changes have an impact on the information system being developed. However, the Management System of LP3I at Politeknik LP3I Kampus Tasikmalaya, uses a monolithic architecture. This architecture was considered difficult to develop, so it was refactored into a microservice architecture. Refactoring microservices has the goal of facilitating system development without disrupting existing systems. The Strangler pattern is used as a method in 10 stages of the architectural refactoring process and two stages of database refactoring, namely changing the database schema and then moving data. After that, then tested using the heuristic usability approach with a value above 73%. As a result, a microservice architecture using the strangler pattern was successfully designed and recommended in the form of a blueprint.*

***Keywords****: Refactoring, Microservice architecture, Metode strangler pattern.*

## 1. Pendahuluan

Lembaga pendidikan dan pengembangan profesi (LP3I) cabang Tasikmalaya sebagai usaha jasa layanan pendidikan telah resmi sebagai perguruan tinggi vokasi. Sehingga, kebijakan ini mempengaruhi sistem yang sudah ada pada lembaga tersebut. Politeknik LP3I Kampus Tasikmalaya merencanakan untuk tercapainya tujuan organisasi dan pelayanan terhadap *stakeholder*. Sistem dituntut untuk melakukan pengembangan dan pembaharuan dengan proses bisnis yang berbeda. Namun, Aplikasi yang dibangun menggunakan arsitektur monolitik membuat aplikasi sulit untuk dipahami dan dikembangkan.

Penelitian ini memfokuskan untuk mengembangkan sistem dengan tujuan untuk mengarahkan dan menyesuaikan terhadap kebijakan perguruan tinggi vokasi. Sistem ini diharapkan memudahkan penggunaan data agar dapat menghemat waktu dalam pelayanan kepada mahasiswa. Untuk itu arsitektur *microservices* ini digunakan untuk menyelesaikan permasalahan di atas. Arsitektur *microservices* adalah gaya arsitektur perangkat lunak yang memerlukan dekomposisi fungsional dari suatu aplikasi [1] . Dengan

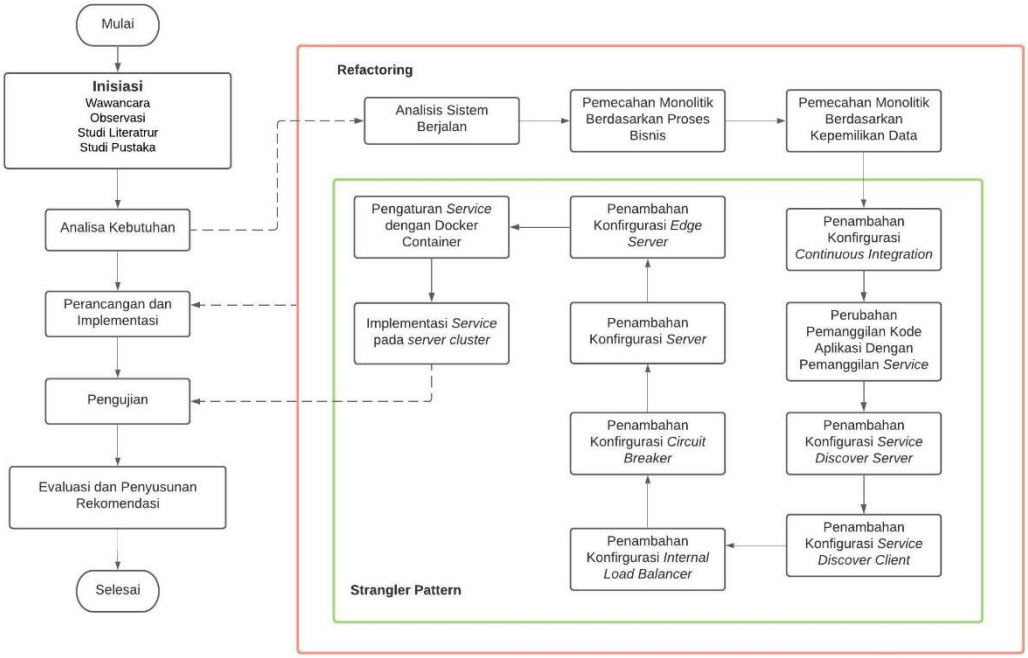
menggunakan arsitektur *microservices* ini dapat menyelesaikan permasalahan di atas, karena dipecahnya *service* secara detail [2] [3].

Management Information System of LP3I (MISIL) merupakan sebuah sistem informasi berbasis web yang digunakan oleh *staff* Politeknik LP3I Kampus Tasikmalaya. Sebagai media informasi untuk kebutuhan mengelola dari penerimaan mahasiswa baru sampai, penempatan kerja. Politeknik LP3I Kampus Tasikmalaya mengalami kesulitan karena sistem yang akan dikembangkan berdasarkan pada unit tertentu dan kebijakan yang berubah.

Selain itu, permasalahan yang dihadapi dalam pengembangan sistem informasi yaitu tidak menghilangkan jasa layanan *collage* bahkan menambah layanan baru berupa *short course*, sehingga pembangunan sistem informasi yang hanya berdasarkan kepada kebutuhan saat itu belum tepat atau memiliki nilai manfaat yang optimal. Oleh karena itu, perlu dilakukan *refactoring* arsitektur monolitik menjadi arsitektur *microservice* pada aplikasi MISIL berdasarkan unit tertentu yang sesuai dengan tujuan Lembaga.

## 2. Metode Penelitian

Metodologi penelitian yang digunakan dalam proses *refactoring* arsitektur monolitik hingga model arsitektur *microservice* yaitu *strangler pattern* [4] [5]. Dari tahapan – tahapan *refactoring* diawali dengan inisiasi kemudian 13 tahapan [6] yang dapat dilihat pada gambar 1. Menjelaskan kronologis penelitian, termasuk desain penelitian, prosedur penelitian (dalam bentuk algoritma, *pseudocode*, atau lainnya), bagaimana menguji dan mengumpulkan data [7] [8]. Deskripsi alur penelitian harus didukung oleh referensi sehingga penjelasan yang ditulis dapat diterima secara ilmiah.




Gambar 1. Tahapan Penelitian

Tahapan penelitian diawali dengan inisiasi. Inisiasi adalah proses pengumpulan data untuk penelitian ini yaitu wawancara, observasi, Studi Literatur, Studi Pustaka. Hasil dari inisiasi ini menjadi bukti kenapa penelitian ini dilakukan. Kemudian data – data yang terkumpul dijadikan sumber untuk melakukan pengolahan dan pengujian.

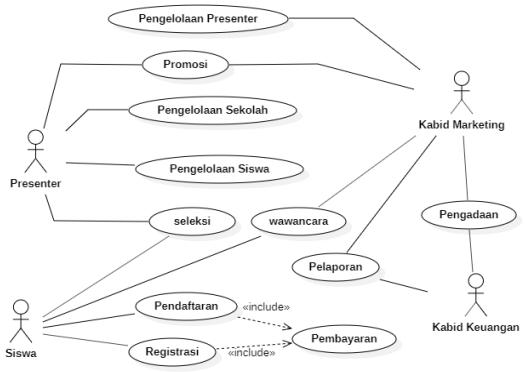
Tahapan selanjutnya yaitu analisa kebutuhan, pada tahap ini dilakukan perancangan dan implementasi mengikuti tahap *refactoring*. 13 tahapan yang terlihat pada gambar 1 dibatasi warna merah merupakan proses *refactoring*. Sebagian besar proses *refactoring* menggunakan *Strangler Pattern* yang dapat dilihat dengan batas warna hijau pada gambar 1 [9].

Tahapan ketiga adalah pengujian menggunakan pendekatan heuristik secara *usability*, Pada tahapan ini, pengujian perancangan menggunakan pendekatan heuristic *usability*. Dengan mengumpulkan hasil kuesioner, dengan jumlah responden 11 di antaranya 7 staf *marketing*, 2 staf keuangan dan 2 staf akademik. Analisa *Usability* [10].

Tahapan terakhir adalah evaluasi dan penyusunan rekomendasi, pada tahap ini hasil dari penelitian ini disusun berupa *Blueprint* Arsitektur untuk diajukan ke Politeknik LP3I Kampus Tasikmalaya.

## 3. Hasil dan Pembahasan

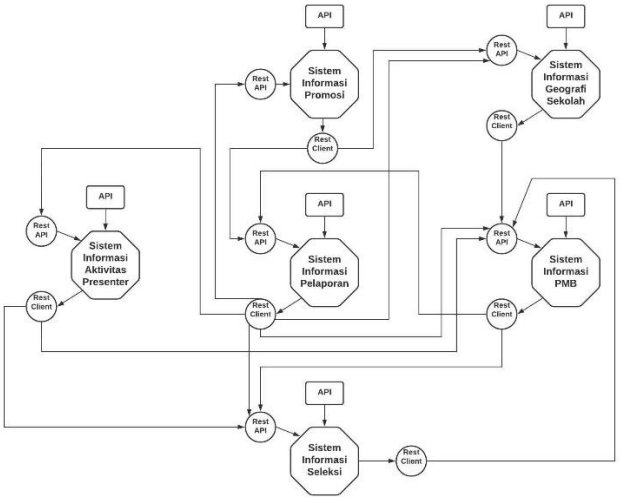
Hasil sistem yang dibangun menggunakan arsitektur *microservice* pada penelitian ini, memiliki beberapa hasil dari analisa sistem yang berjalan pada gambar 2. Di antaranya hasil pemecahan proses bisnis, hasil pemecahan kepemilikan data, hasil pengujian dan hasil *refactoring* aristektur.




Gambar 2. Sistem Yang Berjalan
**(**Sumber: Dokumen Politeknik LP3I Kampus Tasikmalaya)

### 3.1. Hasil Pemecahan Proses Bisnis

Proses bisnis dan analisis sistem berjalan yang telah dibahas sebelumnya. Dilanjutkan pada tahap *refactoring* ini dengan tiga strategi [11] [12] yaitu stop *digging*, *split frontend & backend, dan extract service*. Hasil dari pemecahan proses bisnis dapat dilihat pada Gambar 2 menjadikan bentuk arsitektur monolitik yang kecil. Kemudian relasinya juga diubah menggunakan REST API & REST Client.



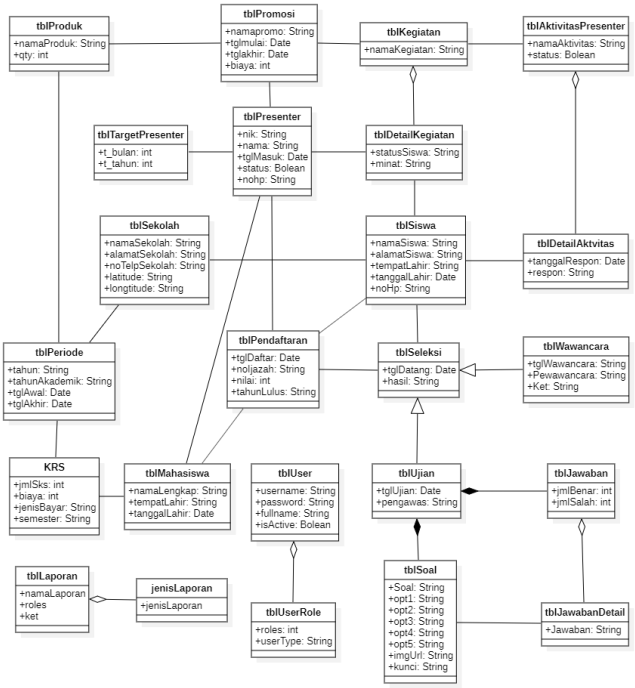

Gambar 3. Proses Bisnis *Microservice*

### 3.2. Hasil Pemecahan Kepemilikan Data

Arsitektur monolitik yang sedang berjalan memiliki perancangan tabel-tabel yang terdapat dalam *database*. Tabel tersebut perlu dilakukan pemecahan berdasarkan kepemilikan data. Langkah sebelum dilakukan pemecahan yaitu dengan mendefinisikan tabel-tabel pada Gambar 3. Selanjutnya dilakukan *mapping* proses bisnis dan table pada gambar 4. Tahap terakhir proses *refactoring database* pada Gambar 4.

### 3.2.1. Mendefinisikan Tabel

Tabel – tabel pada Gambar 3 menunjukan relasi antar tabel yang perlu di pisahkan, karena struktur tabel tersebut belum memenuhi kebutuhan arsitektur *microservice*. Selain itu, dengan adanya perubahan proses bisnis maka terjadi penambahan juga pada stuktur tabel.




Gambar 4. Tabel Sistem Berjalan

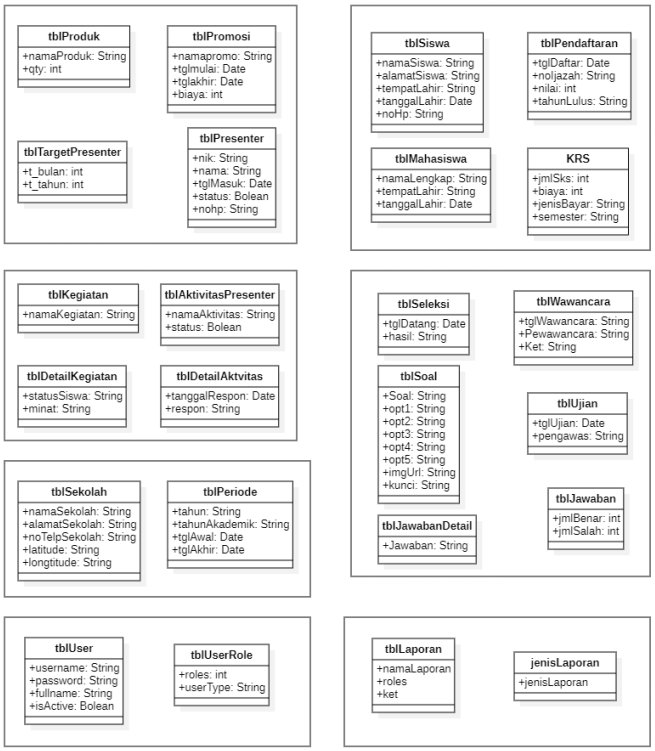
### 3.2.2. *Mapping* Proses Bisnis dan *Table*

Proses bisnis pada gambar 3 diambil dari hasil pemecahan proses bisnis sebelumnya. Dan tabel diambil dari struktur tabel yang ada pada gambar 4. Dengan dilakukannya pemetaan ini dilakukan pada Gambar 5 sudah terlihat tabel mana yang akan dikelompokkan nantinya.


Gambar 5. *Mapping* Diagram Proses Bisnis dan Tabel

### 3.2.3. *Refactoring Database*

Proses perubahan skema *database* yang dilakukan sesuai dengan *mapping* pada gambar 5. Berikut struktur tabel dan *mapping* yang telah dilakukan proses *refactoring* pada gambar. Struktur tabel pada gambar 6 merupakan struktur akhir dari tahap ini. Selanjutnya dilakukan proses pemindahan data dari skema *database* yang lama ke skema *database* yang baru. Pengolahan data tersebut diproses menggunakan perintah SQL.




Gambar 6. *Refactoring Database*

### 3.3. Penambahan Konfigurasi *Continuous Integration*

*Service* yang sudah dirancang sebelumnya, dilakukan pembuatan penyimpanan kode untuk masing-masing *service* pada aplikasi MISIL Divisi *Marketing*. Setelah itu, proses konfigurasi dimulai menggunakan gitlab *runner* pada setiap *service* [13] sehingga hanya diperlukan konfigurasi dengan kode. Pada penelitian ini, bahasa yang digunakan untuk membangun aplikasi yaitu PHP *Framework Laravel*.

### 3.4. Perubahan Pemanggila Kode Aplikasi Dengan Pemanggilan *Service*

Pada tahap ini, berdasarkan pemecahan sebelumnya menjadi *service.* Maka, seluruh pemanggilan kode menggunakan pemanggilan *service*. Pemanggilan tersebut melalui API *(Application Programming Interface)* [6]

### 3.5. Penambahan Konfigurasi *Service Discovery Server*

Tahap selanjutnya menambahkan konfigurasi *service discovery server* atau *service registry*. Tahap ini untuk mendaftarkan *service – service* pada aplikasi MISIL di Politeknik LP3I Kampus Tasikmalaya. Pengembangan aplikasi *service discovery server* merupakan penambahan konfigurasi ini.

*Spring cloud eureka server* menjadi *framework* yang digunakan pada penelitian ini. Menyiapkan *service registry*, yang menyimpan alamat *instance service* [14]. Diharapkan setiap *service* selama tersedia, *server edge, load balancer, atau service* lain dapat menemukan *service* yang diinginkan secara dinamis melalui *service registry.*

### 3.6. Penambahan Konfigurasi *Service Discovery Client*

Tahap ini dilakukan penambahan konfigurasi *service discovery client* diperlukan untuk *service registry* mengetahui bahwa *instance* baru telah digunakan [15]. Konfigurasi ini juga harus diimplementasikan pada masing-masing *service* pada aplikasi MISIL Politeknik Kampus LP3I

Tasikmalaya yang telah didekomposisi sebelumnya. Pada penelitian ini, konfigurasi tersebut menggunakan *framework spring cloud eureka client*.

### 3.7. Penambahan Konfigurasi *Internal Load Balancer*

Konfigurasi *internal load balancer* akan membantu *discovery server* dan *client* untuk mengatur *service* secara dinamis. Internal *load balancer* hanya mengatur beban internal yang ada pada *service discovery client*.

### 3.8. Penambahan Konfigurasi *Circuit Breaker*

Penambahan Konfigurasi *circuit breaker* ada untuk menghindari kegagalan sistem untuk *service* yang diakses saat sedang mati [16]. Tidak semua *service* yang ditambahkan konfigurasi ini, hanya beberapa *service* diantaranya pengelolaan presenter, pengelolaan tahun periode, pengadaan dan cat.

Pada penelitian ini, teknologi untuk implementasinya yaitu *hystrix circuit breaker*. Dalam *circuit breaker* memiliki tiga kondisi diantaranya *open, closed dan half-open*. Ketiga kondisi ini, ditentukan oleh dua konfigurasi yaitu *sender & receiver.*

### 3.9. Penambahan Konfigurasi Server

Penambahan konfigurasi server merupakan titik terakhir dalam semua bahasa pemrograman. Pada penelitian ini, bagaimana *instance* yang berjalan tidak memerlukan *deploying* secara berulang.

*Spring cloud config server* dipilih sebagai pengembangan konfigurasi menjadi penyimpan seluruh konfigurasi *service*. Karena setiap *service* memiliki teknik pengaturan masing-masing [15].

### 3.10. Penambahan Konfigurasi *Edge Server*

*Framework spring cloud zuul* merupakan pengembang aplikasi pada penelitian ini untuk menambah konfigurasi *edge server* [17]. *Masing* – masing *service* akan ditambahkan konfigurasi yang sama. Tujuannya untuk mempermudah dalam memantau status dan jumlah penggunaan setiap *service*.

### 3.11. Pengaturan *Service* Dengan *Docker Container*

Pengaturan masing – masing *service* pada tahap ini memiliki tiga proses pengaturan *service* dengan *docker container* [18] . Secara garis besar pengaturan ini untuk mengurangi penggunaan *resource* yang berlebihan pada aplikasi MISIL Politeknik LP3I Kampus Tasikmalaya. Tahapannya yaitu menambahkan kofigurasi *gradle*, menambahkan konfigurasi *docker image* dan membuat *docker image* dengan perintah docker kemudian menguploadnya ke *docker hub* [19].

### 3.12. Implementasi *Service* Pada *Server Cluster*

Tahap akhir pada proses *refactoring* dengan *strangler pattern* yaitu implementasi *service* pada *server cluster* [20]. Tujuan dari implementasi ini untuk menjalankan *service* melalui *docker container* di *server* yang berbeda. Pada penelitian ini, konfigurasi *docker swarm* yang digunakan. Teknologi *docker swarm* merupakan salah satu implementasi *server cluster* yang berfungsi untuk mengurangi kegagalan sistem aplikasi jika salah satu server mati [21].

### 3.13. Hasil Pengujian

Hasil dari pemecahan proses bisnis dan kepemilikan data akan diuji menggunakan heuristic *usability*. Berikut range tingkat *usability* yang digunakan pada tabel 1. Dihitung bobot nilai setiap indikator dengan bobot tertinggi 4 (empat) dan bobot terendah 1 (satu) maka ditemukan persentase tersebut.

Tabel 1. *Range* Tingkat *Usability*

| Skor | Kualifikasi | Rekomendasi |
|---|---|---|
| 87 – 91 | Sangat Setuju | Ya |
| 82 – 86 | Setuju | Ya |
| 77 – 81 | Cukup Setuju | Ya |
| 72 – 76 | Kurang Setuju | Tidak |

#### 3.13.1. Hasil Perhitungan Variabel Kualifikasi Sangat Setuju dan Setuju

Berdasarkan pernyataan setiap variabel dalam kategori sangat setuju dan setuju mendapatkan hasil di atas 79%. Hasil dari perhitungan tersebut dapat dilihat pada tabel 2 sebagai berikut:

Tabel 2. Hasil Perhitungan *Variable* Sangat Setuju dan Setuju

| No | Variabel | Pernyataaan |
|---|---|---|
| 1 | H2 - *Match Between System and the Real World* | Pengaturan MISIL dan navigasi menggunakan Bahasa Indonesia {P4} |
| 2 | H3 - *User Control and Freedom* | Mobile Application MISIL untuk Aktivitas Presenter dan Promosi {P5} |
| 3 | H5 - *Error Prevention* | Saat menggunakan Aplikasi MISIL jika terjadi error akan ditampilkan error-nya berupa notifikasi bukan informasi bahasa pemrograman atau blank screen. {P8} |
| 4 | H8 - *Aesthetic and Minimalist Design* | Desain Aplikasi MISIL dengan menggunakan warna yang soft {P10} |

### 3.13.2. Hasil Perhitungan Variabel Kualifikasi Sangat Setuju dan Setuju

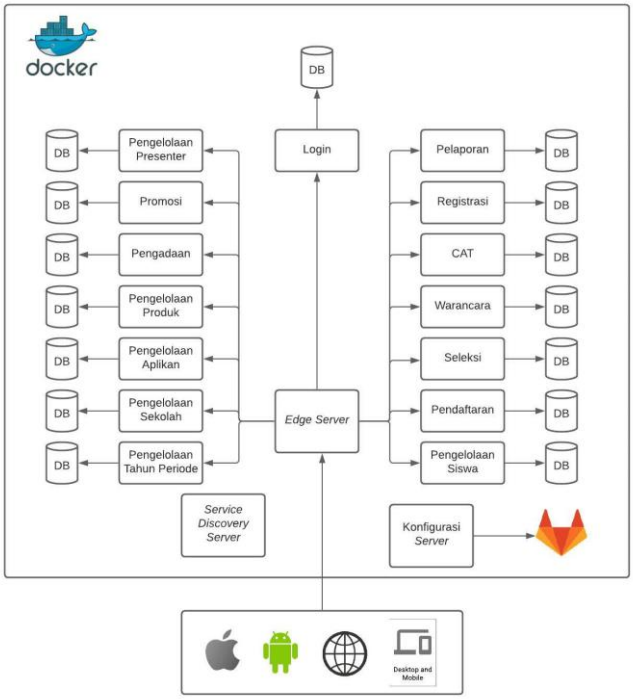
Untuk pernyataan setiap indikator dalam kategori cukup setuju dan kurang setuju masih mendapatkan hasil yang baik yaitu di atas 73%. Hasil perhitungan tersebut dapat dilihat pada tabel 3.

Tabel 3. Hasil Perhitungan *Variable* Cukup Setuju dan Kurang Setuju

| No | Variabel | Pernyataaan |
|---|---|---|
| 1 | H1 - *Visibility of System Status* | Aplikasi MISIL memberikan informasi target presenter pertahun dan perbulan beserta bonus yang didapat {P1}<br>Aplikasi MISIL memberikan informasi lokasi sekolah dengan Peta Geografinya {P2}<br>Aplikasi MISIL dapat membuat pelaporan secara dinamis dan dapat dilihat oleh user yang ditentukan {P3} |
| 2 | H4 - *Consistency and Standard* | Penggunaan bahasa dalam aplikasi MISIL dan dilapangan menggunakan istilah yang sama {P6}<br>Saat mengisi data yang wajib diisi memiliki keterangan "harus diisi!" {P7} |
| 3 | H6 - *Recognition Rather than Recall* | Aplikasi MISIL memberikan pesan atau notifikasi yang jelas. jika terjadi kesalahan input atau ketidaksesuaian data. {P9} |

### 3.14. Hasil *Refactoring* Arsitektur

Hasil dari *refactoring* aplikasi MISIL Politeknik LP3I Tasikmalaya pada divisi *Marketing*. Berdasarkan hasil pengujian perangkat lunak menggunakan pendekatan heuristic *usability* pada arsitektur *microservice*. Dapat disimpulkan bahwa arsitektur *microservice* yang telah dikembangkan dapat direkomendasikan seperti pada gambar 7




Gambar 7. *Blueprint* Arsitektur *Microservice* MISIL Politeknik LP3I Kampus Tasikmalaya

## 4. Kesimpulan

Penelitian ini terlihat jelas perbedaan arsitektur *microservice* dengan monolitik. Secara umum arsitektur monolitik dapat dikembangkan menjadi arsitektur *microservice*. *Refactoring* menggunakan strategi *strangler pattern* dan 13 tahapan *refactoring* menghasilkan rancangan arsitektur *microservice* pada Aplikasi MISIL Politeknik LP3I Kampus Tasikmalaya.

## Daftar Pustaka


[1] A. Messina, R. Rizzo, P. Storniolo dan A. Urso, A Simplified *Database Pattern* for the *Microservice* Architecture, Palermo: IARIA, 2016.

[2] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin dan L. Safina, *Microservices*: Yesterday, Today, and Tomorrow, Springer International Publishing AG, 2017.

[3] T. Ueda, T. Nakaike dan M. Ohara , Workload Characterization for *Microservices*, Tokyo: IEEE, 2016.

[4] K. Finnigan, Enterprise Java *Microservices*, Shelter Island: Manning Publications Co, 2018.

[5] S. D. Santis, L. Florez, D. V. Nguyen dan E. Rosa, Evolve the Monolith to *Microservices* with Java and Node, IBM Corp, 2016.

[6] R. Mufrizal dan D. Indarti, "*Refactoring* Arsitektur *Microservice* Pada Aplikasi Absensi PT. Graha Usaha Teknik," *Jurnal Nasional Teknologi dan Sistem Informasi* , pp. 1-12, 2019.

[7] F. Sulistiana, Strategi Merancang Arsitektur Sistem Informasi Masa Kini, Jakarta: PT. Elex Media Komputindo, 2019.

[8] S. Newman, Monolith to *Microservices*, Sebastopol: O'Reilly Media, Inc, 2018.

[9] A. Balalaie, A. Heydarnoori, P. Jamshidi, D. A. Tamburri dan T. Lynn, "*Microservices* migration *patterns*," *ResearchGate,* pp. 1-25, 2018.

[10] A. Ali, E. Pramana dan S. Tjandra, "Evaluasi Heuristik Pada Web Based Learning Untuk Meningkatkan Aspek *Usability* Sistem," *Jurnal Insand Comtech,* vol. 1, p. 10, 2016.

[11] C. Richardson, *Microservices Patterns* With Example Java, Shelter Island: Manning Publications Co, 2018.

[12] C. Richardson dan F. Smith, Microsevices From Design to Deployment, NGINX, 2016.

[13] T. K. A. W. H. T. J. J. N. Sendy Ferdian, "Continuous Integration and Continuous Delivery Platform Development of Software Engineering and Software Project Management in Higher Education," *JUTISI,* vol. 7, no. 1, pp. 59-70, 2021.

[14] A. L. Davis, Spring Quick Reference Guide, Berkeley, CA.: Apress, 2020.

[15] I. H. S. John Carnell, Spring *Microservices* in Action, Second Edition, Shelter Island, NY: Manning Publication Co., 2021.

[16] K. S. W. D. S. Falahah, "Circuit Breaker in *Microservices*: State of the Art and Future Prospects," *In IOP Conference Series: Materials Science and Engineering,* vol. 1077, no. 1, 2020.

[17] J. Sun, "Design and Implementation of Dormitory Repair Management System based on Spring Cloud Microsevices," *International Core Journal of Engineering,* vol. 7, no. 1, pp. 202-204, 2021.

[18] S. M. T. N. J. C. Kelly Brady, "Docker Container Security in Cloud Computing," dalam *10th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA, 2020.

[19] H. J. B. M. A. Babak Bashari Rad, "An Introduction to Docker and Analysis of its Performance," *International Journal of Computer Science and Network Security,* vol. 17, no. 3, pp. 228-235, 2017.

[20] P. M. Joseph W. Yoder, "*Strangler pattern*s," dalam *Proceedings of the 27th Conference on Pattern Languages of Programs*, 2020.

[21] C. K. Fabrizio Soppelsa, Native Docker Clustering with Swarm, Birmingham, Mumbai: Packt Publishing Ltd., 2016.