Indonesian Journal of Multidisciplinary on Social and Technology
Homepage: https://journal.ilmudata.co.id/index.php/ijmst
Vol. 4 No. 3 (2026) pp: 8121-8130
P-ISSN: 2986-6790, e-ISSN: 2986-6782

Evaluasi Isolasi Kegagalan dan Skalabilitas Arsitektur Microservices pada Platform Layanan Pelanggan UMKM

Reksa Leo Saputra, Bambang Pramono, Isnawaty
Jurusan Teknik Informatika, Fakultas Teknik, Universitas Halu Oleo
muhreksa122@gmail.com, bambang.pramono@uho.ac.id, isnawaty@uho.ac.id

Abstrak

Usaha Mikro, Kecil, dan Menengah (UMKM) di sektor retail seperti fashion dan kuliner umumnya masih menangani pertanyaan pelanggan dan pencatatan pesanan secara manual melalui WhatsApp, sehingga menimbulkan keterlambatan respons dan ketidakkonsistenan data. Penelitian ini merancang, mengimplementasikan, dan mengevaluasi arsitektur microservices untuk platform otomatisasi layanan pelanggan berbasis chatbot WhatsApp dan manajemen pesanan bagi UMKM. Sistem dipecah menjadi empat layanan independen — layanan core, layanan kecerdasan buatan (AI), layanan penghubung WhatsApp, dan frontend — yang dipisahkan berdasarkan perbedaan lingkungan eksekusi, tanggung jawab fungsional, dan kebutuhan isolasi kegagalan, kemudian dikontainerisasi dengan Docker dan diorkestrasi menggunakan Kubernetes. Pengembangan mengikuti Rational Unified Process. Evaluasi difokuskan pada dua atribut kualitas arsitektur, yaitu isolasi kegagalan dan skalabilitas, diuji menggunakan Apache JMeter serta penyuntikan kegagalan pada layanan AI dan layanan WhatsApp, dan dilengkapi pengujian penerimaan pengguna (UAT) kepada 20 pemilik UMKM. Hasil UAT menunjukkan tingkat penerimaan rata-rata 86,1 persen (kategori Sangat Baik). Pengujian skalabilitas pada beban 200 pengguna serentak menunjukkan tingkat kegagalan menurun dari 75,40 persen pada satu replica menjadi 0 persen pada tiga replica, dengan throughput meningkat dari 15,23 menjadi 72,89 permintaan per detik. Pengujian isolasi kegagalan membuktikan pemesanan manual dan akses marketplace tetap berfungsi ketika layanan AI dimatikan, serta chatbot pada halaman toko tetap berfungsi ketika layanan WhatsApp dimatikan. Disimpulkan bahwa arsitektur microservices sesuai diterapkan pada platform dengan karakteristik multi-runtime, multi-kanal, dan ketergantungan pada layanan eksternal seperti yang dihadapi UMKM sektor retail.

Kata kunci: *Arsitektur Microservices, Isolasi Kegagalan, Skalabilitas, UMKM, Chatbot WhatsApp*

Abstract

Micro, Small, and Medium Enterprises (MSMEs) in retail sectors such as fashion and culinary still largely handle customer inquiries and order recording manually through WhatsApp, causing delayed responses and inconsistent order data. This research designs, implements, and evaluates a microservices architecture for a WhatsApp chatbot-based customer service automation and order management platform for MSMEs. The system is decomposed into four independent services — a core service, an AI service, a WhatsApp connector service, and a frontend — separated based on differences in execution runtime, functional responsibility, and failure-isolation needs, then containerized with Docker and orchestrated using Kubernetes. Development follows the Rational Unified Process. The evaluation focuses on two architectural quality attributes, fault isolation and scalability, tested using Apache JMeter and failure injection on the AI and WhatsApp services, complemented by User Acceptance Testing (UAT) involving 20 MSME owners. UAT results show an average acceptance rate of 86.1 percent (Very Good category). Scalability testing at a load of 200 concurrent users shows the error rate dropping from 75.40 percent at one replica to 0 percent at three replicas, with throughput increasing from 15.23 to 72.89 requests per second. Fault-isolation testing confirms that manual ordering and marketplace access remain functional when the AI service is stopped, and that the store-page chatbot remains functional when the WhatsApp service is stopped. It is concluded that a microservices architecture is well suited to platforms characterized by multi-runtime components, multi-channel access, and dependence on external services, as commonly faced by retail MSMEs.

Keywords: *Microservices Architecture, Fault Isolation, Scalability, MSME, WhatsApp Chatbot*

1. Pendahuluan

Usaha Mikro, Kecil, dan Menengah (UMKM) merupakan salah satu penopang utama perekonomian Indonesia, namun sebagian besar masih menghadapi hambatan transformasi digital akibat keterbatasan literasi digital serta sumber daya finansial dan teknis (Hidayat et al., 2024). Pada UMKM sektor retail seperti fashion dan kuliner yang

memiliki intensitas interaksi pelanggan tinggi, layanan pelanggan dan pencatatan pesanan umumnya masih ditangani secara manual melalui aplikasi pesan instan, khususnya WhatsApp (Miraz et al., 2024). Ketergantungan pada proses manual ini menimbulkan keterlambatan respons, ketidakkonsistenan data pesanan, dan meningkatnya beban kerja administratif, sehingga otomatisasi layanan pelanggan berbasis chatbot yang terintegrasi dengan manajemen pesanan ditawarkan sebagai solusi (Mwikali, 2024; Reddy Gali, 2025).

Pada sektor ritel, khususnya usaha fashion dan kuliner, interaksi dengan pelanggan berlangsung secara intensif. Pertanyaan mengenai produk, ketersediaan barang, harga, variasi produk, hingga proses pemesanan dapat terjadi secara berulang dalam waktu yang berdekatan. Dalam praktiknya, komunikasi tersebut banyak dilakukan melalui aplikasi pesan instan, terutama WhatsApp. Apabila seluruh komunikasi dan pencatatan pesanan dilakukan secara manual, beban administrasi akan meningkat dan terdapat kemungkinan terjadinya keterlambatan respons maupun ketidakkonsistenan dalam pencatatan data pesanan. Permasalahan tersebut menunjukkan kebutuhan terhadap sistem yang mampu mengotomatisasi sebagian proses layanan pelanggan sekaligus tetap menyediakan mekanisme pengelolaan transaksi secara terstruktur.

Membangun platform yang mengintegrasikan otomatisasi layanan pelanggan dan manajemen pesanan bukan persoalan sederhana yang dapat diselesaikan oleh aplikasi tunggal, karena melibatkan komponen dengan karakteristik teknis berbeda: modul otomatisasi percakapan berbasis pustaka kecerdasan buatan, modul penghubung kanal WhatsApp yang berjalan pada lingkungan eksekusi (runtime) tersendiri, serta modul pengelolaan produk dan pesanan. Sistem juga bergantung pada layanan eksternal — antarmuka pemrograman model bahasa dan konektivitas WhatsApp — yang keandalannya tidak sepenuhnya berada dalam kendali pengembang. Karakteristik ini secara langsung dapat dipetakan pada keunggulan arsitektur microservices: keberagaman teknologi (technology heterogeneity) menjawab kebutuhan runtime berbeda antara layanan AI berbasis Python dan layanan penghubung WhatsApp berbasis Node.js; isolasi kegagalan (fault isolation) menjawab ketergantungan pada layanan eksternal yang tidak selalu stabil; dan penskalaan selektif menjawab kebutuhan menangani beban yang tidak merata antarkomponen (Sigala, 2025; Cherukuri, 2020).

Namun, sistem layanan pelanggan yang mengintegrasikan chatbot, WhatsApp, pengelolaan transaksi, dan antarmuka web memiliki karakteristik teknis yang berbeda dibandingkan aplikasi web sederhana. Komponen AI dapat menggunakan runtime dan pustaka yang berbeda dengan komponen komunikasi WhatsApp. Selain itu, chatbot bergantung pada layanan model bahasa eksternal, sedangkan konektor WhatsApp bergantung pada konektivitas dan sesi komunikasi dengan platform eksternal. Di sisi lain, fungsi transaksi seperti pengelolaan produk dan pesanan memerlukan tingkat ketersediaan yang lebih tinggi karena berhubungan langsung dengan data bisnis.

Karakteristik tersebut menyebabkan pendekatan arsitektur menjadi aspek penting dalam pengembangan sistem. Arsitektur monolitik dapat memberikan kesederhanaan dalam tahap pengembangan dan deployment, tetapi seluruh komponen aplikasi berada dalam satu unit sehingga gangguan pada bagian tertentu berpotensi memberikan dampak yang lebih luas. Sebaliknya, microservices memungkinkan aplikasi dipecah menjadi sejumlah layanan independen berdasarkan tanggung jawab fungsional dan karakteristik teknisnya. Pemisahan tersebut dapat mendukung isolasi kegagalan, pengembangan dengan teknologi yang berbeda, serta penskalaan layanan secara selektif.

Penerapan arsitektur microservices pada sistem informasi dan layanan publik telah banyak dikaji dan dilaporkan unggul dalam hal skalabilitas serta fleksibilitas pengembangan (Esparza-Peidro et al., 2024; Zuhdi Ali Hisyam et al., 2024). Namun, sebagian besar kajian tersebut berfokus pada penerapan berskala enterprise, sementara evaluasi penerapannya pada konteks UMKM yang menuntut solusi berbiaya rendah namun tetap andal masih terbatas (Pradesa et al., 2023). Penelitian pembanding arsitektur monolitik dan microservices pada domain lain juga menunjukkan bahwa keunggulan arsitektur bersifat kontekstual, bergantung pada karakteristik beban kerja (Syah Redha et al., 2023). Sejauh penelusuran literatur yang dilakukan, penelitian yang secara khusus merancang dan mengevaluasi atribut isolasi kegagalan dan skalabilitas arsitektur microservices pada platform otomatisasi layanan pelanggan multi-kanal (chatbot WhatsApp dan web) untuk UMKM sektor retail belum ditemukan. Kesenjangan inilah yang mendasari penelitian ini.

Berdasarkan permasalahan tersebut, penelitian ini berfokus pada evaluasi dua karakteristik penting, yaitu isolasi kegagalan dan skalabilitas. Isolasi kegagalan diperlukan untuk mengetahui apakah kegagalan pada layanan AI atau layanan WhatsApp dapat dicegah agar tidak menghentikan fungsi bisnis utama. Sementara itu, pengujian skalabilitas diperlukan untuk mengetahui kemampuan sistem dalam mempertahankan kinerja ketika jumlah

DOI: https://doi.org/10.69693/ijmst.v4i3.14040

pengguna meningkat serta untuk melihat pengaruh penambahan jumlah replica terhadap error rate, throughput, dan response time.

Penelitian ini bertujuan untuk merancang dan mengimplementasikan arsitektur microservices pada platform layanan pelanggan UMKM serta mengevaluasi kemampuan arsitektur tersebut dalam menghadapi peningkatan beban dan kegagalan pada layanan tertentu. Hasil evaluasi diharapkan dapat memberikan gambaran empiris mengenai penerapan microservices pada sistem layanan pelanggan UMKM yang memiliki karakteristik multi-runtime, multi-channel, dan ketergantungan terhadap layanan eksternal.

2. Metode Penelitian

2.1. Objek dan Arsitektur Sistem

Objek penelitian adalah platform OM GENT yang dibangun dan dirancang untuk membantu UMKM dalam mengelola interaksi pelanggan, produk, pesanan, pembayaran, dan kanal penjualan. Sistem menggunakan pendekatan microservices dengan membagi fungsi aplikasi ke dalam empat layanan utama, yaitu core service, AI service, WhatsApp connector, dan frontend service.

Core service berfungsi sebagai pusat pengelolaan proses bisnis, meliputi autentikasi, pengelolaan produk, pesanan, pembayaran, toko, dan marketplace. Layanan ini dibangun menggunakan Python dan FastAPI. Core service ditempatkan sebagai layanan inti karena sebagian besar proses transaksi membutuhkan konsistensi terhadap data bisnis. Selain itu, layanan dibuat bersifat stateless sehingga memungkinkan penambahan replica ketika terjadi peningkatan beban.

AI service juga dibangun menggunakan Python dan FastAPI. Layanan ini menangani fungsi chatbot, agent, tool-calling, dan pencarian semantik terhadap data produk. Berbeda dengan core service, AI service memiliki ketergantungan terhadap layanan Large Language Model (LLM) eksternal. Oleh karena itu, layanan AI ditempatkan sebagai service yang terpisah agar kegagalan komunikasi dengan layanan eksternal tersebut tidak secara langsung menghentikan fungsi transaksi utama.

WhatsApp connector dibangun menggunakan Node.js dan Baileys. Layanan ini bertugas menjadi penghubung antara platform dengan WhatsApp. Pemisahan layanan WhatsApp dilakukan karena karakteristik runtime-nya berbeda dengan layanan Python serta memiliki ketergantungan terhadap koneksi dan sesi komunikasi dengan WhatsApp.

Frontend dikembangkan menggunakan React dan Nginx. Layanan ini menyediakan antarmuka untuk toko, marketplace, dan dashboard administrator. Pemisahan frontend dari layanan bisnis memungkinkan antarmuka berkomunikasi melalui API tanpa harus bergantung secara langsung terhadap implementasi internal core service.

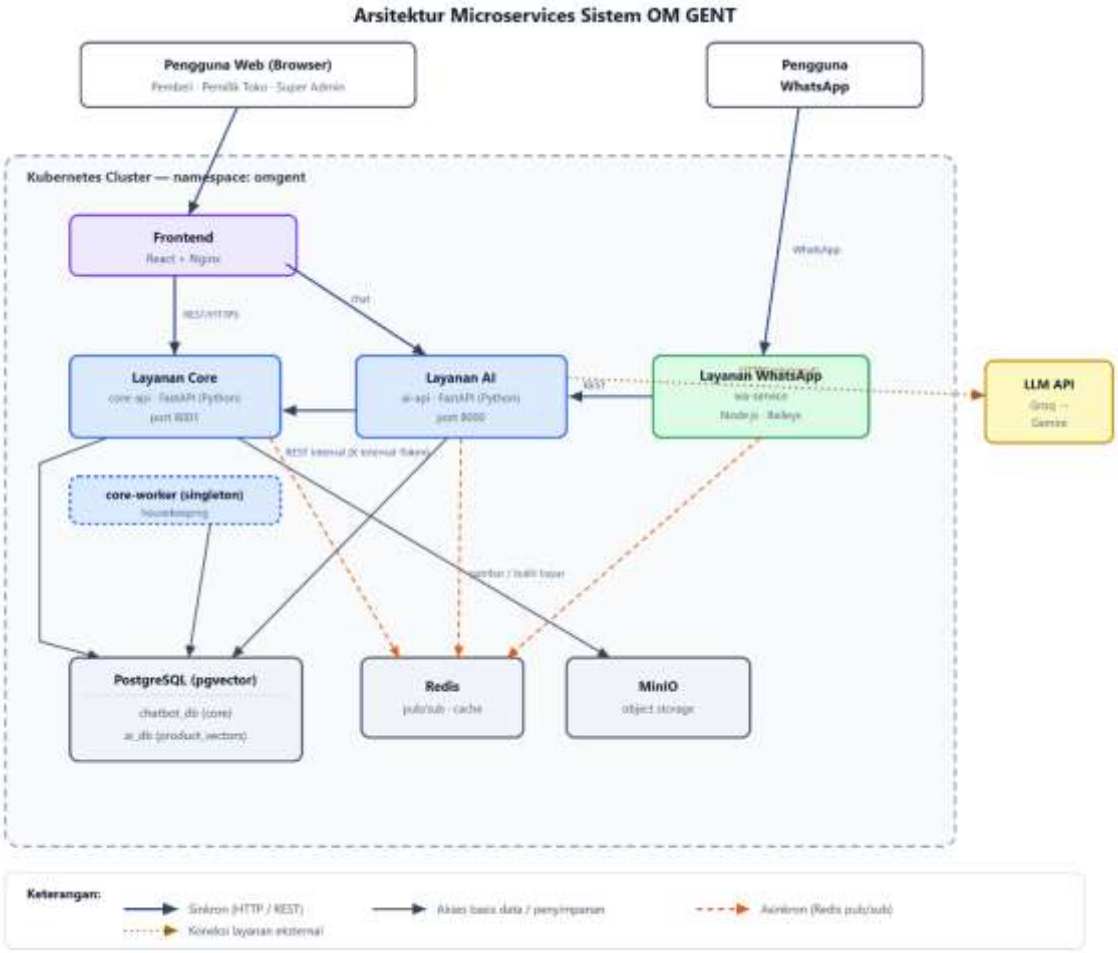
Setiap layanan menggunakan penyimpanan data sesuai kebutuhan fungsionalnya. Core service menggunakan PostgreSQL untuk menyimpan data pengguna, produk, toko, dan pesanan. AI service menggunakan PostgreSQL dengan ekstensi pgvector untuk mendukung penyimpanan dan pencarian vektor produk. Data AI tidak diakses secara langsung dari database core, tetapi diperoleh melalui API yang disediakan core service. Pola tersebut digunakan untuk menjaga batas tanggung jawab antarlayanan dan mengurangi ketergantungan langsung terhadap struktur database layanan lain.

Sistem yang dibangun, OM GENT, dipecah menjadi empat layanan independen berdasarkan perbedaan lingkungan eksekusi (runtime), tanggung jawab fungsional, dan kebutuhan isolasi kegagalan, sebagaimana disajikan pada Tabel 1.

DOI: https://doi.org/10.69693/ijmst.v4i3.14040

Tabel 1. Pemisahan Layanan pada Arsitektur Microservices

Layanan	Runtime	Tanggung Jawab	Dasar Pemisahan
Core (core-api)	Python (FastAPI)	Autentikasi, produk, pesanan, pembayaran, toko, marketplace	Inti logika bisnis; dirancang stateless agar dapat diskalakan
AI (ai-api)	Python (FastAPI)	Percakapan chatbot agentik, tool-calling, pencarian semantik	Bergantung pada layanan LLM eksternal; diisolasi agar kegagalannya tidak memengaruhi transaksi
Penghubung WhatsApp (wa-service)	Node.js (Baileys)	Jembatan komunikasi WhatsApp–layanan AI	Runtime berbeda; bergantung pada konektivitas WhatsApp yang tidak stabil
Frontend	React + Nginx	Antarmuka toko, marketplace, dasbor admin	Terpisah dari logika bisnis (pola client–server)



Gambar 1. Alur Arsitektur Microservice pada Sistem

Setiap layanan memiliki basis data tersendiri (database per service): layanan core menggunakan PostgreSQL untuk data toko, pengguna, produk, dan pesanan, sedangkan layanan AI menggunakan PostgreSQL dengan ekstensi pgvector untuk data vektor produk. Komunikasi antarlayanan dilakukan secara sinkron melalui REST API berbasis HTTP yang dilindungi token internal, dan secara asinkron melalui mekanisme publish–subscribe pada Redis untuk menyebarkan notifikasi perubahan status pesanan. Seluruh layanan dikemas sebagai container Docker dan dijalankan pada cluster Kubernetes single-node, didukung komponen infrastruktur PostgreSQL, Redis, dan penyimpanan objek MinIO untuk berkas gambar dan bukti pembayaran.

DOI: https://doi.org/10.69693/ijmst.v4i3.14040

2.2. Tahapan Pengembangan Sistem

Pengembangan sistem mengikuti Rational Unified Process (RUP) yang terdiri atas empat fase (Irawan Chandra et al., 2024). yaitu inception, elaboration, construction, dan transition.

Pada fase inception, dilakukan identifikasi ruang lingkup sistem, kebutuhan pengguna, serta fungsi utama yang harus tersedia pada platform. Kebutuhan tersebut mencakup pengelolaan produk, pengelolaan pesanan, layanan chatbot, integrasi WhatsApp, marketplace, dan dashboard administrasi.

Fase elaboration digunakan untuk menentukan rancangan arsitektur sistem. Pada fase ini dilakukan identifikasi batas tanggung jawab setiap layanan, mekanisme komunikasi antarlayanan, kebutuhan database, serta ketergantungan terhadap layanan eksternal. Pemisahan service ditentukan berdasarkan perbedaan tanggung jawab fungsional, runtime, dan kebutuhan isolasi kegagalan.

Pada fase construction, masing-masing layanan diimplementasikan menggunakan teknologi yang telah ditentukan. Core dan AI dikembangkan menggunakan FastAPI, layanan WhatsApp menggunakan Node.js dan Baileys, sedangkan frontend menggunakan React. Setiap layanan dikemas dalam container Docker sehingga dapat dikelola secara terpisah pada Kubernetes.

Fase transition digunakan untuk melakukan pengujian terhadap sistem yang telah diimplementasikan. Pengujian mencakup User Acceptance Test (UAT), pengujian skalabilitas menggunakan Apache JMeter, serta pengujian isolasi kegagalan dengan menghentikan layanan tertentu. Sistem kemudian dideploy pada lingkungan cloud/public untuk memastikan fungsi layanan dapat diakses dalam kondisi deployment yang mendekati penggunaan sebenarnya.

2.3. Skenario Pengujian Penerimaan Pengguna (UAT)

Pengujian penerimaan pengguna dilakukan untuk menilai kesesuaian sistem dengan kebutuhan pengguna menggunakan skala Likert lima tingkat, dengan aspek penilaian mengacu pada ISO 25010, yaitu kesesuaian fungsional, efisiensi kinerja, dan kemudahan penggunaan (Thabibi et al., 2025). Kuesioner berisi tujuh pernyataan disebarkan kepada 20 pemilik UMKM sektor fashion dan kuliner di Kota Kendari yang telah mencoba sistem. Skor tiap pernyataan dihitung menggunakan persamaan (1), dan persentase tingkat penerimaan dihitung menggunakan persamaan (2):

$$\text{Score} = \sum(\text{Jumlah responden} \times \text{bobot jawaban}) \quad (1)$$

$$\text{Percentage} = \frac{\text{score}}{\text{score maksimal}} \times 100\% \quad (2)$$

dengan skor maksimal = 5 × jumlah responden. Hasil persentase diinterpretasikan ke dalam lima kategori dari "Sangat Tidak Baik" (0–20%) hingga "Sangat Baik" (81–100%).

Penggunaan UAT dalam penelitian ini tidak hanya digunakan untuk mengetahui apakah fungsi dapat berjalan secara teknis, tetapi juga untuk mengetahui apakah fungsi tersebut dapat diterima dan digunakan oleh pengguna

DOI: https://doi.org/10.69693/ijmst.v4i3.14040

sasaran. Hal ini penting karena peningkatan kualitas arsitektur perangkat lunak tidak hanya ditentukan oleh parameter teknis, tetapi juga oleh kemampuan sistem memenuhi kebutuhan operasional pengguna.

2.4. Skenario Pengujian Skalabilitas[1]

Pengujian skalabilitas dilakukan menggunakan Apache JMeter dengan memberikan beban pada sistem melalui endpoint marketplace. Endpoint tersebut dipilih karena merepresentasikan pola akses baca (read-heavy) yang berpotensi menghasilkan banyak permintaan secara bersamaan.

Variabel utama dalam pengujian adalah jumlah replica core service. Pengujian dilakukan dengan konfigurasi satu, dua, dan tiga replica. Masing-masing konfigurasi diuji menggunakan tiga tingkat concurrent users, yaitu 50, 100, dan 200 pengguna secara bersamaan.

Parameter yang diamati terdiri atas average response time, throughput, dan error rate. Average response time digunakan untuk melihat waktu rata-rata yang diperlukan sistem dalam memberikan respons. Throughput menunjukkan jumlah request yang berhasil diproses sistem dalam satuan request per second. Sementara itu, error rate digunakan untuk menunjukkan persentase permintaan yang gagal.

Pengujian dilakukan melalui Kubernetes Service sehingga permintaan dari JMeter dapat diteruskan kepada pod yang tersedia. Dengan demikian, perubahan jumlah replica dapat diamati pengaruhnya terhadap kemampuan core service dalam menangani peningkatan jumlah permintaan.

Variasi jumlah replica digunakan sebagai variabel independen karena tujuan pengujian bukan hanya mengukur kemampuan sistem pada satu konfigurasi, tetapi juga mengetahui perubahan kinerja ketika kapasitas layanan ditingkatkan. Sementara itu, jumlah concurrent users digunakan sebagai representasi peningkatan beban kerja.

2.5. Skenario Isolasi Kegagalan

Pengujian isolasi kegagalan bertujuan membuktikan bahwa kegagalan pada satu layanan tidak melumpuhkan layanan lain yang tidak bergantung padanya. Pengujian dilakukan dengan menghentikan layanan AI dan layanan WhatsApp secara bergantian, kemudian mengamati apakah fungsi pemesanan manual, akses marketplace dan toko, serta chatbot pada halaman toko tetap berjalan, sebagaimana disajikan pada Tabel 2.

Pada skenario pertama, AI service dihentikan dan sistem diamati dari sisi akses toko, marketplace, serta proses pemesanan manual. Karena fungsi transaksi utama berada pada core service, sistem diharapkan tetap mampu melakukan proses transaksi meskipun chatbot AI tidak tersedia.

Pada skenario kedua, WhatsApp connector dihentikan. Pengujian dilakukan dengan mengakses store page, chatbot pada kanal web, dan proses pemesanan melalui antarmuka web. Tujuan skenario ini adalah mengetahui apakah kegagalan koneksi WhatsApp menyebabkan fungsi web ikut berhenti.

Selain penghentian layanan, dilakukan pula pengujian self-healing dengan menghapus pod secara paksa. Kondisi pod kemudian diamati untuk mengetahui apakah Kubernetes secara otomatis membuat pod pengganti sehingga jumlah replica kembali sesuai konfigurasi deployment.

Tabel 2. Skenario Pengujian Isolasi Kegagalan

No	Layanan yang Dihentikan	Fungsi yang Diamati	Hasil yang Diharapkan
1	AI (ai-api)	Pemesanan melalui kanal manual	Tetap berfungsi
2	AI (ai-api)	Akses marketplace dan toko	Tetap berfungsi
3	WhatsApp (wa-service)	Akses dan pemesanan melalui web	Tetap berfungsi

DOI: https://doi.org/10.69693/ijmst.v4i3.14040

3. Hasil dan Pembahasan

3.1. Implementasi Arsitektur

Seluruh layanan berhasil dijalankan pada cluster Kubernetes dalam kondisi Running dan telah di-deploy pada lingkungan cloud sehingga dapat diakses secara publik. Pemusatan logika bisnis diterapkan pada satu fungsi terpusat di layanan core yang dipanggil baik dari chatbot maupun dari pemesanan manual, sehingga logika pemesanan tetap konsisten di kedua kanal. Pemisahan basis data per layanan membuat layanan AI mengambil data produk hanya melalui REST API layanan core, bukan melalui akses langsung ke basis data, sehingga keterhubungan antarlayanan tetap longgar (loose coupling).

Hasil implementasi menunjukkan bahwa empat layanan dapat dijalankan secara independen pada Kubernetes, yaitu core service, AI service, WhatsApp connector, dan frontend. Masing-masing layanan berada dalam container yang terpisah dan mempunyai tanggung jawab fungsional yang berbeda.

Pemisahan tersebut memberikan batas yang jelas antara fungsi transaksi, fungsi kecerdasan buatan, integrasi WhatsApp, dan antarmuka pengguna. Core service menjadi pusat proses bisnis sehingga pemesanan yang berasal dari chatbot maupun pemesanan manual tetap mengarah pada mekanisme pengelolaan transaksi yang sama. Dengan demikian, pemisahan service tidak menyebabkan adanya duplikasi logika bisnis utama.

Pada sisi AI, data produk diperoleh melalui REST API dari core service. Pendekatan tersebut menyebabkan AI service tidak perlu mengakses database core secara langsung. Ketergantungan AI terhadap struktur database core dapat dikurangi karena perubahan pada penyimpanan internal core tidak harus diketahui oleh AI selama kontrak API tetap dipertahankan.

Pemisahan tersebut juga memberikan keuntungan dari sisi deployment. Perubahan atau pembaruan pada AI service dapat dilakukan tanpa harus melakukan deployment ulang terhadap core service. Hal yang sama berlaku pada WhatsApp connector dan frontend. Dengan demikian, siklus perubahan setiap komponen dapat dilakukan secara lebih independen.

Namun, independensi tersebut juga menghasilkan konsekuensi berupa bertambahnya komunikasi jaringan antarlayanan. Pada arsitektur monolitik, pemanggilan fungsi internal dapat dilakukan dalam satu proses aplikasi, sedangkan pada microservices komunikasi dilakukan melalui HTTP atau mekanisme messaging. Oleh karena itu, desain API dan mekanisme komunikasi menjadi bagian penting untuk menjaga konsistensi dan kinerja sistem.

Gambar 2. Contoh Halaman Dashboard Admin Toko

DOI: https://doi.org/10.69693/ijmst.v4i3.14040

3.2. Hasil Pengujian Penerimaan Pengguna

Hasil rekapitulasi jawaban 20 responden menunjukkan persentase rata-rata tingkat penerimaan sebesar 86,1 persen, termasuk kategori Sangat Baik, sebagaimana disajikan pada Tabel 3. Aspek kesesuaian fungsional — khususnya kemampuan sistem menerima dan mencatat pesanan melalui chat maupun manual — memperoleh persentase tertinggi (91%), menunjukkan bahwa fitur inti sistem berjalan sesuai kebutuhan pengguna. Aspek kemudahan penggunaan juga dinilai baik (89–90%), menandakan antarmuka mudah dipahami meskipun sebagian besar responden belum terbiasa menggunakan sistem serupa.

Tabel 3. Hasil Pengujian Penerimaan Pengguna (UAT)

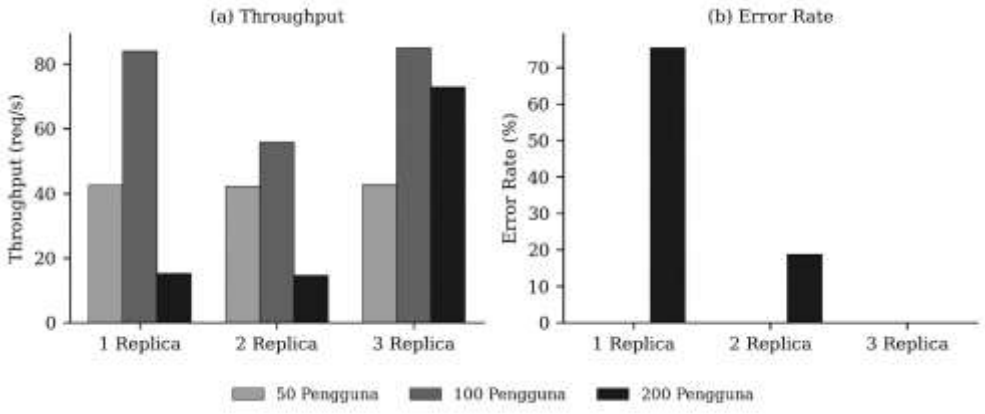
No	Pernyataan	Skor	Persentase	Kategori
1	Halaman login/register untuk toko dan pembeli	88	88%	Sangat Baik
2	Konfirmasi pembayaran melalui bukti transfer	76	76%	Baik
3	Chatbot WhatsApp membantu pertanyaan dan pesanan	83	83%	Sangat Baik
4	Sistem mencatat pesanan melalui chat maupun manual	91	91%	Sangat Baik
5	Notifikasi pesanan diterima dengan cepat	86	86%	Sangat Baik
6	Antarmuka mudah dipahami dan digunakan	90	90%	Sangat Baik
7	Proses menambah dan mengelola produk mudah	89	89%	Sangat Baik
Rata-rata			**86,1%**	**Sangat Baik**

3.3. Hasil Pengujian Skalabilitas

Tabel 4 menyajikan hasil pengujian skalabilitas pada tiga kondisi jumlah replica. Pada beban ringan (50 dan 100 pengguna), perbedaan kinerja antarjumlah replica relatif kecil karena satu replica masih mampu menangani seluruh permintaan tanpa kegagalan. Perbedaan signifikan terlihat pada beban 200 pengguna serentak: pada kondisi satu replica, sistem tidak mampu menangani seluruh permintaan, ditandai tingkat kegagalan mencapai 75,40 persen dan throughput menurun menjadi 15,23 permintaan per detik. Penambahan menjadi dua replica menurunkan tingkat kegagalan menjadi 18,75 persen, dan pada tiga replica sistem mampu menangani seluruh permintaan tanpa kegagalan (error rate 0%) dengan throughput meningkat menjadi 72,89 permintaan per detik.

Tabel 4. Hasil Pengujian Skalabilitas

Jumlah Replica	Beban (pengguna)	Rata-rata Response Time (ms)	Throughput (req/s)	Error Rate (%)
1	50	287,1	42,49	0,00
	100	414,4	83,99	0,00
	200	8.437,4	15,23	75,40
2	50	241,3	42,26	0,00
	100	793,4	55,86	0,00
	200	9.526	14,62	18,75
3	50	250,5	42,66	0,00
	100	436,7	85,12	0,00
	200	2.051,3	72,89	0,00



Gambar 3. Grafik Thoroughput dan Error rate Hasil Pengujian

DOI: https://doi.org/10.69693/ijmst.v4i3.14040

Hasil ini menunjukkan bahwa penambahan jumlah replica pada layanan core dapat meningkatkan kapasitas sistem dalam menangani beban tinggi, membuktikan bahwa arsitektur yang dibangun dapat diskalakan secara horizontal tanpa mengubah kode program — sejalan dengan karakteristik inti arsitektur microservices yang dilaporkan pada domain lain (Gudelli, 2021; Sakti et al., 2025).

3.4. Hasil Pengujian Isolasi Kegagalan

Ketika layanan AI dimatikan, halaman toko tetap dapat diakses dan pemesanan manual tetap dapat dilakukan serta tercatat di halaman admin toko, meskipun fitur chatbot tidak merespons. Ketika layanan WhatsApp dimatikan, chatbot pada halaman toko tetap berfungsi dan pemesanan tetap dapat diproses melalui web. Pengujian pemulihan otomatis menunjukkan Kubernetes berhasil membuat pod pengganti dengan cepat ketika sebuah pod dihapus secara paksa, mempertahankan jumlah replica sesuai konfigurasi. Hasil ini membuktikan bahwa kegagalan pada satu layanan tidak melumpuhkan layanan lain yang tidak bergantung padanya, konsisten dengan prinsip isolasi kegagalan (fault isolation) yang menjadi salah satu justifikasi utama pemilihan arsitektur microservices pada penelitian ini.

3.5. Pembahasan

Ketiga hasil pengujian saling melengkapi dalam menjawab rumusan masalah penelitian. Tingkat penerimaan pengguna sebesar 86,1 persen menunjukkan bahwa pemisahan layanan tidak mengorbankan kesesuaian fungsional dari sudut pandang pengguna akhir. Pengujian skalabilitas menunjukkan bahwa penskalaan selektif — menambah replica hanya pada layanan core yang menjadi jalur transaksi utama — efektif menangani lonjakan beban tanpa perlu menskalakan layanan AI maupun layanan WhatsApp yang bebannya lebih rendah. Pengujian isolasi kegagalan menunjukkan bahwa pemisahan layanan berdasarkan ketergantungan pada API eksternal (model bahasa dan konektivitas WhatsApp) berhasil mencegah gangguan pada layanan tersebut merambat ke transaksi inti. Temuan ini sejalan dengan penelitian yang menegaskan bahwa keunggulan arsitektur bersifat kontekstual dan bergantung pada karakteristik beban serta ketergantungan eksternal sistem (Syah Redha et al., 2023; Esparza-Peidro et al., 2024), dan melengkapi kajian microservices yang selama ini lebih banyak difokuskan pada sistem berskala enterprise dengan menyajikan bukti empiris pada konteks UMKM.

4. Kesimpulan

Penelitian ini berhasil merancang arsitektur microservices untuk platform layanan pelanggan dan manajemen pesanan UMKM dengan memisahkan sistem menjadi empat layanan independen — core, AI, penghubung WhatsApp, dan frontend — berdasarkan perbedaan runtime, tanggung jawab, dan kebutuhan isolasi kegagalan, serta mengimplementasikan dan mengorkestrasikannya menggunakan Docker dan Kubernetes hingga dapat diakses secara publik. Pengujian penerimaan pengguna menunjukkan tingkat penerimaan 86,1 persen (Sangat Baik). Pengujian skalabilitas membuktikan sistem dapat diskalakan secara horizontal, dengan tingkat kegagalan pada beban 200 pengguna serentak menurun dari 75,40 persen pada satu replica menjadi 0 persen pada tiga replica. Pengujian isolasi kegagalan membuktikan bahwa kegagalan pada layanan AI maupun layanan WhatsApp tidak menghentikan fungsi pemesanan manual, akses marketplace dan toko, sehingga sebagian fungsi sistem tetap berjalan meskipun terdapat komponen yang bermasalah. Dengan demikian, arsitektur microservices terbukti sesuai diterapkan pada platform dengan karakteristik multi-runtime dan multi-kanal yang bergantung pada layanan eksternal, sebagaimana dihadapi UMKM sektor retail. Penelitian selanjutnya disarankan menguji orkestrasi pada cluster multi-node untuk mengevaluasi ketahanan terhadap kegagalan tingkat node, serta mengintegrasikan payment gateway otomatis dan kanal layanan pelanggan tambahan selain WhatsApp.

Reference

1. Cherukuri, B. R. (2020). Microservices and containerization: Accelerating web development cycles. *World Journal of Advanced Research and Reviews*, 6(1), 283–296. https://doi.org/10.30574/wjarr.2020.6.1.0087
2. Diantono Abda'u, P., Susanto, A., Supriyono, A. R., & Prasetyanti, D. N. (2024). Perbandingan kinerja antara Gatling dan Apache JMeter pada uji beban RESTful API. *Infotekmesin*, 15(1), 211–215. https://doi.org/10.35970/infotekmesin.v15i1.2176
3. Esparza-Peidro, J., Muñoz-Escoí, F. D., & Bernabéu-Aubán, J. M. (2024). Modeling microservice architectures. *Journal of Systems and Software*, 213, 112041. https://doi.org/10.1016/j.jss.2024.112041
4. Gudelli, V. R. (2021). Kubernetes-based orchestration for scalable cloud solutions. *International Journal of Novel Research and Development*, 6. http://doi.one/10.1729/Journal.43763
5. Hidayat, I., Qurotulaini, D. L., Safitri, N. A., & Novitasari, R. (2024). Transformasi digital pada UMKM di Indonesia dalam menghadapi tantangan dan peluang pada akses pembiayaan digital. *JIIC: Jurnal Intelek Insan Cendikia*, 7414–7423. https://jicnusantara.com/index.php/jiic/article/view/1992

DOI: https://doi.org/10.69693/ijmst.v4i3.14040

6. Irawan Chandra, Y., Ruri Irawati, D., & Riastuti, M. (2024). Penerapan model Rational Unified Process (RUP) dalam membangun aplikasi promosi dan penjualan berbasis web. *IKRAITH-INFORMATIKA*, 107–120. https://doi.org/10.37817/ikraith-informatika.v8i3.4369
7. Iswari, L., & Nasution. (2021). Penerapan React JS pada pengembangan front-end aplikasi startup Ubaform. *Automata*, 193–200. https://journal.uii.ac.id/AUTOMATA/article/view/19532
8. Koç, H., Erdoğan, A. M., Barjakly, Y., & Peker, S. (2021). UML diagrams in software engineering research: A systematic literature review. *Procedia Computer Science*, 74, 13–21. https://doi.org/10.3390/proceedings2021074013
9. Miraz, M. H., Ya'u, A., Adeyinka-Ojo, S., Sarkar, J. B., Hasan, M. T., Hoque, K., & Jin, H. H. (2024). Intention to use determinants of AI chatbots to improve customer relationship management efficiency. *Cogent Business and Management*, 11(1), 1–21. https://doi.org/10.1080/23311975.2024.2411445
10. Mwikali, H. K. (2024). The role of AI and automation in enhancing customer support: Opportunities and challenges. *IDOSR Journal of Current Issues in Social Sciences*, 10(1), 30–32. http://doi.org/10.59298/JCISS/2024/101.19273032
11. Pradesa, E., Syahrani, T., & Sakti, R. E. (2023). Transformasi digital adopsi software as a service layanan cloud accounting oleh UMKM. *BUDGETING: Journal of Business, Management and Accounting*, 5(1), 155–169. https://doi.org/10.31539/budgeting.v5i1.7049
12. Reddy Gali, H. (2025). Understanding order management systems in retail and e-commerce. *Journal of Information Systems Engineering and Management*, 2025, 59s. https://doi.org/10.52783/jisem.v10i59s.12997
13. Sakti, M. D. I., Dirgantara, D., Ramadhan, A. K., & Ibrahim, M. A. (2025). Optimizing asynchronous performance in Node.js with Express and PostgreSQL. *Procedia Computer Science*, 269, 172–181. http://doi.org/10.1016/j.procs.2025.08.270
14. Sigala, N. S. (2025). Microservices architecture in cloud computing: A software engineering perspective on design, deployment, and management. *International Journal of Research and Innovation in Social Science (IJRISS)*, 9(15), 215–239. https://dx.doi.org/10.47772/IJRISS.2025.915EC0013
15. Syah Redha, F., Puspita Sari, R., & Rahmayuda, S. (2023). Perbandingan performa web services yang dibangun menggunakan arsitektur monolithic dan microservices pada sistem point of sales. *Jurnal Teknik Informatika dan Sistem Informasi*, 10, 406–420. https://doi.org/10.35957/jatisi.v10i1.3210
16. Thabibi, H., Wati, S. F. A., & Rinjeni, T. P. (2025). Implementasi User Acceptance Testing (UAT) pada website e-commerce UMKM BBhealthy. *Adopsi Teknologi dan Sistem Informasi (ATASI)*, 4(1), 19–26. https://doi.org/10.30872/atasi.v4i1.2904
17. Zalsahra, R., Sujadi, H., & Suhendri. (2025). Perancangan dan implementasi bot WhatsApp otomatis menggunakan Node JS dan Baileys berbasis website. *Seminar Teknologi Majalengka (STIMA)*, 1, 265–271. https://prosiding.unma.ac.id/index.php/stima/article/view/1348
18. Zuhdi Ali Hisyam, Ridho, F., & Setiyawan, A. (2024). Implementation of a RESTful API-based evolutionary algorithm in a microservices architecture for course timetabling. *Jurnal Aplikasi Statistika & Komputasi Statistik*, 16(2), 175–192. http://doi.org/10.34123/jurnalasks.v16i2.796

DOI: https://doi.org/10.69693/ijmst.v4i3.14040