Jurnal Infosecure Vol.2 No.2, Oktober 2021
p-ISSN: 2087-1023
MTI-UNLA

Perancangan Konsep Komunikasi Data Antar Aplikasi *Smart City* Dengan Bantuan Arsitektur *Microservices*, *Containerization*, *API Gateway* dan *CI/CD*

Pingadi Limajaya

Program Studi Magister Teknik Informatika, Universitas Langlangbuana

pingadi@gmail.com

Abstrak— Perkembangan *smart city* di Indonesia mendorong integrasi berbagai layanan digital pemerintah, namun menghadapi tantangan besar dalam hal komunikasi data antar sistem yang belum terstandarisasi. Ketidakterpaduan ini sering kali menyebabkan duplikasi data, keterlambatan layanan, dan inefisiensi operasional. Penelitian ini menyusun dan menguji model integrasi layanan untuk *smart city* dengan memanfaatkan arsitektur *microservices* dan pemanfaatan *API Gateway* sebagai solusi atas fragmentasi data dan tantangan interoperabilitas sistem.

Metode penelitian mencakup studi literatur, analisis kebutuhan sistem, perancangan arsitektur *microservices*, implementasi simulasi berbasis *Docker* dan *Kubernetes*, serta pengujian performa dan efisiensi. Evaluasi dilakukan dengan membandingkan model arsitektur konvensional dan arsitektur berbasis *API Gateway* menggunakan metrik seperti response time, success rate, penggunaan sumber daya, dan *code reuse ratio*.

Dari hasil simulasi, tingkat keberhasilan komunikasi meningkat dari 88% menjadi 97%, walaupun disertai penambahan rata-rata *response time* sebesar ±30 *ms* akibat proses autentikasi dan *routing* terpusat. Selain itu, terjadi peningkatan efisiensi pengembangan dengan reduksi fungsi redundan hingga 40% dan pengelolaan sumber daya yang lebih optimal melalui fitur *auto-scaling Kubernetes*.

Kesimpulannya, arsitektur integrasi sistem berbasis microservices dan *API Gateway* menawarkan solusi yang modular, efisien, dan scalable untuk mendukung pengembangan sistem *smart city* yang terintegrasi dan berkelanjutan.

Kata kunci— smart city, microservices, API Gateway, arsitektur modular

I. PENDAHULUAN

A. Latar Belakang

Dalam sebuah ekosistem *smart city* terdapat banyak sekali perangkat keras dan perangkat lunak yang saling terhubung satu sama lain baik secara fisik maupun secara digital. Masing-masing aplikasi akan mengumpulkan data dari berbagai sumber baik berupa perangkat keras maupun perangkat lunak, seperti sensor, basis data pemerintahan dan perusahaan (contoh: perusahaan jasa transportasi publik), *social media*, data yang diberikan oleh masyarakat melalui aplikasi-aplikasi, dlsb. [1]. Terlepas dari kenyataan bahwa semakin banyak kota yang menawarkan layanan baru dalam lingkup smart city, tantangan teknologi terbuka untuk mencapai layanan yang lebih baik dan lebih cepat [2], inisiatif saat ini telah gagal membangun lapisan interoperabilitas untuk komunikasi dan pemrosesan data karena tidak dibangun sesuai dengan standar saat ini yang terkait dengan domain ini [3], [4].

Banyaknya perangkat keras dan perangkat lunak yang terlibat dalam alur komunikasi *smart city* tentunya perlu diperhatikan kompleksitasnya. Selain itu tentunya *smart city* perlu didukung oleh sumber daya pendukung lainnya seperti sumber daya manusia dan lain sebagainya. Dikarenakan banyaknya lapisan dalam sebuah ekosistem *smart city* seperti infrastruktur, perangkat keras, perangkat lunak dan sumber daya pendukung lainnya tersebut, maka dapat dikatakan implementasi *smart city* itu mahal, sehingga diperlukan suatu konsep yang dapat membuat integrasi antar sistem lebih efisien dan tepat guna. *Smart city* bukanlah sistem yang berdiri sendiri melainkan sistem yang seharusnya saling terintegrasi sehingga dalam komunikasi data secara digital antar perangkat lunak.

Seperti yang sudah diketahui bersama, integrasi antar sistem itu sulit dan membutuhkan waktu yang tidak sedikit dan tentunya berkorelasi pada biaya yang tidak sedikit juga, sehingga diperlukan sebuah patokan atau standar atau *channel* komunikasi yang dapat digunakan oleh setiap perangkat lunak dalam ekosistem *smart city*. Masalah integrasi banyak aplikasi dalam ekosistem *smart city* dapat diatasi dengan pendekatan modular menggunakan konsep *microservices*, sehingga dapat mempercepat proses pengembangan. Diharapkan dengan penggunaan konsep *microservices* ini, pengembangan aplikasi dapat lebih cepat digunakan oleh pengguna di lapangan dan tidak ada fungsi redundan yang tidak disengaja dalam pembangunan aplikasi. Selain itu heterogenitas teknologi juga dimungkinkan dengan menggunakan konsep *microservices* ini. Sehingga tidak perlu memaksakan menggunakan teknologi standar yang sama untuk setiap organisasi atau institusi yang terkait dalam ekosistem *smart city*. Namun tentunya dalam masalah interkoneksi data antara satu *service* dengan *service* yang lain diperlukan jalur komunikasi dan format data yang terstandarisasi. Namun tentunya untuk membuat jalur komunikasi dan format data yang terstandarisasi tidaklah mudah sehingga diperlukan

sebuah *tools* yang memiliki fungsional *mapping request* dan *response* yang biasanya dimiliki oleh *API Gateway*.

Tujuan dari penggunaan fungsional *mapping request* dan *response* yang biasanya dimiliki oleh *API Gateway* agar setiap *services* yang ada yang tidak dibangun atau dikembangkan oleh *Project Management Officer (PMO)* atau *programmer* yang sama dapat tetap berinteraksi satu sama lain tanpa perlu mengetahui bagaimana masing-masing *programmer* mempersiapkan data karena tentunya dari data-data yang nantinya akan dikirimkan dan diterima dapat dilakukan *mapping* menggunakan fungsional tersebut sehingga lebih mudah dikembangkan dan dikelola di kemudian hari.

Sampai saat ini belum ada penelitian lain yang pernah dilakukan untuk menggunakan *Microservices* dan *API Gateway* secara bersamaan dalam pembangunan ekosistem *smart city*, secara spesifik di Indonesia untuk menjawab kebijakan Satu Data Indonesia dan *super apps* nasional yang belum dibahas secara komprehensif dalam literatur global. Meskipun penelitian seperti Xu dkk. (2019) [5] dan Zhao dkk. (2018) [6] telah mengkaji penggunaan *API Gateway* dan *microservices* dalam konteks *edge computing* dan manajemen *API*, namun penelitian ini memfokuskan pada kombinasi *microservices*, *API Gateway*, dan *containerization* secara simultan dalam konteks pengembangan *smart city* di Indonesia. Penelitian ini merupakan kontribusi teknis yang bersifat inkremental yang menggabungkan beberapa pendekatan yang sudah ada – *microservices*, *API Gateway*, dan *containerization* – ke dalam satu model integrasi yang terfokus pada kebutuhan *smart city* Indonesia. Dengan konsep penggunaan *Microservices* dan *API Gateway* dalam berkomunikasi antar sistem (aplikasi dan konten) dalam sebuah ekosistem *smart city*, diharapkan dapat mempermudah integrasi antar sistem, mempercepat pengembangan sistem dalam ekosistem *smart city*, serta dapat mengurangi biaya yang perlu dikeluarkan dalam pengembangan ekosistem *smart city*. Dengan demikian bukanlah hal yang tidak mungkin untuk mempercepat pengembangan ekosistem *smart city* di Indonesia untuk mendukung Gerakan Menuju 100 *Smart City* yang sudah dicanangkan dalam program Bersama antara Kementerian Komunikasi dan Informatika (KemKomInfo), Kementerian Dalam Negeri, Kementerian Pekerjaan Umum dan Perumahan Rakyat (PUPR), Badan Perencanaan Pembangunan Nasional (Bappenas), dan Kantor Staf Kepresidenan.

Berdasarkan hasil evaluasi dari program Gerakan Menuju 100 *Smart City* yang diselenggarakan oleh Kementerian Komunikasi dan Informatika (KemKomInfo) bersama Bappenas dan kementerian terkait lainnya, ditemukan beberapa kendala signifikan dalam integrasi data antar sistem di tingkat pemerintah daerah, antara lain:

1. Ketiadaan Standar Komunikasi Data
   Banyak daerah peserta belum memiliki standar komunikasi data yang seragam antar aplikasi dan layanan. Hal ini menyebabkan data tidak dapat dipertukarkan secara otomatis antar sistem, yang menghambat integrasi layanan publik.
2. Duplikasi Data dan Ketidakefisienan
   Beberapa kota mengalami duplikasi data akibat pengumpulan informasi yang sama oleh berbagai aplikasi tanpa koordinasi yang baik. Hal ini tidak hanya memboroskan sumber daya, tetapi juga menyebabkan inkonsistensi data yang dapat menurunkan kualitas layanan publik.
3. Kurangnya Kelembagaan Pengelola *Smart City*
   Sebagian besar daerah belum memiliki lembaga atau tim khusus yang bertanggung jawab atas pengelolaan dan integrasi sistem *smart city*. Ketiadaan struktur kelembagaan ini menyulitkan koordinasi antar unit kerja dan pengambilan keputusan yang cepat dan tepat.
4. Keterbatasan Infrastruktur Teknologi Informasi
   Infrastruktur TI yang belum memadai di beberapa daerah menghambat implementasi sistem yang terintegrasi. Keterbatasan ini mencakup jaringan internet yang belum stabil, perangkat keras yang usang, dan kurangnya sistem keamanan data yang andal.

Teknologi yang dipilih dalam perancangan konsep pada penelitian ini diutamakan untuk menggunakan teknologi yang bersifat *open-source* yang berlandaskan kepada Surat Edaran Menteri Negara Pendayagunaan Aparatur Negara tertanggal 30 Maret 2009 mengenai Pemanfaatan Perangkat Lunak Legal dan *Open Source Software (OSS)* dengan nomor SE/01/M.PAN/3/2009 yang merujuk kepada Surat Edaran Menteri Komunikasi dan Informatika nomor 05/SE/M.KOMINFO/10/2005 tentang Pemakaian dan Pemanfaatan Penggunaan Piranti Lunak Legal di Lingkungan Instansi Pemerintah.

Selain itu di tengah perkembangan teknologi dan semakin banyaknya aplikasi yang digunakan di dalam lingkungan pemerintahan dewasa ini, Pemerintah Indonesia juga tengah menyiapkan *super apps* untuk layanan publik terpadu yang dalam implementasinya berlandaskan pada *data driven policy* untuk menghasilkan Satu Data Indonesia. Satu Data Indonesia ini sudah memiliki landasan hukum berdasarkan Perpres No. 39 tahun 2019 tentang Satu Data Indonesia [7] dan Permen Kominfo No. 1 tahun 2023 tentang Interoperabilitas Data Dalam Penyelenggaraan Sistem Pemerintahan Berbasis Elektronik dan Satu Data Indonesia [8]. Penulis berharap penelitian ini dapat membantu pengembangan *super apps* di Indonesia sehingga seluruh layanan yang diberikan pemerintah kepada masyarakat maupun layanan antar instansi pemerintahan dapat berjalan dengan baik dan saling terintegrasi satu sama lain yang tentunya harus tetap terjaga keamanan dan privasi datanya seperti sudah diatur pada Undang-Undang No. 27 tahun 2022 mengenai Perlindungan Data Pribadi (PDP).

B. Rumusan Masalah

Berdasarkan latar belakang permasalahan mengenai kompleksitas integrasi sistem dalam ekosistem *smart city*, maka rumusan masalah dalam penelitian ini adalah sebagai berikut:

RQ1. Bagaimana merancang standar komunikasi data yang dapat digunakan untuk integrasi antar aplikasi dalam ekosistem *smart city*?
RQ2. Bagaimana peran *API Gateway* dalam menyederhanakan komunikasi antar layanan yang dikembangkan secara modular (*microservices*)?
RQ3. Apa saja tantangan dan kompleksitas yang dihadapi dalam mengimplementasikan *API Gateway* sebagai solusi integrasi sistem *smart city*?
RQ4. Sejauh mana pendekatan arsitektur *microservices* dan penggunaan *API Gateway* dapat mengurangi redundansi fungsi dan meningkatkan efisiensi pengembangan sistem?
RQ5. Bagaimana cara mengukur efektivitas penggunaan standar komunikasi data melalui *API Gateway* dalam mendukung integrasi sistem *smart city*?

C. Batasan Masalah

Dalam penelitian ini, penulis membatasi ruang lingkup pada komunikasi data antar aplikasi perangkat lunak dalam ekosistem *smart city*. Fokus penelitian adalah pada perancangan standar komunikasi data berbasis *API Gateway* yang digunakan untuk mengintegrasikan layanan-layanan yang dikembangkan menggunakan pendekatan *microservices*. Penelitian tidak membahas komunikasi antara perangkat keras (seperti *IoT*, sensor, dsb.) dan perangkat lunak, karena umumnya perangkat keras telah memiliki protokol atau *SDK* bawaan dari manufaktur masing-masing.

D. Tujuan

Merancang konsep standar komunikasi data berbasis *API Gateway* untuk mendukung integrasi sistem dalam ekosistem *smart city* berbasis arsitektur microservices. Berdasarkan poin-poin *Research Questions (RQ)* pada Rumusan Masalah yang sudah dituliskan di atas berikut ini tujuan penelitian / *Research Objectives (RO)* yang akan dilakukan oleh penulis:

RO1. Menganalisis kebutuhan komunikasi data antar aplikasi dalam ekosistem *smart city* untuk mengidentifikasi kendala interoperabilitas dan kebutuhan akan standarisasi.
RO2. Merancang peran dan fungsi *API Gateway* sebagai jalur komunikasi utama antar layanan *microservices* agar mampu menyederhanakan proses integrasi sistem.
RO3. Mengevaluasi tantangan dan kompleksitas teknis dalam penerapan *API Gateway* dibandingkan integrasi konvensional antar sistem *(point-to-point integration).*
RO4. Merancang sekaligus membangun model integrasi sistem berbasis *microservices* dan *API Gateway* dengan fokus pada eliminasi duplikasi fungsi serta peningkatan efisiensi pengembangan aplikasi.
RO5. Mengukur efektivitas komunikasi data terstandar melalui pengujian pada *prototype* sistem, dengan metrik seperti waktu respons, tingkat keberhasilan pertukaran data, dan pengurangan redundansi fungsi.

II. METODE

Penelitian ini menggunakan pendekatan eksperimen rekayasa perangkat lunak yang bertujuan untuk merancang, mengimplementasikan, dan mengevaluasi model integrasi sistem berbasis arsitektur *microservices* dan *API Gateway* dalam konteks pengembangan ekosistem smart city. Penelitian difokuskan pada komunikasi data antar perangkat lunak (aplikasi), tidak termasuk komunikasi antara perangkat keras seperti *IoT* dan perangkat lunak.

A. Tahapan Penelitian

1. Studi Literatur:
   a. Mengkaji konsep *smart city*, integrasi sistem, arsitektur *microservices*, *API Gateway*, dan standar komunikasi data.
   b. Mengumpulkan referensi dari kebijakan nasional seperti Satu Data Indonesia, Perpres No. 39 Tahun 2019, dan studi terdahulu terkait integrasi sistem *smart city*.
2. Analisis Kebutuhan Komunikasi Data:
   a. Mengidentifikasi pola pertukaran data dan tantangan komunikasi antar aplikasi.
3. Perancangan Arsitektur Sistem:
   a. Merancang arsitektur *microservices* untuk modularisasi aplikasi *smart city*.
   b. Mendesain skema komunikasi data dan peran *API Gateway* dalam pengaturan pertukaran data antar layanan.
4. Pengembangan Konsep:
   a. Mengimplementasikan *prototype* sistem dengan 2–3 *service* fungsional menggunakan *microservices*.
   b. Membangun *API Gateway* sebagai perantara komunikasi antar *service*, serta menerapkan standar format data (misalnya *JSON schema* atau *OpenAPI*).
5. Evaluasi dan Validasi:
   a. Menguji performa dan efektivitas integrasi menggunakan metrik:
      i. *Response time* (rata-rata waktu permintaan–balasan *API*),
      ii. *Success rate* (jumlah keberhasilan komunikasi antar *service*),
      iii. Jumlah fungsi redundan yang dieliminasi,
      iv. Tingkat keterpakaian ulang service (*reusability*).

   b. Evaluasi dilakukan melalui simulasi permintaan data antar *service* dan pengamatan hasilnya.
6. Analisis dan Kesimpulan:
   a. Menganalisis hasil eksperimen untuk menjawab pertanyaan penelitian.
   b. Menyusun kesimpulan dan rekomendasi terhadap penerapan standar komunikasi data di *smart city*.

B. Teknik Analisis Data

- Analisis kuantitatif terhadap metrik performa sistem (*response time*, *success rate*, dsb).
- Analisis kualitatif terhadap tantangan teknis dan pengalaman implementasi.
- Komparatif antara pendekatan *microservices* dan *API Gateway* dibandingkan dengan pendekatan integrasi konvensional *(one-to-one)*.

C. Perancangan Strategi Implementasi Arsitektur Berbasis Microservices

Dalam mengimplementasikan arsitektur sistem *smart city* yang mengadopsi pendekatan *microservices*, diperlukan strategi yang memperhatikan kondisi nyata di lingkungan pemerintahan, khususnya keterbatasan pada sistem lama (*legacy*) yang tidak memungkinkan untuk diubah secara menyeluruh. Oleh karena itu, pendekatan yang diusulkan adalah dengan membangun layanan-layanan baru secara modular dan terpisah, yang dapat langsung mengakses data dari sistem *legacy* melalui skema integrasi terbatas seperti akses basis data *read-only* atau pemanfaatan adaptor. Setiap layanan microservice dirancang untuk menangani domain fungsional yang spesifik (seperti layanan kependudukan, transportasi, dan pelaporan), dan semua lalu lintas komunikasi antar aplikasi dikelola melalui sebuah *API Gateway* sebagai titik konsolidasi serta pengendalian akses layanan.

Agar implementasi arsitektur ini dapat berjalan efektif dan berkelanjutan, terdapat sejumlah prasyarat penting yang harus dipenuhi sebelum tahap penerapan. Prasyarat tersebut mencakup kesiapan infrastruktur dasar seperti ketersediaan container runtime dan orkestrasi (*Docker* dan *Kubernetes*), sistem kendali versi dan *CI/CD* (*GitHub Actions*), serta sistem monitoring (*Prometheus* dan *Grafana*). Selain itu, faktor non-teknis seperti keberadaan tim pengelola arsitektur, dukungan lintas instansi, dan penerapan tata kelola akses data berbasis peran dan kebijakan keamanan juga menjadi kunci sukses implementasi. Strategi ini memberikan jalur migrasi yang fleksibel dari sistem lama menuju arsitektur yang lebih modern, tanpa mengganggu operasional eksisting dan tetap menjaga prinsip interoperabilitas antar layanan pemerintah daerah.

III. HASIL DAN PEMBAHASAN

A. Evaluasi Standar Format Komunikasi Data

Dalam konteks integrasi sistem *smart city* yang melibatkan berbagai layanan modular, dibutuhkan format komunikasi data yang seragam, terstruktur, dan mudah dipahami oleh seluruh layanan. Oleh karena itu, penelitian ini merancang dan menerapkan standar format komunikasi berbasis *RESTful API* dengan struktur data *JSON* sebagai media pertukaran informasi antar layanan.

Standar ini diadopsi secara konsisten oleh seluruh *service* (layanan pelaporan masyarakat, layanan informasi transportasi, dan layanan kependudukan) dan telah diuji dalam simulasi. Implementasi format *JSON* ini mencakup dua kategori utama yaitu *response* sukses dan *response error*, dengan tambahan elemen metadata untuk mendukung skalabilitas dan keandalan sistem.

Dengan penerapan format komunikasi yang terstandarisasi ini, sistem tidak hanya menjadi lebih robust dan terbuka, tetapi juga lebih siap untuk mendukung integrasi jangka panjang antar layanan *smart city*.

Standarisasi *Response* Sukses

Seluruh *response* yang berhasil akan menggunakan pola tertentu yang dapat membantu sistem *backend* dalam melakukan *response* terhadap *request* dari sistem *frontend*.

```
{
    "status": "success",
    "meta": {
        "total": <total data keseluruhan>,
        "page": <page saat ini>,
        "limit": <limit data yang diambil per page>,
        "retrieved_at": "<tanggal pengambilan data>"
    },
    "data": [
        {
            <objek data>
        }, {
            …
        }
    ]
}
```

Elemen metadata disediakan untuk mendukung fitur pagination dan pengukuran konteks sistem seperti jumlah total data, waktu akses *(retrieved_at)*, dan sebagainya. Format ini memudahkan antarmuka *frontend* dan sistem eksternal dalam membaca serta menavigasi *response* data.

Standarisasi *Response Error*

Seluruh kondisi gagal (input tidak valid, data tidak ditemukan, *server error*, dsb.) disajikan dalam format *error* standar.

```
{
   "status": "success",
   "error_code": <kode angka error>,
   "error_name": <nama atau jenis error>,
   "message":  "<error message yang menggambarkan
               masalah yang ditemukan>"
}
```

Struktur ini memungkinkan kesalahan ditangani secara seragam di sisi *frontend* atau antar sistem. *Field*

error_name menggunakan konvensi penamaan *UPPERCASE* untuk kejelasan *parsing* logika pemrosesan.

Standarisasi Kode Kesalahan

Dalam sistem integrasi layanan berbasis microservices, pengelolaan kesalahan *(error handling)* yang konsisten sangat penting untuk memastikan transparansi, kemudahan *debugging*, serta keandalan sistem dalam mengatasi kondisi tak terduga. Untuk itu, penelitian ini menetapkan standar kode kesalahan yang digunakan secara seragam di seluruh layanan dalam sistem *smart city*.

Tabel berikut merupakan contoh standar kode kesalahan yang diimplementasikan, yang mengacu pada kode status *HTTP* yang umum digunakan serta dilengkapi dengan penamaan *error (error name)*, *error message* dan juga deskripsi kesalahan untuk memperjelas penyebab kesalahan.

TABEL I
CONTOH TABLE STANDAR KODE KESALAHAN

Error Code	***Error Name***	***Error Message***	**Deskripsi**
400	*INVALID_INPUT*	Input tidak valid, cek kembali *field* data yang dikirimkan apakah sudah sesuai atau belum.	Terjadi jika data yang dikirimkan tidak lengkap, kosong, atau salah format.
401	*UNAUTHORIZED*	Autentikasi gagal, cek kembali penggunan token yang sesuai.	Terjadi jika permintaan tidak menyertakan *API Key* atau menggunakan token yang tidak valid.
403	*FORBIDDEN*	Anda tidak memiliki akses ke fitur ini.	Digunakan jika pengguna tidak memiliki hak akses terhadap *resource* tertentu.
404	*NOT_FOUND*	Data tidak ditemukan.	Terjadi jika *resource* yang diminta tidak tersedia dalam sistem.
500	*SERVER_ERROR*	Terjadi kesalahan pada sistem.	Digunakan jika terjadi kesalahan internal yang tidak terduga di sisi server.

Tentu saja pada lingkungan produksi, standar kode kegagalan ini harus disesuaikan dan didokumentasikan sedemikian rupa agar dapat terus digunakan oleh sistem internal maupun sistem eksternal.

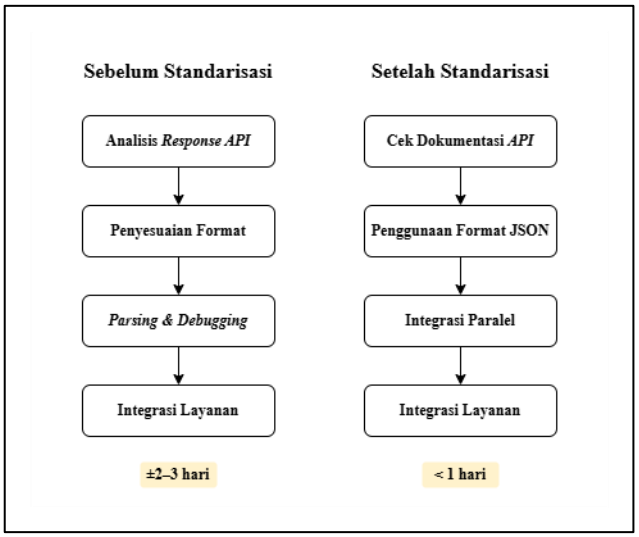
Analisis Dampak Standarisasi Format Komunikasi

Untuk mengukur dampak nyata dari penerapan format komunikasi standar ini, dilakukan pengamatan terhadap proses pengembangan dan integrasi layanan selama simulasi sistem. Hasil analisis menunjukkan peningkatan signifikan dalam beberapa aspek berikut:

Waktu Integrasi Lintas Layanan— Sebelum penggunaan format standar, proses integrasi antar layanan (misalnya antara layanan pelaporan dan layanan kependudukan) memerlukan proses penyesuaian format *response* secara manual. Hal ini menyebabkan keterlambatan hingga 2-3 hari per integrasi. Setelah penerapan struktur *JSON* terstandarisasi, proses integrasi dapat dilakukan dalam waktu kurang dari satu hari per layanan, karena struktur *response* dan kesalahan telah bersifat seragam dan dapat diprediksi. Selain itu dengan bantuan *Postman* atau *Swagger*, setiap integrasi yang dibutuhkan dapat terdokumentasikan dengan baik sehingga meningkatkan konsistensi teknis yang merupakan fondasi penting dalam integrasi sistem.

TABEL II
PERBANDINGAN DAMPAK STANDARISASI FORMAT KOMUNIKASI

Aspek	**Sebelum Standarisasi**	**Sesudah Standarisasi**
Waktu integrasi per layanan	±2–3 hari	< 1 hari
Jumlah *error parsing* JSON	Tinggi	Rendah / Hampir nol
Konsistensi *response* antar *API*	Tidak Konsisten	Konsisten



GAMBAR 1 PERBANDINGAN ALUR INTEGRASI SEBELUM DAN SESUDAH STANDARISASI

Reduksi *Error Parsing* dan *Debugging*— Kesalahan parsing response antar layanan berkurang drastis karena struktur respons *JSON* memiliki skema yang tetap. Hal ini memungkinkan sistem monitoring mendeteksi ketidaksesuaian lebih awal.

Pengembangan *Frontend* Secara Paralel— Dengan adanya format *success* dan *error* yang konsisten, tim frontend dapat mengembangkan antarmuka aplikasi bahkan sebelum backend selesai, cukup dengan dokumentasi *Swagger* dan skema *JSON*. Ini mempercepat siklus pengembangan sistem dari kedua sisi.

Dukungan terhadap Observabilitas dan Audit— Struktur metadata menyediakan peluang pencatatan *log* yang kaya, seperti waktu pengambilan data, batasan permintaan, dan jumlah hasil yang dikembalikan. Hal ini penting untuk sistem publik yang membutuhkan audit trail, serta berguna untuk manajemen beban trafik *(load-aware architecture)*.

Dari hasil observasi tersebut, dapat disimpulkan bahwa penggunaan format komunikasi data yang terstandarisasi dalam bentuk struktur *JSON* tidak hanya memberikan konsistensi teknis, tetapi juga mendukung efisiensi kerja lintas tim, mempercepat integrasi, dan meningkatkan keandalan sistem dalam jangka panjang. Standarisasi ini menjadi fondasi penting dalam membangun arsitektur sistem *smart city* yang modular, terbuka, dan mudah dikembangkan.

B. Evaluasi Sistem

Penulis melakukan pengujian dengan mensimulasikan interaksi antar layanan melalui *API Gateway* menggunakan 3 (tiga) *prototype* layanan dengan beberapa jenis permintaan (*GET*, *POST*) yang dikirimkan dari sisi *backend* ke pengguna *frontend*. Dalam pembuatan konsep ini, penulis melakukan 100 kali pengujian pada masing-masing iterasi untuk masing-masing metode. Tabel di bawah ini menampilkan kesimpulan dari hasil pengujian yang akan penulis jabarkan satu per satu secara lebih terperinci baik dari sisi analisis maupun kesimpulan.

TABEL III
KESIMPULAN HASIL PENGUJIAN

Metrik	Konvensional	*Microservices + API Gateway*
Response Time (ms)	180.9	211.3
Success Rate (%)	87.1	97.2
CPU Usage (%) per service	7-12	4-7
Memory Usage (MB) per service	80-100	60-80
Code Reuse Ratio	Rendah, fungsi validasi / format / *logging* diulang di tiap *service*	~40% fungsi umum dapat dikurangi, dikelola terpusat di *API Gateway*, meningkatkan efisiensi pengembangan
Keamanan	Terbatas, autentikasi dilakukan *per service* (tidak terstandarisasi)	Tersentralisasi *via API key* di *API Gateway*, belum menggunakan *mutual TLS*, namun sudah relatif lebih aman dan lebih mudah dikontrol
Skalabilitas	Kompleks, karena integrasi *point-to-point* menyebabkan rigiditas struktur	Mudah ditingkatkan, layanan bersifat independent dan dapat dilakukan *auto-scaling*, sangat cocok untuk sistem berkembang

Response Time— Terdapat tambahan waktu karena *overhead* autentikasi dan *routing*, namun tetap dalam batas efisiensi yang wajar (<300 *ms*). Hasil pengujian menunjukkan penambahan *latency* sebesar 30.4 *ms* (16,8%) ketika menggunakan *API Gateway*, sebuah *trade-off* yang dapat diterima mengingat peningkatan *success rate* sebesar 10,1%.

Success Rate— Kenaikan persentase *success rate* menunjukkan stabilitas dan konsistensi layanan lebih tinggi melalui penggunaan *API Gateway*. Implementasi *API Gateway* berhasil menekan kegagalan komunikasi dari 12,9% ke hanya 2,9%, dengan pola kegagalan yang kini terpusat dan lebih mudah ditelusuri.

Penggunaan *Resource*— Berdasarkan hasil pengujian dan pemantauan, pendekatan integrasi sistem menggunakan *API Gateway* dan *microservices* yang di-deploy dalam lingkungan *Kubernetes* menunjukkan efisiensi yang lebih tinggi dibandingkan pendekatan konvensional melalui penurunan beban *CPU* per layanan, penggunaan *memory* yang lebih terkendali, and optimisasi *runtime* melalui *Kubernetes*. Pada pendekatan konvensional, sumber daya cenderung terbuang karena fungsi duplikatif, *deployment* manual, serta ketiadaan kontrol terpusat terhadap konsumsi *resource*. Sumber daya yang terbuang ini dalam jangka panjang lebih besar dampaknya daripada konsumsi *resource* dari penambahan *tools API Gateway* dan *Kubernetes* yang tidak digunakan pada pendekatan konvensional. Dengan demikian, arsitektur berbasis *API Gateway* dan *Kubernetes* tidak hanya mendukung pengelolaan sistem yang lebih efisien, tetapi juga meningkatkan performa dan skalabilitas sistem integrasi *smart city* secara berkelanjutan.

Reduksi fungsi redundan dan reusabilitas layanan— Pendekatan sistem konvensional memiliki kelemahan berupa duplikasi fungsi di banyak layanan, seperti validasi pengguna, konversi data, dan format respon, yang harus diimplementasikan berulang sehingga menambah beban pengembangan, meningkatkan risiko inkonsistensi, dan menyulitkan pemeliharaan; setelah diintegrasikan melalui *API Gateway*, fungsi-fungsi umum tersebut dipusatkan dalam *middleware* yang reusable lintas layanan, meningkatkan *code reuse ratio* sekitar 40% dan menyederhanakan kompleksitas kode tiap service, sehingga mempercepat *debugging*, mempermudah pengembangan, dan memungkinkan setiap layanan fokus pada domainnya masing-masing, menjadikan pendekatan ini jauh lebih efisien dan cocok untuk skala besar seperti *smart city*.

Keamanan dan skalabilitas— Penggunaan *API Gateway* secara signifikan meningkatkan keamanan layanan, khususnya dalam komunikasi eksternal, dengan mekanisme autentikasi terpusat melalui *API Key* yang mempermudah kontrol dan pemantauan, meskipun belum mencakup fitur keamanan lanjutan seperti *mTLS* atau *OAuth2*; di sisi lain, arsitektur *microservices* dengan *API Gateway* juga menawarkan skalabilitas yang lebih baik berkat kemampuan pengembangan, *deployment*, dan

scaling mandiri tiap layanan menggunakan *Kubernetes*, berbeda dengan arsitektur konvensional yang masih *point-to-point* dan cenderung statis serta sulit diadaptasi untuk kebutuhan pengembangan jangka panjang seperti *smart city*.

Trade-off: Success Rate vs. Response Time— Hasil eksperimen menunjukkan bahwa penerapan *API Gateway* meningkatkan *success rate* dari sekitar 88% menjadi 97% berkat validasi awal, *routing* cerdas, *rate limiting*, serta *monitoring* terpusat yang menjaga stabilitas dan keandalan sistem, meskipun diiringi kenaikan *response time* dari sekitar 180 *ms* menjadi 210 *ms* akibat adanya tambahan proses seperti autentikasi, transformasi *payload*, dan pencatatan trafik di *API Gateway*; *trade-off* ini dinilai wajar dan layak diterima terutama dalam konteks *smart city*, di mana keberhasilan dan kestabilan layanan publik lebih diutamakan dibanding sekadar kecepatan.

Trade-off: Latency (Response Time) vs. Maintainability— Penerapan arsitektur *microservices* berbasis *API Gateway* memang menyebabkan peningkatan latency sekitar 30 *ms*—dari rata-rata 180 *ms* pada sistem konvensional menjadi 210 *ms*—karena tambahan proses seperti *routing*, validasi, autentikasi, dan *logging*; namun, kompromi ini dinilai wajar dan dapat diterima dalam konteks layanan publik *non-realtime*, mengingat keuntungan besar yang diperoleh dari sisi *maintainability*, termasuk sentralisasi fungsi umum, modularitas layanan, observabilitas terpusat, serta kemudahan pemeliharaan dan pengembangan sistem secara berkelanjutan, sehingga *trade-off* ini mendukung fleksibilitas dan skalabilitas sistem *smart city* di jangka panjang.

C. Strategi Implementasi pada Sistem Smart City Pemerintah

Penerapan arsitektur *microservices* dalam sistem *smart city* pemerintahan menghadapi tantangan signifikan akibat dominasi sistem *legacy* yang telah berjalan bertahun-tahun dan tidak dapat digantikan secara langsung karena keterbatasan anggaran, sumber daya manusia, serta ketergantungan operasional. Penelitian ini mengusulkan pendekatan integrasi bertahap, dengan membangun layanan-layanan *microservices* baru yang secara modular mengakses data dari sistem lama melalui mekanisme *read-only* atau adaptor layanan. Setiap layanan diatur melalui *API Gateway* sebagai titik sentral komunikasi, serta dikemas dalam container dan dikelola menggunakan *Kubernetes* untuk menjamin skalabilitas dan isolasi antar layanan.

Agar strategi implementasi ini dapat berhasil, diperlukan sejumlah prasyarat teknis dan non-teknis, seperti infrastruktur kontainerisasi yang memadai, penerapan *pipeline CI/CD* melalui *GitHub Actions*, serta sistem pemantauan layanan menggunakan *Prometheus* dan *Grafana*. Di samping itu, pengelolaan tata kelola arsitektur, standar *API* lintas instansi, serta pengaturan manajemen akses data berdasarkan prinsip keamanan informasi menjadi elemen penting dalam mendukung interoperabilitas sistem. Pendekatan ini memungkinkan proses modernisasi arsitektur sistem informasi pemerintah dilakukan secara terukur dan berkelanjutan, tanpa mengganggu sistem eksisting yang masih berjalan.

Melalui integrasi bertahap ini, pemerintah daerah dapat membangun fondasi arsitektur layanan digital yang lebih fleksibel, efisien, dan sesuai dengan regulasi nasional seperti UU Perlindungan Data Pribadi, Perpres Satu Data Indonesia, dan kebijakan interoperabilitas dari Kementerian Komunikasi dan Informatika. Hasilnya adalah sistem *smart city* yang lebih adaptif, terstandarisasi, dan siap dikembangkan di masa mendatang.

IV. SIMPULAN

Tujuan utama dari penelitian ini adalah menyusun, membangun, dan menguji model integrasi sistem untuk lingkungan *smart city* dengan pendekatan arsitektur microservices dan pemanfaatan *API Gateway*. Berdasarkan hasil analisis dan evaluasi dari penelitian yang telah dilakukan, diperoleh beberapa temuan utama yang menjawab seluruh pertanyaan penelitian (*Research Questions*) berdasarkan rumusan masalah pada subbab I.B sebagai berikut:

1. Menjawab *RQ1*: Bagaimana merancang standar komunikasi data yang dapat digunakan untuk integrasi antar aplikasi dalam ekosistem *smart city*?— Penelitian ini berhasil merancang standar komunikasi data yang dapat digunakan untuk integrasi antar aplikasi dalam ekosistem *smart city*. Standar ini diimplementasikan melalui pendekatan *RESTful API* dengan format *JSON* yang konsisten, dan dikontrol secara terpusat melalui *API Gateway*. Dengan rancangan ini, proses pertukaran data antar layanan menjadi lebih seragam dan dapat diandalkan. Selain itu dari sisi keamanan, sentralisasi proses autentikasi oleh *API Gateway* memperkuat aspek keamanan sistem serta menyederhanakan mekanisme pengaturan hak akses dan proteksi data antar aplikasi.
2. Menjawab *RQ2*: Bagaimana peran *API Gateway* dalam menyederhanakan komunikasi antar layanan yang dikembangkan secara modular (*microservices*)?— Peran *API Gateway* terbukti signifikan dalam menyederhanakan komunikasi antar layanan. Fitur-fitur seperti *routing*, autentikasi, *logging*, dan validasi input yang sebelumnya harus diimplementasikan di setiap *service* kini terpusat di *API Gateway*, sehingga mengurangi kompleksitas dan inkonsistensi pada level *service* secara individual. Pendekatan ini juga mempermudah pemantauan dan pengelolaan trafik layanan. Integrasi dengan pendekatan *microservices* dan *API Gateway* memiliki keunggulan signifikan dibandingkan model *point-to-point*, terutama dalam hal skalabilitas, pemeliharaan, dan pengelolaan lalu lintas data.

3. Menjawab *RQ3*: Apa saja tantangan dan kompleksitas yang dihadapi dalam mengimplementasikan *API Gateway* sebagai solusi integrasi sistem *smart city*?— Tantangan yang dihadapi dalam implementasi *API Gateway* antara lain adalah penambahan *response time* / latency sekitar 30 *ms* akibat proses *routing* dan validasi terpusat. Selain itu, pengelolaan konfigurasi *route* dan autentikasi memerlukan kehati-hatian agar tidak menjadi *single point of failure*. Namun, tantangan tersebut dapat diatasi dengan pengaturan sistem yang tepat dan pemanfaatan *container orchestration* melalui *Kubernetes*. Meskipun terjadi kenaikan *latency* sekitar 30 *ms*, keuntungan dari sisi *maintainability* jauh lebih besar, dengan pengurangan duplikasi kode dan penyederhanaan arsitektur sistem.
4. Menjawab *RQ4*: Sejauh mana pendekatan arsitektur *microservices* dan penggunaan *API Gateway* dapat mengurangi redundansi fungsi dan meningkatkan efisiensi pengembangan sistem?— Pendekatan berbasis *microservices* dan *API Gateway* terbukti dapat mengurangi redundansi fungsi seperti validasi input, *logging*, dan autentikasi. Penelitian ini mencatat *peningkatan code reuse ratio* sebesar ±40%, yang berdampak langsung pada efisiensi pengembangan dan pemeliharaan sistem. Arsitektur ini memungkinkan pengembangan layanan secara modular, terisolasi, dan dapat diskalakan secara independen. Arsitektur modular juga memungkinkan setiap layanan dikembangkan dan diuji secara terpisah. Hal ini cocok untuk kebutuhan *smart city* yang dinamis dan terus berkembang.
5. Menjawab *RQ5*: Bagaimana cara mengukur efektivitas penggunaan standar komunikasi data melalui *API Gateway* dalam mendukung integrasi sistem *smart city*?— Efektivitas pendekatan integrasi ini diukur melalui metrik performa sistem, antara lain: *success rate*, *response time*, dan *resource usage*. Dibandingkan pendekatan konvensional, *success rate* meningkat dari 87% menjadi 97%, dengan konsumsi *CPU* menurun dari 7–12% menjadi 4–7% dan penggunaan *memory* yang lebih efisien berkat manajemen *container* oleh *Kubernetes*. Ini menunjukkan bahwa penggunaan standar komunikasi melalui *API Gateway* efektif mendukung integrasi sistem *smart city*.

Secara keseluruhan, pendekatan ini layak untuk diterapkan dalam pengembangan sistem *smart city* di Indonesia, terutama sebagai fondasi dalam pengembangan *super apps* pemerintah dan dukungan terhadap kebijakan Satu Data Indonesia.

REFERENSI

[1] G. Ortiz *et al.*, "*A microservice architecture for real-time IoT data processing: A reusable Web of things approach for smart ports*," *Comput. Stand. Interfaces*, vol. 81, p. 103604, Apr. 2022, doi: 10.1016/j.csi.2021.103604.

[2] N. Gavrilović and A. Mishra, "*Software architecture of the internet of things (IoT) for smart city, healthcare and agriculture: analysis and improvement directions*," *J. Ambient Intell. Humaniz. Comput.*, vol. 12, no. 1, pp. 1315–1336, Jan. 2021, doi: 10.1007/s12652-020-02197-3.

[3] "*Web of Things (WoT) Architecture*." Accessed: Nov. 04, 2022. [Online]. Available: https://www.w3.org/TR/wot-architecture/

[4] "*Web of Things (WoT) Thing Description.*" Accessed: Nov. 04, 2022. [Online]. Available: https://www.w3.org/TR/wot-thing-description/

[5] R. Xu, W. Jin, and D. Kim, "*Microservice Security Agent Based On API Gateway in Edge Computing*," *Sensors*, vol. 19, no. 22, Art. no. 22, Jan. 2019, doi: 10.3390/s19224905.

[6] J. Zhao, S. Jing, and L. Jiang, "*Management of API Gateway Based on Micro-service Architecture*," *J. Phys. Conf. Ser.*, vol. 1087, p. 032032, Sep. 2018, doi: 10.1088/1742-6596/1087/3/032032.

[7] Presiden Republik Indonesia, "Peraturan Presiden Republik Indonesia Tentang Satu Data Indonesia." 2019.

[8] Menteri Komunikasi dan Informatika Republik Indonesia, "Peraturan Menteri Komunikasi dan Informatika Republik Indonesia Tentang Interoperabilitas Data Dalam Penyelenggaraan Sistem Pemerintahan Berbasis Elektronik Dan Satu Data Indonesia." 2023.