

Implementasi dan Evaluasi Fungsional serta Performa Arsitektur *Microservices* pada Sistem Informasi Tahfiz

Rifa Sania*, Rosa Ariani Sukamto, Ani Anisyah

Program Studi Ilmu Komputer, Fakultas Pendidikan Matematika dan Ilmu Pengetahuan Alam, Universitas Pendidikan Indonesia, Bandung, Indonesia

Email: ¹*rifasania@upi.edu, ²rosa.ariani@upi.edu, ³anianisyah@upi.edu

Email Penulis Korespondensi: rifasania@upi.edu*

Submitted: 15/06/2026; Accepted: 18/08/2026; Published: 30/09/2026

Abstrak— Pengelolaan program tahfiz di sekolah menengah Islam umumnya masih dilakukan secara manual dan terpisah, sehingga berpotensi menimbulkan inefisiensi pencatatan dan kesulitan pemantauan. Penelitian ini bertujuan mengimplementasikan dan mengevaluasi arsitektur *microservices* pada Sistem Informasi Tahfiz untuk mendukung pengelolaan hafalan secara terstruktur. Penelitian menggunakan pendekatan studi kasus pada institusi pendidikan menengah di Bandung. Data kebutuhan diperoleh melalui wawancara semi-terstruktur dengan satu informan, sedangkan *prototype* dievaluasi oleh empat pengguna. Sistem terdiri atas tujuh *service*, API Gateway sebagai *single entry point*, dan pola *database per service*. Sistem di-deploy pada VPS Ubuntu dengan 2 vCPU dan RAM 2 GB, sedangkan pengujian menggunakan k6. Evaluasi mencakup *blackbox testing*, *load testing*, dan *endurance testing*. Seluruh 78 kasus uji *blackbox* berhasil dengan tingkat keberhasilan 100%. Setiap skenario *load testing* pada 10, 23, 50, dan 100 *virtual users* dijalankan sebanyak tiga kali. Pada 100 *virtual users*, p(95) berada pada rentang 2.385,20–2.918,52 ms, *throughput* 38,54–41,18 req/s, dan *error rate* 0,00%–0,05%. Satu kali *endurance testing* selama empat jam menggunakan 23 *virtual users* menghasilkan p(95) 936,31 ms, *error rate* 0,01%, dan penggunaan memori 1.800–1.900 MB. Hasil penelitian menunjukkan bahwa arsitektur *microservices* dapat diterapkan secara modular dan terintegrasi pada lingkungan yang diuji, sedangkan hasil *endurance testing* menggambarkan stabilitas performa selama skenario pengujian dan tidak digunakan untuk menyimpulkan *resilience* sistem.

Kata Kunci: Arsitektur *Microservices*; API Gateway; *Database per Service*; Pengujian Performa; Sistem Informasi Tahfiz

Abstract— Tahfiz program management in Islamic secondary schools is manual and fragmented, causing inefficient record keeping and monitoring. This study implements and evaluates a microservices architecture for a Tahfiz Information System at an institution in Bandung. Requirements were obtained through one semi-structured interview, and the prototype was evaluated by four users. The system comprised seven services, an API Gateway, and a database-per-service pattern. It was deployed on an Ubuntu VPS with 2 vCPU and 2 GB RAM and tested using k6. Evaluation included 78 black-box test cases, load testing, and endurance testing. All black-box test cases passed. Each load-testing scenario at 10, 23, 50, and 100 virtual users was executed three times. At 100 virtual users, p(95) ranged from 2,385.20 to 2,918.52 ms, throughput from 38.54 to 41.18 requests/s, and error rate from 0.00% to 0.05%. A single four-hour endurance test using 23 virtual users produced a p(95) of 936.31 ms, an error rate of 0.01%, and memory usage of 1,800–1,900 MB. The results show a modular and integrated implementation in the tested environment. The endurance test describes performance stability under the tested scenario without establishing resilience.

Keywords: Microservices Architecture; API Gateway; Database per Service; Performance Testing; Tahfiz Information System

1. PENDAHULUAN

Program Tahfiz Al-Qur'an merupakan salah satu program khusus yang diterapkan pada sejumlah lembaga pendidikan Islam. Dalam praktiknya, pengelolaan program tahfiz pada beberapa lembaga pendidikan masih dilakukan secara manual dan menggunakan media yang terpisah. Pencatatan data hafalan dan evaluasi masih ditemukan menggunakan lembaran kertas atau aplikasi *spreadsheet* seperti Microsoft Excel [1], [2]. Kondisi tersebut juga ditemukan pada salah satu sekolah tingkat menengah di Bandung yang menyelenggarakan program tahfiz bagi 369 siswa dengan dukungan lima Guru Tahsin dan Tahfiz Al-Qur'an (TTQ). Pencatatan perkembangan hafalan masih dilakukan secara manual oleh masing-masing Guru TTQ sehingga pada akhir semester data perlu direkap kembali ke dalam satu file Microsoft Excel sebagai laporan gabungan. Pengelolaan data secara manual dan terpisah dapat menimbulkan inefisiensi serta menyulitkan pemantauan dan evaluasi perkembangan hafalan secara cepat dan terintegrasi [1], [3], [4].

Penerapan Sistem Informasi (SI) menjadi salah satu langkah strategis untuk mendukung digitalisasi pengelolaan program tahfiz. Sistem informasi berbasis web dapat mendukung efisiensi pengelolaan data serta penyebaran informasi akademik dan nonakademik di lingkungan sekolah [5]. Namun, sistem informasi tahfiz tidak hanya membutuhkan fitur pencatatan, tetapi juga perlu mendukung beberapa proses utama, seperti pengelolaan pengguna dan akademik, pencatatan hafalan, pemantauan progres, evaluasi hafalan per juz, aktivitas harian, dan pelaporan. Banyaknya proses tersebut membutuhkan rancangan arsitektur yang mampu memisahkan tanggung jawab setiap bagian sistem secara lebih jelas.

Sebagian sistem informasi pendidikan, khususnya sistem *e-learning* atau *learning management system*, masih dikembangkan menggunakan arsitektur monolitik yang menggabungkan berbagai fungsi ke dalam satu kesatuan aplikasi [6]. Pendekatan monolitik memiliki keterikatan yang tinggi antarbagian sistem, sehingga perubahan pada satu modul dapat memengaruhi bagian lain dan menyulitkan proses pengembangan maupun

pemeliharaan sistem [7], [8]. Kondisi tersebut menjadi dasar perlunya pendekatan arsitektur yang lebih modular. Arsitektur *microservices* menjadi salah satu pendekatan yang dapat digunakan karena membagi aplikasi menjadi layanan-layanan kecil yang independen dan memiliki tanggung jawab fungsional tertentu, sehingga dapat meningkatkan modularitas, kemudahan pemeliharaan, dan fleksibilitas pengembangan sistem [6], [7], [9].

Perkembangan penelitian arsitektur perangkat lunak menunjukkan bahwa *microservices* semakin banyak digunakan sebagai alternatif terhadap arsitektur monolitik pada sistem yang memiliki kebutuhan pengembangan berkelanjutan. Auer et al. [10] membahas kerangka penilaian dalam proses transisi dari sistem monolitik menuju *microservices*, sedangkan Wijayanto et al. [11] merancang arsitektur *super app* layanan akademik yang skalabel dengan pendekatan *microservices*. Dirgantara et al. [12] dan Blinowski et al. [13] menunjukkan bahwa perbedaan antara arsitektur monolitik dan *microservices* tidak hanya terletak pada struktur aplikasi, tetapi juga dapat berdampak pada performa dan penggunaan sumber daya sistem. Dengan demikian, pemilihan arsitektur perlu disesuaikan dengan karakteristik sistem yang dikembangkan, bukan hanya berdasarkan tren teknologi.

Pada konteks sistem pendidikan, beberapa penelitian telah menerapkan *microservices* untuk mendukung pengelolaan layanan yang memiliki banyak proses. Dinova dan Utomo [6] mengembangkan arsitektur *microservice* pada *learning management system e-learning* menggunakan metode *web service*. Dahri et al. [14] menerapkan *microservices* pada *learning management system* berbasis *e-course* dengan memisahkan layanan agar pengembangan sistem dapat dilakukan secara lebih terarah. Ren et al. [15] juga menggunakan pendekatan *microservices* pada platform pembelajaran daring dengan menekankan pembagian modul sesuai fungsi layanan. Beberapa penelitian tersebut memperlihatkan bahwa *microservices* dapat digunakan pada sistem pendidikan yang memiliki kebutuhan layanan beragam. Namun, penerapannya tetap perlu disesuaikan dengan karakteristik dan alur proses pada masing-masing domain karena kebutuhan sistem pembelajaran daring berbeda dengan sistem pengelolaan tahfiz.

Penerapan *microservices* tidak hanya berkaitan dengan pemisahan aplikasi menjadi beberapa layanan, tetapi juga dengan penentuan batas tanggung jawab setiap layanan. Dekomposisi layanan berdasarkan kebutuhan bisnis menjadi salah satu pendekatan penting dalam perancangan sistem *microservices* [16]. Selain itu, API Gateway dapat digunakan sebagai *single entry point* untuk menghubungkan klien dengan berbagai layanan dalam arsitektur *microservices* [17]. Pada sisi data, *microservices* mendorong dekomposisi aplikasi ke dalam layanan kecil yang mandiri serta memungkinkan pengelolaan basis data sesuai kebutuhan tiap layanan [18]. Migrasi dari *database* monolitik ke arsitektur *microservices* juga perlu mempertimbangkan transformasi model data dan pendekatan penyimpanan yang sesuai [19]. Hal tersebut sejalan dengan pola *database per service* yang menempatkan data sebagai bagian dari tanggung jawab masing-masing layanan [20].

Salah satu penelitian yang relevan dengan domain tahfiz dilakukan oleh Al Hilmi et al. [21] melalui penerapan *microservice* pada aplikasi manajemen kegiatan Rumah Tahfiz Qur'an Ulil Albab Kabupaten Indramayu. Penelitian tersebut membagi aplikasi menjadi dua layanan, yaitu layanan data master dan layanan data akademik, dengan basis data yang terpisah. Evaluasi pada penelitian tersebut dilakukan menggunakan *blackbox testing* untuk memeriksa fungsi utama aplikasi, tetapi belum mencakup evaluasi performa dan stabilitas sistem. Sementara itu, hasil evaluasi performa *microservices* perlu dilihat berdasarkan lingkungan *deployment* dan skenario pengujian yang digunakan [13]. *Response time*, *throughput*, dan *error rate* juga digunakan sebagai metrik dalam *load testing* pada *microservices* [22].

Penelitian ini memperluas cakupan penelitian Al Hilmi et al. [21] pada aspek arsitektur, fitur, dan evaluasi. Dari sisi arsitektur, proses utama Sistem Informasi Tahfiz dipisahkan ke dalam tujuh layanan, menerapkan API Gateway sebagai *single entry point*, dan menggunakan pola *database per service*. Dari sisi fitur, sistem mencakup pengelolaan pengguna dan akademik, pencatatan hafalan, ujian periodik, aktivitas harian, evaluasi hafalan per juz, pemantauan progres, dan pelaporan bagi lima jenis pengguna. Dari sisi evaluasi, pengujian dilakukan menggunakan 78 kasus uji *blackbox*, empat skenario *load testing* pada beban 10 hingga 100 *virtual users* yang masing-masing dijalankan sebanyak tiga kali, serta satu kali *endurance testing* menggunakan 23 *virtual users* selama empat jam. Dengan demikian, kontribusi teknis penelitian ini terletak pada penerapan tujuh layanan berdasarkan fungsi utama pengelolaan tahfiz serta evaluasi implementasi dari sisi fungsi, karakteristik performa pada beban bertahap, dan stabilitas performa selama beban berkelanjutan.

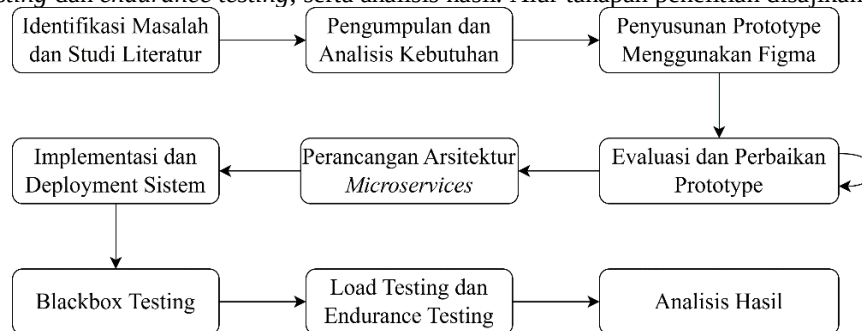
Berdasarkan uraian tersebut, penelitian ini bertujuan untuk mengimplementasikan dan mengevaluasi arsitektur *microservices* pada Sistem Informasi Tahfiz. Implementasi difokuskan pada pemisahan tanggung jawab layanan, pengaturan akses melalui API Gateway, dan pemisahan kepemilikan data, sedangkan evaluasi dilakukan melalui *blackbox testing*, *load testing*, dan *endurance testing*. Hasil penelitian diharapkan dapat memberikan gambaran penerapan arsitektur *microservices* yang modular dan terintegrasi pada sistem informasi tahfiz serta karakteristik kinerjanya pada lingkungan dan skenario pengujian yang digunakan.

2. METODOLOGI PENELITIAN

2.1 Tahapan Penelitian

Penelitian ini menggunakan pendekatan studi kasus pada sebuah institusi pendidikan tingkat menengah yang menyelenggarakan program tahfiz di Bandung. Tahapannya meliputi identifikasi masalah dan studi literatur,

pengumpulan dan analisis kebutuhan, penyusunan *prototype* menggunakan Figma, evaluasi dan perbaikan *prototype* secara iteratif, perancangan arsitektur *microservices*, implementasi dan *deployment* sistem, *blackbox testing*, *load testing* dan *endurance testing*, serta analisis hasil. Alur tahapan penelitian disajikan pada Gambar 1.



Gambar 1. Tahapan Penelitian

Identifikasi masalah dilakukan untuk memahami kondisi pengelolaan program tahfiz yang sedang berjalan, sedangkan studi literatur digunakan untuk menentukan dasar pengembangan sistem, perancangan *microservices*, dan metode pengujian. Pengumpulan kebutuhan dilakukan melalui satu sesi wawancara semi-terstruktur selama kurang lebih 1,5 jam dengan Wakil Kepala Sekolah Bidang Kurikulum sebagai informan utama. Informan dipilih karena memahami kebijakan, alur kegiatan, dan kebutuhan pengelolaan program tahfiz di sekolah. Instrumen yang digunakan berupa panduan pertanyaan wawancara dan rancangan fitur awal. Data wawancara dicatat dan dirangkum berdasarkan informasi mengenai aktor yang terlibat, alur kegiatan, data yang dikelola, kendala pencatatan, pemantauan progres, dan kebutuhan pelaporan.

Hasil wawancara dikelompokkan secara deskriptif ke dalam kebutuhan pengguna, alur proses, data yang dikelola, serta informasi keluaran untuk merumuskan kebutuhan awal sistem. Metode *prototyping* digunakan untuk mengonfirmasi kebutuhan dan alur sistem sebelum implementasi dilakukan [23], [24]. Kebutuhan awal tersebut diterjemahkan ke dalam *prototype* antarmuka menggunakan Figma, kemudian dievaluasi oleh empat pengguna yang terdiri atas satu Wakil Kepala Sekolah Bidang Kurikulum, satu Koordinator TTQ, dan dua Guru TTQ. Evaluasi dilakukan dengan meninjau kesesuaian fungsi, alur, dan informasi pada *prototype*. Masukan pengguna dibandingkan dengan kebutuhan awal dan digunakan untuk memperbaiki rancangan sebelum menjadi acuan implementasi. Evaluasi *prototype* digunakan sebagai konfirmasi kebutuhan sistem dan tidak diposisikan sebagai triangulasi sumber terhadap data wawancara. Karena wawancara hanya melibatkan satu informan dan penelitian dilakukan pada satu institusi, hasil analisis kebutuhan dibatasi pada konteks operasional institusi studi kasus.

Setelah kebutuhan dikonfirmasi, fungsi sistem dipetakan ke dalam tujuh *service* berdasarkan proses utama pengelolaan tahfiz. Hasil perancangan menjadi dasar implementasi dan *deployment* sistem serta penyusunan kasus uji *blackbox*. Sistem yang telah di-*deploy* kemudian diuji melalui *blackbox testing*, *load testing*, dan *endurance testing*. Data hasil pengujian selanjutnya dianalisis untuk menilai fungsi, performa, dan stabilitas sistem pada lingkungan serta skenario yang digunakan.

2.2 Lingkungan Implementasi dan Deployment

Frontend sistem dikembangkan menggunakan React, Vite, dan TypeScript 6.0.2, sedangkan *backend* dikembangkan menggunakan Express.js 5.2.1 dan TypeScript 6.0.2. Node.js 22.22.3 digunakan sebagai runtime backend, Prisma 5.22.0 sebagai Object-Relational Mapping, dan PostgreSQL 16 Alpine sebagai basis data. Pengujian non-fungsional menggunakan k6 2.0.0 yang dijalankan melalui Command Line Interface.

Sistem di-*deploy* pada Virtual Private Server dengan spesifikasi 2 vCPU, RAM 2 GB, penyimpanan 60 GB, dan sistem operasi Ubuntu 22.04.5 LTS. API Gateway dan tujuh *service backend* dijalankan dalam *container* terpisah menggunakan Docker Compose dengan satu *instance* untuk setiap komponen. PostgreSQL dijalankan dalam satu *container* dengan enam basis data logis yang dipisahkan berdasarkan kepemilikan data masing-masing *service*, sedangkan Reporting Service tidak memiliki basis data operasional. Setiap *service* memiliki konfigurasi *build*, dependensi, dan Dockerfile tersendiri sehingga dapat dibangun dan dijalankan secara terpisah, meskipun *deployment* pada penelitian ini dikelola bersama menggunakan Docker Compose. Konfigurasi *deployment service* disajikan pada Tabel 1.

Tabel 1. Konfigurasi *Deployment* Sistem

Komponen	Port Internal	Basis Data	Instance
API Gateway	3000	-	1
Auth Service	3001	auth_db	1
User Service	3002	user_db	1
Academic Service	3003	academic_db	1
Memorization Service	3004	memorization_db	1

Komponen	Port Internal	Basis Data	Instance
Juz Evaluation Service	3005	juz_evaluation_db	1
Habit Service	3006	habit_db	1
Reporting Service	3007	-	1

Frontend mengakses layanan *backend* melalui API Gateway menggunakan *base path* /api. *Load generator* dijalankan pada laptop dengan sistem operasi Windows 11 Pro, prosesor Intel Core i5-8365U, dan RAM 16 GB. Pengujian dilakukan terhadap sistem yang telah di-deploy pada VPS melalui API Gateway.

2.3 Rancangan Pengujian dan Analisis Data

Kasus uji *blackbox* dan skenario k6 disusun serta dijalankan oleh peneliti berdasarkan fungsi dan hak akses setiap jenis pengguna. Pengujian fungsional menggunakan *blackbox testing* untuk memverifikasi kesesuaian *input* dan *output* sistem tanpa memeriksa struktur internal kode [25], [26]. Sebanyak 78 kasus uji disusun berdasarkan fungsi utama pada modul autentikasi, siswa, Guru TTQ, Koordinator TTQ, admin, dan tim yayasan. Skenario pengujian mencakup kondisi positif dan negatif. Setiap kasus uji memuat input, hasil yang diharapkan, hasil aktual, dan status pengujian. Kasus uji dinyatakan berhasil apabila hasil aktual sesuai dengan hasil yang diharapkan.

Pengujian non-fungsional terdiri atas *load testing* dan *endurance testing* menggunakan k6. Pengujian dijalankan melalui API Gateway agar pola akses menyerupai komunikasi *frontend* menuju layanan *backend*. Akun dan data uji disiapkan untuk lima jenis pengguna, yaitu siswa, Guru TTQ, Koordinator TTQ, admin, dan tim yayasan. Distribusi pengguna pada skrip terdiri atas 30% siswa, 40% Guru TTQ, serta masing-masing 10% Koordinator TTQ, admin, dan tim yayasan.

Akun dan data uji disimpan dalam berkas JSON berdasarkan jenis pengguna dan dipilih sesuai indeks VU. Aktivitas setiap jenis pengguna dipilih secara acak berdasarkan proporsi yang ditetapkan. Skenario siswa terdiri atas 50% akses riwayat sidang, 30% pengelolaan target harian, dan 20% aktivitas mandiri. Skenario Guru TTQ terdiri atas 40% akses rekapitulasi, 40% pencatatan aktivitas harian, dan 20% penilaian ujian periodik. Skenario Koordinator TTQ terdiri atas 50% persetujuan sidang, 40% pengelolaan target bulanan, dan 10% ekspor laporan. Skenario admin terdiri atas 60% akses data, 20% pengelolaan pengguna, dan 20% pengelolaan data akademik, sedangkan skenario Tim Yayasan mencakup akses dashboard dan grafik pencapaian.

Setiap iterasi diawali dengan autentikasi dan dilanjutkan dengan aktivitas sesuai jenis pengguna. Berdasarkan skrip, setiap iterasi dapat menghasilkan sekitar 3–9 *request*, bergantung pada jenis pengguna, aktivitas yang dipilih, dan ketersediaan data. Jeda acak selama 1–5 detik diberikan sebelum aktivitas dimulai dan 1–3 detik di antara beberapa aktivitas untuk menyerupai waktu interaksi pengguna.

Load testing menggunakan *executor shared-iterations* dengan empat variasi beban sebagaimana ditunjukkan pada Tabel 2. Setiap skenario dijalankan sebanyak tiga kali atau tiga *run* pada lingkungan *deployment*, skrip, dan parameter beban yang sama. Setiap *run* memiliki batas durasi maksimum sepuluh menit. Aktivitas pengguna tetap dipilih secara acak sesuai proporsi skenario sehingga jumlah dan komposisi *request* dapat berbeda antar-*run*. Pengujian tidak menggunakan tahap *warm-up*, *ramp-up*, atau *ramp-down* secara terpisah.

Tabel 2. Skenario *Load Testing*

Skenario	Virtual Users	Jumlah Iterasi
LT-01	10	50
LT-02	23	123
LT-03	50	250
LT-04	100	500

Endurance testing dilakukan satu kali setelah *load testing* menggunakan *executor constant-vus*. Pengujian dijalankan menggunakan 23 VU selama empat jam. Beban 23 VU dipilih sebagai beban menengah untuk mengamati perubahan metrik aplikasi dan konsumsi sumber daya dalam durasi yang lebih panjang dibandingkan *load testing*. Pengujian dilakukan pada kondisi operasi normal dan tidak mencakup penghentian *container*, gangguan jaringan, kegagalan *service*, *fault injection*, *failover*, maupun pengukuran *recovery time*.

Metrik yang dianalisis meliputi *average response time*, *p(95)*, *p(99)*, *throughput*, *error rate*, dan *check success*. Nilai *p(95)* dan *p(99)* digunakan untuk mengamati distribusi waktu respons, *throughput* menunjukkan laju *request* yang diproses, sedangkan *error rate* menunjukkan persentase *request* yang gagal. Pemilihan *response time*, *throughput*, dan *error rate* mengacu pada metrik pengujian performa *microservices* yang digunakan pada penelitian sebelumnya [13], [22]. Penggunaan CPU dan memori pada VPS juga diamati selama *endurance testing* untuk melihat perubahan konsumsi sumber daya dalam penggunaan berkelanjutan [12].

Data *blackbox testing* dianalisis berdasarkan kesesuaian hasil aktual dengan hasil yang diharapkan. Data *load testing* dianalisis secara deskriptif-komparatif dengan membandingkan *average response time*, *p(95)*, *p(99)*, *throughput*, *error rate*, dan total *request* pada tiga *run* untuk setiap tingkat beban. Data *endurance testing* dianalisis berdasarkan perubahan *response time*, *throughput*, *error rate*, *check success*, CPU, dan memori selama empat jam.

Hasil *endurance testing* digunakan untuk menggambarkan stabilitas performa selama skenario pengujian dan tidak digunakan untuk menyimpulkan *resilience*, *fault tolerance*, atau kemampuan pemulihan sistem.

3. HASIL DAN PEMBAHASAN

Bagian ini menyajikan hasil penelitian sesuai dengan tahapan yang telah dijelaskan pada metode penelitian. Pembahasan diawali dengan hasil analisis kebutuhan dan perancangan sistem, kemudian dilanjutkan dengan implementasi arsitektur *microservices*, pengujian fungsional, serta evaluasi performa pada beban bertahap dan stabilitas performa selama beban berkelanjutan. Penyajian ini disusun untuk memperlihatkan keterkaitan antara kebutuhan sistem, dekomposisi layanan, mekanisme akses melalui API Gateway, pemisahan basis data, komunikasi antarlayanan, dan hasil pengujian yang diperoleh untuk menjawab tujuan penelitian.

3.1 Analisis Kebutuhan dan Perancangan Sistem

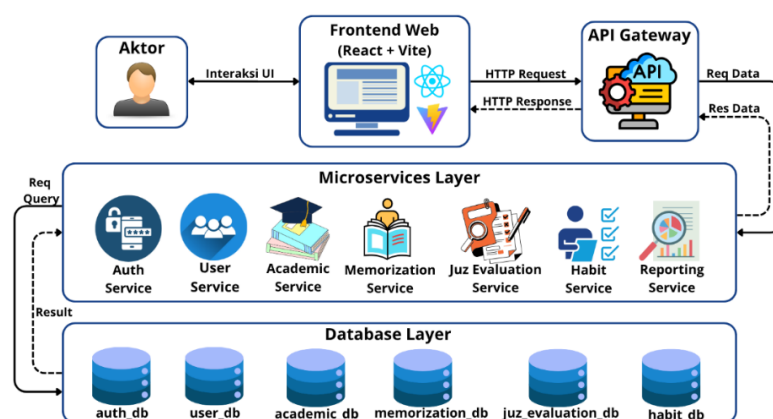
Hasil wawancara dan evaluasi *prototype* menunjukkan bahwa sistem melibatkan lima jenis pengguna, yaitu siswa, Guru TTQ, Koordinator TTQ, admin, dan tim yayasan. Evaluasi *prototype* mengonfirmasi kesesuaian fungsi, alur, dan informasi yang dirancang. Setiap pengguna memiliki kebutuhan yang berbeda dalam proses pencatatan, penilaian, pemantauan, dan pelaporan program tahfiz.

Guru TTQ membutuhkan sistem untuk mencatat setoran hafalan, mengelola penilaian ujian periodik, memantau aktivitas siswa, mengajukan sidang hafalan, serta mencatat hasil sidang sesuai kewenangannya. Guru TTQ juga membutuhkan akses terhadap progres, riwayat sidang, dan rekap setoran siswa dalam kelompok binaannya. Siswa membutuhkan akses untuk memantau capaian hafalan, target yang sedang berjalan, riwayat sidang, serta mencatat aktivitas mandiri seperti tilawah dan murojaah.

Koordinator TTQ membutuhkan fitur untuk memantau progres seluruh siswa, menerima atau menolak pengajuan sidang, menetapkan target hafalan bulanan, serta memperoleh laporan evaluasi program. Admin berperan dalam pengelolaan data pengguna, siswa, guru, kelas, kelompok tahfiz, dan tahun akademik. Sementara itu, tim yayasan membutuhkan informasi ringkas mengenai capaian program tahfiz sebagai bahan evaluasi pada tingkat manajerial.

Kebutuhan fungsional tersebut dikelompokkan ke dalam proses autentikasi, pengelolaan pengguna, pengelolaan akademik, pencatatan hafalan, evaluasi hafalan per juz, aktivitas harian, dan pelaporan. Pengelompokan ini menjadi dasar penentuan batas tanggung jawab *service* pada arsitektur *microservices*. *Prototype* antarmuka yang telah diperbaiki berdasarkan masukan pengguna kemudian digunakan sebagai acuan pengembangan *frontend* dan *backend*. Berdasarkan hasil tersebut, sistem dirancang menjadi tujuh *service* utama, yaitu Auth Service, User Service, Academic Service, Memorization Service, Juz Evaluation Service, Habit Service, dan Reporting Service. Hasil analisis kebutuhan tersebut merepresentasikan kondisi operasional institusi studi kasus dan tidak dimaksudkan untuk digeneralisasi secara langsung pada seluruh institusi penyelenggara program tahfiz.

3.2 Implementasi Arsitektur *Microservices*



Gambar 2. Visualisasi Arsitektur Sistem Informasi Tahfiz Berbasis *Microservices*

Gambar 2 menunjukkan arsitektur sistem yang terdiri atas lapisan aktor, antarmuka pengguna, API Gateway, *microservices*, dan basis data. Pengguna mengakses sistem melalui *frontend web* yang mengirimkan *HTTP request* ke API Gateway sebagai *single entry point*. API Gateway kemudian meneruskan *request* ke *service* yang sesuai berdasarkan *endpoint* yang diakses.

Lapisan *backend* terdiri atas tujuh *service* yang menjalankan logika bisnis sesuai tanggung jawabnya. Setiap *service* pemilik data mengakses basis data logisnya sendiri, sedangkan hasil pemrosesan dikembalikan melalui API Gateway hingga ditampilkan pada *frontend*. Dalam *deployment*, API Gateway dan setiap *service backend*

dijalankan sebagai proses terpisah pada *container* masing-masing dengan port internal yang berbeda. Pemisahan tersebut menunjukkan bahwa fungsi *backend* tidak dijalankan dalam satu proses aplikasi yang sama.

3.2.1 Routing API Gateway

Pada implementasi ini, API Gateway menerima *request* dari *frontend* melalui prefix */api*, kemudian meneruskannya ke *service backend* yang sesuai. Mekanisme tersebut membuat *frontend* tidak perlu mengetahui alamat dan port setiap *service* secara langsung. API Gateway juga menerapkan CORS dengan membatasi origin yang diizinkan pada alamat frontend dan mengaktifkan pengiriman kredensial. Setiap request diberikan *requestId* serta dicatat berdasarkan waktu, metode HTTP, *endpoint*, status respons, dan durasi pemrosesan.

Request beserta *header* autentikasi diteruskan oleh API Gateway kepada *service* tujuan. Validasi token dan pemeriksaan hak akses dilakukan pada *middleware service* terkait. API Gateway memberikan respons kesalahan untuk *endpoint* yang tidak dikenali dan kesalahan internal, tetapi belum dilengkapi *rate limiting* serta mekanisme penanganan gangguan otomatis terhadap kegagalan *service*. Pola *routing* API Gateway ditunjukkan pada Tabel 3.

Tabel 3. Routing API Gateway menuju Layanan *Backend*

Prefix Endpoint	Service Tujuan	Fungsi Utama
/api/auth/*	Auth Service	Pengelolaan akun, login, validasi token, status akun, dan role pengguna.
/api/users/*	User Service	Pengelolaan profil siswa dan Guru TTQ.
/api/academic/*	Academic Service	Pengelolaan tahun akademik, kelas, kelompok tahfiz, target bulanan, dan data referensi Al-Qur'an.
/api/memorization/*	Memorization Service	Pencatatan setoran hafalan, penilaian ujian periodik, dan aktivitas mandiri.
/api/juz-evaluation/*	Juz Evaluation Service	Pengelolaan sidang hafalan per juz, meliputi pengajuan, persetujuan, penjadwalan, penentuan guru penguji, penilaian, dan hasil sidang.
/api/habits/*	Habit Service	Pengelolaan target harian tilawah dan murojaah serta pemantauan <i>streak</i> pencapaian target.
/api/reports/*	Reporting Service	Penyajian dashboard, rekap, monitoring progres, dan laporan melalui agregasi data dari service lain.

Melalui mekanisme *routing* pada Tabel 3, *request* dari pengguna dapat diteruskan menuju layanan yang memiliki tanggung jawab terhadap proses yang diakses. Penggunaan API Gateway juga memungkinkan *frontend* berkomunikasi dengan *backend* melalui satu jalur akses tanpa mengakses setiap *service* secara langsung, sesuai fungsi API Gateway sebagai *single entry point* [17].

3.2.2 Dekomposisi Layanan dan Database per Service

Pada lapisan *backend*, sistem tidak dikembangkan sebagai satu aplikasi monolitik, tetapi dipisahkan ke dalam tujuh *service* utama berdasarkan proses utama pengelolaan tahfiz. Dekomposisi aplikasi ke dalam *microservices* banyak dilakukan dengan mempertimbangkan pendekatan *domain-driven design* dan *business capability* [16]. Dalam penelitian ini, pendekatan tersebut diterapkan dengan memisahkan layanan berdasarkan proses utama pengelolaan tahfiz, yaitu autentikasi, pengelolaan pengguna, akademik, pencatatan hafalan, evaluasi hafalan per juz, aktivitas harian, dan pelaporan. Setiap *service* memiliki fokus fungsionalitas dan basis data masing-masing sebagaimana ditunjukkan pada Tabel 4.

Tabel 4. Dekomposisi Layanan, Basis Data, dan Fokus Fungsional

Nama Layanan	Basis Data	Fokus Fungsionalitas Utama
Auth Service	auth_db	Mengelola akun, login, validasi token, status akun, dan role pengguna.
User Service	user_db	Mengelola data siswa dan guru, termasuk nomor induk siswa dan nomor induk guru.
Academic Service	academic_db	Mengelola tahun akademik, kelas, kelompok tahfiz, record akademik siswa, target bulanan, dan data referensi Al-Qur'an.
Memorization Service	memorization_db	Mengelola setoran hafalan, ujian periodik, dan aktivitas mandiri siswa.
Juz Evaluation Service	juz_evaluation_db	Mengelola pengajuan, persetujuan, jadwal, guru penguji, nilai, dan hasil sidang hafalan per juz.
Habit Service	habit_db	Mengelola target harian tilawah dan murojaah serta pemantauan <i>streak</i> pencapaian target siswa.

Nama Layanan	Basis Data	Fokus Fungsionalitas Utama
Reporting Service	-	Mengagregasi data dari layanan lain untuk <i>dashboard</i> , monitoring progres, rekap, laporan, dan ekspor data.

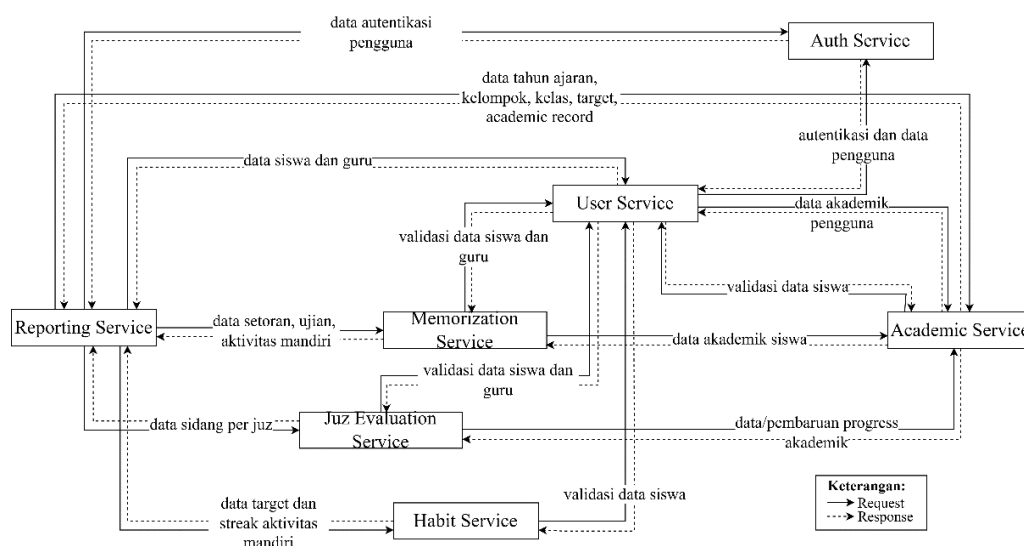
Tabel 4 menunjukkan bahwa setiap *service* menangani proses dan data sesuai ruang lingkupnya. Pemisahan ini mencegah fungsi autentikasi, pengguna, akademik, hafalan, evaluasi hafalan per juz, habit, dan pelaporan dikelola dalam satu *service* yang sama. Pola *database per service* diterapkan melalui enam basis data logis PostgreSQL yang dipisahkan berdasarkan kepemilikan data, sedangkan Reporting Service tidak memiliki basis data operasional. Dengan pola tersebut, setiap *service* pemilik data mengelola basis datanya sendiri dan tidak mengakses basis data *service* lain secara langsung [18], [20].

Keterkaitan data lintas *service* disimpan dalam bentuk ID dan divalidasi melalui komunikasi REST API secara sinkron kepada *service* pemilik data. Setiap *service* hanya melakukan perubahan pada basis datanya sendiri sehingga transaksi dan *rollback* terbatas pada masing-masing *service*. Transaksi terdistribusi dan mekanisme kompensasi otomatis belum diterapkan sehingga kegagalan pada proses lintas *service* belum ditangani melalui pemulihan otomatis. Perubahan skema basis data dikelola secara terpisah pada setiap *service* menggunakan Prisma Migrate. Data disimpan melalui *persistent volume*, tetapi mekanisme *backup* terjadwal dan pemulihan data secara otomatis belum diterapkan.

3.2.3 Komunikasi Antarlayanan dan Reporting Service

Selain komunikasi antara *frontend* dan layanan *backend* melalui API Gateway, beberapa proses pada sistem membutuhkan komunikasi antarlayanan ketika suatu *service* memerlukan data yang dikelola oleh *service* lain. Komunikasi dilakukan secara sinkron melalui pola *request-response* pada REST API tanpa mengakses basis data *service* lain secara langsung. Oleh karena itu, keterlambatan atau kegagalan *service* tujuan dapat memengaruhi proses yang bergantung padanya. Implementasi saat ini masih menggunakan log aplikasi untuk pemantauan dan belum dilengkapi mekanisme penanganan gangguan dan *observability* terpusat, seperti pengaturan *timeout* khusus, *retry* otomatis, *circuit breaker*, *idempotency key*, dan *distributed tracing*.

Gambar 3 menunjukkan komunikasi antarlayanan pada proses utama Sistem Informasi Tahfiz. Garis penuh menunjukkan *request* yang dikirimkan oleh suatu layanan kepada layanan lain, sedangkan garis putus-putus menunjukkan *response* yang dikembalikan setelah *request* diproses. Diagram tersebut menggambarkan pertukaran data antarlayanan tanpa akses langsung ke basis data milik layanan lain.



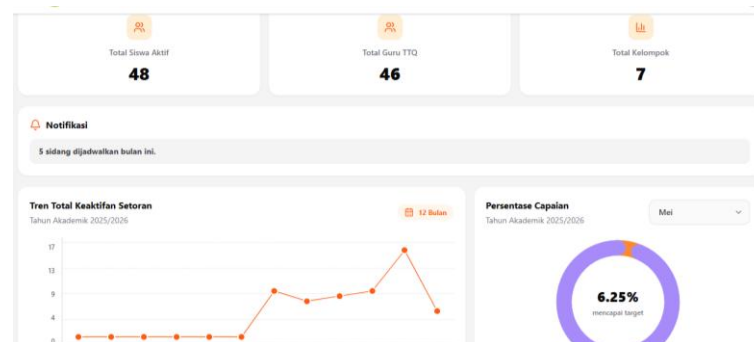
Gambar 3. Diagram Komunikasi Antarlayanan pada Sistem Informasi Tahfiz

Memorization Service berkomunikasi dengan User Service untuk memvalidasi siswa dan Guru TTQ sebelum memproses setoran hafalan, ujian periodik, dan aktivitas mandiri. Habit Service juga menggunakan User Service untuk memvalidasi siswa pada pengelolaan target harian. Pada proses sidang hafalan per juz, Juz Evaluation Service memperoleh data siswa dan guru dari User Service serta data akademik dari Academic Service. Ketika siswa dinyatakan lulus, pembaruan capaian hafalan dilakukan melalui Academic Service sebagai pemilik data akademik.

Reporting Service tidak memiliki basis data operasional dan mengagregasi data melalui *endpoint service* terkait. Data pengguna diperoleh dari Auth Service dan User Service, data akademik dari Academic Service, data setoran dan ujian periodik dari Memorization Service, data sidang dari Juz Evaluation Service, serta data target harian dan *streak* dari Habit Service. Data tersebut digunakan untuk membentuk *dashboard*, monitoring progres, rekap, laporan, dan ekspor data.

Data agregat diperoleh secara langsung dari *service* sumber setiap kali *request* diproses. Pada beberapa pemanggilan, kegagalan memperoleh salah satu data ditangani dengan mengembalikan nilai *null* agar data lain tetap dapat diproses. Namun, Reporting Service belum menggunakan *retry*, *cache*, maupun *fallback* berupa data sebelumnya sehingga laporan dapat menjadi tidak lengkap apabila *service* sumber tidak tersedia. Data yang ditampilkan juga mencerminkan data yang berhasil diperoleh pada saat *request* dan belum membentuk satu *snapshot* transaksi yang seragam.

Sebagai contoh hasil implementasi pada sisi antarmuka, Gambar 4 menunjukkan dashboard koordinator TTQ yang menyajikan ringkasan siswa, Guru TTQ, kelompok tahfiz, jadwal sidang, tren setoran, dan capaian target. Gambar 5 menampilkan halaman monitoring progres yang menyediakan pencarian serta filter kelompok, juz, dan jenis aktivitas. Data pada kedua antarmuka tersebut diperoleh melalui Reporting Service. Implementasi ini menunjukkan bahwa data operasional tetap dikelola oleh *service* pemilik data, sedangkan informasi lintas proses disajikan secara terintegrasi melalui mekanisme agregasi.



Gambar 4. Antarmuka Dashboard Koordinator TTQ

Monitoring Progress
Pantau perkembangan setoran dan evaluasi siswa.

Tabel Monitoring Grafik Pencapaian

Cari berdasarkan nama

Semua Kelompok Semua Juz Setoran Harian

No	Nama Lengkap	Juz Ongoing	Tahfiz Terakhir	Tahfuz Terakhir	Kelompok	Pembina
1	Laila Anggraini	Juz 28	Insaniyah hal 10-10	Ziyadah Al-Tin 1-8	Tahfiz X-1	Siti Rahma
2	Jihan Nuraini	Juz 29	As-Saffat 46-46	Ziyadah Nuh 1-15	Tahfiz X-1	Siti Rahma
3	Pritha Andini	Juz 28	As-Saffat 48-49	Munjaah Al-Muk 1-30	Tahfiz X-1	Siti Rahma
4	Tania Putri	Juz 28	As-Saffat 49-50	Ziyadah Al-Mujadalah 5-6	Tahfiz X-1	Siti Rahma
5	Nadia Aulia	Juz 27	Ali Imran 1-29	Ziyadah Al-Fatihah 1-18	Tahfiz X-1	Siti Rahma
6	Safira Fadilah	Juz 27	Ar-Rahman 1-20	Ziyadah Al-Hadid 1-5	Tahfiz X-1	Siti Rahma

Gambar 5. Antarmuka Monitoring Progres

3.3 Pengujian Fungsional Sistem

Pengujian fungsional dilakukan menggunakan *blackbox testing* untuk memeriksa kesesuaian *input* dan *output* sistem tanpa melihat struktur internal kode [25]. Sebanyak 78 kasus uji disusun berdasarkan fungsi utama pada modul autentikasi, siswa, Guru TTQ, Koordinator TTQ, admin, dan tim yayasan. Skenario mencakup kondisi positif dan negatif, termasuk penanganan input yang tidak valid. Hasil pengujian disajikan pada Tabel 5.

Tabel 5. Rekapitulasi Hasil Pengujian Fungsional Sistem

Kelompok Modul	Jumlah Kasus Uji	Berhasil
Autentikasi	4	4
Modul Siswa	15	15
Modul Guru	14	14
Modul Koordinator	13	13
Modul Admin	30	30
Modul Tim Yayasan	2	2
Total Akumulasi	78	78

Berdasarkan Tabel 5, seluruh 78 kasus uji menghasilkan keluaran sesuai dengan hasil yang diharapkan atau mencapai tingkat keberhasilan 100%. Pengujian mencakup autentikasi, pencatatan setoran, ujian periodik, aktivitas mandiri, pengelolaan sidang, pemantauan progres, pengelolaan data induk, serta penyajian laporan. Untuk menunjukkan kesesuaian *input* dan *output* secara lebih rinci, Tabel 6 menyajikan sampel kasus uji yang mewakili fungsi utama pengguna serta skenario positif dan negatif.

Tabel 6. Sampel Kasus Uji Fungsional

Modul	Fungsi dan Skenario	Jenis	Hasil yang Diharapkan	Hasil Aktual	Status
Autentikasi	Login menggunakan email dan kata sandi yang sesuai	Positif	Pengguna diarahkan ke dashboard sesuai peran	Pengguna berhasil masuk dan diarahkan ke dashboard sesuai peran	Lulus
Autentikasi	Login menggunakan email yang tidak terdaftar	Negatif	Autentikasi ditolak dan pesan kesalahan ditampilkan	Sistem menampilkan pesan “Email atau password salah”	Lulus
Siswa	Menambahkan aktivitas tilawah mandiri	Positif	Aktivitas tilawah siswa disimpan	Aktivitas berhasil disimpan	Lulus
Guru TTQ	Memasukkan setoran harian sesuai jenis aktivitas dan cakupan bacaan	Positif	Data setoran disimpan sesuai input	Data setoran berhasil disimpan	Lulus
Koordinator TTQ	Mengekspor laporan berdasarkan tahun akademik dan semester	Positif	Laporan dihasilkan dalam format Microsoft Excel	Laporan berhasil dihasilkan dan diunduh	Lulus
Admin	Menambahkan pengguna dengan data yang valid	Positif	Data pengguna baru disimpan	Data pengguna berhasil disimpan	Lulus
Admin	Menambahkan pengguna dengan email yang telah terdaftar	Negatif	Penyimpanan ditolak dan pesan kesalahan ditampilkan	Sistem menampilkan pesan “Email sudah terdaftar”	Lulus
Tim Yayasan	Membuka grafik pencapaian berdasarkan filter	Positif	Grafik ditampilkan sesuai filter	Grafik berhasil ditampilkan sesuai filter	Lulus

Sampel pada Tabel 6 menunjukkan bahwa sistem menghasilkan keluaran yang sesuai pada skenario positif serta menolak *input* yang tidak valid pada skenario negatif. Secara keseluruhan, hasil pengujian menunjukkan bahwa alur fungsi dari antarmuka pengguna melalui API Gateway dan *service backend* dapat dijalankan sesuai skenario yang diuji. Sistem selanjutnya dievaluasi melalui pengujian non-fungsional untuk mengamati performa dan stabilitas implementasi *microservices*.

3.4 Evaluasi Performa dan Stabilitas pada Beban Berkelanjutan

Evaluasi performa dilakukan melalui *load testing* dan *endurance testing* berdasarkan skenario yang telah dijelaskan pada Bagian 2.3. Pengujian dijalankan melalui API Gateway sehingga hasil yang diperoleh mencakup proses perutean *request*, pemrosesan *service backend*, komunikasi antarlayanan, dan akses basis data. Evaluasi dilakukan menggunakan metrik waktu respons, *throughput*, *error rate*, serta penggunaan CPU dan memori untuk menggambarkan kinerja sistem pada lingkungan *deployment* dan skenario pengujian yang digunakan [13], [22].

Load testing dilakukan pada beban 10, 23, 50, dan 100 VU. Setiap skenario dijalankan dalam tiga *run* menggunakan lingkungan *deployment*, skrip, jumlah VU, dan jumlah iterasi yang sama. Tabel 7 menyajikan total *request* dan metrik performa pada setiap *run* agar variasi hasil antar-*run* dapat diamati secara langsung.

Tabel 7. Rekapitulasi Metrik Pengujian Beban (*Load Testing*)

Skenario	Jumlah VU	Run	Total Request	Average Response Time (ms)	p(95) Response Time (ms)	p(99) Response Time (ms)	Throughput (req/s)	Error Rate
LT-01	10	1	215	106,40	292,14	554,35	5,62	0,00%
		2	215	98,00	249,76	475,51	5,19	0,00%
		3	216	108,87	385,26	511,09	5,36	0,00%
LT-02	23	1	536	113,35	249,44	896,02	11,09	0,00%
		2	526	210,31	906,17	2.204,45	10,15	0,00%
		3	518	129,39	357,68	1.109,45	10,51	0,00%
LT-03	50	1	1.095	152,76	365,75	1.922,19	25,06	0,00%
		2	1.100	156,27	433,34	1.966,76	23,81	0,00%
		3	1.094	161,09	456,63	1.988,04	24,49	0,00%
LT-04	100	1	2.163	554,79	2.462,95	5.419,26	38,54	0,05%
		2	2.153	522,13	2.385,20	4.692,08	41,18	0,00%
		3	2.189	606,93	2.918,52	5.100,43	39,20	0,05%

Total *request* pada tingkat beban yang sama sedikit berbeda karena aktivitas pengguna dipilih secara acak dan setiap aktivitas dapat menghasilkan jumlah *request* yang berbeda. Hasil ketiga *run* menunjukkan bahwa *throughput* meningkat seiring penambahan beban, dari 5,19–5,62 *req/s* pada 10 VU menjadi 10,15–11,09 *req/s* pada 23 VU, 23,81–25,06 *req/s* pada 50 VU, dan 38,54–41,18 *req/s* pada 100 VU.

Peningkatan waktu respons paling terlihat pada 100 VU. Pada skenario tersebut, *average response time* berada pada rentang 522,13–606,93 ms, *p*(95) sebesar 2.385,20–2.918,52 ms, dan *p*(99) sebesar 4.692,08–5.419,26 ms. *Error rate* tercatat 0,00% pada seluruh *run* dengan beban 10–50 VU, sedangkan pada 100 VU berada pada rentang 0,00%–0,05%. Nilai *p*(99) yang lebih tinggi daripada *average response time* dan *p*(95) menunjukkan bahwa sebagian kecil *request* mengalami waktu pemrosesan yang lebih panjang pada beban tertinggi.

Variasi antar-*run* paling terlihat pada skenario 23 VU, terutama pada *p*(95) dan *p*(99). Nilai *p*(95) berada pada rentang 249,44–906,17 ms, sedangkan *p*(99) berada pada rentang 896,02–2.204,45 ms. Variasi tersebut dapat dipengaruhi oleh pemilihan aktivitas dan *endpoint* secara acak. Namun, komposisi aktivitas aktual dan latensi setiap *endpoint* tidak direkam secara terpisah sehingga penyebab variasi tersebut belum dapat ditentukan secara khusus.

Peningkatan latensi pada 100 VU menunjukkan penurunan responsivitas pada tingkat sistem dalam konfigurasi yang diuji, tetapi belum dapat digunakan untuk menentukan komponen tertentu sebagai *bottleneck*. Pengujian belum merekam penggunaan CPU dan memori setiap *container*, latensi per *endpoint*, durasi komunikasi antarlayanan, jumlah koneksi basis data, dan waktu eksekusi *query*. Oleh karena itu, analisis *bottleneck* dibatasi pada perubahan *response time*, *throughput*, dan *error rate* pada tingkat sistem secara keseluruhan. Hasil pengujian tidak mengidentifikasi API Gateway, *service*, komunikasi antarlayanan, basis data, maupun sumber daya VPS tertentu sebagai penyebab tunggal peningkatan latensi.

Tahap berikutnya adalah *endurance testing* yang dilakukan satu kali menggunakan 23 VU selama empat jam untuk mengamati perubahan performa dan konsumsi sumber daya pada beban berkelanjutan. Pengujian dilakukan pada kondisi operasi normal dan tidak mencakup penghentian *service*, gangguan jaringan, *fault injection*, *failover*, maupun pengukuran *recovery time*. Hasil *endurance testing* disajikan pada Tabel 8.

Tabel 8. Hasil *Endurance Testing*

Durasi Pengujian	Jumlah VU	Iterasi	Total Request	Average Response Time (ms)	<i>p</i> (95) Response Time (ms)	<i>p</i> (99) Response Time (ms)	Throughput (req/s)	Error Rate	Check Success
4 jam	23	46.967	203.884	257,67	936,31	2.460,27	14,15	0,01%	99,99%

Berdasarkan Tabel 8, sistem mampu memproses 46.967 iterasi selama 4 jam dengan *average response time* sebesar 257,67 ms. Nilai *p*(95) sebesar 936,31 ms dan *p*(99) sebesar 2.460,27 ms menunjukkan bahwa 95% *request* selesai dalam waktu hingga 936,31 ms, sedangkan 99% *request* selesai dalam waktu hingga 2.460,27 ms pada skenario pengujian ini. *Throughput* tercatat sebesar 14,15 *req/s* dengan *error rate* 0,01% dan *check success* 99,99%. Hasil tersebut menunjukkan bahwa metrik aplikasi relatif terjaga selama durasi pengujian 4 jam pada beban dan lingkungan yang digunakan.

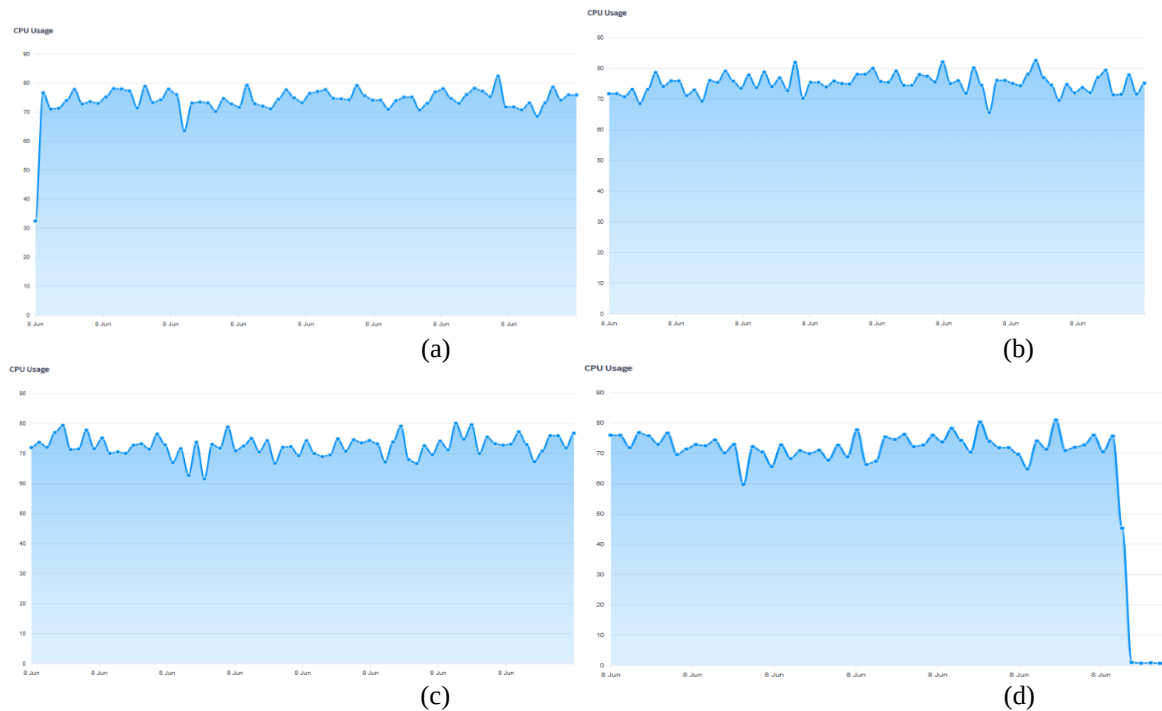
Penggunaan CPU dan memori VPS dipantau melalui *dashboard* monitoring bawaan penyedia VPS selama *endurance testing*. CPU ditampilkan dalam satuan persen, sedangkan memori ditampilkan dalam *megabyte*. *Dashboard* merekam metrik secara otomatis, tetapi interval sampling internal tidak ditampilkan secara eksplisit. Hasil pemantauan dibagi menjadi empat jendela grafik berurutan yang masing-masing berdurasi satu jam. Dengan demikian, panel (a) sampai (d) pada Gambar 6 dan Gambar 7 secara keseluruhan merepresentasikan empat jam *endurance testing*.

Hasil pemantauan CPU pada Gambar 6 menunjukkan bahwa penggunaan CPU selama empat jam umumnya berfluktuasi pada kisaran 60%–82%. Fluktuasi tersebut tidak membentuk pola peningkatan yang terus-menerus dari jam pertama hingga jam keempat. Pada bagian akhir jam keempat, penggunaan CPU menurun setelah proses k6 selesai dan beban *request* dihentikan. Pola tersebut menunjukkan bahwa penggunaan CPU VPS berada pada rentang yang relatif sama selama pengujian. Namun, pemantauan hanya dilakukan pada tingkat VPS sehingga penggunaan CPU masing-masing *container* dan komponen sistem tidak dapat diketahui secara terpisah.

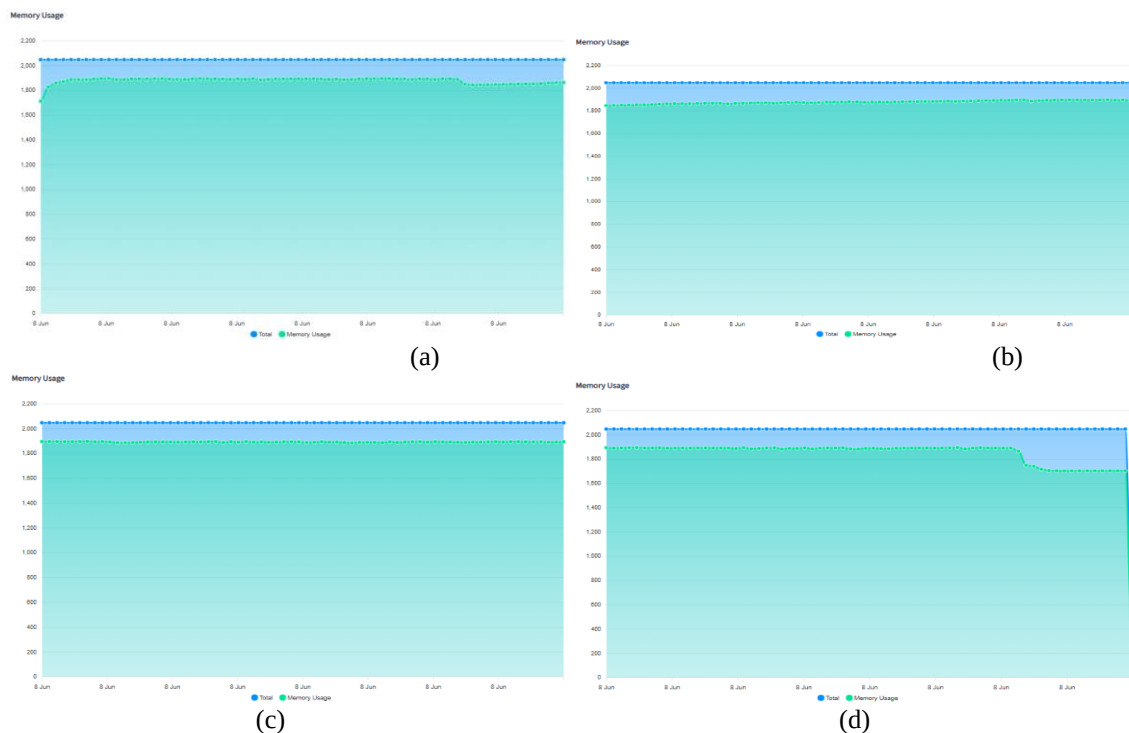
Selain CPU, penggunaan memori pada Gambar 7 juga diamati untuk melihat kestabilan konsumsi sumber daya selama pengujian berlangsung. Pada grafik memori, penggunaan memori meningkat dari sekitar 1.700 MB pada awal jam pertama menjadi sekitar 1.800–1.900 MB, kemudian berada pada rentang yang relatif sama hingga jam keempat. Setelah pengujian selesai, penggunaan memori menurun mendekati 1.700 MB. Titik bernilai nol pada akhir grafik tidak menunjukkan penggunaan memori aktual karena garis kapasitas total memori juga turun menjadi nol, tetapi menunjukkan berakhirnya data pada jendela monitoring.

Meskipun tidak terlihat peningkatan memori yang terus-menerus selama empat jam, penggunaan sebesar 1.800–1.900 MB setara dengan sekitar 88%–93% dari kapasitas RAM 2 GB. Kondisi tersebut menunjukkan bahwa ruang memori yang tersedia relatif terbatas dan dapat menjadi risiko apabila beban pengguna meningkat, terdapat proses tambahan, atau sistem digunakan dalam durasi yang lebih panjang. Namun, hasil ini belum membuktikan bahwa memori merupakan sumber *bottleneck* karena penggunaan memori setiap *container*, basis data, dan

komponen aplikasi tidak diukur secara terpisah. Penambahan kapasitas RAM atau optimasi konsumsi sumber daya dapat dipertimbangkan untuk skenario beban yang lebih tinggi.



Gambar 6. Penggunaan CPU (%) Selama Empat Jam *Endurance Testing*: (a) Jam Ke-1, (b) Jam Ke-2, (c) Jam Ke-3, dan (d) Jam Ke-4



Gambar 7. Penggunaan Memori (MB) Selama Empat Jam *Endurance Testing*: (a) Jam Ke-1, (b) Jam Ke-2, (c) Jam Ke-3, dan (d) Jam Ke-4

Secara keseluruhan, *endurance testing* menunjukkan bahwa *response time*, *throughput*, dan tingkat keberhasilan *request* relatif terjaga pada 23 VU selama empat jam. Hasil tersebut terbatas pada konfigurasi VPS, beban, durasi, dan kondisi operasi normal yang digunakan. Penggunaan memori yang mendekati kapasitas RAM serta tidak tersedianya metrik per *service* menjadi keterbatasan dalam menilai stabilitas jangka panjang dan menentukan sumber *bottleneck*. Karena pengujian tidak mencakup gangguan terkontrol, *failover*, atau pengukuran

recovery time, hasilnya tidak digunakan sebagai bukti *resilience*, *fault tolerance*, atau kemampuan pemulihan sistem.

3.5 Pembahasan Implementasi dan Evaluasi Arsitektur *Microservices*

Pembahasan pada bagian sebelumnya memperlihatkan bahwa pemisahan proses utama pada Sistem Informasi Tahfiz menghasilkan tujuh *service* dengan batas tanggung jawab yang berbeda. Perbandingan cakupan implementasi dan evaluasi penelitian dengan penelitian terdahulu disajikan pada Tabel 9.

Tabel 9. Perbandingan Cakupan Implementasi dan Evaluasi Penelitian

Aspek	Penelitian Terdahulu	Penelitian Ini	Kebaruan Penelitian
Implementasi <i>microservices</i>	Al Hilmi et al. [21] menerapkan 2 <i>service</i> untuk 2 jenis pengguna dengan basis data terpisah	Sistem menerapkan 7 <i>service</i> untuk 5 jenis pengguna, API Gateway, dan 6 basis data operasional	Memperluas dekomposisi <i>service</i> dan cakupan proses pengelolaan tahfiz
Evaluasi fungsional	Al Hilmi et al. [21] menggunakan <i>blackbox testing</i> tanpa melaporkan jumlah kasus uji	Sebanyak 78 dari 78 kasus uji berhasil	Menyajikan evaluasi fungsional yang lebih terukur
Evaluasi performa	Al Fansha et al. [22] melakukan <i>load testing</i> menggunakan <i>response time</i> , <i>throughput</i> , dan <i>error rate</i> tanpa <i>endurance testing</i>	Empat skenario <i>load testing</i> pada 10 hingga 100 VU masing-masing dijalankan dalam tiga <i>run</i> , sedangkan <i>endurance testing</i> dijalankan satu kali menggunakan 23 VU selama empat jam	Menambahkan evaluasi performa bertahap dan pengamatan stabilitas performa selama beban berkelanjutan
Metrik performa	Al Fansha et al. [22] menggunakan <i>response time</i> , <i>throughput</i> , dan <i>error rate</i>	Pengujian mencakup <i>average response time</i> , <i>p(95)</i> , <i>p(99)</i> , <i>throughput</i> , <i>error rate</i> , <i>check success</i> , serta CPU dan memori pada <i>endurance testing</i>	Memperluas evaluasi hingga distribusi waktu respons dan penggunaan sumber daya

Perbandingan pada Tabel 9 menunjukkan bahwa penelitian ini memperluas penerapan *microservices* pada domain tahfiz dari dua menjadi tujuh *service* dan dari dua menjadi lima jenis pengguna. Implementasi didukung oleh API Gateway, pola *database per service*, dan Reporting Service sebagai layanan agregasi. Perbandingan tersebut menunjukkan perbedaan cakupan implementasi dan evaluasi, bukan perbandingan performa antarsistem secara langsung karena objek, lingkungan, dan skenario pengujiannya berbeda.

Hasil pengujian menunjukkan bahwa pemisahan *service* dan komunikasi antarlayanan tetap mendukung fungsi utama sistem. Tiga *run load testing* menunjukkan peningkatan *throughput* hingga 38,54–41,18 *req/s* pada 100 VU, dengan *p(95)* sebesar 2.385,20–2.918,52 ms dan *error rate* 0,00%–0,05%. *Endurance testing* menunjukkan metrik aplikasi yang relatif terjaga selama empat jam, sedangkan penggunaan memori sebesar 88%–93% kapasitas RAM menunjukkan ruang sumber daya yang terbatas. Kebaruan penelitian terletak pada penerapan tujuh *microservices* serta evaluasi fungsi dan performa melalui pengujian beban berulang. Hasil penelitian masih terbatas pada satu VPS, aktivitas acak, dan belum tersedianya pengukuran performa pada tingkat komponen.

4. KESIMPULAN

Penelitian ini berhasil mengimplementasikan arsitektur *microservices* pada Sistem Informasi Tahfiz melalui tujuh *service* untuk lima jenis pengguna, API Gateway sebagai *single entry point*, dan pola *database per service*. Seluruh 78 kasus uji *blackbox* berhasil dengan tingkat keberhasilan 100%. Setiap skenario *load testing* pada beban 10, 23, 50, dan 100 VU dijalankan sebanyak tiga kali. Pada beban 10 hingga 50 VU, seluruh *request* pada ketiga *run* diproses tanpa kegagalan. Pada 100 VU, *p(95)* berada pada rentang 2.385,20–2.918,52 ms, *throughput* sebesar 38,54–41,18 *req/s*, dan *error rate* sebesar 0,00%–0,05%. *Endurance testing* menggunakan 23 VU selama empat jam menghasilkan *p(95)* 936,31 ms, *error rate* 0,01%, dan *check success* 99,99%. Penggunaan CPU tidak menunjukkan peningkatan berkelanjutan, sedangkan penggunaan memori mencapai 1.800–1.900 MB atau 88%–93% dari kapasitas RAM. Hasil tersebut menunjukkan bahwa arsitektur *microservices* dapat mendukung pengelolaan tahfiz secara modular dan terintegrasi pada lingkungan yang diuji. Evaluasi masih terbatas pada satu VPS dan satu kali *endurance testing* selama empat jam. Pengujian belum mencakup penggunaan sumber daya setiap *container*, latensi setiap *endpoint*, komunikasi antarlayanan, koneksi basis data, dan waktu eksekusi *query*, sehingga sumber *bottleneck* belum dapat ditentukan secara khusus. *Endurance testing* juga dilakukan pada kondisi operasi normal tanpa pengujian gangguan, *failover*, dan pemulihan sehingga hasilnya tidak digunakan untuk menyimpulkan *resilience* sistem. Penelitian selanjutnya dapat memperluas variasi beban, jumlah data, durasi, dan lingkungan *deployment*, serta melakukan pemantauan setiap *service*, pengulangan *endurance testing*, dan pengujian gangguan terkontrol.

REFERENCES

- [1] R. Ridwan and C. Mutia, “Pengembangan aplikasi web dan mobile pada monitoring dan evaluasi program tahfidz Quran,” *CIRCUIT J. Ilm. Pendidik. Tek. Elektro*, vol. 6, no. 1, pp. 47–53, 2022, doi: 10.22373/crc.v6i1.11700.
- [2] M. Masrur, M. Muhtarom, S. Salamun, N. Kusuma, A. Wahyudi, and S. H. Khotimah, “Evaluation of digital-based Al-Qur’an tahfiz learning in boarding school,” *TARBIYA J. Educ. Muslim Soc.*, vol. 12, no. 1, pp. 1–16, Apr. 2025, doi: 10.15408/tjems.v12i1.44758.
- [3] U. Akhror and A. Setiawan, “Aplikasi media digital monitoring tahfidz untuk meningkatkan kualitas hafalan santri SMA Al-I’tishom,” *JIKA (Jurnal Informatika)*, vol. 9, no. 3, pp. 302–311, 2025, doi: 10.31000/jika.v9i3.14309.
- [4] S. M. Marier and P. F. Dewi, “Tahfidz Quran monitoring system in Islamic boarding schools,” *Telemat. J. Telemat. dan Teknol. Inf.*, vol. 18, no. 1, pp. 1–11, 2021, doi: 10.31315/telematika.v18i1.3931.
- [5] R. J. Lontaan and A. R. Sinadia, “Design and development of a web-based school information system,” *CogITO Smart J.*, vol. 10, no. 2, pp. 593–606, 2024, doi: 10.31154/cogito.v10i2.858.593-606.
- [6] C. A. Dinova and I. C. Utomo, “Pengembangan arsitektur microservice pada learning management system e-learning menggunakan metode web service,” *J. Ilmu Komput. dan Inform.*, vol. 3, no. 2, pp. 125–141, 2024, doi: 10.54082/jiki.102.
- [7] S. Baškarada, V. Nguyen, and A. Koronios, “Architecting microservices: Practical opportunities and challenges,” *J. Comput. Inf. Syst.*, vol. 60, no. 5, pp. 428–436, 2020, doi: 10.1080/08874417.2018.1520056.
- [8] A. Garindra, T. Wati, and I. W. W. Pradnyana, “Perancangan arsitektur microservices untuk resiliensi sistem informasi perpustakaan pusat (Studi kasus UPN Veteran Jakarta),” *Format J. Ilm. Tek. Inform.*, vol. 9, no. 2, pp. 99–109, 2021, doi: 10.22441/format.2020.v9i2.001.
- [9] N. K. Hadi, S. H. A. Hamad, S. J. Abbas, G. F. Ali, and M. M. M. Maadi, “Enhancing software reusability in higher education applications through microservices architecture,” *J. Wirel. Mob. Networks, Ubiquitous Comput. Dependable Appl.*, vol. 16, no. 1, pp. 390–409, 2025, doi: 10.58346/JOWUA.2025.I1.024.
- [10] F. Auer, V. Lenarduzzi, M. Felderer, and D. Taibi, “From monolithic systems to microservices: An assessment framework,” *Inf. Softw. Technol.*, vol. 137, Art. no. 106600, 2021, doi: 10.1016/j.infsof.2021.106600.
- [11] B. Wijayanto, D. Iskandar, and S. P. Rahayu, “From monoliths to microservices: Designing a scalable super app architecture for academic services at Universitas Jenderal Soedirman,” *J. Tek. Inform.*, vol. 6, no. 4, pp. 2380–2391, 2025, doi: 10.52436/1.jutif.2025.6.4.5237.
- [12] D. P. Dirgantara, D. S. Kusumo, and R. G. Utomo, “Docker-based monolithic and microservices architecture performance comparison,” *J. Tek. Inform.*, vol. 5, no. 2, pp. 357–365, 2024, doi: 10.52436/1.jutif.2024.5.2.1338.
- [13] G. Blinowski, A. Ojdowska, and A. Przybyłek, “Monolithic vs. microservice architecture: A performance and scalability evaluation,” *IEEE Access*, vol. 10, pp. 20357–20374, 2022, doi: 10.1109/ACCESS.2022.3152803.
- [14] F. Dahri, A. M. El Hanafi, D. Handoko, and N. Wulan, “Implementation of microservices architecture in learning management system e-course using web service method,” *Sink. J. dan Penelit. Tek. Inform.*, vol. 6, no. 1, pp. 76–82, 2022, doi: 10.33395/sinkron.v7i1.11229.
- [15] X. Ren, H. Wang, and T. T. Cai, “Design and implementation of a microservices-based online learning platform,” in *Proceedings of the 2023 2nd International Conference on Educational Innovation and Multimedia Technology (EIMT 2023)*, 2023, pp. 455–460, doi: 10.2991/978-94-6463-192-0_60.
- [16] M. Waseem, P. Liang, M. Shahin, A. Di Salle, and G. Márquez, “Design, monitoring, and testing of microservices systems: The practitioners’ perspective,” *J. Syst. Softw.*, vol. 182, Art. no. 111061, 2021, doi: 10.1016/j.jss.2021.111061.
- [17] A. Suljkanović, B. Milosavljević, V. Indić, and I. Dejanović, “Developing microservice-based applications using the Silvera domain-specific language,” *Appl. Sci.*, vol. 12, no. 13, Art. no. 6679, 2022, doi: 10.3390/app12136679.
- [18] R. Laigner, Y. Zhou, M. A. V. Salles, Y. Liu, and M. Kalinowski, “Data management in microservices: State of the practice, challenges, and research directions,” *Proc. VLDB Endow.*, vol. 14, no. 13, pp. 3348–3361, 2021, doi: 10.14778/3484224.3484232.
- [19] J. Kazanavičius, D. Mažeika, and D. Kalibaitien, “An approach to migrate a monolith database into multi-model polyglot persistence based on microservice architecture: A case study for mainframe database,” *Appl. Sci.*, vol. 12, no. 12, Art. no. 6189, 2022, doi: 10.3390/app12126189.
- [20] C. Richardson, “Pattern: Database per service,” *Microservices.io*. Accessed: Jun. 12, 2026. [Online]. Available: <https://microservices.io/patterns/data/database-per-service.html>.
- [21] M. A. Al Hilmi, F. P. B. Muhamad, K. A. Cahyanto, and S. R. Mutahari, “Penerapan microservice pada pengembangan aplikasi manajemen kegiatan Rumah Tahfiz Qur’an Ulil Albab Kabupaten Indramayu,” *J. IKRAITH-INFORMATIKA*, vol. 6, no. 2, pp. 63–72, 2022.
- [22] D. Al Fansha, M. Y. H. Setyawan, and M. N. Fauzan, “Load test pada microservice yang menerapkan CQRS dan event sourcing,” *J. Buana Inform.*, vol. 12, no. 2, pp. 126–134, 2021, doi: 10.24002/jbi.v12i2.4749.
- [23] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner’s Approach*, 9th ed. New York, NY, USA: McGraw-Hill Education, 2019.
- [24] I. Sommerville, *Software Engineering, Global Edition*, 10th ed. Harlow, U.K.: Pearson Education, 2016.
- [25] M. R. Maulana, B. Susanto, and P. A. Christianto, “Pengujian black box dengan teknik equivalence partitioning pada aplikasi monitoring pemberian obat filariasis berbasis Android,” *KLIK Kaji. Ilm. Inform. dan Komput.*, vol. 4, no. 4, pp. 2179–2187, 2024, doi: 10.30865/klik.v4i4.1603.
- [26] F. In’am Pratama, E. M. N. Subroto, R. M. Haira, and M. A. Yaqin, “Pengujian black box pada aplikasi e-commerce OpenCart dengan metode equivalence partitioning dan boundary value analysis,” *J. Ilm. Inform.*, vol. 8, no. 1, pp. 54–64, 2023, doi: 10.35316/jimi.v8i1.54-64.