PERANCANGAN ARSITEKTUR PERANGKAT LUNAK *MICROSERVICES* PADA APLIKASI *OPEN LIBRARY* UNIVERSITAS TELKOM MENGGUNAKAN gRPC

Jeremiah Ferdinand[1], Alvi Syahrina[2], Ahmad Musnansyah[3]
[1,2,3] Universitas Telkom, Bandung
[1]jeremiahferdinand@student.telkomuniversity.ac.id, [2]syahrina@telkomuniversity.ac.id, [3]musnansyah@telkomuniversity.ac.id

Abstrak

Aplikasi Perpustakaan *Open Library* Universitas Telkom merupakan sebuah *website* yang dikembangkan sebagai sarana informasi terbuka untuk setiap buku, karya ilmiah, dan jurnal dengan beberapa fitur seperti pencarian, katalog, dan reservasi buku. Berdasarkan pengumpulan data yang peneliti lakukan berupa *non-functional software testing*, aplikasi *existing* membutuhkan peningkatan pada *scalability dan performance* dikarenakan sistem tidak dapat melayani *request* pada jumlah dan waktu tertentu sesuai dengan skenario yang ditentukan. Penelitian ini bertujuan untuk mengimplementasikan arsitektur perangkat lunak dengan gaya arsitektur *Microservices*. Teknik yang digunakan untuk melakukan evaluasi dari penelitian ini dengan melakukan *non-functional software testing* untuk mengukur batas *user* yang bisa mengakses pada waktu bersamaan. Peneliti melakukan migrasi perangkat lunak dimulai dengan mengumpulkan setiap fungsi dan proses dari *existing* sistem dan memisahkan setiap proses bisnis ke *domain* independen dengan menerapkan konsep *domain driven design*. Setelah dilakukan penguraian proses bisnis, peneliti melakukan migrasi arsitektur perangkat lunak monolitik ke *microservices* dengan menerapkan konsep *strangler pattern* serta pengembangan sampai tahap *functional testing* pada setiap modul dan *non-functional testing* skenario yang telah ditentukan. Hasil dari pengembangan arsitektur perangkat lunak menggunakan *microservices*, organisasi dapat meningkatkan performa dan kebebasan dalam memilih teknologi tertentu sesuai dengan masalah yang sedang dihadapi.

Kata Kunci: ***Microservices, Open Library*, gRPC**

Abstract

Universitas Telkom Open Library application is a website developed as an open information facility for every book, scientific work, and journal with several features. Based on the data collection that the researcher did in the form of non-functional software testing. Existing applications require improvements in scalability and performance, the system cannot serve requests at a certain amount and time according to the specified scenario. This research aims to implement a software architecture with a Microservices architectural style. The technique used to evaluate this research is to perform non-functional software testing to measure the limit of users who can access it at the same time. Researchers do software migration starting by collecting every function and process from the existing system and separating each business process into an independent domain by applying the concept of domain-driven design. After describing the business process, the researcher migrated the monolithic software architecture to microservices by applying the strangler pattern concept as well as development to the functional testing stage for each module and non-functional testing scenarios that have been determined. As a result of developing software architecture using microservices, organizations can improve performance and freedom in choosing certain technologies according to the problems at hand.

Keywords: Microservices, Open Library, gRPC

I. PENDAHULUAN

Aplikasi Perpustakaan *Open Library* Universitas Telkom merupakan sebuah *website* yang dikembangkan sebagai sarana informasi terbuka untuk setiap buku, karya ilmiah, dan jurnal dengan beberapa fitur seperti pencarian, katalog, dan reservasi buku. Ketika jam sibuk, peneliti melakukan penelitian dimulai dari mengumpulkan data dengan *non-functional software testing* berupa *load test* untuk mengukur kemampuan dan kecepatan sistem dalam menangani *request* dalam jumlah besar, peneliti mendapatkan sistem tidak dapat menangani 200 *concurrent user* secara bersamaan dengan maksimal, terbukti dengan *down*-nya aplikasi di tengah pengetesan dan buruknya *response-time* yang peneliti dapatkan dalam pengetesan. Maka dari itu, peneliti menyimpulkan aplikasi *existing* Perpustakaan *Open Library* Universitas Telkom membutuhkan peningkatan pada sisi *scalability, availability, dan performance*.

Open Library merupakan organisasi dibawah naungan Universitas Telkom yang dimana kampus tersebut memiliki 7 fakultas dan kurang lebih 40 prodi dan 22348 jumlah mahasiswa serta 746 dosen, hal tersebut menjadi pertimbangan peneliti untuk memastikan bahwa aplikasi *Open Library* Universitas Telkom membutuhkan *scalable*

ISSN : 2828-0075

dan *performance* yang baik guna melayani berbagai *request* dari *user*.

Terdapat beberapa cara untuk melakukan komunikasi antara layanan dalam *distributed systems* khususnya *microservices*, salah satunya menggunakan gRPC yang terkenal dengan kecepatan dalam memproses suatu data secara *synchronous blocking*.

Berdasarkan permasalahan yang ada, peneliti mengusulkan pengembangan arsitektur dengan konsep *microservices* dengan menggunakan gRPC sebagai komunikasi antar *services* guna meningkatkan *Scalability, Availability, dan Stability* sebuah sistem.

Adapun tujuan penelitian ini adalah untuk mengetahui dampak penggunaan arsitektur perangkat lunak dengan gaya arsitektur *Microservices* dengan menggunakan gRPC sebagai pertukaran data atau informasi dari *services* yang dimiliki oleh aplikasi *Open Library* Universitas Telkom guna meningkatkan atau menerapkan *Scalability, Availability, & Stability Pattern* dari sebuah sistem yang dirancang.

II. KAJIAN TEORI

A. *Scalability, Availability,* & *Performance Patterns*

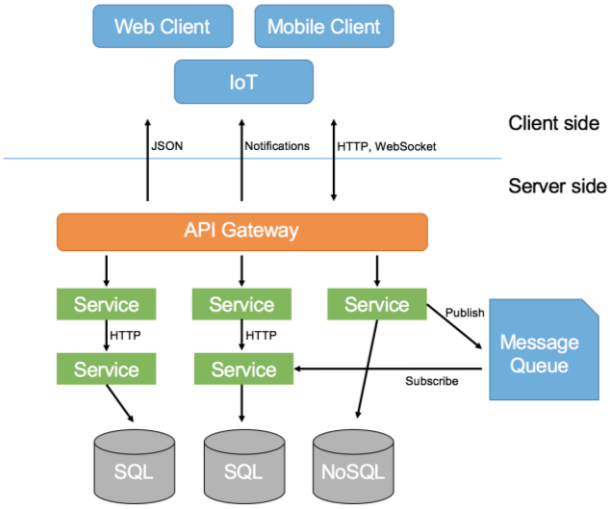
Scalability dalam sebuah sistem merupakan hal yang sangat krusial dalam kesuksesan sebuah organisasi, organisasi memerlukan platform yang *scalable* untuk menangani baik pertumbuhan *traffic* dan pertumbuhan volume data. *Scalability* merupakan kapabilitas sebuah aplikasi *enterprise* dan komponen ekosistemnya untuk menangani peningkatan beban dan permintaan tanpa mengorbankan efisiensi keseluruhannya [1] .

Availability aplikasi yang berkesinambungan dan maksimum sangat penting untuk kelangsungan dan kesuksesan bisnis. *High availability* tinggi merupakan faktor penting untuk aplikasi *critical, real-time*, dan perdagangan karena secara langsung mempengaruhi pendapatan sebuah bisnis dan *traffic* sebuah situs. *Availability* perangkat lunak selalu menjadi aspek penting dari proses arsitektur. *High availability* merupakan kemampuan sistem untuk terus tersedia bagi pengguna tanpa kehilangan sebuah layanan. *Availability* menunjukkan keseluruhan waktu aplikasi atau layanan tersedia untuk memenuhi kebutuhan pengguna. *High availability* sangat penting untuk operasi dan kelangsungan bisnis serta untuk mencapai tujuan akhir dari aplikasi perangkat lunak seperti pendapatan *online* dan kepuasan pelanggan. Untuk mencapai ketersediaan lima sembilan (99,999%), *downtime* maksimum yang ditoleransi adalah 300 detik per tahun. Karena faktor-faktor ini, ketersediaan tinggi sekarang menjadi persyaratan penting dan "harus dimiliki" untuk sebagian besar aplikasi perusahaan. Hal tersebut juga berdampak pada loyalitas pelanggan dan meningkatkan keunggulan kompetitif [1].

Kecepatan dan keresponsifan sebuah halaman *web* merupakan hal yang sangat penting sebagai faktor keberhasilan sebuah *website*. *Web* yang optimal memberikan keunggulan terhadap kompetitor dan meningkatkan *user experiences* secara keseluruhan. Hal tersebut memberikan dampak secara langsung terhadap arah bisnis terutama *traffic, conversion ratio*, dan, *index* kepuasan pelanggan. Dengan menjamurnya perangkat dan platform seluler di mana-mana, kecepatan *web* menjadi lebih relevan dari sebelumnya [1].

B. Arsitektur *Microservices*

Merupakan sebuah gaya arsitektur pada perangkat lunak dengan kumpulan antar layanan yang mudah di-*maintain* dan di-*test*, *loosely coupled*, di-*deploy* independen, terorganisir berdasarkan kapabilitas bisnis, dan dapat dimiliki oleh tim kecil. Setiap layanan yang ada pada arsitektur *Microservices* dapat menggunakan teknologi yang sesuai dengan kebutuhan tanpa mengetahui teknologi yang digunakan sebelumnya. *Microservices* memperkuat struktur modular yang sangat penting bagi tim yang sangat besar. Menurut Martin Fowler, ini adalah *key benefit* yang juga aneh jika dikatakan kelebihan, karena tidak ada alasan apapun mengapa *Microservices* memiliki struktur modular yang lebih kuat daripada monolitik.



GAMBAR 1
ARSITEKTUR *MICROSERVICES*

C. Migrasi Perangkat Lunak dengan *Strangler Pattern*

Strangler Pattern merupakan cara untuk melakukan *refactor microservices* dengan cara melakukan modernisasi di aplikasi baru dan menghapus aplikasi lama secara bertahap. *Strangler application* terdiri dari 2 tipe *services.* Pertama, ada layanan yang mengimplementasikan fungsionalitas yang sebelumnya berada di monolit. Kedua, ada layanan yang mengimplementasikan fitur baru [2].

D. gRPC

gRPC merupakan bagian *remote procedure* yang *open source* dan dikembangkan oleh Google pada tahun 2015. gRPC menggunakan HTTP/2, *Protocol Buffer* sebagai transportasi *interface* setiap pertukaran data. gRPC

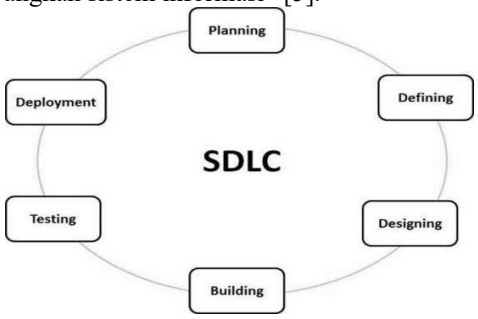
menyediakan fitur untuk autentikasi, *bidirectional streaming*, dan *flow control*. gRPC mendukung berbagai bahasa sebagai klien dan peladen untuk komunikasi data. Menurut *benchmark* yang telah dilakukan, *Protocol Buffers* dengan gRPC lebih unggul dari segi kecepatan dan performa karena metode serialisasi biner cepat dan sangat disarankan jika kecepatan dan performa menjadi pertimbangan dalam organisasi [3].

Pada implementasinya dalam gRPC, aplikasi klien dapat dengan langsung memanggil *method* yang ada di peladen di mesin yang berbeda sama seperti ketika komunikasi/pemanggilan *method* tersebut berada pada mesin yang sama.

E. *Software Development Life Cycle* (SDLC)

Software Development Life Cycle (SDLC) adalah sebuah tahapan atau siklus dalam industri pengembangan perangkat lunak untuk melakukan *design*, pengembangan, dan pengujian dari perangkat lunak dengan kualitas yang baik [4]. Dalam pengertian lain, Dora dan Dubey mengatakan,

"*Software Development Life Cycle (SDLC)* merupakan kerangka kerja yang digunakan untuk merencanakan, mengatur, dan melakukan kontrol proses untuk mengembangkan sistem informasi" [5].



GAMBAR 2
TAHAPAN PADA SDLC

E. *Software Testing*

Software Testing merupakan metode untuk memeriksa kelayakan sebuah produk perangkat lunak dengan kebutuhan yang telah ditentukan sebelumnya, dan untuk memastikan tidak ada kecacatan dalam produk. *Software Testing* melibatkan eksekusi manual dan otomatis. Tujuan dari *software testing* adalah untuk mengidentifikasi *error*, celah, dan kebutuhan yang belum terpenuhi dari kebutuhan awal yang telah ditentukan. Dalam implementasinya, melakukan testing untuk perangkat lunak dibagi menjadi beberapa jenis, diantara lainnya yang terkait dengan penelitian ini adalah sebagai berikut:

1. Functional Test

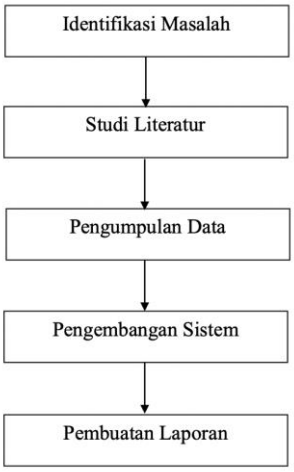
Functional Testing merupakan tipe *testing* untuk melakukan validasi sistem dari perangkat lunak dengan dengan spesifikasi dan kebutuhan yang telah ditentukan di awal. Tujuan dari *functional test* adalah untuk melakukan validasi pada setiap fungsi dari perangkat lunak dengan memberikan sebuah *input* atau masukan, dan melakukan verifikasi dengan *output* yang diberikan yang akan dibandingkan dengan spesifikasi dan kebutuhan yang telah ditentukan di awal. *Functional Test* terdiri dari *Unit Test* dan *Integration Test*.

2. Non Functional Test / Performance Testing

Williams mengemukakan: "Dalam konteks *web development*, *performance testing memerlukan* sebuah *tools software* untuk mensimulasikan bagaimana aplikasi bekerja dalam keadaan tertentu. Kuantitatif *performance testing* melihat dari sebuah metrik berupa *response time* dibandingkan pengujian kualitatif mengukur skalabilitas, kestabilan, dan interoperabilitas[6]. *Performance testing* lebih menekankan kepada ketahanan sebuah perangkat lunak pada kondisi tertentu terutama ketika perangkat lunak mendapatkan *traffic* dalam jumlah banyak ketimbang menguji fungsionalitas sebuah perangkat lunak. *Performance testing* terdiri dari *Load Test*, *Stress Test*, *Spike Test*, *Endurance Test*, *Scalability Test* dan *Volume Test.*

III. METODE

Untuk mencapai penelitian yang efisien, dan terstruktur, penyusun menggunakan kerangka kerja penelitian dalam menyusun penelitian ini. Kerangka kerja ini merupakan langkah-langkah yang dilakukan dalam penyelesaian masalah yang diteliti.



GAMBAR 3
KERANGKA KERJA PENELITIAN

Pada tahap identifikasi masalah, peneliti melakukan observasi terhadap sistem yang saat ini digunakan dengan melakukan sebuah pengujian berupa *non-functional test* terhadap Aplikasi *Open Library* Universitas Telkom. Dalam tahap studi literatur, dilakukan teknik penyusunan sistematis untuk memudahkan langkah-langkah yang diambil. Dengan melakukan studi literatur pada buku, jurnal dengan topik *Software Architecture & Design*, *Scalability, Availability, & Perfromance Patterns*, *Microservices*, *Migrating Monolith to Microservices*, gRPC, serta penelitian terdahulu yang berkaitan dengan *Software*. Data yang didapatkan dari studi

literatur digunakan sebagai acuan pada penelitian yang peneliti lakukan. Pada tahapan pengumpulan data, peneliti mengumpulkan data dengan penelitian lapangan berupa pengujian data terhadap sistem dengan sumber dari data Aplikasi *Open Library* Universitas Telkom, serta penelitian pustaka lewat dokumentasi teknologi terkait yang digunakan.

Dalam tahap pengembangan sistem terdiri dari dekomposisi layanan, migrasi arsitektur perangkat lunak dan implementasi pengembangan. Pada tahap dekomposisi layanan, untuk beralih ke arsitektur *Microservices*, layanan *Monolith existing system* dipecah menjadi layanan kecil yang independen. Peneliti menggunakan prinsip dan pola *Domain Driven Design* (DDD) untuk memudahkan menentukan batas dan memecah layanan berdasarkan domain. Pada tahap migrasi arsitektur perangkat lunak, peneliti melakukan migrasi arsitektur perangkat lunak dengan pola / *pattern Strangler Application.* Pada tahap implementasi pengembangan, peneliti melakukan implementasi pengembangan sistem sesuai dengan metode pengembangan yang telah ditentukan yaitu *Software Development Life Cycle* (SDLC). Setelah melakukan identifikasi dari permasalahan, riset dengan studi literatur, mengumpulkan data, dan melakukan pengembangan sistem, peneliti membuat sebuah laporan penelitian.

Untuk melakukan pengembangan arsitektur perangkat lunak dengan usulan baru, peneliti menggunakan *Software Development Life Cycle* (SDLC) sebagai acuan tahapan untuk pengembangan *backend* Aplikasi Perpustakaan *Open Library* Universitas Telkom. Peneliti juga menggunakan strategi *strangler pattern* untuk melakukan migrasi arsitektur perangkat lunak, strategi tersebut dirasa cocok dengan studi kasus *backend* Aplikasi Perpustakaan *Open Library* Universitas Telkom saat ini dengan bertahapnya pengembangan layanan baru guna melancarkan migrasi arsitektur perangkat lunak.

IV. HASIL DAN PEMBAHASAN

Analisis dan perancangan pada *backend* sistem Aplikasi Perpustakaan *Open Library* Universitas Telkom dilakukan dengan menganalisis masalah, solusi, dan kebutuhan fitur, dekomposisi layanan, migrasi arsitektur perangkat lunak, desain perangkat lunak berupa UML Diagram

A. Perancangan Migrasi Perangkat Lunak

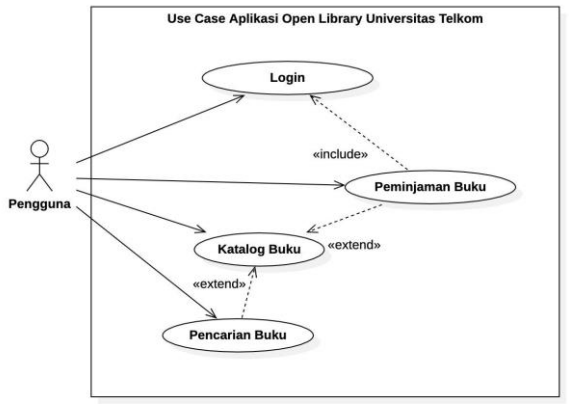
Untuk mengembangkan migrasi arsitektur perangkat lunak dari arsitektur *monolitik* ke arsitektur *microservices*, diperlukan dekomposisi pada proses bisnis utama dan memecahnya menjadi bagian kecil. Setiap layanan kecil seharusnya mengimplementasikan sebuah proses sesuai dengan fungsi proses bisnis yang terkait dalam bagian-bagian kecil (*cohesive),* dan dirancang secara bebas dan independen dan mudah dimodifikasi tanpa mengganggu komponen lain (*loosely coupled)*. Setelah peneliti melakukan dekomposisi layanan pada arsitektur perangkat lunak, selanjutnya peneliti melakukan migrasi perangkat lunak dari arsitektur *Monolith* ke *Microservices* dengan pola *Strangler Application*. Merujuk pada migrasi perangkat lunak dengan pola *Strangler Application*, peneliti melakukan migrasi / modernisasi aplikasi dengan cara menambahkan layanan baru dengan penerapan *microservices* secara bertahap sering pengembangan waktu. Layanan arsitektur *monolitik* Aplikasi *Monolith Open Library* Universitas Telkom akan berkurang dan *services-services* baru dengan penerapan *microservices* akan bertambah seiring berjalannya waktu.

Kebutuhan perangkat lunak yang digunakan pada migrasi arsitektur perangkat lunak *Open Library* Universitas Telkom, kategori perangkat lunak yang digunakan antara lain sistem operasi, *web server*, bahasa pemrograman, basis data, dan *cache.* Untuk merealisasikan pengembangan, dibutuhkan bahasa pemrograman, peneliti menggunakan beberapa bahasa pemrograman seperti Go (untuk layanan pencarian buku dan transaksi), NodeJs (untuk layanan katalog buku) dan PHP (untuk layanan autentikasi). Bahasa pemrograman digunakan pada penelitian ini digunakan untuk merealisasikan program ke dalam bentuk *backend* sistem yang didalamnya terdapat proses sesuai *requirement* yang telah ditentukan. Basis data digunakan sebagai wadah informasi pada sistem yang dikembangkan. Bahasa pemrograman dan basis data pada penelitian ini ditentukan berdasarkan kelebihan suatu bahasa pemrograman atau basis data dan permasalahan yang ingin dipecahkan pada penelitian ini.

Setiap layanan yang dibangun dalam pengembangan, dibutuhkan perangkat keras untuk menjalankan sistem. Daftar perangkat keras yang digunakan adalah 2 Core 2GB RAM dan 120GB SSD dari *provider* AWS (untuk layanan aplikasi *backend*), 4GB RAM dan 120GB SSD dari *provider* AWS (untuk layanan basis data dan tembolok / *cache*). Setiap *server* memiliki masing masing layanan dan teknologi mengacu pada konsep *Microservices.* Spesifikasi RAM dan SSD pada basis data ditentukan berdasarkan rata-rata *minimum* dan *recomended spesification* yang dianjurkan pada laman situs basis data dan *cache*. Untuk aplikasi *backend*, spesifikasi *core* dan RAM ditentukan berdasarkan beratnya proses yang ingin diselesaikan pada masing-masing layanan.

B. Perancangan Desain Perangkat Lunak

Use Case Diagram merupakan diagram yang menggambarkan fungsionalitas sistem serta kebutuhan yang diinginkan atau dikehendaki oleh pengguna. Pada Gambar 2 secara keseluruhan pada sistem Aplikasi *Open Library* Universitas Telkom. Fitur terdiri dari l*ogin* yang merupakan proses *authentication* antar *request* user dan sistem, fitur peminjaman buku untuk melakukan peminjaman dan pengembalian buku oleh *user*, fitur katalog buku untuk menampilkan katalog yang ada pada perpustakaan berupa *list* dan *detail*, Fitur Pencarian yang akan menampilkan *list* katalog buku paling mirip dengan masing masing buku dengan *keyword* yang dimasukan oleh *user*.



GAMBAR 2
USES A CASE DIAGRAM

C. Implementasi Perancangan Produk

API Autentikasi digunakan ketika pengguna melakukan *login*, API Autentikasi berada di layanan / *service authentication*. API Autentikasi menerima *body* berupa JSON dengan *attribute username* dan *password*, ketika autentikasi berhasil dilakukan, API Autentikasi akan melakukan keluaran JSON berupa pesan berhasil dan *token* untuk *credentials* pengguna. API *List* Katalog Buku digunakan ketika pengguna ingin melihat *listing* buku, API *List* Katalog berada di layanan / *service* katalog. API memberikan *output* berupa JSON berisikan data katalog buku. API Detail Buku digunakan ketika pengguna ingin melihat katalog buku, API Detail Buku berada di layanan / *service* buku. API memberikan *output* berupa JSON berisikan data detail katalog buku. API Pencarian Buku digunakan ketika pengguna melakukan pencarian dari katalog buku, API Pencarian Buku berada di layanan / *service search*. API Pencarian Buku menerima *body* berupa JSON dengan *attribute keyword*, ketika pencarian ditemukan, API Pencarian Buku akan mengeluarkan keluaran JSON berupa data sesuai pencarian pada katalog.

API Pinjam Buku digunakan ketika pengguna melakukan peminjaman buku dari katalog buku, API Peminjaman Buku berada di layanan / *service transaction*. API Peminjaman Buku menerima *body* berupa JSON dengan *attribute* id buku dan tipe buku, API Peminjaman Buku akan mengeluarkan keluaran JSON berupa pesan berhasil.

D. Skenario *Unit Test*, *Integration Test* dan *Performance Test*

Untuk melakukan pengujian kualitas pada setiap *modules* yang telah dibangun, dibutuhkan beberapa kasus atau *scenario unit test* yang berisi fungsi dan ekspektasi keluaran sebuah sistem. Untuk melakukan pengujian kualitas dalam konteks integrasi antar *system* dan modul, dilakukan sebuah skenario *integration test* dengan skenario yang telah ditentukan pada tahap pengumpulan *requirement*. Untuk melakukan pengujian kecepatan dan keandalan sebuah sistem dilakukan *performance test*. Tabel 1 menunjukan fungsi yang telah dirumuskan pada tahap pengumpulan *requirement* dan ekspektasi yang akan dilakukan oleh *unit* sistem.

TABEL 1
SKENARIO *TEST*

No	Fungsi	Ekspektasi			
		Unit Test	*Integration Test*	*Performance Test*	
				Jumlah *User*	Periode
1	*List* Katalog Buku	Fungsi menampilkan *list* katalog buku	*Return status code 200*	200	30 detik
2	Detail Buku	Fungsi menampilkan detail buku	*Return status code 200*	200	30 detik
3	Proses Pengembalian Buku	Fungsi melakukan proses pengembalian buku	*Return status code 200*	200	30 detik
4	Proses Peminjaman Buku	Fungsi melakukan *update* peminjaman buku	*Return status code 201*	200	30 detik
5	Autentikasi (*Login*)	Fungsi menampilkan autentikasi	*Return status code 200*	200	30 detik
6	Pencarian Buku	Fungsi menampilkan data hasil pencarian	*Return status code 200*	200	30 detik

E. Pembanding *Backend* Aplikasi Perpustakaan *Open Library* Universitas Telkom dengan Arsitektur Monolitik

Pada Tabel 2, dihasilkan pengujian dari *performance & load* test untuk mengukur kemampuan keandalan dan kecepatan aplikasi monolitik yang dibangun, *stress test* untuk mengukur kemampuan keandalan dengan pengujian dengan jumlah *user* konstan secara maksimal selama 30 detik dari aplikasi monolitik yang dibangun serta *spike test* untuk mengukur kemampuan keandalan dengan pengujian dengan jumlah *user* yang meningkat dan menurun secara dinamis selama 30 detik dari aplikasi monolitik yang dibangun.

TABEL 2 HASIL *TEST*

No	Nama Proses	*Performance & Load Test*				*Stress Test*		*Spike Test*			
		Average Response Time	Jumlah *Request*	*Failure Request*	Jumlah *User* Maksimal	*Average Response Time*	Jumlah *User* Maksimal	*Average Response Time*	Jumlah *Request*	*Failure Request*	Jumlah *User* Maksimal
1	*List* Katalog Buku	31.62 detik	117	5883	139	0.5 detik	200	0.5 detik	6000	0	200
2	Detail Buku	32.26 detik	122	5878	135	0.3 detik	200	0.3 detik	6000	0	200
3	Proses Pengembalian Buku	30.26 detik	102	5898	147	0.8 detik	200	0.8 detik	6000	0	200
4	Proses Peminjaman Buku	30.26 detik	102	5898	147	0.8 detik	200	0.8 detik	6000	0	200
5	Autentikasi (*Login*)	25.99 detik	80	5920	164	0.6 detik	200	0.34 detik	6000	0	200
6	Pencarian Buku	23.18 detik	137	5863	150	0.3 detik	200	0.3 detik	6000	0	200

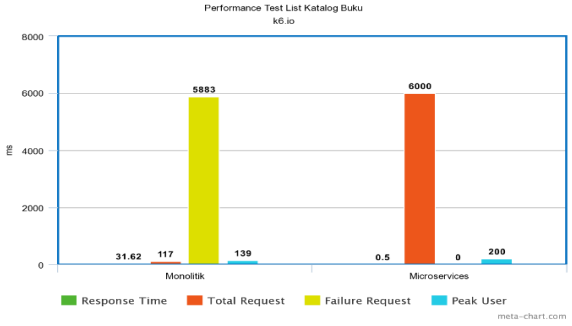
F. *Backend* Aplikasi Perpustakaan Open Library Universitas Telkom dengan Arsitektur *Microservices*

Pengujian dilakukan terhadap modul katalog buku, detail katalog buku, pencarian buku dan *integration test* dengan skenario yang telah ditentukan pada Tabel 3, hasil dari pengujian pada modul ini menunjukkan *test passed* atau sesuai dengan skenario dan ekspektasi.

TABEL 3
NON-FUNCTIONAL TEST

No	Nama Proses	*Performance & Load Test*				*Spike Test*		*Stress Test*			
		Average Response Time	Jumlah *Request*	*Failure Request*	Jumlah *User* Maksimal	*Average Response Time*	Jumlah *User* Maksimal	*Average Response Time*	Jumlah *Request*	*Failure Request*	Jumlah *User* Maksimal
1	*List* Katalog Buku	0.5 detik	6000	0	200	0.5 detik	200	0.5 detik	6000	0	200
2	Detail Buku	0.3 detik	6000	0	200	0.4 detik	200	0.3 detik	6000	0	200
3	Proses Pengembalian Buku	0.8 detik	6000	0	200	0.8 detik	200	0.8 detik	6000	0	200
4	Proses Peminjaman Buku	0.8 detik	6000	0	200	0.8 detik	200	0.8 detik	6000	0	200
5	Autentikasi (*Login*)	0.6 detik	6000	0	200	0.6 detik	200	0.6 detik	6000	0	200
6	Pencarian Buku	0.3 detik	6000	0	200	0.3 detik	200	0.3 detik	6000	0	200

Dengan pengujian *non-functional test* yang telah dilakukan, dapat disimpulkan bahwa proses pengujian setelah dilakukan migrasi perangkat lunak, performa sistem meningkat terlihat dari *Average Response Time* yang berkurang dan jumlah *user* maksimal yang meningkat. *Average Response Time* merupakan satuan kecepatan rata-rata sistem dalam melayani suatu proses. Jumlah *user* maksimal merupakan satuan waktu *user* yang bisa dilayani oleh sistem dalam satu waktu. Gambar 3 menunjukan peningkatan signifikan *response time* yang kecil pada *microservices* dan peningkatan jumlah *peak user* atau *user* yang bisa mengakses sistem secara bersamaan.



GAMBAR 3
BAGAN PENGUJIAN

V. KESIMPULAN

Berdasarkan penelitian yang dilakukan mengenai implementasi arsitektur perangkat lunak *microservices* pada aplikasi *Open Library* Universitas Telkom menggunakan gRPC, kesimpulan yang dapat ditarik yaitu terdapat beberapa cara untuk meningkatkan kecepatan dan skalabilitas sistem, pada penelitian ini, peneliti menerapkan konsep *distributed systems* yang diaplikasikan ke arsitektur perangkat lunak *microservices*, dengan kebebasan pemilihan teknologi, pembagian layanan secara independen menghasilkan performa yang lebih optimal dibandingkan dengan *existing* sistem dan pembanding arsitektur perangkat lunak monolitik. Penerapan gRPC pada penelitian ini dilakukan disetiap layanan yang berkomunikasi terhadap layanan lainnya menerapkan konsep arsitektur perangkat lunak *microservices.* Pada penerapannya, layanan yang membutuhkan data akan berkomunikasi dengan gRPC *server* lewat gRPC *client* yang dipasang pada setiap layanan.

REFERENSI

[1] Shivakumar S. K. *Architecting High Performing, Scalable and Available Enterprise Web Applications*. Massachusetts: Elsevier, 2014, pp. 1-57. doi.org/10.1016/C2013-0-14272-9

[2] Richardson C. *Microservices Patterns*. New York: *Manning Publications*, 2018, pp. 129.

[3] Kiraly S., dan Szekely S. "*Analysing RPC and Testing the Performance of Solutions*". Informatica, vol. 42, pp. 555-561, Sep. 2018. doi.org/10.31449/inf.v42i4.1510

[4] Dora, S. K., dan Dubey, P. *"Software Development Life Cycle (SDLC) Analytical Comparison and Survey on Traditional and Agile Methodology*". *National Monthly Refereed Journal of Research in Science & Technology*, vol. 2, pp. 22-30, Aug. 2013.

[5] Surya L. "*Systems Development Life Cycle Models". International Journal of Creative Research Thoughts* (IJRT), vol. 5, pp. 200-204, Oct.