Indonesian Journal of Multidisciplinary on Social and Technology
Homepage: https://journal.ilmudata.co.id/index.php/ijmst
Vol. 4 No. 3 (2026) pp: 5866-5878
P-ISSN: 2986-6790, e-ISSN: 2986-6782

Sistem Pelaporan Kekerasan terhadap Perempuan dan Anak Menggunakan Arsitektur *Microservices*

Muh. Afdal Ziqri Ramadhan, Bambang Pramono, Isnawaty
Jurusan Teknik Informatika, Fakultas Teknik, Universitas Halu Oleo
glxyull@gmail.com, bambang.pramono@uho.ac.id, isnawaty@uho.ac.id

Abstrak

Penanganan kasus kekerasan terhadap perempuan dan anak di UPTD PPA Kota Kendari masih mempertukarkan status kasus secara manual antartahap, sehingga informasi mudah terputus dan sulit dipantau. Penelitian ini membangun sistem pelaporan dan manajemen kasus berbasis web dengan arsitektur microservices yang memisahkan layanan pelaporan dan layanan manajemen kasus, serta menerapkan RabbitMQ sebagai message broker untuk melakukan decoupling penyelarasan status antarlayanan, kemudian mengevaluasi apakah kompleksitas tersebut sepadan dengan karakteristik organisasi penggunanya. Sistem dikembangkan mengikuti Rational Unified Process pada kontainer Docker. Evaluasi mencakup dua eksperimen terpisah, yaitu pengujian beban menggunakan Apache JMeter terhadap lima endpoint pada beban 50, 100, dan 150 virtual user dengan pembanding versi monolitik, serta pengujian ketahanan melalui penyuntikan kegagalan terhadap 200 perubahan status ketika layanan pelaporan dimatikan. Hasilnya menunjukkan arsitektur monolitik mencapai performa setara pada empat dari lima endpoint meskipun hanya memperoleh separuh jatah CPU. Keunggulan microservices yang konsisten hanya ditemukan pada jalur tulis lintas layanan, dengan waktu respons 57,2 persen lebih rendah dan throughput 45,2 persen lebih tinggi pada beban 150 virtual user. Pengujian ketahanan menunjukkan seluruh perubahan status tertahan pada antrean dan tersinkron sepenuhnya setelah layanan pulih, sedangkan pada rancangan tanpa message broker seluruhnya hilang permanen. Disimpulkan bahwa kompleksitas microservices melampaui kebutuhan operasional instansi tersebut, sehingga pemilihan arsitektur perlu didasarkan pada karakteristik organisasi dan beban kerja yang sesungguhnya.

Kata kunci: Sistem Pelaporan, Kekerasan Perempuan Dan Anak, Arsitektur Microservices, Message Broker, Evaluasi Arsitektur

Abstract

The handling of violence cases against women and children at the UPTD PPA of Kendari City still involves manual status exchanges between stages, making information prone to disconnection and difficult to monitor. This research develops a web-based reporting and case management system using a microservices architecture that separates reporting and case management services. It implements RabbitMQ as a message broker to decouple status synchronization between services, evaluating whether this complexity aligns with the organization's characteristics. The system was developed following the Rational Unified Process on Docker containers. The evaluation includes two experiments: load testing using Apache JMeter on five endpoints with 50, 100, and 150 virtual users compared to a monolithic version, and resilience testing via failure injection on 200 status changes while the reporting service is down.Results indicate the monolithic architecture achieves equivalent performance on four of five endpoints, despite receiving half the CPU allocation. The consistent advantage of microservices is only observed in cross-service write paths, showing 57.2 percent lower response time and 45.2 percent higher throughput at 150 virtual users. Resilience testing demonstrates all status changes are retained in the queue and fully synchronized after service recovery, whereas they are permanently lost without a message broker. Ultimately, the microservices' complexity exceeds the institution's operational needs; thus, architectural selection must depend on actual organizational characteristics and workloads.

Keywords: Reporting System, Violence Against Women And Children, Microservices Architecture, Message Broker, Architecture Evaluation.

1. Pendahuluan

Kekerasan terhadap perempuan dan anak merupakan isu sosial yang menuntut penanganan serius, khususnya pada tingkat pemerintah daerah yang menjadi ujung tombak pelayanan bagi korban. Pada Dinas Pemberdayaan Perempuan dan Perlindungan Anak (DPPPPA) Kota Kendari, proses penanganan kasus kekerasan terhadap perempuan dan anak masih menghadapi kendala koordinasi antara tahap pelaporan dan tahap penanganan, karena

status kasus dipertukarkan secara manual antarbidang sehingga informasi mudah terputus dan sulit dipantau secara real-time. Pemanfaatan sistem informasi berbasis web untuk membenahi tata kelola layanan instansi pemerintah daerah di Sulawesi Tenggara sendiri telah beberapa kali dilakukan, baik melalui perancangan ulang sistem yang telah berjalan (Nirwana et al., 2025) maupun melalui penyajian data kewilayahan secara terpusat (Nisa et al., 2024), sehingga pendekatan serupa relevan diterapkan pada penanganan kasus kekerasan terhadap perempuan dan anak.

Beberapa penelitian terdahulu telah menerapkan arsitektur microservices pada sistem informasi pemerintahan maupun layanan publik dan melaporkan keunggulannya dalam hal skalabilitas serta fleksibilitas pengembangan (Calderón-Gómez et al., 2021; Tjahyono et al., 2022). Namun penelitian pembanding menunjukkan bahwa keunggulan tersebut bersifat kontekstual dan bergantung pada karakteristik beban kerja serta organisasi penggunanya, sehingga tidak berlaku secara umum (Blinowski et al., 2022; Dirgantara et al., 2024). Sementara itu, sistem informasi penanganan KtP/A yang telah dibangun di beberapa daerah umumnya mengadopsi arsitektur monolitik dan belum menyertakan mekanisme sinkronisasi status kasus antarlayanan (Adi et al., 2023; Hasan et al., 2026; Putri & Ritonga, 2024; Rumondor et al., 2025).

Sejauh penelusuran literatur yang dilakukan, penelitian yang menerapkan arsitektur microservices dengan message broker untuk melakukan decoupling status kasus KtP/A antara layanan pelaporan dan layanan penanganan belum ditemukan, meskipun pola komunikasi asinkron melalui message broker telah banyak diterapkan pada domain lain (Karabey Aksakalli et al., 2021; Imanuel et al., 2024). Yang juga belum ditemukan adalah penilaian atas kesepadanan arsitektur tersebut pada instansi pemerintah daerah berskala kecil, yaitu apakah kompleksitas yang dituntutnya sebanding dengan kebutuhan operasional yang sebenarnya. Kesenjangan kedua inilah yang mendasari penelitian ini.

Penelitian ini bertujuan membangun sistem pelaporan dan manajemen kasus KtP/A berbasis web dengan arsitektur microservices yang menerapkan message broker untuk melakukan decoupling status kasus antara layanan pelaporan dan layanan penanganan, kemudian mengevaluasi hasilnya secara empiris melalui pengujian beban terhadap pembanding monolitik serta pengujian ketahanan dengan penyuntikan kegagalan. Evaluasi tersebut diarahkan untuk menilai apakah kompleksitas arsitektur yang diterapkan sepadan dengan karakteristik organisasi penggunanya.

2. Metode Penelitian

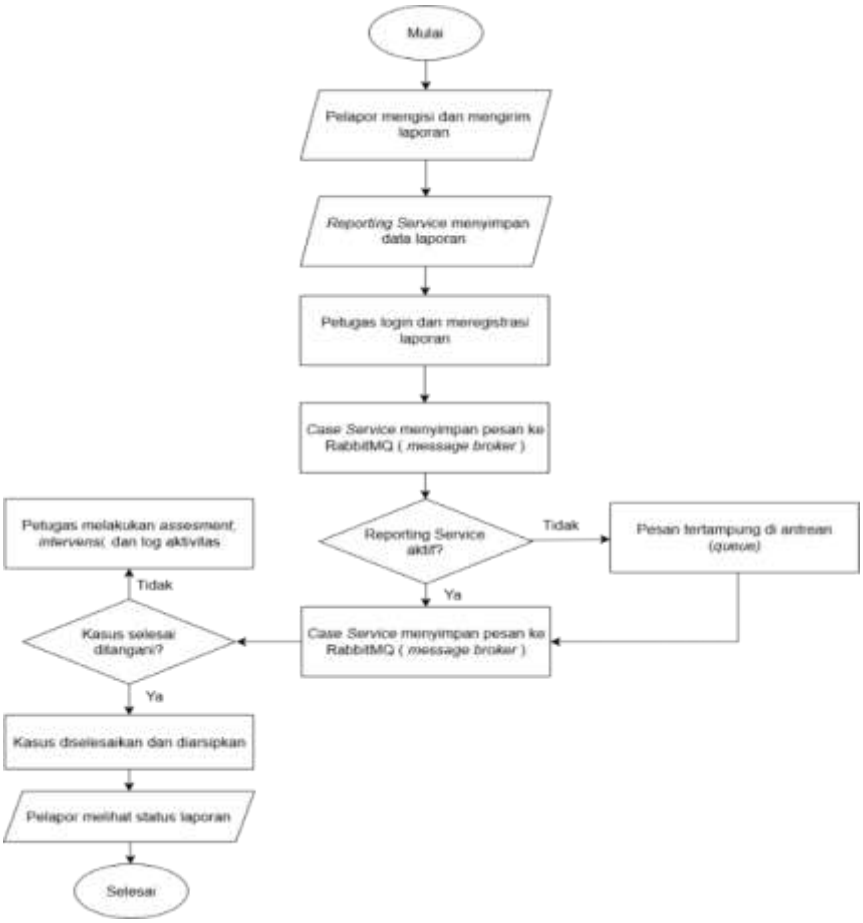
2.1. Desain Penelitian

2.1.1. Alur Sistem

Alur kerja sistem mengikuti tahapan penanganan kasus yang berlaku di UPTD PPA Kota Kendari, sebagaimana ditunjukkan pada Gambar 1. Masyarakat mengirimkan laporan melalui antarmuka publik tanpa perlu membuat akun, lalu layanan pelaporan menyimpannya dengan status menunggu registrasi. Petugas yang telah masuk ke sistem meregistrasi laporan tersebut sehingga terbentuk satu berkas kasus, kemudian menjalankan assessment, memulai intervensi layanan, dan mencatat log pendampingan sampai seluruh kebutuhan korban terpenuhi. Kasus lalu ditutup dan diarsipkan, sedangkan pelapor dapat memeriksa perkembangan laporannya melalui antarmuka publik menggunakan kode laporan yang diterima saat pengiriman.

Perubahan status ditangani melalui pola yang sama pada tiga titik, yaitu saat registrasi laporan, saat intervensi dimulai, dan saat kasus dinyatakan selesai. Layanan manajemen kasus menyimpan perubahan ke basis datanya sendiri terlebih dahulu, lalu menerbitkan pesan ke antrean pada message broker, dan layanan pelaporan mengonsumsi pesan itu untuk memperbarui status laporan. Karena pesan tertahan di antrean, pembaruan tetap tersimpan meskipun layanan pelaporan sedang tidak berjalan. Assessment dan pencatatan log tidak termasuk di dalamnya karena hanya memperbarui data kasus tanpa mengubah status laporan.

DOI: https://doi.org/10.69693/ijmst.v4i3.13707



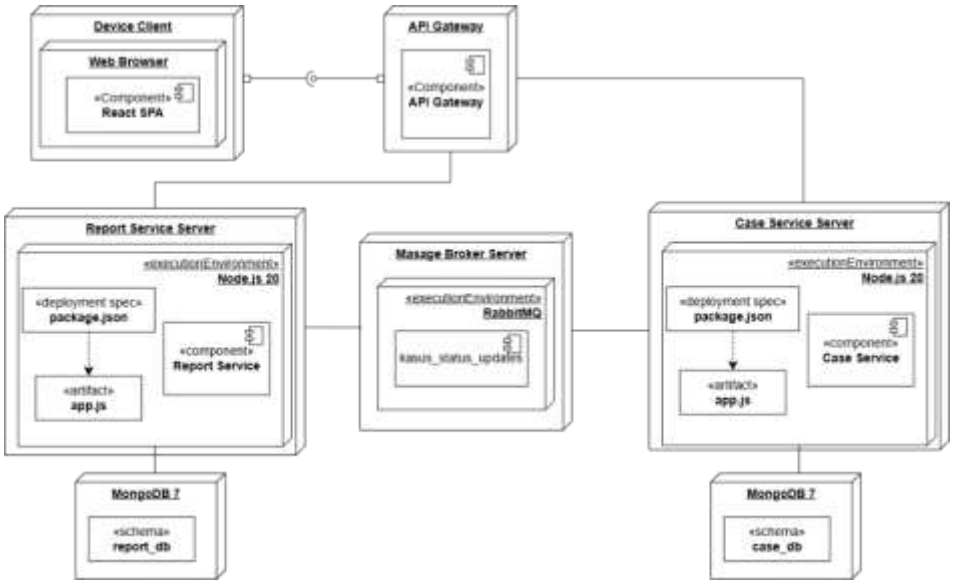
Gambar 1. Alur Penanganan Laporan pada Sistem SIPEKA

2.1.2. Arsitektur *Microservices* dan *Message Broker*

Sistem dibagi menjadi dua layanan mandiri berdasarkan dekomposisi menurut kapabilitas bisnis, yaitu pendekatan yang lazim dipakai untuk menentukan batas layanan pada arsitektur microservices (Sangabriel-Alarcón et al., 2024). Pembagiannya terdiri atas layanan pelaporan yang menangani penerimaan dan penyajian laporan masyarakat, serta layanan manajemen kasus yang menangani registrasi hingga penutupan kasus sekaligus pengelolaan akun petugas. Autentikasi tidak dijadikan layanan tersendiri karena merupakan kebutuhan teknis lintas layanan, bukan kapabilitas bisnis yang dimiliki satu unit kerja. Setiap layanan memiliki basis data MongoDB sendiri, dibangun menggunakan Node.js dengan framework Express, dan dijalankan pada kontainer Docker yang terpisah (Riyanto et al., 2022). Seluruh permintaan dari antarmuka React melewati nginx yang berperan murni sebagai API Gateway, sedangkan verifikasi JSON Web Token dilakukan secara lokal pada masing-masing layanan.

Komunikasi perubahan status antarlayanan dialihkan dari pemanggilan HTTP sinkron menjadi pertukaran pesan asinkron melalui RabbitMQ, yang pada pengujian terdahulu menunjukkan kestabilan lebih baik dibandingkan komunikasi berbasis REST ketika jumlah permintaan meningkat (Aprianto et al., 2024). Layanan manajemen kasus berperan sebagai produser dan layanan pelaporan sebagai konsumen, dengan antrean kasus_status_updates yang bersifat durable dan pesan yang ditandai persistent, sehingga pesan tetap tersimpan ketika konsumen tidak berjalan dan diproses kembali setelah layanan tersebut pulih. Penyebaran seluruh komponen pada lingkungan penerapan ditunjukkan pada Gambar 2, mencakup peramban klien, API Gateway, kedua server layanan yang berjalan pada lingkungan eksekusi Node.js 20 beserta basis datanya masing-masing, dan server message broker.

DOI: https://doi.org/10.69693/ijmst.v4i3.13707



Gambar 2. *Diagram Deployment* Sistem SIPEKA

2.1.3. Arsitektur Monolitik sebagai Pembanding

Untuk memperoleh pembanding yang sah, dibangun pula versi monolitik dari sistem yang sama. Versi ini berkedudukan sebagai instrumen pengujian, bukan sebagai produk kedua dari penelitian. Seluruh controller dan model diambil utuh dari basis kode kedua layanan microservices sehingga yang berbeda hanya mekanisme komunikasi antarmodul. Pemanggilan lintas layanan yang semula melewati message broker digantikan pemanggilan fungsi langsung di dalam satu proses Express, dan seluruh koleksi data ditempatkan pada satu basis data. Dengan cara ini, selisih hasil pengukuran tidak dapat dikaitkan dengan perbedaan kualitas implementasi, melainkan hanya dengan perbedaan arsitekturnya.

Batas sumber daya setiap kontainer aplikasi disamakan pada 0,5 CPU dan 512 MB memori mengikuti pola penelitian rujukan. Konsekuensinya, total sumber daya yang diterima kedua arsitektur tidak sama karena arsitektur microservices menjalankan dua kontainer aplikasi sekaligus komponen pendukung tambahan. Perbedaan ini tidak dinetralkan, melainkan dilaporkan apa adanya sebagaimana disajikan pada Tabel 1, karena kebutuhan sumber daya yang lebih besar merupakan biaya melekat dari arsitektur microservices dan justru termasuk hal yang sedang dievaluasi dalam penelitian ini.

Tabel 1. Perbandingan Footprint Sumber Daya Kedua Arsitektur

Arsitektur	Kontainer aplikasi	Total batas CPU	Total batas memori	Komponen pendukung
Microservices	2	1,0	1024 MB	nginx, RabbitMQ, MongoDB
Monolitik	1	0,5	512 MB	MongoDB

2.2. Metode Pengumpulan Data

Pengumpulan data dilakukan melalui wawancara dengan Kepala UPTD PPA Kota Kendari selaku penanggung jawab operasional penanganan kasus. Wawancara diarahkan untuk menggali kebutuhan fungsional sistem sekaligus karakteristik organisasi yang menjadi dasar penilaian kesesuaian arsitektur, meliputi jumlah dan pembagian kerja petugas, volume laporan yang masuk, pola lonjakan beban, serta kebutuhan integrasi dengan sistem eksternal. Hasil wawancara menunjukkan bahwa penanganan dijalankan oleh satu tim yang sama tanpa pembagian bidang, dengan volume rata-rata sekitar sepuluh laporan per bulan dan tanpa riwayat lonjakan laporan serentak.

Studi pustaka dilakukan terhadap penelitian terdahulu mengenai penerapan arsitektur microservices pada sistem informasi pemerintahan maupun layanan kesehatan (Calderón-Gómez et al., 2021; Tjahyono et al., 2022), perbandingan performa antara arsitektur monolitik dan microservices (Blinowski et al., 2022; Dirgantara et al.,

DOI: https://doi.org/10.69693/ijmst.v4i3.13707

2024), serta sistem informasi penanganan kasus KtP/A yang telah dibangun di daerah lain (Adi et al., 2023; Balahadia et al., 2022; Rumondor et al., 2025).

Data yang dipakai pada pengujian beban merupakan data sintetis, bukan data kasus sebenarnya, karena data kasus KtP/A bersifat rahasia dan volumenya jauh di bawah beban uji yang diperlukan. Sebelum setiap kali pengujian dijalankan, basis data kedua arsitektur disiapkan ulang dengan komposisi yang identik, yaitu 500 dokumen kasus sebagai beban baca, 500 laporan sebagai data latar, dan 60.000 laporan berstatus menunggu registrasi sebagai sasaran skenario tulis lintas layanan. Penyiapan ulang ini diperlukan karena skenario registrasi menambah dokumen kasus baru pada setiap pengujian, sehingga volume data yang dibaca akan berbeda pada pengujian berikutnya apabila sisa data dibiarkan.

2.3. Tahapan Pengembangan Sistem

Pengembangan sistem mengikuti metode Rational Unified Process yang terdiri atas empat fase. Fase inception menyusun ruang lingkup dan kebutuhan awal sistem berdasarkan kendala koordinasi status kasus yang teridentifikasi dari hasil pengumpulan data. Fase elaboration merancang arsitektur yang membagi fungsi utama ke dalam layanan pelaporan dan layanan manajemen kasus, sekaligus menetapkan pola publikasi dan konsumsi pesan melalui message broker sebagai mekanisme penyelarasan status antarlayanan.

Fase construction mengimplementasikan rancangan tersebut menggunakan Node.js dengan framework Express untuk kedua layanan backend, React untuk antarmuka pengguna, dan MongoDB sebagai basis data, dengan setiap komponen dijalankan pada kontainer Docker yang terpisah. Pada fase ini pula dibangun versi monolitik yang berfungsi sebagai pembanding pengujian. Fase transition menjalankan pengujian beban dan pengujian ketahanan melalui penyuntikan kegagalan, serta menyiapkan penerapan sistem di lingkungan UPTD PPA Kota Kendari.

2.4. Skenario Pengujian

Pengujian dijalankan pada satu lingkungan perangkat keras yang sama, yaitu komputer jinjing dengan prosesor Intel Core i5-10300H berkecepatan 2,50 GHz yang memiliki empat inti dan delapan thread, memori 8 GB, dan sistem operasi Windows 11 build 26200. Seluruh kontainer dijalankan menggunakan Docker Engine 28.2.2 dengan Docker Compose 2.37.1 di atas WSL2 berkernel 6.6.87.2, yang memperoleh alokasi memori sebesar 3,74 GB dari total memori mesin. Karena mesin yang sama sekaligus menjalankan seluruh kontainer dan perangkat penguji beban, terdapat konsekuensi terhadap penafsiran hasil yang dibahas pada bagian 3.3..

2.4.1. Pengujian Beban

Pengujian beban dilakukan menggunakan Apache JMeter terhadap lima endpoint yang dipilih berdasarkan perbedaan karakter bebannya, sebagaimana disajikan pada Tabel 2. Pemilihan ini mempertimbangkan bahwa pengujian yang hanya memuat endpoint layanan tunggal tidak akan pernah melewati jalur komunikasi antarlayanan, padahal jalur itulah yang membedakan kedua arsitektur. Karena itu ditambahkan satu endpoint tulis lintas layanan yang melewati message broker dan satu endpoint publik paling ringan yang berfungsi mengisolasi biaya lompatan melalui API Gateway (Setiawan & Enda, 2025).

Tabel 2. Endpoint yang Diuji pada Pengujian Beban

No	Endpoint	Metode	Karakter beban
1	/api/master/kekerasan	GET	Baseline paling ringan dan bersifat publik; yang terukur praktis biaya lompatan melalui API Gateway
2	/api/auth/login	POST	Terikat CPU karena verifikasi bcrypt dengan cost 10
3	/api/laporan	POST	Tulis publik, merupakan jalur utama pelapor
4	/api/penanganan	GET	Baca terproteksi JWT yang membaca seluruh koleksi tanpa paginasi
5	/api/penanganan/registrasi	POST	Tulis lintas layanan; satu-satunya jalur yang melewati message broker

Setiap endpoint diuji pada tiga tingkat beban, yaitu 50, 100, dan 150 virtual user. Pengujian disusun berbasis durasi, bukan berbasis jumlah iterasi tetap: setiap endpoint dibebani selama 35 detik dengan waktu ramp-up 5 detik, dan setiap virtual user mengulang permintaan tanpa henti sampai durasi tersebut habis. Pengaturan ini dipilih karena dengan jumlah iterasi tetap, endpoint yang ringan selesai sebelum seluruh thread sempat menyala sehingga

DOI: https://doi.org/10.69693/ijmst.v4i3.13707

tingkat konkurensi yang diuji tidak pernah mendekati target. Konsekuensinya, jumlah sampel per endpoint merupakan hasil pengukuran dan bukan angka yang ditetapkan di awal.

Pengujian pada arsitektur microservices diarahkan melalui API Gateway, sedangkan pengujian pada arsitektur monolitik diarahkan langsung ke porta layanannya. Endpoint keempat sengaja dijalankan sebelum endpoint kelima karena registrasi menambah dokumen kasus baru yang akan mengubah volume data yang dibaca endpoint keempat.

Endpoint analisis kronologi dikecualikan dari pengujian meskipun tersedia pada sistem. Endpoint tersebut memanggil model bahasa eksternal dengan batas waktu 45 detik dan dibatasi sepuluh permintaan per jendela waktu, sehingga yang akan terukur adalah performa layanan eksternal beserta penolakan buatan dari pembatas laju, bukan performa arsitektur sistem yang sedang dibandingkan.

2.4.2. Pengujian Ketahanan melalui Penyuntikan Kegagalan

Pengujian ketahanan dilakukan dengan menghentikan layanan pelaporan secara sengaja, kemudian mengirimkan 200 perubahan status selagi layanan tersebut tidak berjalan. Durasi penghentian tidak ditetapkan sebagai angka tetap, melainkan berlangsung sepanjang proses pengiriman. Setelah seluruh perubahan terkirim, layanan pelaporan dihidupkan kembali dan diamati berapa banyak pesan yang tertahan pada antrean, berapa banyak yang akhirnya diterapkan pada data laporan, serta berapa lama waktu yang dibutuhkan sejak perintah menghidupkan layanan hingga seluruh antrean habis diproses.

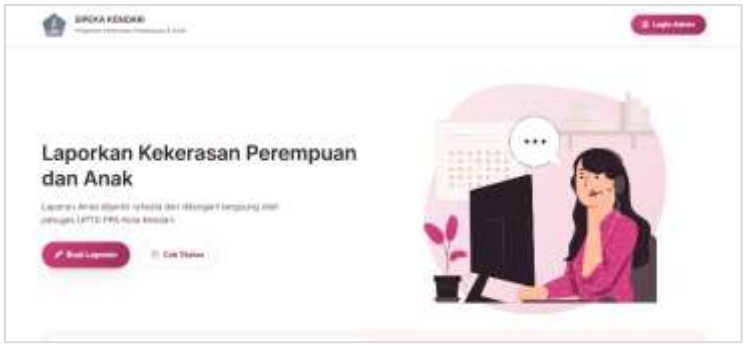
Skenario yang sama dijalankan pada kondisi pembanding, yaitu ketika perubahan status dikirim melalui pemanggilan HTTP sinkron tanpa message broker. Jalur tersebut sudah tidak ada lagi pada layanan manajemen kasus karena telah digantikan mekanisme antrean, sehingga kondisi pembanding direproduksi oleh skrip penguji yang memanggil endpoint pembaruan status pada layanan pelaporan secara langsung, dengan metode, muatan, dan penanganan kegagalan yang sama seperti rancangan lama. Panggilan tersebut melewati API Gateway karena porta layanan pelaporan tidak dipublikasikan ke luar jaringan kontainer; perbedaannya hanya pada bentuk galat yang diterima, sedangkan akibatnya sama, yaitu permintaan gagal tanpa mekanisme pengulangan.

Perbandingan kedua kondisi tersebut menjadi dasar penilaian sejauh mana penggunaan message broker memengaruhi keandalan penyelarasan status antarlayanan. Pengujian ini hanya berlaku pada arsitektur microservices, karena arsitektur monolitik berjalan dalam satu proses dan tidak memiliki mode kegagalan antarlayanan yang independen.

3. Hasil dan Pembahasan

3.1. Implementasi Antarmuka Sistem

Sistem yang dibangun diberi nama SIPEKA dan memiliki dua kelompok antarmuka, yaitu antarmuka publik untuk masyarakat pelapor dan antarmuka internal untuk petugas UPTD PPA. Antarmuka publik dapat diakses tanpa proses masuk sehingga pelapor tidak perlu membuat akun terlebih dahulu, sebagaimana ditunjukkan pada Gambar 3. Halaman ini memuat formulir pelaporan beserta fasilitas penelusuran status laporan menggunakan kode laporan yang diberikan sistem setelah laporan terkirim.



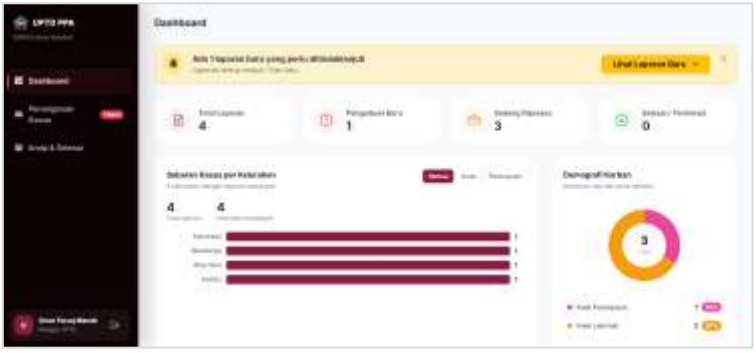
Gambar 3. Halaman Pelaporan Publik

DOI: https://doi.org/10.69693/ijmst.v4i3.13707

Antarmuka internal dilindungi oleh proses masuk berbasis peran seperti pada Gambar 4. Sistem membedakan dua peran, yaitu petugas layanan yang menangani kasus sehari-hari dan super admin yang mengelola akun pengguna serta data master. Token yang diterbitkan pada proses masuk diverifikasi kembali secara lokal oleh setiap layanan pada setiap permintaan berikutnya.

Gambar 4. Halaman Masuk Petugas

Dasbor petugas pada Gambar 5 menampilkan daftar laporan yang menunggu registrasi beserta kasus yang sedang ditangani, lengkap dengan status masing-masing. Melalui halaman ini petugas menjalankan registrasi laporan, mencatat hasil assessment, memulai intervensi, menambahkan log perkembangan, hingga menutup kasus. Setiap perubahan status yang dilakukan pada halaman ini memicu penerbitan pesan ke message broker.



Gambar 5. Dasbor Petugas UPTD PPA

Dasbor super admin pada Gambar 6 menyajikan rekapitulasi jumlah laporan dan kasus, statistik kinerja penanganan, serta fasilitas ekspor data. Fasilitas ekspor disediakan untuk membantu petugas menyiapkan data yang masih harus dimasukkan secara manual ke sistem pelaporan nasional, karena sistem nasional tersebut belum menyediakan antarmuka pemrograman yang memungkinkan pengiriman data secara otomatis.

Gambar 6. Dasbor Super Admin

DOI: https://doi.org/10.69693/ijmst.v4i3.13707

3.2. Implementasi Mekanisme Produser dan Konsumen

Mekanisme penyelarasan status antarlayanan diimplementasikan melalui satu antrean durable bernama kasus_status_updates. Sisi produser berada pada layanan manajemen kasus dan dijalankan setelah perubahan data kasus berhasil disimpan, bukan sebelumnya, agar data kasus yang merupakan sumber kebenaran tetap aman apabila penerbitan pesan gagal. Realisasi kedua sisi mekanisme tersebut ditunjukkan pada Gambar 7 dan Gambar 8.

```
async function connect() {
  if (channel) return channel;

  const connection = await amqp.connect(RABBITMQ_URL);
  connection.on('close', () => { channel = null; });
  connection.on('error', () => { channel = null; });

  channel = await connection.createChannel();
  await channel.assertQueue(QUEUE_NAME, { durable: true });

  return channel;
}

// Publish perubahan status kasus agar report-service menyinkronkan status laporan.
// Dipakai sebagai pengganti axios.patch langsung ke report-service (lihat kasusController.js):
// kalau report-service/RabbitMQ sedang down, pesan tetap tertampung di queue durable
// sampai report-service kembali hidup dan memprosesnya.
async function publishStatusUpdate({ laporan_id, status, catatan }) {
  const ch = await connect();
  const payload = Buffer.from(JSON.stringify({ laporan_id, status, catatan }));
  ch.sendToQueue(QUEUE_NAME, payload, { persistent: true });
}
```

Gambar 7. Penerbitan Pesan Perubahan Status pada Layanan Manajemen Kasus

Fungsi penerbitan pesan membuka kanal ke broker hanya sekali dan memakainya kembali untuk penerbitan berikutnya. Antrean dideklarasikan dengan atribut durable sehingga tetap ada setelah broker dimulai ulang, sedangkan setiap pesan dikirim dengan penanda persistent sehingga isinya ditulis ke penyimpanan dan tidak hanya berada di memori. Kegagalan penerbitan ditangani di sisi pemanggil dengan mencatatnya pada log tanpa membatalkan perubahan data kasus yang telah tersimpan.

Sisi konsumen berada pada layanan pelaporan dan dijalankan sebagai proses yang terus mendengarkan antrean. Konsumen mendeklarasikan antrean yang sama lalu menetapkan prefetch bernilai satu sehingga hanya satu pesan diproses pada satu waktu. Konfirmasi penerimaan dikirim secara manual setelah pembaruan status laporan benar-benar berhasil. Apabila terjadi galat basis data, pesan dikembalikan ke antrean untuk diproses ulang, sedangkan pesan yang isinya tidak dapat diurai dikonfirmasi dan dibuang agar tidak menyumbat antrean. Konsumen juga menghubungkan diri kembali secara berkala ketika koneksi ke broker terputus.

```
reporting-service/src/queue/consumer.js
async function startConsumer() {
  let connection;
  try {
    connection = await amqp.connect(RABBITMQ_URL);
  } catch (err) {
    console.error('Gagal konek RabbitMQ, retry dalam 5 detik:', err.message);
    await sleep(RETRY_DELAY_MS);
    return startConsumer();
  }

  connection.on('close', () => {
    console.error('[x] RabbitMQ terputus, mencoba reconnect...');
    setTimeout(startConsumer, RETRY_DELAY_MS);
  });
  connection.on('error', () => {});

  const channel = await connection.createChannel();
  await channel.assertQueue(QUEUE_NAME, { durable: true });
  channel.prefetch(1);

  console.log(`Menunggu pesan di queue "${QUEUE_NAME}"...`);

  channel.consume(QUEUE_NAME, async (msg) => {
    if (!msg) return;

    let payload;
    try {
      payload = JSON.parse(msg.content.toString());
    } catch (parseErr) {
      console.error('Pesan tidak valid, dibuang:', parseErr.message);
      return channel.ack(msg);
    }
```

Gambar 8. Konsumsi Pesan Perubahan Status pada Layanan Pelaporan

DOI: https://doi.org/10.69693/ijmst.v4i3.13707

Perbedaan mendasar dibandingkan rancangan sebelumnya terletak pada perlakuan terhadap kegagalan. Pada rancangan lama, perubahan status dikirim melalui pemanggilan HTTP sinkron tanpa mekanisme pengulangan, sehingga kegagalan pengiriman menyebabkan pembaruan status hilang secara permanen dan hanya tercatat pada berkas log. Pada rancangan baru, kegagalan pada sisi konsumen tidak menghilangkan pesan karena pesan tetap tertahan pada antrean hingga konsumen kembali berjalan.

3.3. Hasil Pengujian Beban Perbandingan Arsitektur

Pengujian beban menghasilkan tiga besaran yang dibandingkan antararsitektur, yaitu rata-rata waktu respons, throughput, dan tingkat kesalahan. Keterangan MS pada Tabel 3 sampai Tabel 5 menunjukkan arsitektur microservices dan Mono menunjukkan arsitektur monolitik. Rata-rata waktu respons setiap endpoint pada ketiga tingkat beban disajikan pada Tabel 3.

Tabel 3. Rata-rata Waktu Respons Kedua Arsitektur (milidetik)

Endpoint	MS 50 VU	Mono 50 VU	MS 100 VU	Mono 100 VU	MS 150 VU	Mono 150 VU
/api/master/kekerasan	251,3	264,2	681,0	477,3	742,4	752,7
/api/auth/login	7086,1	7112,2	13557,2	12646,2	17240,8	18707,2
/api/laporan	288,1	291,1	749,2	586,9	745,9	776,1
/api/penanganan	10446,4	8876,4	17046,0	16889,8	23405,3	23589,4
/api/penanganan/registrasi	559,7	645,8	1200,1	1331,9	1479,1	2324,4

Endpoint masuk dan endpoint pembacaan data penanganan mencatat waktu respons yang jauh lebih tinggi dibandingkan endpoint lainnya pada kedua arsitektur. Keduanya berada dalam kondisi jenuh: endpoint masuk terbatas oleh biaya komputasi bcrypt sehingga throughput tertahan di sekitar enam permintaan per detik, sedangkan endpoint pembacaan data penanganan membaca seluruh koleksi tanpa paginasi sehingga throughput tertahan di sekitar empat sampai lima permintaan per detik. Rata-rata pada kedua endpoint tersebut juga menyembunyikan sebaran yang sangat lebar; sebagai contoh, endpoint pembacaan data penanganan pada beban 150 virtual user memiliki rata-rata 23405,3 milidetik dengan persentil ke-95 mencapai 58683 milidetik pada arsitektur microservices dan 60007 milidetik pada arsitektur monolitik. Angka pada kedua endpoint ini karena itu mencerminkan batas implementasi, bukan perbedaan arsitektur.

Throughput yang dicatat pada pengujian yang sama disajikan pada Tabel 4, sedangkan tingkat kesalahan disajikan pada Tabel 5.

Tabel 4. Throughput Kedua Arsitektur (permintaan per detik)

Endpoint	MS 50 VU	Mono 50 VU	MS 100 VU	Mono 100 VU	MS 150 VU	Mono 150 VU
/api/master/kekerasan	183,92	175,57	135,17	193,79	171,99	183,91
/api/auth/login	6,21	6,22	5,76	6,30	6,17	5,95
/api/laporan	161,09	159,20	122,93	157,80	171,48	178,06
/api/penanganan	4,05	4,67	4,66	4,59	4,39	4,91
/api/penanganan/registrasi	82,67	71,82	76,83	69,28	86,11	59,32

Tabel 5. Tingkat Kesalahan Kedua Arsitektur (persen)

Endpoint	MS 50 VU	Mono 50 VU	MS 100 VU	Mono 100 VU	MS 150 VU	Mono 150 VU
/api/master/kekerasan	0,00	0,00	0,00	0,00	0,00	0,00
/api/auth/login	0,00	0,00	0,00	0,00	0,00	0,00
/api/laporan	0,11	0,07	0,07	0,20	0,15	0,12
/api/penanganan	0,00	0,00	0,00	0,00	1,95	7,21
/api/penanganan/registrasi	0,00	0,00	0,07	0,00	0,00	0,00

Tingkat kesalahan bernilai nol pada hampir seluruh kombinasi. Satu-satunya selisih yang berarti muncul pada endpoint pembacaan data penanganan di beban 150 virtual user, yaitu 1,95 persen pada arsitektur microservices

DOI: https://doi.org/10.69693/ijmst.v4i3.13707

berbanding 7,21 persen pada arsitektur monolitik. Seluruh kegagalan tersebut berupa habisnya batas waktu soket dan hanya muncul ketika endpoint telah berada dalam kondisi jenuh. Kesalahan kecil pada endpoint pelaporan publik berupa penolakan validasi dan tidak berkaitan dengan arsitektur.

Seluruh pengujian dijalankan pada satu mesin yang sekaligus menjalankan kontainer Docker dan perangkat penguji beban, sehingga terdapat ragam antar-run yang tidak berkaitan dengan arsitektur. Besarnya ragam tersebut dapat diukur karena konfigurasi microservices pada beban 100 virtual user sempat dijalankan dua kali, sebagaimana disajikan pada Tabel 6. Pola serupa juga terlihat pada throughput endpoint data master yang tidak berubah secara monoton seiring naiknya beban pada kedua arsitektur.

Tabel 6. Selisih Dua Kali Pengukuran pada Konfigurasi yang Sama (Microservices, 100 Virtual User)

Endpoint	Pengukuran 1 (ms)	Pengukuran 2 (ms)	Selisih relatif (%)
/api/master/kekerasan	566,4	681,0	20,2
/api/auth/login	12908,2	13557,2	5,0
/api/laporan	581,9	749,2	28,7
/api/penanganan	16068,0	17046,0	6,1

Karena ragam pada konfigurasi yang identik mencapai 28,7 persen, penafsiran hasil dibatasi pada selisih yang arahnya konsisten di seluruh tingkat beban, bukan pada besar selisih di satu titik beban saja. Dengan kriteria tersebut, endpoint data master, endpoint masuk, endpoint pelaporan publik, dan endpoint pembacaan data penanganan tidak menunjukkan perbedaan arsitektur yang dapat disimpulkan, karena arah keunggulannya berganti-ganti antartingkat beban. Satu pengecualian perlu dicatat secara terbuka: pada beban 100 virtual user, endpoint data master mencatat waktu respons 42,7 persen lebih lambat pada arsitektur microservices, dan selisih itu melampaui ragam yang teramati. Penyebabnya belum dapat dipastikan karena konfigurasi monolitik pada beban tersebut hanya diukur satu kali, sehingga belum dapat dibedakan apakah arsitektur microservices yang melemah atau pengukuran monolitik yang kebetulan luar biasa cepat.

Satu-satunya perbedaan yang memenuhi kriteria konsistensi adalah endpoint registrasi, yaitu jalur tulis lintas layanan yang melewati message broker. Arsitektur monolitik mencatat waktu respons 15,4 persen lebih lambat pada beban 50 virtual user, 11,0 persen lebih lambat pada 100 virtual user, dan 57,2 persen lebih lambat pada 150 virtual user, sedangkan throughput arsitektur microservices lebih tinggi 15,1 persen, 10,9 persen, dan 45,2 persen pada ketiga tingkat beban tersebut. Enam dari enam perbandingan mengarah ke sisi yang sama.

Keunggulan tersebut perlu dibaca dengan hati-hati. Pada arsitektur microservices, permintaan registrasi dinyatakan selesai segera setelah pesan berhasil diterbitkan ke antrean, sedangkan pembaruan status laporan dikerjakan konsumen setelahnya. Waktu respons yang lebih rendah pada jalur ini karena itu tidak menunjukkan pekerjaan yang lebih ringan, melainkan pekerjaan yang ditunda dengan konsekuensi konsistensi yang bersifat eventual. Arsitektur monolitik menanggung seluruh biaya penulisan tersebut di dalam permintaan yang sama.

Temuan pada bagian ini sejalan dengan Blinowski et al. (2022) dan Dirgantara et al. (2024) yang menyimpulkan bahwa keunggulan antararsitektur bersifat kontekstual dan tidak berlaku menyeluruh pada semua jenis beban. Yang perlu ditegaskan adalah asimetri sumber daya pada pengujian ini: arsitektur microservices memperoleh total batas CPU dua kali lipat dibandingkan arsitektur monolitik sebagaimana tercantum pada Tabel 1, dan menjalankan dua komponen pendukung tambahan berupa API Gateway dan message broker. Dengan separuh jatah CPU dan tanpa komponen pendukung tersebut, arsitektur monolitik tetap mencapai hasil yang setara pada empat dari lima endpoint.

3.4. Hasil Pengujian Ketahanan Penyelarasan Status

Pengujian penyuntikan kegagalan dijalankan dengan mematikan layanan pelaporan sementara layanan manajemen kasus tetap memproses perubahan status. Hasil pengujian pada kedua kondisi disajikan pada Tabel 7.

DOI: https://doi.org/10.69693/ijmst.v4i3.13707

Tabel 7. Hasil Penyelarasan Status saat Layanan Pelaporan Dimatikan

Kondisi pengujian	Perubahan status dikirim	Berhasil tersinkron	Hilang permanen	Keberhasilan (%)
Tanpa message broker (HTTP sinkron)	200	0	200	0,00
Dengan message broker (RabbitMQ)	200	200	0	100,00

Pada kondisi tanpa message broker, seluruh 200 perubahan status gagal seketika karena layanan tujuan tidak merespons, dan kegagalan tersebut tidak memiliki mekanisme pengulangan sehingga pembaruan hilang secara permanen. Pemeriksaan langsung ke basis data setelah layanan pulih memastikan seluruh laporan tetap berada pada status sebelumnya. Pada kondisi dengan message broker, seluruh 200 pesan tertahan pada antrean durable selama layanan konsumen tidak berjalan, kemudian diproses seluruhnya dalam waktu 23,71 detik terhitung sejak perintah menghidupkan layanan diberikan. Angka tersebut mencakup waktu penyalaan kontainer dan pembentukan ulang koneksi ke broker, bukan hanya waktu pemrosesan antrean. Seluruh 200 laporan berubah ke status yang dituju, sehingga tingkat keberhasilan penyelarasan mencapai 100 persen.

Hasil tersebut menunjukkan bahwa message broker memindahkan titik kritis komunikasi ke komponen yang memiliki mekanisme antrean dan pengulangan bawaan, bukan menghilangkan seluruh titik kegagalan sistem. Broker sendiri tetap merupakan komponen yang dapat mengalami gangguan, dan pada penelitian ini broker dijalankan sebagai instans tunggal tanpa klaster. Selain itu, perlindungan ini tidak berlaku apabila layanan produser berhenti sebelum sempat menerbitkan pesan. Risiko tersebut tetap terbatas karena perubahan data kasus telah disimpan lebih dahulu ke basis data, sehingga yang perlu diselaraskan ulang hanya status pada sisi laporan.

3.5. Kesesuaian Arsitektur dengan Karakteristik Organisasi

Ketika hasil pengujian dipertemukan dengan hasil wawancara, terlihat bahwa kompleksitas arsitektur yang diterapkan melampaui kebutuhan operasional UPTD PPA Kota Kendari pada kondisi saat ini. Alasan yang lazim dipakai untuk memilih microservices tidak ditemukan pada organisasi ini: penanganan dijalankan oleh satu tim yang sama tanpa pembagian bidang sehingga tidak ada batas kepemilikan yang menuntut layanan terpisah, volume laporan hanya sekitar sepuluh per bulan tanpa riwayat lonjakan serentak sehingga kebutuhan penskalaan mandiri tidak muncul, dan kebutuhan integrasi ke sistem pelaporan nasional tidak dapat diwujudkan secara otomatis karena sistem tersebut belum menyediakan antarmuka pemrograman.

Hasil pengujian beban menguatkan penilaian tersebut. Arsitektur monolitik mencapai performa yang setara pada empat dari lima endpoint meskipun hanya memperoleh separuh jatah CPU dan tidak menjalankan API Gateway maupun message broker. Keunggulan arsitektur microservices terbatas pada satu jalur tulis lintas layanan, dan keunggulan itu pun berasal dari penundaan pekerjaan ke konsumen, bukan dari beban kerja yang lebih ringan. Dengan kata lain, penambahan sumber daya dan komponen yang dituntut arsitektur ini tidak berbanding lurus dengan perbaikan performa yang diperoleh.

Temuan tersebut sejalan dengan kesimpulan Blinowski et al. (2022) bahwa pemilihan arsitektur bergantung pada karakteristik organisasi dan beban kerja, bukan pada keunggulan teknis yang berlaku umum. Perbedaannya, penelitian tersebut menarik kesimpulan dari lingkungan uji sintetis, sedangkan penelitian ini menyandingkan hasil pengukuran dengan data wawancara pada satu instansi nyata. Penelitian terdahulu yang menerapkan microservices pada sistem pelaporan pemerintahan (Tjahyono et al., 2022) juga menitikberatkan pembahasan pada perancangan dan penerapannya, tanpa menilai kembali apakah kompleksitas tersebut sepadan dengan karakteristik instansi penggunanya.

Nilai yang tetap dapat dipertahankan dari penerapan message broker terletak pada perbaikan keandalan penyelarasan status sebagaimana ditunjukkan Tabel 7, serta pada tingkat kesalahan yang lebih rendah ketika sistem berada dalam kondisi jenuh. Perbaikan tersebut nyata dan terukur, tetapi cakupannya terbatas pada mode kegagalan yang justru dimunculkan oleh pemisahan layanan itu sendiri. Pada arsitektur monolitik, mode kegagalan tersebut memang tidak ada sejak awal karena seluruh perubahan dikerjakan dalam satu proses.

Keterbatasan lain yang perlu diakui berkaitan dengan pemeliharaan pascapenelitian. UPTD PPA Kota Kendari tidak memiliki staf teknologi informasi internal dan bergantung pada pihak ketiga, sedangkan penerapan sistem ini menuntut pengelolaan beberapa kontainer sekaligus satu message broker. Kebutuhan kompetensi operasional

DOI: https://doi.org/10.69693/ijmst.v4i3.13707

tersebut merupakan biaya yang melekat pada arsitektur yang dipilih dan perlu diperhitungkan pada penerapan sesungguhnya. Keterbatasan pengujian juga perlu dinyatakan: setiap kombinasi arsitektur dan tingkat beban umumnya hanya diukur satu kali, sehingga sebagian selisih kecil tidak dapat dipisahkan dari ragam antar-run.

4. Kesimpulan

Penelitian ini berhasil membangun sistem pelaporan dan manajemen kasus kekerasan terhadap perempuan dan anak berbasis web dengan arsitektur microservices yang terdiri atas layanan pelaporan dan layanan manajemen kasus, dengan RabbitMQ sebagai message broker yang melakukan decoupling penyelarasan status antarlayanan. Pengujian beban terhadap lima endpoint pada tiga tingkat beban menunjukkan bahwa arsitektur monolitik mencapai performa setara pada empat dari lima endpoint meskipun hanya memperoleh separuh jatah CPU dan tanpa komponen API Gateway maupun message broker. Keunggulan arsitektur microservices yang konsisten pada seluruh tingkat beban hanya ditemukan pada jalur tulis lintas layanan, dengan waktu respons hingga 57,2 persen lebih rendah dan throughput hingga 45,2 persen lebih tinggi pada beban 150 virtual user, serta pada tingkat kesalahan saat sistem jenuh, yaitu 1,95 persen berbanding 7,21 persen. Pengujian penyuntikan kegagalan terhadap 200 perubahan status menunjukkan seluruhnya tertahan pada antrean saat layanan pelaporan dimatikan dan berhasil diterapkan setelah layanan pulih dengan tingkat keberhasilan 100 persen, sedangkan pada rancangan pembanding tanpa message broker seluruh perubahan tersebut hilang secara permanen dengan tingkat keberhasilan 0 persen. Hasil pengukuran tersebut, ketika disandingkan dengan karakteristik organisasi pengguna, menunjukkan bahwa kompleksitas arsitektur microservices melampaui kebutuhan operasional UPTD PPA Kota Kendari pada kondisi saat ini, karena tidak ditemukan pembagian bidang kerja, lonjakan beban, maupun kebutuhan integrasi otomatis yang menuntut pemisahan layanan. Keunggulan yang diperoleh terbatas pada ketahanan komunikasi antarlayanan, yaitu mode kegagalan yang justru dimunculkan oleh pemisahan layanan itu sendiri. Temuan ini menegaskan bahwa pemilihan arsitektur perlu didasarkan pada karakteristik organisasi dan beban kerja yang sesungguhnya, bukan semata pada keunggulan teknis arsitektur tersebut. Sistem yang dihasilkan tetap bermanfaat sebagai purwarupa yang menggantikan pertukaran status secara manual antartahap penanganan. Penelitian selanjutnya disarankan mengulang pengukuran beberapa kali pada setiap kombinasi agar ragam antar-run dapat dipisahkan dari perbedaan arsitektur, serta menguji sistem pada instansi dengan pembagian bidang kerja dan volume laporan yang lebih besar. Penerapan paginasi pada endpoint pembacaan data penanganan perlu didahulukan karena endpoint tersebut terbukti menjadi titik jenuh pada kedua arsitektur. Penerapan message broker sebagai klaster, penambahan mekanisme pemantauan antrean, serta integrasi otomatis ke sistem pelaporan nasional apabila antarmuka pemrogramannya telah tersedia juga merupakan arah pengembangan yang layak ditempuh.

Reference

1. Adi, D. A., Terttiavini, & Marcelina, D. (2023). Sistem informasi pelayanan pengaduan kekerasan terhadap perempuan dan anak berbasis web. Jurnal Ilmiah Binary, 5(2), 174–184. https://doi.org/10.52303/jb.v5i2.124
2. Aprianto, A. N., Girsang, A. S., Nugroho, Y., & Putra, W. K. (2024). Performance analysis of RabbitMQ and Nats Streaming for communication in microservice. Teknologi: Jurnal Ilmiah Sistem Informasi, 14(1), 37–47. https://doi.org/10.26594/teknologi.v14i1.4498
3. Balahadia, F. F., Astoveza, Z. J. M., Jamolin, G. R., & Astoveza, N. E. A. L. (2022). Development and implementation of violence against women and their children report system mobile application. International Journal of Science, Technology, Engineering and Mathematics, 2(3), 17–42. https://doi.org/10.53378/352906
4. Blinowski, G., Ojdowska, A., & Przybyłek, A. (2022). Monolithic vs. microservice architecture: A performance and scalability evaluation. IEEE Access, 10, 20357–20374. https://doi.org/10.1109/ACCESS.2022.3152803
5. Calderón-Gómez, H., Mendoza-Pittí, L., Vargas-Lombardo, M., Gómez-Pulido, J. M., Rodríguez-Puyol, D., Sención, G., & Polo-Luque, M. L. (2021). Evaluating service-oriented and microservice architecture patterns to deploy eHealth applications in cloud computing environment. Applied Sciences, 11(10), 4350. https://doi.org/10.3390/app11104350
6. Dirgantara, D. P., Kusumo, D. S., & Utomo, R. G. (2024). Docker-based monolithic and microservices architecture performance comparison. Jurnal Teknik Informatika (JUTIF), 5(2), 357–365. https://doi.org/10.52436/1.jutif.2024.5.2.1338
7. Hasan, P., Totombu, R. D., Lahallo, J., & Sutejo, H. (2026). Sistem pengelolaan pengaduan kekerasan pada perempuan dan anak berbasis website di UPTD PPA Kota Jayapura. Nusantara of Engineering (NOE), 9(1), 57–67. https://doi.org/10.29407/noe.v9i01.26568
8. Imanuel, D., Rini, M. N. A., & Susanto, B. (2024). Penerapan choreography message broker untuk transaksi data berbasis asynchronous RESTful. Jurnal Terapan Teknologi Informasi, 8(1), 53–68. https://doi.org/10.21460/jutei.2024.81.321
9. Karabey Aksakalli, I., Çelik, T., Can, A. B., & Tekinerdoğan, B. (2021). Deployment and communication patterns in microservice architectures: A systematic literature review. Journal of Systems and Software, 180, 111014. https://doi.org/10.1016/j.jss.2021.111014
10. Nirwana, N., Hayat, T. A., Nurhidaya, A., Pramono, B., & Isnawaty, I. (2025). Perancangan ulang sistem informasi Balai Penjaminan Mutu Pendidikan (Studi kasus: BPMP Sulawesi Tenggara). Jurnal Sistem Informasi dan Sistem Komputer (SIMKOM), 10(1), 18–30. https://doi.org/10.51717/simkom.v10i1.555
11. Nisa, F. H., Thamrin, M. B. A. A., Jaya, G., & Isnawaty, I. (2024). Rancang bangun sistem informasi persebaran lokasi industri manufaktur Sulawesi Tenggara berbasis website. Jurnal Sistem Informasi dan Sistem Komputer (SIMKOM), 9(1), 47–58. https://doi.org/10.51717/simkom.v9i1.357

DOI: https://doi.org/10.69693/ijmst.v4i3.13707

12. Putri, A. A., & Ritonga, F. U. (2024). Proses penanganan kasus kekerasan seksual pada anak berkebutuhan khusus di Unit Pelaksana Teknis Daerah Perlindungan Perempuan dan Anak (UPTD PPA) Kota Medan. Sosmaniora: Jurnal Ilmu Sosial dan Humaniora, 3(1), 15–30. https://doi.org/10.55123/sosmaniora.v3i1.3045
13. Riyanto, D. J., Pizaini, Safaat H., N., & Affandes, M. (2022). Implementasi service choreography pattern arsitektur microservice classroom akademik menggunakan Docker. JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika), 7(3), 768–779. https://doi.org/10.29100/jipi.v7i3.3126
14. Rumondor, A., Wensen, V. A., Budiman, M. J., & Kapoh, H. (2025). Case reporting system with tracking and chat machine learning for the Department of Women's Empowerment and Child Protection Manado. Journal La Multiapp, 6(3), 496–507. https://doi.org/10.37899/journallamultiapp.v6i3.278
15. Sangabriel-Alarcón, J., Ocharán-Hernández, J. O., Limón, X., & Cortés-Verdín, M. K. (2024). Domain-driven design in microservices-based systems development: A systematic literature review and thematic analysis. Programming and Computer Software, 50(8), 742–770. https://doi.org/10.1134/S0361768824700749
16. Setiawan, H., & Enda, D. (2025). Pengujian load testing website Jurusan Teknik Informatika Politeknik Negeri Bengkalis. JEKIN – Jurnal Teknik Informatika, 5(1), 1–12. https://doi.org/10.58794/jekin.v5i1.820
17. Tjahyono, A., Muslim, A., & Anggraini, D. (2022). Perancangan sistem informasi laporan kegiatan penanaman modal dengan menggunakan arsitektur microservices pada Kementerian Investasi/Badan Koordinasi Penanaman Modal. Indonesian Journal on Computing, 7(3), 33–52. https://doi.org/10.34818/INDOJC.2022.7.3.674

DOI: https://doi.org/10.69693/ijmst.v4i3.13707